

CX

Generated by Doxygen 1.8.6

Sun Aug 31 2014 17:33:25

## Contents

<b>1</b>	<b>Main Page</b>	<b>1</b>
1.1	Installation . . . . .	1
1.2	System Requirements . . . . .	1
1.3	Examples and Tutorials . . . . .	1
1.4	Creating a Blank Experiment . . . . .	3
<b>2</b>	<b>Blocking Code</b>	<b>3</b>
<b>3</b>	<b>Framebuffers and Buffer Swapping</b>	<b>4</b>
<b>4</b>	<b>Modules</b>	<b>5</b>
<b>5</b>	<b>Program Model</b>	<b>5</b>
<b>6</b>	<b>license</b>	<b>8</b>
<b>7</b>	<b>Module Index</b>	<b>8</b>
7.1	Modules . . . . .	8
<b>8</b>	<b>Namespace Index</b>	<b>9</b>
8.1	Namespace List . . . . .	9
<b>9</b>	<b>Hierarchical Index</b>	<b>9</b>
9.1	Class Hierarchy . . . . .	9
<b>10</b>	<b>Class Index</b>	<b>12</b>
10.1	Class List . . . . .	12
<b>11</b>	<b>Module Documentation</b>	<b>14</b>
11.1	Data . . . . .	14
11.1.1	Detailed Description . . . . .	14
11.2	Entry Point . . . . .	15
11.2.1	Detailed Description . . . . .	15
11.2.2	Function Documentation . . . . .	15
11.2.3	Variable Documentation . . . . .	15
11.3	Input Devices . . . . .	16
11.3.1	Detailed Description . . . . .	16
11.4	Message Logging . . . . .	17
11.4.1	Detailed Description . . . . .	17
11.4.2	Enumeration Type Documentation . . . . .	17

11.5 Randomization . . . . .	18
11.5.1 Detailed Description . . . . .	18
11.6 Sound . . . . .	19
11.6.1 Detailed Description . . . . .	19
11.7 Timing . . . . .	20
11.7.1 Detailed Description . . . . .	20
11.7.2 Variable Documentation . . . . .	20
11.8 Utility . . . . .	21
11.8.1 Detailed Description . . . . .	21
11.9 Video . . . . .	22
11.9.1 Detailed Description . . . . .	22
<b>12 Namespace Documentation</b>	<b>23</b>
12.1 CX Namespace Reference . . . . .	23
12.1.1 Detailed Description . . . . .	24
12.1.2 Function Documentation . . . . .	24
12.2 CX::Algo Namespace Reference . . . . .	25
12.2.1 Detailed Description . . . . .	25
12.2.2 Function Documentation . . . . .	26
12.3 CX::Draw Namespace Reference . . . . .	28
12.3.1 Detailed Description . . . . .	29
12.3.2 Function Documentation . . . . .	29
12.4 CX::Instances Namespace Reference . . . . .	39
12.4.1 Detailed Description . . . . .	39
12.5 CX::Synth Namespace Reference . . . . .	39
12.5.1 Detailed Description . . . . .	40
12.5.2 Function Documentation . . . . .	40
12.6 CX::Util Namespace Reference . . . . .	42
12.6.1 Detailed Description . . . . .	43
12.6.2 Enumeration Type Documentation . . . . .	43
12.6.3 Function Documentation . . . . .	44
<b>13 Class Documentation</b>	<b>54</b>
13.1 CX::Synth::Adder Class Reference . . . . .	54
13.1.1 Detailed Description . . . . .	54
13.1.2 Member Function Documentation . . . . .	54
13.2 CX::Synth::AdditiveSynth Class Reference . . . . .	54
13.2.1 Detailed Description . . . . .	55

13.2.2	Member Enumeration Documentation	55
13.2.3	Member Function Documentation	55
13.3	CX::Algo::BlockSampler< T > Class Template Reference	58
13.3.1	Detailed Description	58
13.3.2	Constructor & Destructor Documentation	58
13.3.3	Member Function Documentation	58
13.4	CX::Synth::Clamper Class Reference	59
13.4.1	Detailed Description	59
13.4.2	Member Function Documentation	60
13.5	CX::CX_SlidePresenter::Configuration Struct Reference	60
13.5.1	Detailed Description	60
13.6	CX::CX_SoundStream::Configuration Struct Reference	61
13.6.1	Detailed Description	61
13.6.2	Member Data Documentation	61
13.7	CX::CX_BaseClockImplementation Class Reference	62
13.7.1	Detailed Description	62
13.8	CX::Util::CX_BaseUnitConverter Class Reference	62
13.8.1	Detailed Description	62
13.8.2	Member Function Documentation	63
13.9	CX::CX_Clock Class Reference	63
13.9.1	Detailed Description	63
13.9.2	Member Function Documentation	63
13.10	CX::Util::CX_CoordinateConverter Class Reference	66
13.10.1	Detailed Description	66
13.10.2	Constructor & Destructor Documentation	67
13.10.3	Member Function Documentation	67
13.11	CX::CX_DataFrame Class Reference	69
13.11.1	Detailed Description	71
13.11.2	Member Function Documentation	71
13.12	CX::CX_DataFrameCell Class Reference	79
13.12.1	Detailed Description	80
13.12.2	Constructor & Destructor Documentation	80
13.12.3	Member Function Documentation	80
13.13	CX::CX_DataFrameColumn Class Reference	82
13.13.1	Detailed Description	82
13.13.2	Member Function Documentation	82
13.14	CX::CX_DataFrameRow Class Reference	83

13.14.1 Detailed Description . . . . .	83
13.15CX::Util::CX_DegreeToPixelConverter Class Reference . . . . .	83
13.15.1 Detailed Description . . . . .	83
13.15.2 Constructor & Destructor Documentation . . . . .	84
13.15.3 Member Function Documentation . . . . .	84
13.16CX::CX_Display Class Reference . . . . .	86
13.16.1 Detailed Description . . . . .	87
13.16.2 Member Function Documentation . . . . .	87
13.17CX::CX_InputManager Class Reference . . . . .	94
13.17.1 Detailed Description . . . . .	95
13.17.2 Member Function Documentation . . . . .	95
13.18CX::CX_Joystick Class Reference . . . . .	96
13.18.1 Detailed Description . . . . .	96
13.18.2 Member Function Documentation . . . . .	96
13.19CX::CX_Keyboard Class Reference . . . . .	97
13.19.1 Detailed Description . . . . .	98
13.19.2 Member Enumeration Documentation . . . . .	98
13.19.3 Member Function Documentation . . . . .	98
13.20CX::Util::CX_LapTimer Class Reference . . . . .	100
13.20.1 Detailed Description . . . . .	101
13.20.2 Constructor & Destructor Documentation . . . . .	101
13.20.3 Member Function Documentation . . . . .	101
13.21CX::Util::CX_LengthToPixelConverter Class Reference . . . . .	102
13.21.1 Detailed Description . . . . .	102
13.21.2 Constructor & Destructor Documentation . . . . .	102
13.21.3 Member Function Documentation . . . . .	102
13.22CX::CX_Logger Class Reference . . . . .	104
13.22.1 Detailed Description . . . . .	104
13.22.2 Member Function Documentation . . . . .	104
13.23CX::CX_MessageFlushData Struct Reference . . . . .	107
13.23.1 Detailed Description . . . . .	107
13.23.2 Constructor & Destructor Documentation . . . . .	107
13.24CX::CX_Mouse Class Reference . . . . .	107
13.24.1 Detailed Description . . . . .	108
13.24.2 Member Enumeration Documentation . . . . .	108
13.24.3 Member Function Documentation . . . . .	108
13.25CX::Draw::CX_PatternProperties_t Class Reference . . . . .	110

13.25.1 Detailed Description . . . . .	110
13.25.2 Member Enumeration Documentation . . . . .	110
13.25.3 Member Data Documentation . . . . .	110
13.26CX::CX_RandomNumberGenerator Class Reference . . . . .	111
13.26.1 Detailed Description . . . . .	112
13.26.2 Constructor & Destructor Documentation . . . . .	112
13.26.3 Member Function Documentation . . . . .	113
13.27CX::Util::CX_SegmentProfiler Class Reference . . . . .	121
13.27.1 Detailed Description . . . . .	121
13.27.2 Constructor & Destructor Documentation . . . . .	121
13.27.3 Member Function Documentation . . . . .	122
13.28CX::CX_SlidePresenter Class Reference . . . . .	122
13.28.1 Detailed Description . . . . .	123
13.28.2 Member Enumeration Documentation . . . . .	124
13.28.3 Member Function Documentation . . . . .	124
13.29CX::CX_SoundBuffer Class Reference . . . . .	129
13.29.1 Detailed Description . . . . .	130
13.29.2 Member Function Documentation . . . . .	130
13.30CX::CX_SoundBufferPlayer Class Reference . . . . .	135
13.30.1 Detailed Description . . . . .	135
13.30.2 Member Function Documentation . . . . .	135
13.31CX::CX_SoundBufferRecorder Class Reference . . . . .	137
13.31.1 Detailed Description . . . . .	138
13.31.2 Member Function Documentation . . . . .	138
13.32CX::CX_SoundStream Class Reference . . . . .	139
13.32.1 Detailed Description . . . . .	140
13.32.2 Member Function Documentation . . . . .	141
13.33CX::CX_Time_t< T > Class Template Reference . . . . .	146
13.33.1 Detailed Description . . . . .	148
13.33.2 Constructor & Destructor Documentation . . . . .	149
13.33.3 Member Function Documentation . . . . .	149
13.34CX::Util::CX_TrialController Class Reference . . . . .	150
13.34.1 Detailed Description . . . . .	151
13.34.2 Member Function Documentation . . . . .	151
13.35CX::CX_WindowConfiguration_t Struct Reference . . . . .	152
13.35.1 Detailed Description . . . . .	152
13.36CX::Synth::Envelope Class Reference . . . . .	153

13.36.1 Detailed Description . . . . .	153
13.36.2 Member Function Documentation . . . . .	153
13.36.3 Member Data Documentation . . . . .	153
13.37CX::CX_Keyboard::Event Struct Reference . . . . .	154
13.37.1 Detailed Description . . . . .	154
13.37.2 Member Data Documentation . . . . .	154
13.38CX::CX_Joystick::Event Struct Reference . . . . .	155
13.38.1 Detailed Description . . . . .	155
13.38.2 Member Enumeration Documentation . . . . .	155
13.39CX::CX_Mouse::Event Struct Reference . . . . .	156
13.39.1 Detailed Description . . . . .	156
13.40CX::Synth::Filter Class Reference . . . . .	156
13.40.1 Detailed Description . . . . .	157
13.40.2 Member Enumeration Documentation . . . . .	157
13.40.3 Member Function Documentation . . . . .	157
13.40.4 Member Data Documentation . . . . .	157
13.41CX::CX_SlidePresenter::FinalSlideFunctionArgs Struct Reference . . . . .	157
13.41.1 Detailed Description . . . . .	158
13.42CX::Synth::FIRFilter Class Reference . . . . .	158
13.42.1 Detailed Description . . . . .	158
13.42.2 Member Enumeration Documentation . . . . .	158
13.42.3 Member Function Documentation . . . . .	159
13.43CX::Synth::GenericOutput Class Reference . . . . .	159
13.43.1 Detailed Description . . . . .	159
13.43.2 Member Function Documentation . . . . .	159
13.44CX::CX_SoundStream::InputEventArgs Struct Reference . . . . .	159
13.44.1 Detailed Description . . . . .	160
13.45CX::CX_DataFrame::IoOptions Class Reference . . . . .	160
13.45.1 Detailed Description . . . . .	160
13.46CX::Algo::LatinSquare Class Reference . . . . .	160
13.46.1 Detailed Description . . . . .	161
13.46.2 Constructor & Destructor Documentation . . . . .	161
13.46.3 Member Function Documentation . . . . .	162
13.47CX::Synth::Mixer Class Reference . . . . .	163
13.47.1 Detailed Description . . . . .	163
13.47.2 Member Function Documentation . . . . .	163
13.48CX::Synth::ModuleBase Class Reference . . . . .	164

13.48.1 Detailed Description	165
13.48.2 Member Function Documentation	165
13.49CX::Synth::ModuleParameter Class Reference	166
13.49.1 Detailed Description	166
13.49.2 Member Function Documentation	166
13.50CX::Synth::Multiplier Class Reference	167
13.50.1 Detailed Description	167
13.50.2 Member Function Documentation	167
13.51CX::Synth::Oscillator Class Reference	169
13.51.1 Detailed Description	169
13.51.2 Member Function Documentation	169
13.52CX::CX_SoundStream::OutputEventArgs Struct Reference	170
13.52.1 Detailed Description	170
13.53CX::CX_Time_t< T >::PartitionedTime Struct Reference	170
13.53.1 Detailed Description	171
13.54CX::CX_SlidePresenter::PresentationErrorInfo Struct Reference	171
13.54.1 Detailed Description	171
13.54.2 Member Data Documentation	172
13.55CX::CX_SlidePresenter::Slide Struct Reference	172
13.55.1 Detailed Description	173
13.56CX::CX_SlidePresenter::SlideTimingInfo Struct Reference	173
13.56.1 Detailed Description	173
13.57CX::Synth::SoundBufferInput Class Reference	173
13.57.1 Detailed Description	174
13.57.2 Member Function Documentation	174
13.58CX::Synth::SoundBufferOutput Class Reference	174
13.58.1 Detailed Description	175
13.58.2 Member Function Documentation	175
13.59CX::Synth::Splitter Class Reference	175
13.59.1 Detailed Description	175
13.59.2 Member Function Documentation	176
13.60CX::Synth::StereoSoundBufferOutput Class Reference	176
13.60.1 Detailed Description	176
13.60.2 Member Function Documentation	177
13.61CX::Synth::StereoStreamOutput Class Reference	177
13.61.1 Detailed Description	177
13.62CX::Synth::StreamOutput Class Reference	178



13.62.1 Detailed Description . . . . .	178
13.63CX::Synth::TrivialGenerator Class Reference . . . . .	178
13.63.1 Detailed Description . . . . .	178
13.63.2 Member Function Documentation . . . . .	179

## Index

180

# 1 Main Page

ofxCX (aka the C++ Experiment System; hereafter referred to as **CX**) is a "total conversion mod" for openFrameworks (often abbreviated oF) that is designed to be used for creating psychology experiments. OpenFrameworks and **CX** are based on C++, which is a very good programming language for anything requiring a high degree of timing precision. OpenFrameworks and **CX** are both free and open source, distributed under the MIT license.

One of the features that **CX** has is the ability to run without a substantial installation process. When a **CX** program is compiled and linked, the resulting artifact is an executable binary that can be run directly without needing another program to be installed. The collection of files needed to run a **CX** program is about 10 MB (possibly more, depending on stimuli). Installing the program just requires unzipping a file.

## 1.1 Installation

In order to use **CX**, you must have openFrameworks installed. Currently only version 0.8.0 of openFrameworks is supported by **CX**, which you can download from this page: <http://openframeworks.cc/download/older.-html>. The main openFrameworks download page (<http://openframeworks.cc/download/>) has information about how to install openFrameworks for use with some of the more popular development environments.

Once you have installed openFrameworks, you can install **CX** by putting the contents of the **CX** repository into a subdirectory under OFDIR/addons (typically OFDIR/addons/ofxCX), where OFDIR is where you put openFrameworks when you installed it. To use **CX** in a project, use the openFrameworks project generator and select ofxCX as an addon (see the instructions for using the [Examples and Tutorials](#) for information).

## 1.2 System Requirements

openFrameworks works on a wide variety of hardware and software, some of which are not supported by **CX**. **CX** works on computers with certain versions of Windows, Linux, or OSx operating systems. Windows 7 and XP are both supported.

As far as hardware is concerned, the minimum requirements for **CX** are very low. However, if your video card is too old, you won't be able to use some types of rendering. Having a video card that supports OpenGL version 3.2 at least is good, although older ones will work, potentially with reduced functionality. Also, a 2+ core CPU helps with some things and is generally a good idea for psychology experiments, because one core can be hogged by **CX** and the operating system can use the other core for other things. Basically, use a computer made after 2010 and you will have no worries whatsoever. However, **CX** has been found to work with reduced functionality on computers from the mid 90's, so there is that option, although I cannot make any guarantees that it will work on any given computer of that vintage.

## 1.3 Examples and Tutorials

There are several examples of how to use **CX**. The example files can be found in the **CX** directory (see [Installation](#)) in subfolders with names beginning with "example-". Some of the examples are on a specific topic and others are sample experiments that integrate together different features of **CX**.

In order to use the examples and tutorials, do the following:

1. Use the oF project generator (OFDIR/projectGenerator/projectGeneratorSimple.exe) to create a new project that uses the ofxCX addon. The project generator asks you what to name your project, where to put it (defaults to OFDIR/apps/myApps/), and has the option of selecting addons. Click the "addons" button and check to box next to ofxCX. If ofxCX does not appear in the list of addons, you probably didn't put the ofxCX directory in the right place.
2. Go to the newly-created project directory (that you chose when creating the project in step 1) and go into the src subdirectory.
3. Delete all of the files in the src directory (main.cpp, testApp.h, and testApp.cpp).
4. Copy the example .cpp file into this src directory.
5. If the example has a data folder, copy the contents of that folder into yourProjectDirectory/bin/data. The bin/data folder probably won't exist at this point. You can create it.
6. This step depends on your compiler, but you'll need to tell it to use the example source file that you copied in step 4 when it compiles the project (and possibly to specifically not use the files you deleted from the src directory in step 3).
7. Compile and run the project.

#### Tutorials:

- soundBuffer - Tutorial covering a number of things that you can do with CX\_SoundObjects, including loading sound files, combining sounds, and playing them.
- modularSynth - This tutorial demonstrates a number of ways to generate auditory stimuli using the synthesizer modules in the [CX::Synth](#) namespace.
- dataframe - Tutorial covering use of CX\_DataFrame, which is a container for storing data of various types that is collected in an experiment.
- logging - Tutorial explaining how the error logging system of [CX](#) works and how you can use it in your experiments.
- animation - A simple example of a simple way to draw moving things in [CX](#). Also includes some mouse input handling: cursor movement, clicks, and scroll wheel activity.

#### Experiments:

- changeDetection - A very straightforward change-detection task demonstrating some of the features of [CX](#) like presentation of time-locked stimuli, keyboard response collection, and use of the CX\_RandomNumberGenerator. There is also an advanced version of the changeDetection task that shows how to do data storage and output with a CX\_DataFrame and how to use a custom coordinate system with visual stimuli so that you don't have to work in pixels.
- nBack - Demonstrates advanced use of CX\_SlidePresenter in the implementation of an N-Back task. An advanced version of this example contrasts two methods of rendering stimuli with a CX\_SlidePresenter, demonstrating the advantages of each.

**Misc.:**

- helloWorld - A very basic getting started program.
- renderingTest - Includes several examples of how to draw stuff using ofFbo (a kind of offscreen buffer), ofImage (for opening image files: .png, .jpg, etc.), a variety of basic of drawing functions (ofCircle, ofRect, ofTriangle, etc.), and a number of [CX](#) drawing functions from the [CX::Draw](#) namespace that supplement openFrameworks's drawing capabilities.

## 1.4 Creating a Blank Experiment

To create your first experiment, follow the steps for using the examples in [Examples and Tutorials](#) up to and including step 3. Then create an empty .cpp file in the source directory of the project folder you made. In the file, you will need to include [CX\\_EntryPoint.h](#) and define runExperiment, like in the example below:

```
#include "CX_EntryPoint.h"

void runExperiment (void) {
    //Do everything you need to do for your experiment
}
```

That's all you need to do to get started. You should look at the [Examples and Tutorials](#) in order to learn more about how [CX](#) works. You should start with the helloWorld example.

## Topics

The best way to get an overview of how [CX](#) works is to look at the [Examples and Tutorials](#).

- To learn about presenting visual stimuli, go to the [Video](#) page or see the renderingTest or animation examples or the nBack or changeDetection example experiments.
- To learn about playing, recording, and generating sounds, go to the [Sound](#) page or see the soundObject or modularSynth examples.
- To learn how to store and output experiment data, see the [Data](#) page or see the dataFrame example.
- To learn about random number generation, see the [Randomization](#) page.
- To learn about how [CX](#) logs errors and other runtime information, see the [Message Logging](#) page.

You can look at the [Modules](#) page to see the other modules that [CX](#) has.

## 2 Blocking Code

Blocking code is code that either takes a long time to complete or that waits until some event occurs before allowing code execution to continue. An example of blocking code that waits is

```
do {
    Input.pollEvents();
} while (Input.Keyboard.availableEvents() == 0);
```

This code waits until the keyboard has been used in some way. No code past it can be executed until the keyboard is used, which could take a long time. Any code that blocks while waiting for a human to do something is blocking.

An example of blocking code that takes a long time (or at least could take a long time) is

```
vector<double> d = CX::Util::sequence<double>(0, 1000000, .033);
```

which requires the allocation of about 300 MB of RAM. This code doesn't wait for anything to happen, it just takes a long time to execute.

Blocking code is potentially harmful because it prevents some parts of [CX](#) from working in some situations. It is not a cardinal sin and there are times when using blocking code is acceptable. However, blocking code should not be used when trying to present stimuli or when responses are being made. The reason for this is that [CX](#) expects to be able to repeatedly check information related to stimulus presentation and input at very short intervals (at least every millisecond), but that cannot happen if a piece of code is blocking. There is of course an exception to the "no blocking while waiting for responses" rule, which is when your blocking code is doing nothing but waiting for a response and constantly polling for user input. For example, the following code waits until any response is made:

```
while(!Input.pollEvents())
;
//Process the inputs.
```

### 3 Framebuffers and Buffer Swapping

Some pieces of terminology that come up a lot in the documentation for [CX](#) are framebuffer, front buffer, back buffer, and swap buffers. What exactly are these things?

A framebuffer is fairly easy to explain in the rough by example. The contents of the screen of a computer are stored in a framebuffer. A framebuffer is essentially a rectangle of pixels where each pixel can be set to display any color. Framebuffers do not always have the same number of pixels as the screen: you can have framebuffers that are smaller or larger than the size of the screen. Framebuffers larger than the screen don't really do much for you as you cannot fit the whole thing on the screen. In [CX](#), framebuffers are typically worked with through the abstraction of an `ofFbo`.

There are two special framebuffers: The front buffer and the back buffer. These are created by OpenGL automatically as part of starting OpenGL. The size of these special framebuffers is functionally the same as the size of the window (or the whole screen, if in full screen mode). The front buffer contains what is shown on the screen. The back buffer is not presented on the screen, so it can be rendered to at any time without affecting what is visible on the screen. Typically, when you render stuff in [CX](#), you call `CX::CX_Display::beginDrawingToBackBuffer()` and `CX::CX_Display::endDrawingToBackBuffer()` around whatever you are rendering. This causes drawing that happens between the two function calls to be rendered to the back buffer.

What you have rendered to the back buffer has no effect on what you see on screen until you swap the contents of the front and back buffers. This isn't always a true swap, in that the back buffer does not end up with the contents of the front buffer in it. On many systems, the back buffer is copied to the front buffer and is itself unchanged. This swap can be done by using different functions of the `CX_Display`: `CX::CX_Display::swapBuffers()`, `CX::CX_Display::swapBuffersInThread()`, or `CX::CX_Display::setAutomaticSwapping()`. These functions are not interchangeable, so make sure you are using the right one for your application.

#### Vertical Synchronization

Vertical synchronization (Vsync) is the process by which the swaps of the front and back are synchronized to the refreshes of the monitor in order to prevent vertical tearing. Vertical tearing happens when one part of a scene is being drawn onto the monitor while a different scene is copied into the front buffer, causing parts of both scenes to be drawn at once. The "tearing" happens on the monitor where one scene abruptly becomes the other. In order to use Vsync, there must be some control over when the front buffer is drawn to. The ideal process might be that when the user requests a buffer swap the video card waits until the next vertical blank to swap the buffers. Unfortunately, what actually happens is implementation dependent, which makes writing software that will always work properly difficult.

One problem that I have observed is that even with Vsync enabled if there have been no buffer swaps for some time (several screen refresh periods), buffer swaps can happen more quickly than expected. For example, if the buffers

have not been swapped for 2.5 refresh periods and a buffer swap is requested, the buffer swap function can return immediately, not waiting until 3 refresh periods have passed to queue the swap. One process that could explain this is if when the user requests a buffer swap, if at least one vertical blank has passed since the last buffer swap, the buffers are swapped immediately. This can cause problems if the surrounding code is expecting the buffer swap to wait until the next refresh has occurred to return. One possible solution to this is, after a buffer swap has been requested, to tell OpenGL to wait until all ongoing processes have completed before continuing. This can be done with `CX::CX_Display::waitForOpenGL()` and results in a kind of "software" Vsync, as opposed to the "hardware" Vsync that is done by OpenGL internally. Calling the buffer swap function and then `CX::CX_Display::waitForOpenGL()` works sometimes, but it isn't perfect. On some systems, this will result in a wait of two frame periods before continuing (don't ask me why). On other systems, it works just fine. You can turn on hardware or software Vsync with `CX::CX_Display::useHardwareVSync()` and `CX::CX_Display::useSoftwareVSync()`.

So how do you know if you are having problems with video presentation that are related to Vsync? Probably the easiest way is to use a feature of `CX::CX_SlidePresenter` to learn about the timing of your stimuli. `CX::CX_SlidePresenter::printLastPresentationInformation()` provides a lot of timing information related to slide presentation so that you can check for errors easily. The errors can take the form of incorrect slide durations or frame counts (depending on presentation mode). If slides are consistently not started at the intended start time but the copy to the back buffer is happening in time, the most likely culprit is that something strange is going on with Vsync. If you are experiencing problems in windowed mode but not in full screen mode, you shouldn't worry. Vsync does not work properly in windowed mode in most modern operating systems due to the way in which they do window compositing.

`CX` provides some functionality to help deal with the Vsync annoyances. `CX::CX_Display::useHardwareVSync()` and `CX::CX_Display::useSoftwareVSync()` can turn "hardware" and "software" Vsync on or off independently so that if you experience problems you can try different solutions. You can test different combinations of Vsync using `CX::CX_Display::testBufferSwapping()`. You can also try different buffer swapping modes of `CX::CX_SlidePresenter` (see `CX::CX_SlidePresenter::SwappingMode`). One of the swapping modes (`MULTI_CORE`) swaps the buffers every frame in a secondary thread which avoids issues that arise from not swapping the buffers every frame. However, this mode can really only be used effectively with a 2+ core CPU, so if you are working with old computers, this may not be for you.

Another option to help deal with Vsync issues is to force Vsync on or off in your video card driver. Modern AMD and Nvidia drivers allow you to force Vsync on or off for specific applications or globally, which in my experience seems to be more reliable than turning Vsync on or off from within `CX`. If you force Vsync to a setting in the video card driver, `CX::CX_Display::useHardwareVSync()` will probably not do anything, but `CX::CX_Display::useSoftwareVSync()` probably would (although it is not clear that you would want to have both hardware and software Vsync enabled at the same time).

## 4 Modules

## 5 Program Model

### Program Flow

One of the foundational aspects of `CX` is the design of the overall program flow, which includes things such as how responses are collected, how stimuli are drawn to the screen, and other similar concepts. The best way to learn about program flow is to examine the [examples](#). The examples cover most of the critical topics and introduce the major components of `CX`.

The most important thing to understand is that in `CX`, nothing happens that your code does not explicitly ask for, with the exception of a small amount of setup, which is discussed below (see Pre-experiment Setup). For example, `CX` does not magically collect and timestamp user responses for you. Your code must poll for user input in order to get timestamps for input. This is explained more in the input section. In `CX`, there is no code running in the background that makes everything work out for your experiment, you have to design your experiment in such a way that you are covering all of your bases. That said, `CX` is specifically designed to make doing that as easy and painless as possible, while still giving you as much control over your experiment as is reasonably possible.

## Internals/Attributions

**CX** is not a monolithic entity. It is based on a huge amount of open source software written by many different authors over the years. Clearly, **CX** is based on **openFrameworks**, but **openFrameworks** itself is based on many different libraries. Window handling, which involves creating a window that can be rendered to and receiving user input events from the operating system, is managed by **GLFW** (<http://www.glfw.org/>). The actual rendering of visual stimuli is done using **OpenGL** (<http://www.opengl.org/>), which is wrapped by several **openFrameworks** abstractions (e.g. **ofGLProgrammableRenderer** at a lower level, e.g. **ofPath** at the level at which a typical user would use).

Audio is processed in different ways depending on the type of audio player used. **CX\_SoundBufferPlayer** and **CX\_SoundBufferRecorder** wrap **CX\_SoundStream** which wraps **RtAudio** (<https://www.music.mcgill.ca/~gary/rtaudio/>). If you are using **ofSoundPlayer**, depending on your operating system it might eventually use **FMOD** on Windows or **OSx** (<http://www.fmod.org/>; although the **openFrameworks** maintainers are considering moving away from **FMOD**) or **OpenAL** on Linux (<http://en.wikipedia.org/wiki/OpenAL>). However, you should check that this information is correct.

There are other libraries that are a part of **openFrameworks** that I am not as familiar with, including **Poco** (<http://pocoproject.org/>), which provides a variety of very useful utility functions and networking, **FreeType** (<http://www.freetype.org/>) which does font rendering, and many others.

Additionally, **CX** uses the **colorspace** package by Pascal Getreuer (<http://www.getreuer.info/home/colorspace>).

**CX** would not have been possible without the existence of these high-quality open-source projects.

## Overriding openFrameworks

Although **CX** is technically an addon to **openFrameworks**, there are a number of ways in which **CX** hijacks normal **oF** functionality in order to work better. As such, you cannot assume that all **oF** functionality is available to you.

Generally, drawing visual stimuli using **oF** classes and functions is fully supported. See the **renderingTest** example to see a plethora of ways to put things on the screen.

Audio output using **ofSoundPlayer** is supported, although no timing guarantees are made. Prefer **CX::CX\_SoundBufferPlayer**.

The input events (e.g. **ofEvents().mousePressed**) technically work, but with two serious limitations. 1) The events only fire when **CX\_InputManager::pollEvents()** is called (which internally calls **glfwPollEvents()** to actually kick off the events firing). 2) The standard **oF** events do not have timestamps, which limits their usefulness.

The following functions' behavior is superseded by functionality provided by **CX\_Display** (see also **CX::Instances::Display**): **ofGetFrameNum()** is replaced by **CX\_Display::getFrameNumber()** **ofGetLastFrameTime()** is replaced by **CX\_Display::getLastSwapTime()**

The following functions do nothing: **ofGetFrameRate()**, **ofSetFrameRate()**, **ofGetTargetFrameRate()**

A variety of behaviors related to **ofBaseApp** do not function because **CX** is not based on a class derived from **ofBaseApp** nor does it use **ofRunApp()** to begin the program. For example, a standard **oF** app class should have **steup()**, **update()**, and **draw()** functions that will be called by **oF** during program execution. **CX** has a different model that does not force object-orientation (at least at some levels).

## Pre-experiment Setup

There is very little that **CX** does without you asking for it. The one major exception is pre-experiment setup, in which a number of basic operations are performed in order to set up a platform on which the rest of the experiment can run. The most significant step is to open a window and set up the **OpenGL** rendering environment. The main pseudorandom number generator (**CX::Instances::RNG**) is seeded. The logging system is prepared for use. The main clock (**CX::Instances::Clock**) is prepared for use.

## Input Timing

The way user input is handled by `CX` is easily explained by giving the process of receiving a mouse click. Assume that a `CX` program is running in a window. The user clicks inside of the window. At this point, the operating system (or at least the windowing subsystem of the operating system, but I will choose to conflate them) detects that the click has occurred and notes that the location of the click is within the window. The operating system then attempts to tell the program that a mouse click has occurred. In order to be notified about input events like mouse click, the program has previously set up a message queue for incoming messages from the operating system. The OS puts the mouse event into the message queue. In order for the program to find out about the message it needs to check the message queue. This is what happens when `CX::CX_InputManager::pollEvents()` is called: The message queue is checked and all messages in the queue are processed, given timestamps, and routed to the next queue (e.g. the message queue in `CX::CX_Mouse` that is accessed with `CX::CX_Mouse::availableEvents()` and `CX::CX_Mouse::getNextEvent()`). The timestamps are not given by the operating system\*, so if `pollEvents` is not called regularly, input events will be received and everything will appear to be working correctly, but the timestamps will be wrong.

Of course, the actual process extends all the way back to the input device itself. The user presses the button and the microcontroller in the input device senses that a button has been pressed. It places this button press event into its outgoing message queue. At the next polling interval (typically 1 ms), the USB host controller on the computer polls the device for messages, discovers that a message is waiting and copies the message to the computer. At some point, the operating system checks to see if the USB host controller has received messages from any devices. It discovers the message and moves the message into the message queue of the program. At each step in which the message moves from one message queue to the next, the data contained in the message likely changes a little. At the start in the mouse, the message might just be "button 1 pressed". At the next step in the USB host controller, the message might be "input device 1 (type is mouse) button 1 pressed". Once the operating system gets the message it might be "mouse button 1 pressed while cursor at (367, 200) relative to clicked window". Eventually, the message gets into the message queue that users of `CX` work with, in `CX::CX_Mouse`, for example.

This process sounds very long and complicated, suggesting that it might take a long time to complete, throwing off timing data. That is true: Input timing data collected by `CX` is not veridical, there are invariably delays, including non-systematic delays. However, there are several steps in the process that no experiment software can get around, so the problems with timing data are not unique to `CX`. It might be possible to write a custom driver for the mouse or keyboard that allows the software to bypass the operating system's message queue, but it is very difficult to avoid the USB hardware delays, which can be on the order of milliseconds for many kinds of standard input devices. The next layer of the problem is that we are really interested in response time to a specific stimulus, but the time at which the stimulus was actually presented may be misreported by audio or video hardware/software, so even if the response timestamp had no error whatsoever, when it is compared with the stimulus presentation time, the response latency would be wrong due to errors in measures stimulus presentation time. Based on this large set of problems with collecting accurate response latency data, it is my firm belief that the only way to accurately measure response latency is with a button box that measures actual stimulus onset time with a light or sound sensor and also measures the time of a button press or other response. If you don't use such a system, the expectation is that you simply allow any error in response latencies to be dealt with statistically. Typically, any systematic error in response times will be subtracted out when conditions are compared with one another. Any random error will simply slightly inflate the estimated variance, but probably not to any meaningful extent.

If you would like to learn more about the internals of how input is handled in `CX`, you can see how GLFW and openFrameworks manage input by examining the source code in the respective repositories.

\*Technically, on Windows the messages that are given to a program do have a timestamp. However, the documentation doesn't actually say what the timestamp represents. My searching turns up the suggestion that it is a timestamp in milliseconds from system boot, but that the timestamp is set using the `GetTickCount` function, which typically has worse than 10 ms precision. This makes the timestamp attached to the message of very little value. See this page for documentation of what information comes with a Windows message: <http://msdn.microsoft.com/en-us/library/windows/desktop/ms644958%28v=vs.85%29.aspx>. The only page on which I actually found a definition of what the time member stores is this page <http://msdn.microsoft.com/en-us/library/aa929818.aspx>, which gives information pertaining to Windows Mobile 6.5, which is an obsolete smartphone operating system.



## Stimulus Timing

Although the kinds of error introduced into response time data can often be dealt with statistically, errors in stimulus presentation can be more serious. For example, if a visual stimulus is systematically presented for an extra frame throughout an experiment, then the method of the experiment has been altered without the experimenter learning about the alteration. Even if the extra frame does not always happen, on average participants are seeing more of that stimulus than they should be. An error on the magnitude of an extra frame is nearly impossible to detect by eye in most cases, so it is important that there is some way to detect errors in stimulus presentation. The primary method of presenting time-locked visual stimuli is the `CX_SlidePresenter`. It has built in error-detection features that pick up on certain kinds of errors. Information about presentation errors can be found by using `CX::CX_SlidePresenter::checkForPresentationErrors()`.

Although it is nice to be made aware of errors when they occur, it is better to not have the errors happen in the first place. For this reason, stimulus presentation in `CX` is designed around avoiding errors. For visual stimuli, the `CX_SlidePresenter` provides a very easy-to-use way to present visual stimuli. Because the interface is so simple, user error is minimized. The backend code of the `CX_SlidePresenter` is designed to minimize the potential for timing errors by carefully tracking the passage of time, monitor refreshes, and timing of stimulus rendering.

On the audio front, `CX` provides the `CX_SoundBufferPlayer`, which plays `CX::CX_SoundBuffer` "CX\_SoundBuffers". If several sounds are to be presented in a time-locked sequence, playing the sounds individually at their intended onset time can result in unequal startup delays for each sound, but if all of the sounds are combined together into a single audio buffer this possibility is eliminated. `CX_SoundObjects` are designed to make combining multiple sound stimuli together easy, which helps to prevent timing errors that could have otherwise occurred between sounds. `CX` also includes `CX_SoundStream`, which provides a method for directly accessing and manipulating the contents of audio buffers that are received from or sent to audio hardware.

## 6 license

The code in this repository is available under the MIT License ([https://secure.wikimedia.org/wikipedia/en/wiki/-Mit\\_license](https://secure.wikimedia.org/wikipedia/en/wiki/-Mit_license)).

Copyright (c) 2014 Kyle Hardman

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the "Software"), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

## 7 Module Index

### 7.1 Modules

Here is a list of all modules:



Data	14
Entry Point	15
Input Devices	16
Message Logging	17
Randomization	18
Sound	19
Timing	20
Utility	21
Video	22

## 8 Namespace Index

### 8.1 Namespace List

Here is a list of all documented namespaces with brief descriptions:

<a href="#">CX</a>	23
<a href="#">CX::Algo</a>	25
<a href="#">CX::Draw</a>	28
<a href="#">CX::Instances</a>	39
<a href="#">CX::Synth</a>	39
<a href="#">CX::Util</a>	42

## 9 Hierarchical Index

### 9.1 Class Hierarchy

This inheritance list is sorted roughly, but not completely, alphabetically:

<a href="#">CX::Algo::BlockSampler&lt; T &gt;</a>	58
<a href="#">CX::CX_SlidePresenter::Configuration</a>	60
<a href="#">CX::CX_SoundStream::Configuration</a>	61
<a href="#">CX::CX_BaseClockImplementation</a>	62
<a href="#">CX::Util::CX_BaseUnitConverter</a>	62
<a href="#">CX::Util::CX_DegreeToPixelConverter</a>	83

<b>CX::Util::CX_LengthToPixelConverter</b>	<b>102</b>
<b>CX::CX_Clock</b>	<b>63</b>
<b>CX::Util::CX_CoordinateConverter</b>	<b>66</b>
<b>CX::CX_DataFrame</b>	<b>69</b>
<b>CX::CX_DataFrameCell</b>	<b>79</b>
<b>CX::CX_DataFrameColumn</b>	<b>82</b>
<b>CX::CX_DataFrameRow</b>	<b>83</b>
<b>CX::CX_Display</b>	<b>86</b>
<b>CX::CX_InputManager</b>	<b>94</b>
<b>CX::CX_Joystick</b>	<b>96</b>
<b>CX::CX_Keyboard</b>	<b>97</b>
<b>CX::Util::CX_LapTimer</b>	<b>100</b>
<b>CX::CX_Logger</b>	<b>104</b>
<b>CX::CX_MessageFlushData</b>	<b>107</b>
<b>CX::CX_Mouse</b>	<b>107</b>
<b>CX::Draw::CX_PatternProperties_t</b>	<b>110</b>
<b>CX::CX_RandomNumberGenerator</b>	<b>111</b>
<b>CX::Util::CX_SegmentProfiler</b>	<b>121</b>
<b>CX::CX_SlidePresenter</b>	<b>122</b>
<b>CX::CX_SoundBuffer</b>	<b>129</b>
<b>CX::CX_SoundBufferPlayer</b>	<b>135</b>
<b>CX::CX_SoundBufferRecorder</b>	<b>137</b>
<b>CX::CX_SoundStream</b>	<b>139</b>
<b>CX::CX_Time_t&lt; T &gt;</b>	<b>146</b>
<b>CX::CX_Time_t&lt; std::ratio&lt; 1, 1000 &gt; &gt;</b>	<b>146</b>
<b>CX::Util::CX_TrialController</b>	<b>150</b>
<b>CX::CX_WindowConfiguration_t</b>	<b>152</b>
<b>CX::CX_Keyboard::Event</b>	<b>154</b>
<b>CX::CX_Joystick::Event</b>	<b>155</b>
<b>CX::CX_Mouse::Event</b>	<b>156</b>

CX::CX_SlidePresenter::FinalSlideFunctionArgs	157
CX::CX_SoundStream::InputEventArgs	159
CX::CX_DataFrame::IoOptions	160
CX::Algo::LatinSquare	160
CX::Synth::ModuleBase	164
CX::Synth::Adder	54
CX::Synth::AdditiveSynth	54
CX::Synth::Clamper	59
CX::Synth::Envelope	153
CX::Synth::Filter	156
CX::Synth::FIRFilter	158
CX::Synth::GenericOutput	159
CX::Synth::Mixer	163
CX::Synth::Multiplier	167
CX::Synth::Oscillator	169
CX::Synth::SoundBufferInput	173
CX::Synth::SoundBufferOutput	174
CX::Synth::Splitter	175
CX::Synth::StreamOutput	178
CX::Synth::TrivialGenerator	178
CX::Synth::ModuleParameter	166
CX::CX_SoundStream::OutputEventArgs	170
CX::CX_Time_t< T >::PartitionedTime	170
CX::CX_SlidePresenter::PresentationErrorInfo	171
CX::CX_SlidePresenter::Slide	172
CX::CX_SlidePresenter::SlideTimingInfo	173
CX::Synth::StereoSoundBufferOutput	176
CX::Synth::StereoStreamOutput	177

## 10 Class Index

### 10.1 Class List

Here are the classes, structs, unions and interfaces with brief descriptions:

<a href="#">CX::Synth::Adder</a>	54
<a href="#">CX::Synth::AdditiveSynth</a>	54
<a href="#">CX::Algo::BlockSampler&lt; T &gt;</a>	58
<a href="#">CX::Synth::Clamper</a>	59
<a href="#">CX::CX_SlidePresenter::Configuration</a>	60
<a href="#">CX::CX_SoundStream::Configuration</a>	61
<a href="#">CX::CX_BaseClockImplementation</a>	62
<a href="#">CX::Util::CX_BaseUnitConverter</a>	62
<a href="#">CX::CX_Clock</a>	63
<a href="#">CX::Util::CX_CoordinateConverter</a>	66
<a href="#">CX::CX_DataFrame</a>	69
<a href="#">CX::CX_DataFrameCell</a>	79
<a href="#">CX::CX_DataFrameColumn</a>	82
<a href="#">CX::CX_DataFrameRow</a>	83
<a href="#">CX::Util::CX_DegreeToPixelConverter</a>	83
<a href="#">CX::CX_Display</a>	86
<a href="#">CX::CX_InputManager</a>	94
<a href="#">CX::CX_Joystick</a>	96
<a href="#">CX::CX_Keyboard</a>	97
<a href="#">CX::Util::CX_LapTimer</a>	100
<a href="#">CX::Util::CX_LengthToPixelConverter</a>	102
<a href="#">CX::CX_Logger</a>	104
<a href="#">CX::CX_MessageFlushData</a>	107
<a href="#">CX::CX_Mouse</a>	107
<a href="#">CX::Draw::CX_PatternProperties_t</a>	110
<a href="#">CX::CX_RandomNumberGenerator</a>	111

CX::Util::CX_SegmentProfiler	121
CX::CX_SlidePresenter	122
CX::CX_SoundBuffer	129
CX::CX_SoundBufferPlayer	135
CX::CX_SoundBufferRecorder	137
CX::CX_SoundStream	139
CX::CX_Time_t< T >	146
CX::Util::CX_TrialController	150
CX::CX_WindowConfiguration_t	152
CX::Synth::Envelope	153
CX::CX_Keyboard::Event	154
CX::CX_Joystick::Event	155
CX::CX_Mouse::Event	156
CX::Synth::Filter	156
CX::CX_SlidePresenter::FinalSlideFunctionArgs	157
CX::Synth::FIRFilter	158
CX::Synth::GenericOutput	159
CX::CX_SoundStream::InputEventArgs	159
CX::CX_DataFrame::IoOptions	160
CX::Algo::LatinSquare	160
CX::Synth::Mixer	163
CX::Synth::ModuleBase	164
CX::Synth::ModuleParameter	166
CX::Synth::Multiplier	167
CX::Synth::Oscillator	169
CX::CX_SoundStream::OutputEventArgs	170
CX::CX_Time_t< T >::PartitionedTime	170
CX::CX_SlidePresenter::PresentationErrorInfo	171
CX::CX_SlidePresenter::Slide	172
CX::CX_SlidePresenter::SlideTimingInfo	173

<a href="#">CX::Synth::SoundBufferInput</a>	173
<a href="#">CX::Synth::SoundBufferOutput</a>	174
<a href="#">CX::Synth::Splitter</a>	175
<a href="#">CX::Synth::StereoSoundBufferOutput</a>	176
<a href="#">CX::Synth::StereoStreamOutput</a>	177
<a href="#">CX::Synth::StreamOutput</a>	178
<a href="#">CX::Synth::TrivialGenerator</a>	178

## 11 Module Documentation

### 11.1 Data

#### Classes

- class [CX::CX\\_DataFrame](#)
- class [CX::CX\\_DataFrameColumn](#)
- class [CX::CX\\_DataFrameRow](#)
- class [CX::CX\\_DataFrameCell](#)

#### 11.1.1 Detailed Description

This module is related to storing experimental data. [CX\\_DataFrame](#) is the most important class in this module.

## 11.2 Entry Point

### Functions

- void `runExperiment` (void)

### Variables

- `CX_InputManager CX::Instances::Input` = `CX::Private::inputManagerFactory()`
- `CX_Display CX::Instances::Display`
- `CX_Logger CX::Instances::Log`
- `CX_RandomNumberGenerator CX::Instances::RNG`

### 11.2.1 Detailed Description

The entry point provides access to a few instances of classes that can be used by user code. It also provides declarations (but not definitions) of a function which the user should define (see `runExperiment()`).

### 11.2.2 Function Documentation

#### 11.2.2.1 `runExperiment ( void )`

The user code should define a function with this name and type signature (takes no arguments and returns nothing). This function will be called once setup is done for `CX`. When `runExperiment` returns, the program will exit.

```
void runExperiment (void) {
    //Do your experiment.

    return; //Return when done to exit the program. You don't have to explicitly return; you can just fall
           off the end of the function.
           //You can alternately call std::exit() or ofExit() at any point.
}
```

### 11.2.3 Variable Documentation

#### 11.2.3.1 `CX::CX_Display CX::Instances::Display`

An instance of `CX::CX_Display` that is lightly hooked into the `CX` backend. `setup()` is called for `Display` before `runExperiment()` is called.

#### 11.2.3.2 `CX::CX_InputManager CX::Instances::Input = CX::Private::inputManagerFactory()`

An instance of `CX_InputManager` that is very lightly hooked into the `CX` backend.

#### 11.2.3.3 `CX::CX_Logger CX::Instances::Log`

This is an instance of `CX::CX_Logger` that is hooked into the `CX` backend. All log messages generated by `CX` and `openFrameworks` go through this instance.

#### 11.2.3.4 `CX::CX_RandomNumberGenerator CX::Instances::RNG`

An instance of `CX_RandomNumberGenerator` that is (lightly) hooked into the `CX` backend.

## 11.3 Input Devices

### Classes

- class [CX::CX\\_InputManager](#)
- class [CX::CX\\_Joystick](#)
- class [CX::CX\\_Keyboard](#)
- class [CX::CX\\_Mouse](#)

### 11.3.1 Detailed Description

There are a number of different classes that together perform the input handling functions of [CX](#). Start by looking at [CX::CX\\_InputManager](#) and the instance of that class that is created for you: [CX::Instances::Input](#).

For interfacing with serial ports, use `ofSerial` (<http://www.openframeworks.cc/documentation/communication/of-Serial.html>).

### See Also

[CX::CX\\_InputManager](#) for the primary interface to input devices.  
[CX::CX\\_Keyboard](#) for keyboard specific information.  
[CX::CX\\_Mouse](#) for mouse specific information.  
[CX::CX\\_Joystick](#) for joystick specific information.



## 11.4 Message Logging

### Classes

- class [CX::CX\\_Logger](#)

### Enumerations

- enum [CX::CX\\_LogLevel](#) {  
    **LOG\_ALL** = 0, **LOG\_VERBOSE** = 1, **LOG\_NOTICE** = 2, **LOG\_WARNING** = 3,  
    **LOG\_ERROR** = 4, **LOG\_FATAL\_ERROR** = 5, **LOG\_NONE** = 6 }

#### 11.4.1 Detailed Description

This module is designed for logging error, warnings, and other messages. The primary interface is the [CX\\_Logger](#) class, in particular the preinstantiated [CX::Instances::Log](#).

#### 11.4.2 Enumeration Type Documentation

##### 11.4.2.1 enum [CX::CX\\_LogLevel](#) [strong]

Log levels for log messages. Depending on the log level chosen, the name of the level will be printed before the message. Depending on the settings set using `level()`, `levelForConsole()`, or `levelForFile()`, if the log level of a message is below the level set for the module or logging target it will not be printed. For example, if `LOG_ERROR` is the level for the console and `LOG_NOTICE` is the level for the module "test", then messages logged to the "test" module will be completely ignored if at verbose level (because of the module setting) and will not be printed to the console if they are below the level of an error (because of the console setting).

## 11.5 Randomization

### Classes

- class [CX::CX\\_RandomNumberGenerator](#)

### 11.5.1 Detailed Description

This module provides a class that is used for random number generation.

## 11.6 Sound

### Namespaces

- [CX::Synth](#)

### Classes

- class [CX::CX\\_SoundBufferPlayer](#)
- class [CX::CX\\_SoundBufferRecorder](#)
- class [CX::CX\\_SoundBuffer](#)
- class [CX::CX\\_SoundStream](#)

#### 11.6.1 Detailed Description

There are a few different ways to deal with sounds in [CX](#). The thing that most people want to do is to play sounds, which is done with the `CX_SoundBufferPlayer`. See the `soundBuffer` tutorial for information on how to do that.

If you want to record sound, use the `CX_SoundBufferRecorder`.

If you want to generate sound stimuli through sound synthesis, see the [CX::Synth](#) namespace and `modularSynth` tutorial.

Finally, if you want to have direct control of the data going to and from a sound device, see `CX_SoundStream`.

## 11.7 Timing

### Classes

- class [CX::CX\\_Clock](#)
- class [CX::CX\\_BaseClockImplementation](#)
- class [CX::CX\\_Time\\_t< T >](#)
- class [CX::Util::CX\\_LapTimer](#)
- class [CX::Util::CX\\_SegmentProfiler](#)

### Variables

- [CX\\_Clock](#) [CX::Instances::Clock](#)

#### 11.7.1 Detailed Description

This module provides methods for timestamping events in experiments.

#### 11.7.2 Variable Documentation

##### 11.7.2.1 [CX\\_Clock](#) [CX::Instances::Clock](#)

An instance of [CX::CX\\_Clock](#) that is hooked into the [CX](#) backend. Anything in [CX](#) that requires timing information uses this instance. You should use this instance in your code and not make your own instance of [CX\\_Clock](#). You should never need another instance. You should never use another instance, as the experiment start times will not agree between instances.

## 11.8 Utility

### Namespaces

- [CX::Util](#)

### Classes

- class [CX::Util::CX\\_TrialController](#)
- class [CX::Util::CX\\_DegreeToPixelConverter](#)
- class [CX::Util::CX\\_LengthToPixelConverter](#)
- class [CX::Util::CX\\_CoordinateConverter](#)

#### 11.8.1 Detailed Description

## 11.9 Video

### Namespaces

- [CX::Draw](#)

### Classes

- class [CX::CX\\_Display](#)
- class [CX::CX\\_SlidePresenter](#)

#### 11.9.1 Detailed Description

This module is related to creating and presenting visual stimuli.

The [CX::Draw](#) namespace contains some more complex drawing functions. However, almost all of the drawing of stimuli is done using openFrameworks functions. A lot of the common functions can be found in ofGraphics.h (<http://www.openframeworks.cc/documentation/graphics/ofGraphics.html>), but there are a lot of other ways to draw stimuli with openFrameworks: See the graphics and 3d sections of this page: <http://www.openframeworks.cc/documentation/>.

## 12 Namespace Documentation

### 12.1 CX Namespace Reference

#### Namespaces

- [Algo](#)
- [Draw](#)
- [Instances](#)
- [Synth](#)
- [Util](#)

#### Classes

- class [CX\\_Clock](#)
- class [CX\\_BaseClockImplementation](#)
- class [CX\\_DataFrame](#)
- class [CX\\_DataFrameColumn](#)
- class [CX\\_DataFrameRow](#)
- class [CX\\_DataFrameCell](#)
- class [CX\\_Display](#)
- struct [CX\\_WindowConfiguration\\_t](#)
- class [CX\\_InputManager](#)
- class [CX\\_Joystick](#)
- class [CX\\_Keyboard](#)
- struct [CX\\_MessageFlushData](#)
- class [CX\\_Logger](#)
- class [CX\\_Mouse](#)
- class [CX\\_RandomNumberGenerator](#)
- class [CX\\_SlidePresenter](#)
- class [CX\\_SoundBuffer](#)
- class [CX\\_SoundBufferPlayer](#)
- class [CX\\_SoundBufferRecorder](#)
- class [CX\\_SoundStream](#)
- class [CX\\_Time\\_t](#)

#### Typedefs

- typedef int64\_t [CX\\_RandomInt\\_t](#)  
*The type of integer returned by the [CX\\_RandomNumberGenerator::randomInt\(\)](#) functions.*
- typedef [CX\\_Time\\_t](#)< std::ratio< 3600, 1 > > [CX\\_Hours](#)  
*Hours.*
- typedef [CX\\_Time\\_t](#)< std::ratio< 60, 1 > > [CX\\_Minutes](#)  
*Minutes.*
- typedef [CX\\_Time\\_t](#)< std::ratio< 1, 1 > > [CX\\_Seconds](#)  
*Seconds.*
- typedef [CX\\_Time\\_t](#)< std::ratio< 1, 1000 > > [CX\\_Millis](#)  
*Milliseconds.*
- typedef [CX\\_Time\\_t](#)< std::ratio< 1, 1000000 > > [CX\\_Micros](#)

*Microseconds.*

- typedef `CX_Time_t` < std::ratio< 1, 1000000000 > > `CX_Nanos`

*Nanoseconds.*

## Enumerations

- enum `CX_LogLevel` {  
**LOG\_ALL** = 0, **LOG\_VERBOSE** = 1, **LOG\_NOTICE** = 2, **LOG\_WARNING** = 3,  
**LOG\_ERROR** = 4, **LOG\_FATAL\_ERROR** = 5, **LOG\_NONE** = 6 }

## Functions

- std::ostream & `operator<<` (std::ostream &os, const `CX_DataFrameCell` &cell)
- void `reopenWindow` (`CX_WindowConfiguration_t` config)
- std::ostream & `operator<<` (std::ostream &os, const `CX_Joystick::Event` &ev)  
*Stream insertion operator for the `CX_Joystick::Event` struct.*
- std::istream & `operator>>` (std::istream &is, `CX_Joystick::Event` &ev)  
*Stream extraction operator for the `CX_Joystick::Event` struct.*
- std::ostream & `operator<<` (std::ostream &os, const `CX_Keyboard::Event` &ev)  
*Stream insertion operator for the `CX_Keyboard::Event` struct.*
- std::istream & `operator>>` (std::istream &is, `CX_Keyboard::Event` &ev)  
*Stream extraction operator for the `CX_Keyboard::Event` struct.*
- std::ostream & `operator<<` (std::ostream &os, const `CX_Mouse::Event` &ev)  
*Stream insertion operator for the `CX_Mouse::Event` struct.*
- std::istream & `operator>>` (std::istream &is, `CX_Mouse::Event` &ev)  
*Stream extraction operator for the `CX_Mouse::Event` struct.*
- template<typename TimeUnit >  
std::ostream & `operator<<` (std::ostream &os, const `CX_Time_t`< TimeUnit > &t)
- template<typename TimeUnit >  
std::istream & `operator>>` (std::istream &is, `CX_Time_t`< TimeUnit > &t)
- template<typename T >  
`CX_Time_t`< T > `operator*` (`CX_Time_t`< T > lhs, double rhs)  
*Multiplies a `CX_Time_t` with a numeric value, resulting in a `CX_Time_t` in the same units as *lhs*.*
- template<typename T >  
`CX_Time_t`< T > `operator*` (double lhs, `CX_Time_t`< T > rhs)  
*Multiplies a `CX_Time_t` with a numeric value, resulting in a `CX_Time_t` in the same units as *rhs*.*

### 12.1.1 Detailed Description

This namespace contains all of the symbols related to `CX`, except for `runExperiment()`, which is not namespace qualified.

### 12.1.2 Function Documentation

#### 12.1.2.1 std::ostream & CX::operator<< ( std::ostream & os, const CX\_DataFrameCell & cell )

Stream insertion operator for a `CX_DataFrameCell`.



12.1.2.2 `template<typename TimeUnit > std::ostream& CX::operator<< ( std::ostream & os, const CX_Time_t< TimeUnit > & t )`

Stream insertion operator for [CX\\_Time\\_t](#).

#### Note

If `operator<<` and `operator>>` are used to convert to/from a stream representation, you MUST use the same time units on both ends of the conversion. In addition, converting to/from string is not guaranteed to be a lossless operation, but it usually works out.

```

st::stringstream ss;
CX_Seconds sec(60);

ss << sec;
cout << ss.str() << endl;

CX_Minutes min;
ss >> min; //Oops! This is now holding 60 minutes!

//Better: Use the same type at both ends.
CX_Seconds sec2;
ss << sec;
ss >> sec2;
```

12.1.2.3 `template<typename TimeUnit > std::istream& CX::operator>> ( std::istream & is, CX_Time_t< TimeUnit > & t )`

Stream extraction operator for [CX\\_Time\\_t](#).

12.1.2.4 `void CX::reopenWindow ( CX_WindowConfiguration_t config )`

This function opens a GLFW window that can be rendered to. If another window was already open by the application at the time this is called, that window will be closed. This is useful if you want to control some of the parameters of the window that cannot be changed after the window has been opened.

## 12.2 CX::Algo Namespace Reference

### Classes

- class [LatinSquare](#)
- class [BlockSampler](#)

### Functions

- `template<typename T >`  
`std::vector< T > generateSeparatedValues (int count, double minDistance, std::function< double(T, T)> distanceFunction, std::function< T(void)> randomDeviate, int maxSequentialFailures=200)`
- `template<typename T >`  
`std::vector< std::vector< T > > fullyCross (std::vector< std::vector< T > > factors)`

### 12.2.1 Detailed Description

This namespace contains a few complex algorithms that can be difficult to properly implement or are very experiment-specific.

## 12.2.2 Function Documentation

### 12.2.2.1 `template<typename T> std::vector< std::vector< T > > CX::Algo::fullyCross ( std::vector< std::vector< T > > factors )`

This function fully crosses the levels of the factors of a design. For example, for a 2X3 design, it would give you all 6 combinations of the levels of the design.

## Parameters

<i>factors</i>	A vector of factors, each factor being a vector containing all the levels of that factor.
----------------	---

## Returns

A vector of crossed factor levels. It's length is equal to the product of the levels of the factors. The length of each "row" is equal to the number of factors.

## Example use:

```
std::vector< std::vector<int> > levels(2); //Two factors
levels[0].push_back(1); //The first factor has two levels (1 and 2)
levels[0].push_back(2);
levels[1].push_back(3); //The second factor has three levels (3, 4, and 5)
levels[1].push_back(4);
levels[1].push_back(5);
auto crossed = fullyCross(levels);
```

crossed should contain a vector with six subvectors with the contents:

```
{ {1,3}, {1,4}, {1,5}, {2,3}, {2,4}, {2,5} }
```

## where

```
crossed[3][0] == 2
crossed[3][1] == 3
crossed[0][1] == 3
```

**12.2.2.2** `template<typename T> std::vector< T > CX::Algo::generateSeparatedValues ( int count, double minDistance, std::function< double(T, T)> distanceFunction, std::function< T(void)> randomDeviate, int maxSequentialFailures = 200 )`

This algorithm is designed to deal with the situation in which a number of random values must be generated that are each at least some distance from every other random value. This is a very generic implementation of this algorithm. It works by taking pointers to two functions that work on whatever type of data you are using. The first function is a distance function: it returns the distance between two values of the type. You can define distance in whatever way you would like. The second function generates random values of the type.

## Template Parameters

<i>&lt; T &gt;</i>	The type of data you are working with.
--------------------	--

## Parameters

<i>count</i>	The number of values you want to be generated.
<i>minDistance</i>	The minimum distance between any two values. This will be compared to the result of distance-Function.
<i>distanceFunction</i>	A function that computes the distance, in whatever units you want, between two values of type T.
<i>randomDeviate</i>	A function that generates random values of type T.
<i>maxSequential-Failures</i>	The maximum number of times in a row that a newly-generated value is less than minDistance from at least one other value. This essentially makes sure that if it is not possible to generate a random value that is at least some distance from the others, the algorithm will terminate.

## Returns

A vector of values. If the function terminated prematurely due to `maxSequentialFailures` being reached, the returned vector will have 0 elements.

```
//This example function generates locCount points with both x and y values bounded by minimumValues and
//maximumValues that
//are at least minDistance pixels from each other.
std::vector<ofPoint> getObjectLocations(int locCount, double minDistance, ofPoint minimumValues, ofPoint
maximumValues) {
    auto pointDistance = [] (ofPoint a, ofPoint b) {
        return sqrt(pow(a.x - b.x, 2) + pow(a.y - b.y, 2));
    };

    auto randomPoint = [&]() {
        ofPoint rval;
        rval.x = RNG.randomInt(minimumValues.x, maximumValues.x);
        rval.y = RNG.randomInt(minimumValues.y, maximumValues.y);
        return rval;
    };

    return CX::Algo::generateSeparatedValues<ofPoint>(locCount, minDistance, pointDistance, randomPoint,
1000);
}

//Call of example function
vector<ofPoint> v = getObjectLocations(5, 50, ofPoint(0, 0), ofPoint(400, 400));
```

## 12.3 CX::Draw Namespace Reference

### Classes

- class [CX\\_PatternProperties\\_t](#)

### Enumerations

- enum **LineCornerMode** { **OUTER\_POINT**, **BEZIER\_ARC**, **STRAIGHT\_LINE**, **ARC** }

### Functions

- ofPath [squircleToPath](#) (double radius, double amount)
- ofPath [arrowToPath](#) (float length, float headOffsets, float headSize, float lineWidth)
- std::vector< ofPoint > [getStarVertices](#) (unsigned int numberOfPoints, float innerRadius, float outerRadius, float rotationDeg)
- ofPath [starToPath](#) (unsigned int numberOfPoints, float innerRadius, float outerRadius)
- void [star](#) (ofPoint center, unsigned int numberOfPoints, float innerRadius, float outerRadius, float rotationDeg)
- void [centeredString](#) (int x, int y, std::string s, ofTrueTypeFont &font)
- void [centeredString](#) (ofPoint center, std::string s, ofTrueTypeFont &font)
- ofPixels [greyscalePatternToPixels](#) (const [CX\\_PatternProperties\\_t](#) &properties)
- ofPixels [gaborToPixels](#) (const [CX\\_GaborProperties\\_t](#) &properties)
- ofTexture [gaborToTexture](#) (const [CX\\_GaborProperties\\_t](#) &properties)
- void [gabor](#) (ofPoint center, const [CX\\_GaborProperties\\_t](#) &properties)
- bool [isPointInRegion](#) (ofPoint p, ofPoint r1, ofPoint r2, float tolerance)
- bool [arePointsInLine](#) (ofPoint p1, ofPoint p2, ofPoint p3, float slopeTolerance=1e-3)
- ofPoint [findIntersectionOfLines](#) (LineSegment ls1, LineSegment ls2)
- std::vector< LineSegment > [getParallelLineSegments](#) (LineSegment ls, float distance)
- ofVec3f [getCornerOuterVector](#) (ofPoint p1, ofPoint p2, ofPoint p3, float vectorLength)
- ofPath [lines](#) (std::vector< ofPoint > points, float width, LineCornerMode cornerMode)
- void [lines](#) (std::vector< ofPoint > points, float lineWidth)

- void [line](#) (ofPoint p1, ofPoint p2, float width)
- void [ring](#) (ofPoint center, float radius, float width, unsigned int resolution)
- void [arc](#) (ofPoint center, float radiusX, float radiusY, float width, float angleBegin, float angleEnd, unsigned int resolution)
- std::vector< ofPoint > [getBezierVertices](#) (std::vector< ofPoint > controlPoints, unsigned int resolution)
- void [bezier](#) (std::vector< ofPoint > controlPoints, float width, unsigned int resolution)
- std::vector< double > [convertColors](#) (std::string conversionFormula, double S1, double S2, double S3)
- ofFloatColor [convertToRGB](#) (std::string inputColorSpace, double S1, double S2, double S3)
- std::vector< ofPoint > [getFixationCrossVertices](#) (float armLength, float armWidth)
- ofPath [fixationCrossToPath](#) (float armLength, float armWidth)
- void [fixationCross](#) (ofPoint location, float armLength, float armWidth)
- void [saveFboToFile](#) (ofFbo &fbo, std::string filename)
- template<typename ofColorType >  
std::vector< ofColorType > [getRGBSpectrum](#) (unsigned int colorCount)
- template<typename T >  
ofVbo [colorArcToVbo](#) (ofPoint center, std::vector< ofColor\_< T >> colors, float radiusX, float radiusY, float width, float angleBegin, float angleEnd)
- template<typename T >  
void [colorArc](#) (ofPoint center, std::vector< ofColor\_< T >> colors, float radiusX, float radiusY, float width, float angleBegin, float angleEnd)
- template<typename T >  
ofVbo [colorWheelToVbo](#) (ofPoint center, std::vector< ofColor\_< T >> colors, float radius, float width, float angle)
- template<typename T >  
void [colorWheel](#) (ofPoint center, std::vector< ofColor\_< T >> colors, float radius, float width, float angle)
- template<typename T >  
void [patternMask](#) (ofPoint center, float width, float height, float squareSize, std::vector< ofColor\_< T >> colors=std::vector< ofColor\_< T >>(0))

### 12.3.1 Detailed Description

This namespace contains functions for drawing certain complex stimuli. These functions are provided "as-is": If what they draw looks nice to you, great; however, there are no strong guarantees about what the output of the functions will look like.

### 12.3.2 Function Documentation

**12.3.2.1** void CX::Draw::arc ( ofPoint *center*, float *radiusX*, float *radiusY*, float *width*, float *angleBegin*, float *angleEnd*, unsigned int *resolution* )

[Draw](#) an arc around a central point. If radiusX and radiusY are equal, the arc will be like a section of a circle. If they are unequal, the arc will be a section of an ellipse.

#### Parameters

<i>center</i>	The point around which the arc will be drawn.
<i>radiusX</i>	The radius of the arc in the X-axis.
<i>radiusY</i>	The radius of the arc in the Y-axis.
<i>width</i>	The width of the arc, radially from the center.

<i>angleBegin</i>	The angle at which to begin the arc, in degrees.
<i>angleEnd</i>	The angle at which to end the arc, in degrees. If the arc goes in the "wrong" direction, try giving a negative value for <i>angleEnd</i> .
<i>resolution</i>	The resolution of the arc. The arc will be composed of <i>resolution</i> line segments.

**Note**

This uses an ofVbo internally. If VBOs are not supported by your video card, this may not work at all.

### 12.3.2.2 ofPath CX::Draw::arrowToPath ( float *length*, float *headOffsets*, float *headSize*, float *lineWidth* )

Draws an arrow to an ofPath. The outline of the arrow is drawn with strokes, so you can have the path be filled to have a solid arrow, or you can use non-zero width strokes in order to have the outline of an arrow. The arrow points up by default but you can rotate it with ofPath::rotate().

**Parameters**

<i>length</i>	The length of the arrow in pixels.
<i>headOffsets</i>	The angle between the main arrow body and the two legs of the tip, in degrees.
<i>headSize</i>	The length of the legs of the head in pixels.
<i>lineWidth</i>	The width of the lines used to draw the arrow (i.e. the distance between parallel strokes).

**Returns**

An ofPath containing the arrow. The center of the arrow is at (0,0) in the ofPath.

### 12.3.2.3 void CX::Draw::bezier ( std::vector< ofPoint > *controlPoints*, float *width*, unsigned int *resolution* )

Draws a bezier curve with an arbitrary number of control points. May become slow with a large number of control points. Uses de Casteljau's algorithm to calculate the curve points. See this awesome guide: <http://pomax.github.io/bezierinfo/>

**Parameters**

<i>controlPoints</i>	Control points for the bezier.
<i>width</i>	The width of the lines to be drawn. Uses CX::Draw::lines(std::vector<ofPoint>, float) internally to draw the connecting lines.
<i>resolution</i>	Controls the approximation of the bezier curve. There will be <i>resolution</i> line segments drawn to complete the curve ( <i>resolution</i> + 1 points).

### 12.3.2.4 void CX::Draw::centeredString ( int *x*, int *y*, std::string *s*, ofTrueTypeFont & *font* )

Equivalent to a call to CX::Draw::centeredString(ofPoint(x, y), s, font).

### 12.3.2.5 void CX::Draw::centeredString ( ofPoint *center*, std::string *s*, ofTrueTypeFont & *font* )

Draws a string centered on a given location using the given font. Strings are normally drawn such that the x coordinate gives the left edge of the string and the y coordinate gives the line above which the letters will be drawn, where some characters (like y or g) can descend below the line.

**Parameters**

<i>center</i>	The coordinates of the center of the string.
<i>s</i>	The string to draw.
<i>font</i>	A font that has already been prepared for use.

**12.3.2.6** `template<typename T> void CX::Draw::colorArc ( ofPoint center, std::vector< ofColor_< T >> colors, float radiusX, float radiusY, float width, float angleBegin, float angleEnd )`

Draws an arc with specified colors. The precision of the arc is controlled by how many colors are supplied.

#### Parameters

<i>center</i>	The center of the color wheel.
<i>colors</i>	The colors to use in the color arc.
<i>radiusX</i>	The radius of the color wheel in the X-axis.
<i>radiusY</i>	The radius of the color wheel in the Y-axis.
<i>width</i>	The width of the arc. The arc will extend half of the width in either direction from the radii.
<i>angleBegin</i>	The angle at which to begin the arc, in degrees.
<i>angleEnd</i>	The angle at which to end the arc, in degrees. If the arc goes in the "wrong" direction, try giving a negative value for <i>angleEnd</i> .

**12.3.2.7** `template<typename T> ofVbo CX::Draw::colorArcToVbo ( ofPoint center, std::vector< ofColor_< T >> colors, float radiusX, float radiusY, float width, float angleBegin, float angleEnd )`

See [CX::Draw::colorArc\(\)](#) for documentation. The only difference is that this function returns an ofVbo, which a complicated thing you can learn about here: <http://www.openframeworks.cc/documentation/gl/ofVbo.-html>

**12.3.2.8** `template<typename T> void CX::Draw::colorWheel ( ofPoint center, std::vector< ofColor_< T >> colors, float radius, float width, float angle )`

Draws a color wheel (really, a ring) with specified colors.

#### Parameters

<i>center</i>	The center of the color wheel.
<i>colors</i>	The colors to use in the color wheel.
<i>radius</i>	The radius of the color wheel.
<i>width</i>	The width of the color wheel. The color wheel will extend half of the width in either direction from the radius.
<i>angle</i>	The amount to rotate the color wheel.

```
//This code snippet draws an isoluminant color wheel to the screen using color conversion from LAB to RGB.
//Move the mouse and turn the scroll wheel to see different slices of the LAB space.
#include "CX_EntryPoint.h"
```

```
void runExperiment(void) {
    Input.setup(false, true);

    float L = 50;
    float aOff = 40;
    float bOff = 40;

    while (true) {
        if (Input.pollEvents()) {
            while (Input.Mouse.availableEvents() > 0) {
                CX_Mouse::Event mev = Input.Mouse.getNextEvent();

                if (mev.eventType == CX_Mouse::Event::SCROLLED) {
                    L += mev.y;
                }
            }
        }
    }
}
```

```

    if (mev.eventType == CX_Mouse::Event::MOVED) {
        aOff = mev.x - Display.getCenterOfDisplay().x;
        bOff = mev.y - Display.getCenterOfDisplay().y;
    }

    //Only if input has been received, redraw the color wheel
    vector<ofFloatColor> wheelColors(100);

    for (int i = 0; i < wheelColors.size(); i++) {
        float angle = (float)i / wheelColors.size() * 2 * PI;
        float A = sin(angle) * aOff;
        float B = cos(angle) * bOff;

        wheelColors[i] = Draw::convertToRGB("LAB", L, A, B); //Convert the L, A, and
        B components to the RGB color space.
    }

    Display.beginDrawingToBackBuffer();
    ofBackground(0);
    Draw::colorWheel(Display.getCenterOfDisplay(), wheelColors
    , 200, 70, 0);
    Display.endDrawingToBackBuffer();
    Display.swapBuffers();
}
}
}

```

#### 12.3.2.9 `template<typename T> ofVbo CX::Draw::colorWheelToVbo ( ofPoint center, std::vector< ofColor_< T >> colors, float radius, float width, float angle )`

See `CX::Draw::colorWheel()` for documentation. The only difference is that this function returns an `ofVbo`, which a complicated thing you can learn about here: <http://www.openframeworks.cc/documentation/gl/of-Vbo.html>

#### 12.3.2.10 `std::vector< double > CX::Draw::convertColors ( std::string conversionFormula, double S1, double S2, double S3 )`

Convert between two color spaces. This conversion uses this library internally: <http://www.getreuer.info/home/colospace>

##### Parameters

<i>conversion-Formula</i>	A formula of the format "SRC -> DEST", where SRC and DEST are valid color spaces. For example, if you wanted to convert from HSL to RGB, you would use "HSL -> RGB" as the formula. The whitespace is immaterial, but the arrow must exist (the arrow can point either direction). See this page for options for the color space: <a href="http://www.getreuer.info/home/colospace#TOC-MATLAB-Usage">http://www.getreuer.info/home/colospace#TOC-MATLAB-Usage</a> .
<i>S1</i>	Source coordinate 1. Corresponds to, e.g., the R in RGB. S2 and S3 follow as expected.

##### Returns

An vector of length 3 containing the converted coordinates in the destination color space. The value at index 0 corresponds to the first letter in the resulting color space and the next two indices proceed as expected.

```

vector<double> hslValues = Draw::convertColors("XYZ -> HSL", .7, .4, .6); //Convert
    x=.7, y=.4, z=.6 to HSL values.
hslValues[0]; //Access the hue value.
hslValues[2]; //Access the lightness value.

```

##### Note

The values returned by this function may not be in the allowed range for the destination color space. Make sure they are clamped to reasonable values if they are to be used directly.



## See Also

[CX::Draw::convertToRGB\(\)](#) is a convenience function for the most common conversion that will typically be done (something to RGB).

#### 12.3.2.11 ofFloatColor CX::Draw::convertToRGB ( std::string *inputColorSpace*, double *S1*, double *S2*, double *S3* )

This function converts from an arbitrary color space to the RGB color space. This is convenient, because in order to draw stimuli with a color, you need to have the color in the RGB space.

## Parameters

<i>inputColorSpace</i>	The color space to convert from. For example, if you wanted to convert from LAB coordinates, you would provide the string "LAB". See this page for more options for the color space: <a href="http://www.getreuer.info/home/colospace#TOC-MATLAB-Usage">http://www.getreuer.info/home/colospace#TOC-MATLAB-Usage</a> (ignore the MATLAB title on that page).
<i>S1</i>	Source coordinate 1. Corresponds to, e.g., the R in RGB. <i>S2</i> and <i>S3</i> follow as expected.

## Returns

An ofFloatColor containing the RGB coordinates. [Instances](#) of ofFloatColor can be implicitly converted to ofColor in assignment.

## See Also

Example code in the documentation for [CX::Draw::colorWheel\(\)](#) uses this function.

#### 12.3.2.12 void CX::Draw::fixationCross ( ofPoint *location*, float *armLength*, float *armWidth* )

Draws a standard fixation cross (plus sign).

## Parameters

<i>location</i>	Where to draw the fixation cross.
<i>armLength</i>	The length of the arms of the cross (end to end, not from the center).
<i>armWidth</i>	The width of the arms.

#### 12.3.2.13 ofPath CX::Draw::fixationCrossToPath ( float *armLength*, float *armWidth* )

Draws a standard fixation cross (plus sign) to an ofPath. The fixation cross will be centered on (0,0) in the ofPath.

## Parameters

<i>armLength</i>	The length of the arms of the cross (end to end, not from the center).
<i>armWidth</i>	The width of the arms.

## Returns

An ofPath containing the fixation cross.

#### 12.3.2.14 void CX::Draw::gabor ( ofPoint *center*, const CX\_GaborProperties\_t & *properties* )

Draws a gabor pattern with the specified properties.

## Parameters

<i>center</i>	The location of the center of the pattern.
<i>properties</i>	The settings to be used to generate the pattern.

12.3.2.15 `std::vector< ofPoint > CX::Draw::getBezierVertices ( std::vector< ofPoint > controlPoints, unsigned int resolution )`

Gets the vertices needed to draw a bezier curve. See [CX::Draw::bezier\(\)](#) for parameter meanings.

## Returns

A vector of points created based on the `controlPoints`.

12.3.2.16 `std::vector< ofPoint > CX::Draw::getFixationCrossVertices ( float armLength, float armWidth )`

Gets teh vertices defining the perimeter of a standard fixation cross (plus sign).

## Parameters

<i>armLength</i>	The length of the arms of the cross (end to end, not from the center).
<i>armWidth</i>	The width of the arms.

## Returns

A vector with the 12 needed vertices.

12.3.2.17 `template<typename ofColorType > std::vector<ofColorType> CX::Draw::getRGBSpectrum ( unsigned int colorCount )`

Sample colors from the RGB spectrum with variable precision. Colors will be sampled beginning with red, continue through yellow, green, cyan, blue, violet, and almost, but not quite, back to red.

## Template Parameters

<i>ofColorType</i>	An oF color type. One of: <code>ofColor</code> , <code>ofFloatColor</code> , or <code>ofShortColor</code> , or <code>ofColor_&lt;some-OtherType&gt;</code> .
--------------------	--

## Parameters

<i>colorCount</i>	The number of colors to draw from the RGB spectrum, which will be rounded up to the next multiple of 6.
-------------------	---

## Returns

A vector containing the sampled colors with a number of colors equal to `colorCount` rounded up to the next multiple of 6.

12.3.2.18 `std::vector< ofPoint > CX::Draw::getStarVertices ( unsigned int numberOfPoints, float innerRadius, float outerRadius, float rotationDeg )`

This function obtains the vertices needed to draw an N pointed star.

## Parameters

<i>numberOfPoints</i>	The number of points in the star.
<i>innerRadius</i>	The distance from the center of the star at which the inner points of the star hit.
<i>outerRadius</i>	The distance from the center of the star to the outer points of the star.
<i>rotationDeg</i>	The number of degrees to rotate the star. 0 degrees has one point of the star pointing up. Positive values rotate the star counter-clockwise.

## Returns

A vector of points defining the vertices needed to draw the star. There will be  $2 * \text{numberOfPoints} + 1$  vertices with the last vertex equal to the first vertex. The vertices are centered on (0, 0).

12.3.2.19 ofPixels CX::Draw::greyscalePatternToPixels ( const CX\_PatternProperties\_t & *properties* )

This function draws a greyscale pattern, like a gabor, to an ofPixels object. The results of this function are not intended to be used directly, but to be applied to an image, for example. The pattern lacks color information, but can be used as an alpha mask or used to control color intensity, or otherwise.

## Parameters

<i>properties</i>	The properties that will be used to create the pattern.
-------------------	---

## Returns

An ofPixels object containing the pattern.

## See Also

[CX::Draw::gabor\(\)](#) can be used to draw a [gabor](#) patch with color and transparency, based on a greyscale pattern.

12.3.2.20 void CX::Draw::line ( ofPoint *p1*, ofPoint *p2*, float *width* )

This function draws a line from p1 to p2 with the given width.

## Note

This function supersedes ofLine because the line width of the line drawn with ofLine cannot currently be set to a value greater than 1.

12.3.2.21 void CX::Draw::lines ( std::vector< ofPoint > *points*, float *lineWidth* )

This function draws a series of line segments to connect the given points. At each point, the line segments are joined with a circle, which results in overdraw. As a result, this function does not work well with transparency. A workaround is to draw with max alpha into an fbo and then draw the fbo with transparency.

## Parameters

<i>points</i>	The points to connect with lines.
<i>lineWidth</i>	The width of the line.

## Note

If the last point is the same as the first point, the final line segment junction will be joined with a circle.

12.3.2.22 `template<typename T > void CX::Draw::patternMask ( ofPoint center, float width, float height, float squareSize,  
std::vector< ofColor_< T >> colors = std::vector<ofColor_<T>>(0) )`

This function draws a pattern mask created with a large number of small squares.

## Parameters

<i>width</i>	The width of the area to draw to, in pixels.
<i>height</i>	The height of the area to draw to, in pixels.
<i>pixelSize</i>	The size of each small square making up the shape, in pixels.
<i>colors</i>	Optional. If a vector of colors is provided, colors will be sampled in blocks using an <a href="#">Algo::Block-Sampler</a> from the provided colors. If no colors are provided, each color will be chosen randomly by sampling a hue value in the HSB color space, with the S and B held constant at maximum values (i.e. each color will be a bright, fully saturated color).

12.3.2.23 void CX::Draw::ring ( ofPoint *center*, float *radius*, float *width*, unsigned int *resolution* )

This function draws a ring, i.e. an unfilled circle. The filled area of the ring is between  $\text{radius} + \text{width}/2$  and  $\text{radius} - \text{width}/2$ .

## Parameters

<i>center</i>	The center of the ring.
<i>radius</i>	The radius of the ring.
<i>width</i>	The radial width of the ring.
<i>resolution</i>	The ring will be approximated with a number of line segments, which is controlled with <i>resolution</i> .

## Note

This function supersedes drawing rings with `ofCircle` with `fill` set to `off` because the line width of the unfilled circle cannot be set to a value greater than 1.

12.3.2.24 void CX::Draw::saveFboToFile ( ofFbo & *fbo*, std::string *filename* )

Saves the contents of an `ofFbo` to a file. The file type is hinted by the file extension you provide as part of the file name.

## Parameters

<i>fbo</i>	The framebuffer to save.
<i>filename</i>	The path of the file to save. The file extension determines the type of file that is saved. If no file extension is given, nothing gets saved. Many standard file types are supported: png, bmp, jpg, gif, etc. However, if the fbo has an alpha channel, only png works properly (at least of those I have tested).

12.3.2.25 ofPath CX::Draw::squircleToPath ( double *radius*, double *amount* )

This function draws an approximation of a squircle (<http://en.wikipedia.org/wiki/Squircle>) using Bezier curves. The squircle will be centered on (0,0) in the `ofPath`.

## Parameters

<i>radius</i>	The radius of the largest circle that can be enclosed in the squircle.
<i>amount</i>	The "squirliness" of the squircle. The default (0.9) seems like a pretty good amount for a good approximation of a squircle, but different amounts can give different sorts of shapes.

## Returns

An `ofPath` containing the squircle.

12.3.2.26 void CX::Draw::star ( ofPoint *center*, unsigned int *numberOfPoints*, float *innerRadius*, float *outerRadius*, float *rotationDeg* )

This draws an N-pointed star.

## Parameters

<i>center</i>	The point at the center of the star.
<i>numberOfPoints</i>	The number of points in the star.
<i>innerRadius</i>	The distance from the center of the star to where the inner points of the star hit.
<i>outerRadius</i>	The distance from the center of the star to the outer points of the star.
<i>rotationDeg</i>	The number of degrees to rotate the star. 0 degrees has one point of the star pointing up. Positive values rotate the star counter-clockwise.

12.3.2.27 ofPath CX::Draw::starToPath ( unsigned int *numberOfPoints*, float *innerRadius*, float *outerRadius* )

This draws an N-pointed star to an ofPath. The star will be centered on (0,0) in the ofPath.

## Parameters

<i>numberOfPoints</i>	The number of points in the star.
<i>innerRadius</i>	The distance from the center of the star at which the inner points of the star hit.
<i>outerRadius</i>	The distance from the center of the star to the outer points of the star.

## Returns

An ofPath containing the star.

## 12.4 CX::Instances Namespace Reference

## Variables

- [CX\\_Clock](#) Clock
- [CX\\_Display](#) Display
- [CX\\_InputManager](#) Input = CX::Private::inputManagerFactory()
- [CX\\_Logger](#) Log
- [CX\\_RandomNumberGenerator](#) RNG

## 12.4.1 Detailed Description

This namespace contains instances of some classes that are fundamental to the functioning of [CX](#).

## 12.5 CX::Synth Namespace Reference

## Classes

- class [ModuleBase](#)
- class [ModuleParameter](#)
- class [AdditiveSynth](#)
- class [Adder](#)
- class [Clamper](#)
- class [Envelope](#)
- class [Filter](#)
- class [GenericOutput](#)
- class [Mixer](#)
- class [Multiplier](#)

- class [Oscillator](#)
- class [Splitter](#)
- class [SoundBufferInput](#)
- class [StreamOutput](#)
- class [StereoStreamOutput](#)
- class [SoundBufferOutput](#)
- class [StereoSoundBufferOutput](#)
- class [TrivialGenerator](#)
- class [FIRFilter](#)

## Functions

- double [sinc](#) (double x)
- double [relativeFrequency](#) (double f, double semitoneDifference)
- [ModuleBase](#) & [operator>>](#) ([ModuleBase](#) &l, [ModuleBase](#) &r)
- void [operator>>](#) ([ModuleBase](#) &l, [ModuleParameter](#) &r)

### 12.5.1 Detailed Description

This namespace contains a number of classes that can be combined together to form a modular synthesizer that can be used to procedurally generate sound stimuli. There are methods for saving the sound stimuli to a file for later use or directly outputting the sounds to sound hardware. There is also a way to use the data from a [CX\\_SoundBuffer](#) as the input to the synth.

There are two types of oscillators ([Oscillator](#) and [AdditiveSynth](#)), an ADSR [Envelope](#), two types of filters ([Filter](#) and [FIRFilter](#)), a [Splitter](#) and a [Mixer](#), and some utility classes for adding, multiplying, and clamping values.

Making your own modules is simplified by the fact that all modules inherit from [ModuleBase](#). You only need to overload one function from [ModuleBase](#) in order to have a functional module, although there are some other functions that can be overloaded for advanced uses.

### 12.5.2 Function Documentation

#### 12.5.2.1 [ModuleBase](#) & [CX::Synth::operator>>](#) ( [ModuleBase](#) & l, [ModuleBase](#) & r )

This operator is used to connect modules together. `l` is set as the input for `r`.

```
Oscillator osc;
StreamOutput out;
osc >> out; //Connect osc as the input for out.
```

#### 12.5.2.2 void [CX::Synth::operator>>](#) ( [ModuleBase](#) & l, [ModuleParameter](#) & r )

This operator connects a module to the module parameter. It is not possible to connect a module parameter as an input for anything: They are dead ends.

```
using namespace CX::Synth;
Oscillator osc;
Envelope fenv;
Adder add;
add.amount = 500;
fenv >> add >> osc.frequency; //Connect the envelope as the input for the frequency of the oscillator with
                              an offset of 500 Hz.
```



12.5.2.3 double CX::Synth::relativeFrequency ( double *f*, double *semitoneDifference* )

This function returns the frequency that is *semitoneDifference* semitones from *f*.

## Parameters

$f$	The starting frequency.
<i>semitone-Difference</i>	The difference (positive or negative) from $f$ to the desired output frequency.

## Returns

The final frequency.

12.5.2.4 `double CX::Synth::sinc ( double x )`

The sinc function, defined as  $\sin(x)/x$ .

## 12.6 CX::Util Namespace Reference

## Classes

- class [CX\\_LapTimer](#)
- class [CX\\_SegmentProfiler](#)
- class [CX\\_TrialController](#)
- class [CX\\_BaseUnitConverter](#)
- class [CX\\_DegreeToPixelConverter](#)
- class [CX\\_LengthToPixelConverter](#)
- class [CX\\_CoordinateConverter](#)

## Enumerations

- enum [CX\\_RoundingConfiguration](#) { [CX\\_RoundingConfiguration::ROUND\\_TO\\_NEAREST](#), [CX\\_RoundingConfiguration::ROUND\\_UP](#), [CX\\_RoundingConfiguration::ROUND\\_DOWN](#), [CX\\_RoundingConfiguration::ROUND\\_TOWARD\\_ZERO](#) }

## Functions

- float [degreesToPixels](#) (float degrees, float pixelsPerUnit, float viewingDistance)
- float [pixelsToDegrees](#) (float pixels, float pixelsPerUnit, float viewingDistance)
- unsigned int [getMaaSampleCount](#) (void)
- bool [checkOFVersion](#) (int versionMajor, int versionMinor, int versionPatch)
- bool [writeToFile](#) (std::string filename, std::string data, bool append)
- double [round](#) (double d, int roundingPower, [CX::Util::CX\\_RoundingConfiguration](#) c)
- std::map< std::string, std::string > [readKeyValueFile](#) (std::string filename, std::string delimiter, bool trimWhitespace, std::string commentString)
- float [getAngleBetweenPoints](#) (ofPoint p1, ofPoint p2)
- ofPoint [getRelativePointFromDistanceAndAngle](#) (ofPoint start, float distance, float angle)
- template<typename T >  
std::vector< T > [arrayToVector](#) (T arr[], unsigned int arraySize)
- template<typename T >  
std::vector< T > [sequence](#) (T start, T end, T stepSize)
- template<typename T >  
std::vector< T > [sequenceSteps](#) (T start, unsigned int steps, T stepSize)

- `template<typename T >`  
`std::vector< T > sequenceAlong (T start, T end, unsigned int steps)`
- `template<typename T >`  
`std::vector< T > intVector (T start, T end)`
- `template<typename T >`  
`std::vector< T > repeat (T value, unsigned int times)`
- `template<typename T >`  
`std::vector< T > repeat (std::vector< T > values, unsigned int times, unsigned int each=1)`
- `template<typename T >`  
`std::vector< T > repeat (std::vector< T > values, std::vector< unsigned int > each, unsigned int times=1)`
- `template<typename T >`  
`std::string vectorToString (std::vector< T > values, std::string delimiter=",", int significantDigits=8)`
- `template<typename T >`  
`std::vector< T > stringToVector (std::string s, std::string delimiter)`
- `template<typename T >`  
`T clamp (T val, T minimum, T maximum)`
- `template<typename T >`  
`std::vector< T > clamp (std::vector< T > vals, T minimum, T maximum)`
- `template<typename T >`  
`std::vector< T > unique (std::vector< T > vals)`
- `template<typename T >`  
`std::vector< T > concatenate (const std::vector< T > &A, const std::vector< T > &B)`
- `template<typename T >`  
`std::vector< T > exclude (const std::vector< T > &A, const std::vector< T > &B)`
- `template<typename T >`  
`T max (std::vector< T > vals)`
- `template<typename T >`  
`T min (std::vector< T > vals)`
- `template<typename T >`  
`T mean (std::vector< T > vals)`
- `template<typename T_OUT , typename T_IN >`  
`T_OUT mean (std::vector< T_IN > vals)`
- `template<typename T >`  
`T var (std::vector< T > vals)`
- `template<typename T_OUT , typename T_IN >`  
`T_OUT var (std::vector< T_IN > vals)`

### 12.6.1 Detailed Description

This namespace contains a variety of utility functions.

### 12.6.2 Enumeration Type Documentation

#### 12.6.2.1 enum CX::Util::CX\_RoundingConfiguration [strong]

The way in which numbers should be rounded with [round\(\)](#).

Enumerator

- ROUND\_TO\_NEAREST*** Round to the nearest number.
- ROUND\_UP*** Round to the number above the current number.
- ROUND\_DOWN*** Round to the number below the current number.
- ROUND\_TOWARD\_ZERO*** Round toward zero.

### 12.6.3 Function Documentation

#### 12.6.3.1 `template<typename T> std::vector< T > CX::Util::arrayToVector ( T arr[], unsigned int arraySize )`

Copies arraySize elements of an array of T to a vector<T>.

##### Template Parameters

<code>&lt; T &gt;</code>	The type of the array. Is often inferred by the compiler.
--------------------------	---

##### Parameters

<i>arr</i>	The array of data to put into the vector.
<i>arraySize</i>	The length of the array, or the number of elements to copy from the array if not all of the elements are wanted.

##### Returns

The elements in a vector.

#### 12.6.3.2 `bool CX::Util::checkOFVersion ( int versionMajor, int versionMinor, int versionPatch )`

Checks that the version of oF that is used during compilation matches the requested version. If the desired version was 0.7.1, simply input (0, 7, 1) as the arguments. A warning will be logged if the versions don't match.

##### Returns

True if the versions match, false otherwise.

#### 12.6.3.3 `template<typename T> T CX::Util::clamp ( T val, T minimum, T maximum )`

Clamps a value (i.e. forces the value to be between two bounds). If the value is outside of the bounds, it is set to be equal to the nearest bound.

##### Parameters

<i>val</i>	The value to clamp.
<i>minimum</i>	The lower bound. Must be less than or equal to maximum.
<i>maximum</i>	The upper bound. Must be greater than or equal to minimum.

##### Returns

The clamped value.

#### 12.6.3.4 `template<typename T> std::vector< T > CX::Util::clamp ( std::vector< T > vals, T minimum, T maximum )`

Clamps a vector of values. See [CX::Util::clamp\(\)](#).

##### Parameters

<i>vals</i>	The values to clamp.
<i>minimum</i>	The lower bound. Must be less than or equal to maximum.

<i>maximum</i>	The upper bound. Must be greater than or equal to minimum.
----------------	--

**Returns**

The clamped values.

**12.6.3.5** `template<typename T> std::vector< T > CX::Util::concatenate ( const std::vector< T > & A, const std::vector< T > & B )`

Concatenates together two vectors A and B.

**Parameters**

<i>A</i>	The first vector of values.
<i>B</i>	The second vector of values.

**Returns**

The concatenation of A and B, being a vector containing {A1, A2, ... An, B1, B2, ... Bn}.

**12.6.3.6** `float CX::Util::degreesToPixels ( float degrees, float pixelsPerUnit, float viewingDistance )`

Returns the number of pixels needed to subtend deg degrees of visual angle. You might want to round this if you want to align to pixel boundaries. However, if you are antialiasing your stimuli you might want to use floating point values to get precise subpixel rendering.

**Parameters**

<i>degrees</i>	Number of degrees.
<i>pixelsPerUnit</i>	The number of pixels per distance unit on the target monitor. You can pick any unit of distance, as long as <i>viewingDistance</i> has the same unit.
<i>viewingDistance</i>	The distance of the viewer from the monitor, with the same distance unit as <i>pixelsPerUnit</i> .

**Returns**

The number of pixels needed.

**12.6.3.7** `template<typename T> std::vector< T > CX::Util::exclude ( const std::vector< T > & values, const std::vector< T > & exclude )`

Gets the values from *values* that do not match the values in *exclude*.

**Parameters**

<i>values</i>	The set of values to select from.
<i>exclude</i>	The set of values to exclude from <i>values</i> .

**Returns**

A vector containing the values that were not excluded. This vector may be empty.

**12.6.3.8** `float CX::Util::getAngleBetweenPoints ( ofPoint p1, ofPoint p2 )`

Returns the angle in degrees "between" p1 and p2. If you take the difference between p2 and p1, you get a resulting vector, V, that gives the displacement from p1 to p2. Imagine that you begin at (0, 0) and move to (abs(V.x), 0), creating

a line segment. Now if you "rotate" this line segment clockwise, like the hand of a clock, until you reach V, the angle rotated through is the value returned by this function.

This is useful if you want to know, e.g., what angle you must travel on to get from some arbitrary point on screen to where the mouse cursor is.

#### Parameters

<i>p1</i>	The start point of the vector V.
<i>p2</i>	The end point of V. If p1 and p2 are reversed, the angle will be off by 180 degrees.

#### Returns

The angle in degrees between p1 and p2.

#### 12.6.3.9 unsigned int CX::Util::getMsaaSampleCount ( void )

This function retrieves the MSAA ([http://en.wikipedia.org/wiki/Multisample\\_anti-aliasing](http://en.wikipedia.org/wiki/Multisample_anti-aliasing)) sample count. The sample count can be set by calling CX::relaunchWindow() with the desired sample count set in the argument to relaunchWindow().

#### 12.6.3.10 ofPoint CX::Util::getRelativePointFromDistanceAndAngle ( ofPoint start, float distance, float angle )

This function begins at point *start*, travels *distance* from that point along *angle*, and returns the resulting point.

#### Parameters

<i>start</i>	The starting point.
<i>distance</i>	The distance to travel.
<i>angle</i>	The angle to travel on, in degrees.

#### 12.6.3.11 template<typename T> std::vector< T> CX::Util::intVector ( T start, T end )

Creates a vector of integers going from start to end. start may be greater than end, in which case the returned values will be in descending order. This is similar to using CX::sequence, but the step size is fixed to 1 and it works properly when trying to create a descending sequence of unsigned integers.

#### Returns

A vector of the values int the sequence.

#### 12.6.3.12 template<typename T> T CX::Util::max ( std::vector< T> vals )

Finds the maximum value in a vector of values.

#### Template Parameters

<i>T</i>	The type of data to be operated on. This type must have operator> defined.
----------	--

#### Parameters

<i>vals</i>	The vector of values.
-------------	-----------------------

#### Returns

The maximum value in the vector.

12.6.3.13 `template<typename T> T CX::Util::mean ( std::vector< T > vals )`

Calculates the mean value of a vector of values.

## Template Parameters

<i>T</i>	The type of data to be operated on and returned. This type must have operator+(T) and operator/(unsigned int) defined.
----------	--

## Parameters

<i>vals</i>	The vector of values.
-------------	-----------------------

## Returns

The mean of the vector.

12.6.3.14 `template<typename T_OUT , typename T_IN > T_OUT CX::Util::mean ( std::vector< T_IN > vals )`

Calculates the mean value of a vector of values.

## Template Parameters

<i>T_OUT</i>	The type of data to be returned. This type must have operator+(T_IN) and operator/(unsigned int) defined.
<i>T_IN</i>	The type of data to be operated on.

## Parameters

<i>vals</i>	The vector of values.
-------------	-----------------------

## Returns

The mean of the vector.

12.6.3.15 `template<typename T > T CX::Util::min ( std::vector< T > vals )`

Finds the minimum value in a vector of values.

## Template Parameters

<i>T</i>	The type of data to be operated on. This type must have operator< defined.
----------	--

## Parameters

<i>vals</i>	The vector of values.
-------------	-----------------------

## Returns

The minimum value in the vector.

12.6.3.16 `float CX::Util::pixelsToDegrees ( float pixels, float pixelsPerUnit, float viewingDistance )`

The inverse of [CX::Util::degreesToPixels\(\)](#).

12.6.3.17 `std::map< std::string, std::string > CX::Util::readKeyValueFile ( std::string filename, std::string delimiter, bool trimWhitespace, std::string commentString )`

This function reads in a file containing information stored as key-value pairs. A file of this kind could look like:

```
Key=Value
blue=0000FF
unleash_penguins=true
```



This type of file is often used for configuration of a program. This function simply provides a simple way to read in such data.

## Parameters

<i>filename</i>	The name of the file containing key-value data.
<i>delimiter</i>	The string that separates the key from the value. In the example, it is "=".
<i>trimWhitespace</i>	If true, whitespace characters surrounding both the key and value will be removed.
<i>commentString</i>	If commentString is not the empty string (i.e. ""), everything on a line following the first instance of commentString will be ignored.

## Returns

A map<string, string>, where the keys are the keys to the map.

### 12.6.3.18 `template<typename T> std::vector< T> CX::Util::repeat ( T value, unsigned int times )`

Repeats value "times" times.

## Parameters

<i>value</i>	The value to be repeated.
<i>times</i>	The number of times to repeat the value.

## Returns

A vector containing times copies of the repeated value.

### 12.6.3.19 `template<typename T> std::vector< T> CX::Util::repeat ( std::vector< T> values, unsigned int times, unsigned int each = 1 )`

Repeats the elements of values. Each element of values is repeated "each" times and then the process of repeating the elements is repeated "times" times.

## Parameters

<i>values</i>	Vector of values to be repeated.
<i>times</i>	The number of times the process should be performed.
<i>each</i>	Number of times each element of values should be repeated.

## Returns

A vector of the repeated values.

### 12.6.3.20 `template<typename T> std::vector< T> CX::Util::repeat ( std::vector< T> values, std::vector< unsigned int> each, unsigned int times = 1 )`

Repeats the elements of values. Each element of values is repeated "each" times and then the process of repeating the elements is repeated "times" times.

## Parameters

<i>values</i>	Vector of values to be repeated.
<i>each</i>	Number of times each element of values should be repeated. Must be the same length as values. If not, an error is logged and an empty vector is returned.

<i>times</i>	The number of times the process should be performed.
--------------	--

**Returns**

A vector of the repeated values.

### 12.6.3.21 `double CX::Util::round ( double d, int roundingPower, CX::Util::CX_RoundingConfiguration c )`

Rounds the given double to the given power of 10.

**Parameters**

<i>d</i>	The number to be rounded.
<i>roundingPower</i>	The power of 10 to round <i>d</i> to. For the value 34.56, the results with different rounding powers (and <i>c</i> = ROUND_TO_NEAREST) are as follows: RP = 0 -> 35; RP = 1 -> 30; RP = -1 -> 34.6.
<i>c</i>	The type of rounding to do, from the <a href="#">CX::Util::CX_RoundingConfiguration</a> enum. You can round up, down, to nearest (default), and toward zero.

**Returns**

The rounded value.

### 12.6.3.22 `template<typename T> std::vector< T> CX::Util::sequence ( T start, T end, T stepSize )`

Creates a sequence of numbers from start to end by steps of size stepSize. start may be greater than end, but only if stepSize is less than 0. If start is less than end, stepSize must be greater than 0.

Example call: `sequence<double>(1, 3.3, 2)` results in a vector containing {1, 3}

**Parameters**

<i>start</i>	The start of the sequence. You are guaranteed to get this value in the sequence.
<i>end</i>	The number past which the sequence should end. You are not guaranteed to get this value.
<i>stepSize</i>	A nonzero number.

**Returns**

A vector containing the sequence.

### 12.6.3.23 `template<typename T> std::vector< T> CX::Util::sequenceAlong ( T start, T end, unsigned int outputLength )`

Creates a sequence from start to end, where the size of each step is chosen so that the length of the sequence is equal to outputLength.

**Parameters**

<i>start</i>	The value at which to start the sequence.
<i>end</i>	The value to which to end the sequence.
<i>outputLength</i>	The number of elements in the returned sequence.

**Returns**

A vector containing the sequence.

12.6.3.24 `template<typename T> std::vector< T> CX::Util::sequenceSteps ( T start, unsigned int steps, T stepSize )`

Make a sequence starting from *start* and taking *steps* steps of *stepSize*.

`sequenceSteps( 1.5, 4, 2.5 );`

Creates the sequence {1.5, 4, 6.5, 9, 11.5}

#### Parameters

<i>start</i>	Value from which to start.
<i>steps</i>	The number of steps to take.
<i>stepSize</i>	The size of each step.

#### Returns

A vector containing the sequence.

12.6.3.25 `template<typename T> std::vector< T> CX::Util::stringToVector ( std::string s, std::string delimiter )`

This function takes a string, splits it on the delimiter, and converts each delimited part of the string to T, returning a `vector<T>`.

#### Template Parameters

<i>T</i>	The type of the data encoded in the string.
----------	---

#### Parameters

<i>s</i>	The string containing the encoded data.
<i>delimiter</i>	The string that delimits the elements of the data.

#### Returns

A vector of the encoded data converted to T.

12.6.3.26 `template<typename T> std::vector< T> CX::Util::unique ( std::vector< T> vals )`

Uses `std::unique` to find all of the unique values in *vals* and return copies of those values.

#### Parameters

<i>vals</i>	Th vector of values to find unique values in.
-------------	---

#### Returns

A vector containing the unique values in *vals*.

12.6.3.27 `template<typename T> T CX::Util::var ( std::vector< T> vals )`

Calculates the sample variance of a vector of values.

#### Template Parameters

<i>T</i>	The type of data.
----------	-------------------

**Parameters**

<i>vals</i>	The data.
-------------	-----------

**Returns**

The sample variance.

**12.6.3.28** `template<typename T_OUT , typename T_IN > T_OUT CX::Util::var ( std::vector< T_IN > vals )`

Calculates the sample variance of a vector of values.

**Template Parameters**

<i>T_OUT</i>	The type of data to be returned.
<i>T_IN</i>	The type of data to be operated on.

**Parameters**

<i>vals</i>	The vector of values.
-------------	-----------------------

**Returns**

The mean of the vector.

**12.6.3.29** `template<typename T > std::string CX::Util::vectorToString ( std::vector< T > values, std::string delimiter = " , " , int significantDigits = 8 )`

This function converts a vector of values to a string representation of the values.

**Parameters**

<i>values</i>	The vector of values to convert.
<i>delimiter</i>	A string that is used to separate the elements of <i>value</i> in the final string.
<i>significantDigits</i>	Only for floating point types. The number of significant digits in the value.

**Returns**

A string containing a representation of the vector of values.

**12.6.3.30** `bool CX::Util::writeToFile ( std::string filename, std::string data, bool append )`

Writes data to a file, either appending the data to an existing file or creating a new file, overwriting any existing file with the given filename.

**Parameters**

<i>filename</i>	Name of the file to write to. If it is a relative file name, it will be placed relative the the data directory.
-----------------	---

<i>data</i>	The data to write
<i>append</i>	If true, data will be appended to an existing file, if it exists. If append is false, any existing file will be overwritten and a warning will be logged. If no file exists, a new one will be created.

#### Returns

True if an error was encountered while writing the file, true otherwise. If there was an error, an error message will be logged.

## 13 Class Documentation

### 13.1 CX::Synth::Adder Class Reference

```
#include <CX_ModularSynth.h>
```

Inherits [CX::Synth::ModuleBase](#).

#### Public Member Functions

- double [getNextSample](#) (void) override

#### Public Attributes

- [ModuleParameter](#) **amount**

#### Additional Inherited Members

#### 13.1.1 Detailed Description

This class simply takes an input and adds an `amount` to it. The `amount` can be negative, in which case this class is a subtractor. If there is no input to this module, it behaves as though the input is 0, so the output value will be equal to `amount`. Thus, it can also behave as a numerical constant.

#### 13.1.2 Member Function Documentation

##### 13.1.2.1 double Adder::getNextSample( void ) [override],[virtual]

This function should be overloaded for any derived class that can be used as the input for another module.

Reimplemented from [CX::Synth::ModuleBase](#).

The documentation for this class was generated from the following files:

- CX\_ModularSynth.h
- CX\_ModularSynth.cpp

### 13.2 CX::Synth::AdditiveSynth Class Reference

```
#include <CX_ModularSynth.h>
```

Inherits [CX::Synth::ModuleBase](#).

## Public Types

- enum **HarmonicSeriesType** { HS\_MULTIPLE, HS\_SEMITONE, HS\_USER\_FUNCTION }
- enum [HarmonicAmplitudeType](#) { SINE, SQUARE, SAW, TRIANGLE }
- typedef float **wavePos\_t**
- typedef float **amplitude\_t**

## Public Member Functions

- void [setFundamentalFrequency](#) (double f)
- void [setStandardHarmonicSeries](#) (unsigned int harmonicCount)
- void [setHarmonicSeries](#) (unsigned int harmonicCount, HarmonicSeriesType type, double controlParameter)
- void [setHarmonicSeries](#) (unsigned int harmonicCount, std::function< double(unsigned int)> userFunction)
- void [setAmplitudes](#) (HarmonicAmplitudeType type)
- void [setAmplitudes](#) (HarmonicAmplitudeType t1, [HarmonicAmplitudeType](#) t2, double mixture)
- void [setAmplitudes](#) (std::vector< amplitude\_t > amps)
- std::vector< amplitude\_t > [calculateAmplitudes](#) (HarmonicAmplitudeType type, unsigned int count)
- void [pruneLowAmplitudeHarmonics](#) (double tol)
- double [getNextSample](#) (void) override

## Additional Inherited Members

## 13.2.1 Detailed Description

This class is an implementation of an additive synthesizer. Additive synthesizers are essentially an inverse fourier transform. You specify at which frequencies you want to have a sine wave and the amplitudes of those waves, and they are combined together into a single waveform.

The frequencies are referred to as harmonics, due to the fact that typical audio applications of additive synths use the standard harmonic series ( $f(i) = f\_fundamental * i$ ). However, setting the harmonics to values not found in the standard harmonic series can result in really unusual and interesting sounds.

The output of the additive synth is not easily bounded between -1 and 1 due to various oddities of additive synthesis. For example, although in the limit as the number of harmonics goes to infinity square and sawtooth waves made with additive synthesis are bounded between -1 and 1, with smaller numbers of harmonics the amplitudes actually overshoot these bounds slightly. Of course, if an unusual harmonic series is used with arbitrary amplitudes, it can be hard to know if the output of the synth will be within the bounds. A [Synth::Multiplier](#) can help deal with this.

## 13.2.2 Member Enumeration Documentation

## 13.2.2.1 enum CX::Synth::AdditiveSynth::HarmonicAmplitudeType

Assuming that the standard harmonic series is being used, the values in this enum, when passed to [setAmplitudes\(\)](#), cause the amplitudes of the harmonics to be set in such a way as to produce the desired waveform.

## 13.2.3 Member Function Documentation

13.2.3.1 std::vector< AdditiveSynth::amplitude\_t > AdditiveSynth::calculateAmplitudes ( [HarmonicAmplitudeType](#) type, unsigned int count )

This is a specialty function that only works when the standard harmonic series is being used. If so, it calculates the amplitudes needed for the harmonics so as to produce the specified waveform type.

## Parameters

<i>type</i>	The type of waveform that should be output from the additive synth.
<i>count</i>	The number of harmonics.

## Returns

A vector of amplitudes.

### 13.2.3.2 `double AdditiveSynth::getNextSample ( void ) [override],[virtual]`

This function should be overloaded for any derived class that can be used as the input for another module.

Reimplemented from [CX::Synth::ModuleBase](#).

### 13.2.3.3 `void AdditiveSynth::pruneLowAmplitudeHarmonics ( double tol )`

This function removes all harmonics that have an amplitude that is less than or equal to a tolerance times the amplitude of the harmonic with the greatest absolute amplitude.

The result of this pruning is that the synthesizer will be more computationally efficient but provide a possibly worse approximation of the desired waveform.

## Parameters

<i>tol</i>	<code>tol</code> is interpreted differently depending on its value. If <code>tol</code> is greater than or equal to 0, it is treated as a proportion of the amplitude of the frequency with the greatest amplitude. If <code>tol</code> is less than 0, it is treated as the difference in decibels between the frequency with the greatest amplitude and the tolerance.
------------	--

## Note

Because only harmonics with an amplitude less than or equal to the tolerance times an amplitude are pruned, setting `tol` to 0 will remove harmonics with 0 amplitude, but no others.

### 13.2.3.4 `void AdditiveSynth::setAmplitudes ( HarmonicAmplitudeType type )`

This function sets the amplitudes of the harmonics based on the chosen type. The resulting waveform will only be correct if the harmonic series is the standard harmonic series (see [setStandardHarmonicSeries\(\)](#)).

## Parameters

<i>type</i>	The type of wave calculate amplitudes for.
-------------	--

### 13.2.3.5 `void AdditiveSynth::setAmplitudes ( HarmonicAmplitudeType t1, HarmonicAmplitudeType t2, double mixture )`

This function sets the amplitudes of the harmonics based on a mixture of the chosen types. The resulting waveform will only be correct if the harmonic series is the standard harmonic series (see [setStandardHarmonicSeries\(\)](#)). This is a convenient way to morph between waveforms.

### 13.2.3.6 `void AdditiveSynth::setAmplitudes ( std::vector< amplitude_t > amps )`

This function sets the amplitudes of the harmonics to arbitrary values as specified in `amps`.



## Parameters

<i>amps</i>	The amplitudes of the harmonics. If this vector does not contain as many values as there are harmonics, the unspecified amplitudes will be set to 0.
-------------	--

13.2.3.7 void AdditiveSynth::setFundamentalFrequency ( double *f* )

This sets the fundamental frequency being used by the [AdditiveSynth](#). All harmonics are adjusted based on the new fundamental.

13.2.3.8 void AdditiveSynth::setHarmonicSeries ( unsigned int *harmonicCount*, HarmonicSeriesType *type*, double *controlParameter* )

## Parameters

<i>type</i>	The type of harmonic series to generate. Can be either HS_MULTIPLE or HS_SEMITONE. For HS_MULTIPLE, each harmonic's frequency will be some multiple of the fundamental frequency, depending on the harmonic number and controlParameter. For HS_SEMITONE, each harmonic's frequency will be some number of semitones above the previous frequency, based on controlParameter (specifying the number of semitones).
<i>controlParameter</i>	If <i>type</i> == HS_MULTIPLE, the frequency for harmonic <i>i</i> will be <i>i</i> * controlParameter, where the fundamental gives the value 1 for <i>i</i> . If <i>type</i> == HS_SEMITONE, the frequency for harmonic <i>i</i> will be $\text{pow}(2, (i - 1) * \text{controlParameter}/12)$ , where the fundamental gives the value 1 for <i>i</i> .

## Note

If *type* == HS\_MULTIPLE and *controlParameter* == 1, then the standard harmonic series will be generated.

If *type* == HS\_SEMITONE, *controlParameter* does not need to be an integer.

13.2.3.9 void AdditiveSynth::setHarmonicSeries ( unsigned int *harmonicCount*, std::function< double(unsigned int)> *userFunction* )

This function calculates the harmonic series from a function supplied by the user.

## Parameters

<i>harmonicCount</i>	The number of harmonics to generate.
<i>userFunction</i>	The user function takes an integer representing the harmonic number, where the fundamental has the value 1, and returns the frequency that should be used for that harmonic.

13.2.3.10 void AdditiveSynth::setStandardHarmonicSeries ( unsigned int *harmonicCount* )

The standard harmonic series begins with the fundamental frequency *f1* and each successive harmonic has a frequency equal to *f1* \* *n*, where *n* is the harmonic number for the harmonic. This is the natural harmonic series, one that occurs, e.g., in a vibrating string.

The documentation for this class was generated from the following files:

- CX\_ModularSynth.h
- CX\_ModularSynth.cpp

### 13.3 CX::Algo::BlockSampler< T > Class Template Reference

```
#include <CX_Algorithm.h>
```

#### Public Member Functions

- [BlockSampler](#) ([CX\\_RandomNumberGenerator](#) \*rng, const std::vector< T > &values)
- void [setup](#) ([CX\\_RandomNumberGenerator](#) \*rng, const std::vector< T > &values)
- T [getNextValue](#) (void)
- void [restartSampling](#) (void)
- unsigned int [getBlockNumber](#) (void) const
- unsigned int [getBlockPosition](#) (void) const

#### 13.3.1 Detailed Description

```
template<typename T>class CX::Algo::BlockSampler< T >
```

This class helps with the case where a set of V values must be sampled randomly with the constraint that each block of V samples should have each value in the set. For example, if you want to present a number of trials in four different conditions, where the conditions are intermixed, but you want to observe all four trial types every four trials, you could use this class.

```
#include "CX_EntryPoint.h"

void runExperiment(void) {
    //Construct a BlockSampler using RNG as the random number generator
    //and integer values 1 to 4 as the data to sample from.
    Algo::BlockSampler<int> bs(&RNG, Util::intVector(1, 4));

    //Generate 4 blocks of values and print those values along with information about the block and
    position
    cout << "Block, Position: Value" << endl;
    while (bs.getBlockNumber() < 4) {
        cout << bs.getBlockNumber() << ", " << bs.getBlockPosition() << ": ";
        cout << bs.getNextValue() << endl;
    }
}
```

#### Note

Another way of getting blocked random samples is to use [CX::CX\\_RandomNumberGenerator::sampleBlocks\(\)](#).

#### 13.3.2 Constructor & Destructor Documentation

**13.3.2.1** `template<typename T> CX::Algo::BlockSampler< T >::BlockSampler ( CX_RandomNumberGenerator *rng, const std::vector< T > & values ) [inline]`

Constructs a [BlockSampler](#) with the given settings. See [setup\(\)](#) for the meaning of the parameters.

#### 13.3.3 Member Function Documentation

**13.3.3.1** `template<typename T> unsigned int CX::Algo::BlockSampler< T >::getBlockNumber ( void ) const [inline]`

Returns the index of the block that is currently being sampled. Because it is zero-indexed, you can alternately think of the value as the number of completed blocks.

13.3.3.2 `template<typename T> unsigned int CX::Algo::BlockSampler< T >::getBlockPosition ( void ) const [inline]`

Returns the index of the sample that will be taken the next time [getNextValue\(\)](#) is called. If 0, it means that a block of samples was just finished. If within the current block 4 samples had already been taken, this will return 4

13.3.3.3 `template<typename T> T CX::Algo::BlockSampler< T >::getNextValue ( void ) [inline]`

Get the next value sampled from the provided data.

#### Returns

An element sampled from the provided values, or if there were no values provided, a warning will be logged and a default-constructed instance of T will be returned.

13.3.3.4 `template<typename T> void CX::Algo::BlockSampler< T >::restartSampling ( void ) [inline]`

Restarts sampling to be at the beginning of a block of samples. Also resets the block number (

13.3.3.5 `template<typename T> void CX::Algo::BlockSampler< T >::setup ( CX_RandomNumberGenerator * rng, const std::vector< T > & values ) [inline]`

Set up the [BlockSampler](#).

#### Parameters

<i>rng</i>	A pointer to a <a href="#">CX_RandomNumberGenerator</a> that will be used to randomize the sampled data.
<i>values</i>	A vector of values from which to sample.

The documentation for this class was generated from the following file:

- CX\_Algorithm.h

## 13.4 CX::Synth::Clamper Class Reference

```
#include <CX_ModularSynth.h>
```

Inherits [CX::Synth::ModuleBase](#).

#### Public Member Functions

- double [getNextSample](#) (void) override

#### Public Attributes

- [ModuleParameter](#) **low**
- [ModuleParameter](#) **high**

#### Additional Inherited Members

##### 13.4.1 Detailed Description

This class clamps inputs to be in the interval [low, high], where low and high are the members of this class.

### 13.4.2 Member Function Documentation

#### 13.4.2.1 `double Clamper::getNextSample ( void ) [override],[virtual]`

This function should be overloaded for any derived class that can be used as the input for another module.

Reimplemented from [CX::Synth::ModuleBase](#).

The documentation for this class was generated from the following files:

- CX\_ModularSynth.h
- CX\_ModularSynth.cpp

## 13.5 CX::CX\_SlidePresenter::Configuration Struct Reference

```
#include <CX_SlidePresenter.h>
```

### Public Attributes

- [CX\\_Display \\* display](#)  
*A pointer to the display to use.*
- `std::function< void(CX_SlidePresenter::FinalSlideFunctionArgs &)>` [finalSlideCallback](#)  
*A pointer to a user function that will be called as soon as the final slide is presented.*
- [CX\\_SlidePresenter::ErrorMode errorMode](#)  
*This sets how errors in slide presentation should be handled. Currently, the only available mode is the default, so this should not be changed.*
- `bool` [deallocateCompletedSlides](#)  
*If true, once a slide has been presented, its framebuffer will be deallocated to conserve memory.*
- [SwappingMode swappingMode](#)  
*The mode used for swapping slides. See the SwappingMode enum for the settings. Defaults to MULTI\_CORE.*
- [CX\\_Millis preSwapCPUHoggingDuration](#)  
*Only used if swappingMode is a single core mode. The amount of time, before a slide is swapped from the back buffer to the front buffer, that the CPU is put into a spinloop waiting for the buffers to swap.*
- `bool` [useFenceSync](#)  
*Hint that fence sync should be used to check that slides are fully copied to the back buffer before they are swapped in.*
- `bool` [waitUntilFenceSyncComplete](#)  
*If useFenceSync is false, this is also forced to false. If this is true, new slides will not be swapped in until there is confirmation that the slide has been fully copied into the back buffer. This prevents vertical tearing, but may cause slides to be swapped in late if the copy confirmation is delayed but the copy has actually occurred. Does nothing if swappingMode is MULTI\_CORE.*

### 13.5.1 Detailed Description

This struct is used for configuring a [CX\\_SlidePresenter](#).

The documentation for this struct was generated from the following file:

- CX\_SlidePresenter.h

## 13.6 CX::CX\_SoundStream::Configuration Struct Reference

```
#include <CX_SoundStream.h>
```

### Public Attributes

- int [inputChannels](#)  
*The number of input (e.g. microphone) channels to use. If 0, no input will be used.*
- int [outputChannels](#)  
*The number of output channels to use. Currently only stereo and mono are well-supported. If 0, no output will be used.*
- int [sampleRate](#)
- unsigned int [bufferSize](#)
- RtAudio::Api [api](#)
- RtAudio::StreamOptions [streamOptions](#)
- int [inputDeviceId](#)  
*The ID of the desired input device. A value of -1 will cause the system default input device to be used.*
- int [outputDeviceId](#)  
*The ID of the desired output device. A value of -1 will cause the system default output device to be used.*

### 13.6.1 Detailed Description

This struct controls the configuration of the [CX\\_SoundStream](#).

### 13.6.2 Member Data Documentation

#### 13.6.2.1 RtAudio::Api CX::CX\_SoundStream::Configuration::api

This argument depends on your operating system. Using RtAudio::Api::UNSPECIFIED will pick an available API for your system (if any; see the links below). The API means the type of software interface to use. For example, on Windows, you can choose from Windows Direct Sound (DS) and ASIO. ASIO is commonly used with audio recording equipment because it has lower latency whereas DS is more of a consumer-grade interface. The choice of API does not affect how you use this class, but it may affect the performance of sound playback.

See <http://www.music.mcgill.ca/~gary/rtaudio/classRtAudio.html#ac9b6f625da88249d08a8409a9db> for a listing of the APIs. See <http://www.music.mcgill.ca/~gary/rtaudio/classRtAudio.-html#afd0bfa26deae9804e18faff59d0273d9> for the default ordering of the APIs if RtAudio::Api::UNSPECIFIED is used.

#### 13.6.2.2 unsigned int CX::CX\_SoundStream::Configuration::bufferSize

The size of the audio data buffer to use, in sample frames. A larger buffer size means more latency but also a greater potential for audio glitches (clicks and pops). Buffer size is per channel (i.e. if there are two channels and buffer size is set to 256, the actual buffer size will be 512 samples).

#### 13.6.2.3 int CX::CX\_SoundStream::Configuration::sampleRate

The requested sample rate for the input and output channels. If, for the selected device(s), this sample cannot be used, the nearest greater sample rate will be chosen. If there is no greater sample rate, the next lower sample rate will be used.

### 13.6.2.4 RtAudio::StreamOptions CX::CX\_SoundStream::Configuration::streamOptions

See [http://www.music.mcgill.ca/~gary/rtaudio/structRtAudio\\_1\\_1StreamOptions.-html](http://www.music.mcgill.ca/~gary/rtaudio/structRtAudio_1_1StreamOptions.-html) for more information.

`flags` must not include `RTAUDIO_NONINTERLEAVED`: The audio data used by `CX` is interleaved.

The documentation for this struct was generated from the following file:

- `CX_SoundStream.h`

## 13.7 CX::CX\_BaseClockImplementation Class Reference

```
#include <CX_Clock.h>
```

Inherited by `CX::CX_StdClockWrapper< stdClock >`.

### Public Member Functions

- virtual long long **nanos** (void)=0
- virtual void **resetStartTime** (void)=0
- virtual std::string **getName** (void)

#### 13.7.1 Detailed Description

`CX_Clock` uses classes that are derived from this class for timing. See `CX::CX_Clock::setImplementation()`.

`nanos()` should return the current time in nanoseconds. If the implementation does not have nanosecond precision, it should still return time in nanoseconds, which might just involve a multiplication (e.g. clock ticks are in microseconds, so multiply by 1000 to make each value equal to a nanosecond).

It is assumed that the implementation has some way to subtract off a start time so that `nanos()` counts up from 0 and that `resetStartTime()` can reset the start time so that the clock counts up from 0 after `resetStartTime()` is called.

The documentation for this class was generated from the following file:

- `CX_Clock.h`

## 13.8 CX::Util::CX\_BaseUnitConverter Class Reference

```
#include <CX_UnitConversion.h>
```

Inherited by `CX::Util::CX_DegreeToPixelConverter`, and `CX::Util::CX_LengthToPixelConverter`.

### Public Member Functions

- virtual float **operator()** (float x)
- virtual float **inverse** (float y)

#### 13.8.1 Detailed Description

This class should be inherited from by any unit converters. You should override both `operator()` and `inverse()`. `inverse()` should perform the mathematical inverse of the operation performed by `operator()`.

### 13.8.2 Member Function Documentation

13.8.2.1 virtual float CX::Util::CX\_BaseUnitConverter::inverse ( float y ) [inline], [virtual]

`inverse()` should perform the inverse operation as `operator()`.

Reimplemented in [CX::Util::CX\\_LengthToPixelConverter](#), and [CX::Util::CX\\_DegreeToPixelConverter](#).

13.8.2.2 virtual float CX::Util::CX\_BaseUnitConverter::operator() ( float x ) [inline], [virtual]

`operator()` should perform the unit conversion.

Reimplemented in [CX::Util::CX\\_LengthToPixelConverter](#), and [CX::Util::CX\\_DegreeToPixelConverter](#).

The documentation for this class was generated from the following file:

- [CX\\_UnitConversion.h](#)

## 13.9 CX::CX\_Clock Class Reference

```
#include <CX_Clock.h>
```

### Public Member Functions

- void [setImplementation](#) ([CX\\_BaseClockImplementation](#) \*impl)
- void [precisionTest](#) (unsigned int iterations)
- [CX\\_Millis](#) now (void)
- void [sleep](#) ([CX\\_Millis](#) t)
- void [delay](#) ([CX\\_Millis](#) t)
- void [resetExperimentStartTime](#) (void)
- std::string [getExperimentStartDateTimeString](#) (std::string format="%Y-%b-%e %h-%M-%S %a")

### Static Public Member Functions

- static std::string [getDateTimeString](#) (std::string format="%Y-%b-%e %h-%M-%S %a")

### 13.9.1 Detailed Description

This class is responsible for getting timestamps for anything requiring timestamps. The way to get timing information is the function [now\(\)](#). It returns the current time relative to the start of the experiment in microseconds (on most systems, see [getTickPeriod\(\)](#) to check the actual precision).

An instance of this class is preinstantiated for you. See [CX::Instances::Clock](#).

### 13.9.2 Member Function Documentation

13.9.2.1 void CX\_Clock::delay ( CX\_Millis t )

This functions blocks for the requested period of time. This is likely more precise than [CX\\_Clock::sleep\(\)](#) because it does not give up control to the operating system, but it wastes resources because it just sits in a spinloop for the requested duration. This is functionally a static function.

**13.9.2.2** `std::string CX_Clock::getDateTimeString ( std::string format = "%Y-%b-%e %h-%M-%S %a" ) [static]`

This function returns a string containing the local time encoded according to some format.



## Parameters

<i>format</i>	See <a href="http://pocoproject.org/docs/Poco.DateTimeFormatter.html#4684">http://pocoproject.org/docs/Poco.DateTimeFormatter.html#4684</a> for documentation of the format. E.g. "%Y/%m/%d %H:%M:%S" gives "year/month/day 24Hour-Clock:minute:second" with some zero-padding for most things. The default "%Y-%b-%e %h-%M-%S %a" is "yearWithCentury-abbreviatedMonthName-nonZeroPaddedDay 12HourClock-minuteZeroPadded-secondZeroPadded am/pm".
---------------	--

### 13.9.2.3 std::string CX\_Clock::getExperimentStartDateTimeString ( std::string format = "%Y-%b-%e %h-%M-%S %a" )

Get a string representing the date/time of the start of the experiment encoded according to a format.

## Parameters

<i>format</i>	See <a href="#">getDateTimeString()</a> for the definition of the format.
---------------	---

### 13.9.2.4 CX\_Millis CX\_Clock::now ( void )

This function returns the current time relative to the start of the experiment in milliseconds. The start of the experiment is defined by default as when the [CX\\_Clock](#) instance named Clock (instantiated in this file) is constructed (typically the beginning of program execution).

## Returns

A CX\_Millis object containing the time.

## Note

This cannot be converted to current date/time in any meaningful way. Use [getDateTimeString\(\)](#) for that.

### 13.9.2.5 void CX\_Clock::precisionTest ( unsigned int iterations )

This function tests the precision of the clock used by [CX](#). The results are computer-specific. If the precision of the clock is worse than microsecond accuracy, a warning is logged including information about the actual precision of the clock.

Depending on the number of iterations, this function may be considered blocking. See [Blocking Code](#).

## Parameters

<i>iterations</i>	Number of time duration samples to take. More iterations should give a better estimate.
-------------------	---

### 13.9.2.6 void CX\_Clock::resetExperimentStartTime ( void )

If for some reason you have a long setup period before the experiment proper starts, you could call this function so that the values returned by [CX\\_Clock::now\(\)](#) will count up from 0 starting from when this function was called. This function also resets the experiment start date/time (see [getExperimentStartDateTimeString\(\)](#)).

### 13.9.2.7 void CX\_Clock::setImplementation ( CX::CX\_BaseClockImplementation \* impl )

Set the underlying clock implementation used by this instance of [CX\\_Clock](#). You would use this function if the default clock implementation used by [CX\\_Clock](#) has insufficient precision on your system. You can use CX::CX\_StdClock-Wrapper to wrap any of the clocks from the std::chrono namespace or any clock that conforms to the standard of those clocks.

## Parameters

<i>impl</i>	A pointer to an instance of a class derived from <a href="#">CX::CX_BaseClockImplementation</a> .
-------------	---

## 13.9.2.8 void CX\_Clock::sleep ( CX\_Millis t )

This functions sleeps for the requested period of time. This can be somewhat imprecise because it requests a specific sleep duration from the operating system, but the operating system may not provide the exact sleep time.

## Parameters

<i>t</i>	The requested sleep duration. If 0, the thread yields rather than sleeping.
----------	---

The documentation for this class was generated from the following files:

- CX\_Clock.h
- CX\_Clock.cpp

## 13.10 CX::Util::CX\_CoordinateConverter Class Reference

```
#include <CX_UnitConversion.h>
```

## Public Member Functions

- [CX\\_CoordinateConverter](#) (void)
- [CX\\_CoordinateConverter](#) (ofPoint origin, bool invertX, bool invertY, bool invertZ=false)
- void [setAxisInversion](#) (bool invertX, bool invertY, bool invertZ=false)
- void [setOrigin](#) (ofPoint newOrigin)
- void [setMultiplier](#) (float multiplier)
- void [setUnitConverter](#) ([CX\\_BaseUnitConverter](#) \*converter)
- ofPoint [operator\(\)](#) (ofPoint p)
- ofPoint [operator\(\)](#) (float x, float y, float z=0)
- ofPoint [inverse](#) (ofPoint p)
- ofPoint [inverse](#) (float x, float y, float z=0)

## 13.10.1 Detailed Description

This helper class is used for converting from a somewhat user-defined coordinate system into the standard computer monitor coordinate system. When user coordinates are input into this class, they will be converted into the standard monitor coordinate system. This lets you use coordinates in your own system and convert those coordinates into the standard coordinates that are used by the drawing functions of openFrameworks.

See [CX\\_CoordinateConverter::setUnitConverter\(\)](#) for a way to do change the units of the coordinate system to, for example, inches or degrees of visual angle.

Example use:

```
CX_CoordinateConverter conv(Display.  
    getCenterOfDisplay(), false, true); //Make the center of the display the origin and  
    invert  
//the Y-axis. This makes positive x values go to the right and positive y values go up from the center of  
    the display.  
ofSetColor(255, 0, 0); //Draw a red circle in the center of the display.  
ofCircle(conv(0, 0), 20);  
ofSetColor(0, 255, 0); //Draw a green circle 100 pixels to the right of the center.  
ofCircle(conv(100, 0), 20);  
ofSetColor(0, 0, 255); //Draw a blue circle 100 pixels above the center (inverted y-axis).  
ofCircle(conv(0, 100), 20);
```

Another example of the use of this class can be found in the `advancedChangeDetection` example experiment.

### 13.10.2 Constructor & Destructor Documentation

#### 13.10.2.1 CX::Util::CX\_CoordinateConverter::CX\_CoordinateConverter ( void )

Constructs a `CX_CoordinateConverter` with the default settings. The settings can be changed later with `setAxisInversion()`, `setOrigin()`, `setMultiplier()`, and/or `setUnitConverter()`.

#### 13.10.2.2 CX::Util::CX\_CoordinateConverter::CX\_CoordinateConverter ( ofPoint *origin*, bool *invertX*, bool *invertY*, bool *invertZ* = false )

Constructs a `CX_CoordinateConverter` with the given settings.

##### Parameters

<i>origin</i>	The location within the standard coordinate system at which the origin (the point at which the x, y, and z values are 0) of the user-defined coordinate system is located. If, for example, you want the center of the display to be the origin within your user-defined coordinate system, you could use <code>CX_Display::getCenterOfDisplay()</code> as the value for this argument.
<i>invertX</i>	Invert the x-axis from the default, which is that x increases to the right.
<i>invertY</i>	Invert the y-axis from the default, which is that y increases downward.
<i>invertZ</i>	Invert the z-axis from the default, which is that z increases toward the user (i.e. pointing out of the front of the screen). The other way of saying this is that smaller (increasingly negative) values are farther away.

### 13.10.3 Member Function Documentation

#### 13.10.3.1 ofPoint CX::Util::CX\_CoordinateConverter::inverse ( ofPoint *p* )

Performs the inverse of `operator()`, i.e. converts from standard coordinates to user coordinates.

##### Parameters

<i>p</i>	A point in standard coordinates.
----------	----------------------------------

##### Returns

A point in user coordinates.

#### 13.10.3.2 ofPoint CX::Util::CX\_CoordinateConverter::inverse ( float *x*, float *y*, float *z* = 0 )

Equivalent to `inverse (ofPoint (x, y, z))`;

#### 13.10.3.3 ofPoint CX::Util::CX\_CoordinateConverter::operator() ( ofPoint *p* )

The primary method of conversion between coordinate systems. You supply a point in user coordinates and get in return a point in standard coordinates.

Example use:

```
CX_CoordinateConverter cc(ofPoint(200,200), false, true);
ofPoint p(-50, 100); //P is in user-defined coordinates, 50 units left and 100 units above the origin.
ofPoint res = cc(p); //Use operator() to convert from the user system to the standard system.
//res should contain (150, 100) due to the inverted y axis.
```

**Parameters**

<i>p</i>	The point in user coordinates that should be converted to standard coordinates.
----------	---

**Returns**

The point in standard coordinates.

**13.10.3.4** `ofPoint CX::Util::CX_CoordinateConverter::operator() ( float x, float y, float z = 0 )`

Equivalent to a call to `operator() (ofPoint(x, y, z));`.

**13.10.3.5** `void CX::Util::CX_CoordinateConverter::setAxisInversion ( bool invertX, bool invertY, bool invertZ = false )`

Sets whether each axis within the user-defined system is inverted from the standard coordinate system.

**Parameters**

<i>invertX</i>	Invert the x-axis from the default, which is that x increases to the right.
<i>invertY</i>	Invert the y-axis from the default, which is that y increases downward.
<i>invertZ</i>	Invert the z-axis from the default, which is that z increases toward the viewer (i.e. pointing out of the front of the screen).

**13.10.3.6** `void CX::Util::CX_CoordinateConverter::setMultiplier ( float multiplier )`

This function sets the amount by which user coordinates are multiplied before they are converted to standard coordinates. This allows you to easily scale stimuli, assuming that the [CX\\_CoordinateConverter](#) is used throughout. If it has not been set, the multiplier is 1 by default.

**Parameters**

<i>multiplier</i>	The amount to multiply user coordinates by.
-------------------	---

**13.10.3.7** `void CX::Util::CX_CoordinateConverter::setOrigin ( ofPoint newOrigin )`

Sets the location within the standard coordinate system at which the origin of the user-defined coordinate system is located.

**Parameters**

<i>newOrigin</i>	The location within the standard coordinate system at which the origin (the point at which the x, y, and z values are 0) of the user-defined coordinate system is located. If, for example, you want the center of the display to be the origin within your user-defined coordinate system, you could use <a href="#">CX_Display::getCenterOfDisplay()</a> as the value for this argument.
------------------	--

**13.10.3.8** `void CX::Util::CX_CoordinateConverter::setUnitConverter ( CX_BaseUnitConverter * converter )`

Sets the unit converter that will be used when converting the coordinate system. In this way you can convert both the coordinate system in use and the units used by the coordinate system in one step. See [CX\\_DegreeToPixelConverter](#) and [CX\\_LengthToPixelConverter](#) for examples of the converters that can be used.

Example use:

```
//At global scope:
CX_CoordinateConverter conv(ofPoint(0,0), false, true); //The origin will be set to a
proper value later.
CX_DegreeToPixelConverter d2p(35, 70);
```

```
//During setup:
conv.setOrigin(Display.getCenterOfDisplay());
conv.setUnitConverter(&d2p); //Use degrees of visual angle as the units of the user coordinate system.

//Draw a blue circle 2 degrees of visual angle to the left of the origin and 3 degrees above (inverted
    y-axis) the origin.
ofSetColor(0, 0, 255);
ofCircle(conv(-2, 3), 20);
```

#### Parameters

<i>converter</i>	A pointer to an instance of a class that is a <a href="#">CX_BaseUnitConverter</a> or which has inherited from that class. See <a href="#">CX_UnitConversion.h/cpp</a> for the implementation of <a href="#">CX_LengthToPixelConverter</a> to see an example of how to create you own converter.
------------------	--

#### Note

The origin of the coordinate converter must be in the units that result from the unit conversion. E.g. if you are converting the units from degrees to pixels, the origin must be in pixels. See [setOrigin\(\)](#).

The unit converter passed to this function must continue to exist throughout the lifetime of the coordinate converter. It is not copied.

The documentation for this class was generated from the following files:

- [CX\\_UnitConversion.h](#)
- [CX\\_UnitConversion.cpp](#)

## 13.11 CX::CX\_DataFrame Class Reference

```
#include <CX_DataFrame.h>
```

#### Classes

- class [loOptions](#)

#### Public Types

- typedef std::vector  
    < [CX\\_DataFrameCell](#) >  
    ::size\_type **rowIndex\_t**

#### Public Member Functions

- [CX\\_DataFrame](#) & **operator=** (const [CX\\_DataFrame](#) &df)
- [CX\\_DataFrameCell](#) **operator()** (std::string column, rowIndex\_t row)
- [CX\\_DataFrameCell](#) **operator()** (rowIndex\_t row, std::string column)
- [CX\\_DataFrameCell](#) **at** (rowIndex\_t row, std::string column)
- [CX\\_DataFrameCell](#) **at** (std::string column, rowIndex\_t row)
- [CX\\_DataFrameColumn](#) **operator[]** (std::string column)
- [CX\\_DataFrameRow](#) **operator[]** (rowIndex\_t row)
- void **appendRow** ([CX\\_DataFrameRow](#) row)
- void **setRowCount** (rowIndex\_t rowCount)
- void **addColumn** (std::string columnName)

- void **append** (CX\_DataFrame df)
- std::string **print** (std::string delimiter="\t", bool printRowNumbers=true) const
- std::string **print** (const std::set< std::string > &columns, std::string delimiter="\t", bool printRowNumbers=true) const
- std::string **print** (const std::vector< rowIndex\_t > &rows, std::string delimiter="\t", bool printRowNumbers=true) const
- std::string **print** (const std::set< std::string > &columns, const std::vector< rowIndex\_t > &rows, std::string delimiter="\t", bool printRowNumbers=true) const
- std::string **print** (OutputOptions oOpt) const
- bool **printToFile** (std::string filename, std::string delimiter="\t", bool printRowNumbers=true) const
- bool **printToFile** (std::string filename, const std::set< std::string > &columns, std::string delimiter="\t", bool printRowNumbers=true) const
- bool **printToFile** (std::string filename, const std::vector< rowIndex\_t > &rows, std::string delimiter="\t", bool printRowNumbers=true) const
- bool **printToFile** (std::string filename, const std::set< std::string > &columns, const std::vector< rowIndex\_t > &rows, std::string delimiter="\t", bool printRowNumbers=true) const
- bool **printToFile** (std::string filename, OutputOptions oOpt) const
- bool **readFromFile** (std::string filename, InputOptions iOpt)
- bool **readFromFile** (std::string filename, std::string cellDelimiter="\t", std::string vectorEncloser="\"", std::string vectorElementDelimiter=";")
- void **clear** (void)
- bool **deleteColumn** (std::string columnName)
- bool **deleteRow** (rowIndex\_t row)
- std::vector< std::string > **getColumnNames** (void) const
- bool **columnExists** (std::string columnName) const
- *Returns true if the named column exists in the CX\_DataFrame.*
- bool **columnContainsVectors** (std::string columnName) const
- *Returns true if the named column contains any cells which contain vectors (i.e. have a length > 1).*
- rowIndex\_t **getRowCount** (void) const
- bool **reorderRows** (const vector< CX\_DataFrame::rowIndex\_t > &newOrder)
- CX\_DataFrame **copyRows** (vector< CX\_DataFrame::rowIndex\_t > rowOrder) const
- CX\_DataFrame **copyColumns** (vector< std::string > columns)
- void **shuffleRows** (void)
- void **shuffleRows** (CX\_RandomNumberGenerator &rng)
- template<typename T >  
std::vector< T > **copyColumn** (std::string column) const
- std::vector< std::string > **convertVectorColumnToColumns** (std::string columnName, int startIndex, bool deleteOriginal, std::string newBaseName="")
- void **convertAllVectorColumnsToMultipleColumns** (int startIndex, bool deleteOriginals)

#### Protected Member Functions

- void **\_resizeToFit** (std::string column, rowIndex\_t row)
- void **\_resizeToFit** (rowIndex\_t row)
- void **\_resizeToFit** (std::string column)
- void **\_equalizeRowLengths** (void)

## Protected Attributes

- `std::map< std::string, vector < CX_DataFrameCell > > _data`
- `rowIndex_t _rowCount`
- `IoOptions _ioOptions`

## Friends

- class **CX\_DataFrameRow**
- class **CX\_DataFrameColumn**

## 13.11.1 Detailed Description

This class provides an easy way to store data from an experiment and output that data to a file at the end of the experiment. A [CX\\_DataFrame](#) is a square two-dimensional array of cells, but each cell is capable of holding a vector of data. Each cell is indexed with a column name (a string) and a row number. Cells can store many different kinds of data and the data can be inserted or extracted easily. The standard method of storing data is to use `operator()`, which dynamically resizes the data frame. When an experimental session is complete, the data can be written to a file using [printToFile\(\)](#).

See the example `dataFrame.cpp` for thorough examples of how to use a [CX\\_DataFrame](#).

Several of the member functions of this class could be blocking if the amount of data in the data frame is large enough.

## 13.11.2 Member Function Documentation

13.11.2.1 `void CX_DataFrame::addColumn ( std::string columnName )`

Adds a column to the data frame.

## Parameters

<i>columnName</i>	The name of the column to add. If a column with that name already exists in the data frame, a warning will be logged.
-------------------	---

13.11.2.2 `void CX_DataFrame::append ( CX_DataFrame df )`

Appends a data frame to this data frame. [appendRow\(\)](#) is used to copy over the rows of `df`.

## Parameters

<i>df</i>	The <a href="#">CX_DataFrame</a> to append.
-----------	---

13.11.2.3 `void CX_DataFrame::appendRow ( CX_DataFrameRow row )`

Appends the row to the end of the data frame.

## Parameters

<i>row</i>	The row to add.
------------	-----------------

## Note

If the row has columns that do not exist in the data frame, those columns will be added to the data frame.

#### 13.11.2.4 `CX_DataFrameCell` `CX_DataFrame::at ( rowIndex_t row, std::string column )`

Access the cell at the given row and column with bounds checking. Throws a `std::out_of_range` exception and logs an error if either the row or column is out of bounds.



## Parameters

<i>row</i>	The row number.
<i>column</i>	The column name.

## Returns

A [CX\\_DataFrameCell](#) that can be read from or written to.

## 13.11.2.5 void CX\_DataFrame::clear ( void )

Deletes the contents of the data frame. Resizes the data frame to have no rows and no columns.

## 13.11.2.6 std::vector&lt; std::string &gt; CX\_DataFrame::convertVectorColumnToColumns ( std::string columnName, int startIndex, bool deleteOriginal, std::string newBaseName = " " )

Converts a column which contains vectors of data into multiple columns which are given names with an ascending integer suffix. Each new column will contain the data from one location in the previous vectors of data. For example, if you have length 3 vectors in a column and use this function on that column, you will end up with three columns, each of which contains one of the elements of those vectors, with order maintained, of course.

If you have vectors with different lengths within the same column, this function still works, it just fills empty cells of new columns with the string "NA".

## Parameters

<i>columnName</i>	The name of the column to convert to multiple columns. If the named column does not exist or it does not contain any vectors, this function has no effect.
<i>startIndex</i>	The value at which to start giving suffix indices. For example, if it is 1, the first new column will be named "newBaseName1", the second "newBaseName2", etc..
<i>deleteOriginal</i>	If <code>true</code> , the original column, <code>columnName</code> , will be deleted once the data has been copied into the new columns.
<i>newBaseName</i>	If this is the empty string, <code>columnName</code> will be used as the base for the new column names. Otherwise, <code>newBaseName</code> will be used.

## Returns

A vector of strings containing the new names. If an error occurred or nothing needed to be done, this vector will be of length 0.

## Note

If any of the names of the new columns conflicts with an existing column name, the new column will be created, but its name will be changed by appending "\_NEW". If this new name conflicts with an existing name, the process will be repeated until the new name does not conflict.

## 13.11.2.7 template&lt;typename T &gt; std::vector&lt; T &gt; CX::CX\_DataFrame::copyColumn ( std::string column ) const

Makes a copy of the data contained in the named column, converting it to the specified type (such a conversion must be possible).

## Template Parameters

<i>T</i>	The type of data to extract. Must not be <code>std::vector&lt;C&gt;</code> , where C is any type.
----------	---

## Parameters

<i>column</i>	The name of the column to copy data from.
---------------	---

## Returns

A vector containing the copied data.

13.11.2.8 `CX_DataFrame CX_DataFrame::copyColumns ( vector< std::string > columns )`

Copies the specified columns into a new data frame.

## Parameters

<i>columns</i>	A vector of column names to copy out. If a requested column is not found, a warning will be logged, but the function will otherwise complete successfully.
----------------	--

## Returns

A [CX\\_DataFrame](#) containing the specified columns.

## Note

This function may be [Blocking Code](#) if the amount of copied data is large.

13.11.2.9 `CX_DataFrame CX_DataFrame::copyRows ( vector< CX_DataFrame::rowIndex_t > rowOrder ) const`

Creates [CX\\_DataFrame](#) containing a copy of the rows specified in rowOrder. The new data frame is not linked to the existing data frame.

## Parameters

<i>rowOrder</i>	A vector of <code>CX_DataFrame::rowIndex_t</code> containing the rows from this data frame to be copied out. The indices in rowOrder may be in any order: They don't need to be ascending. Additionally, the same row to be copied may be specified multiple times.
-----------------	---

## Returns

A [CX\\_DataFrame](#) containing the rows specified in rowOrder.

## Note

This function may be [Blocking Code](#) if the amount of copied data is large.

13.11.2.10 `bool CX_DataFrame::deleteColumn ( std::string columnName )`

Deletes the given column of the data frame.

## Parameters

<i>columnName</i>	The name of the column to delete. If the column is not in the data frame, a warning will be logged.
-------------------	---

## Returns

True if the column was found and deleted, false if it was not found.

## 13.11.2.11 bool CX\_DataFrame::deleteRow ( rowIndex\_t row )

Deletes the given row of the data frame.

## Parameters

<i>row</i>	The row to delete (0 indexed). If row is greater than or equal to the number of rows in the data frame, a warning will be logged.
------------	---

## Returns

True if the row was in bounds and was deleted, false if the row was out of bounds.

## 13.11.2.12 std::vector&lt; std::string &gt; CX\_DataFrame::getColumnNames ( void ) const

Returns a vector containing the names of the columns in the data frame.

## Returns

Vector of strings with the column names.

## 13.11.2.13 CX\_DataFrame::rowIndex\_t CX\_DataFrame::getRowCount ( void ) const

Returns the number of rows in the data frame.

## 13.11.2.14 CX\_DataFrameCell CX\_DataFrame::operator() ( std::string column, rowIndex\_t row )

Access the cell at the given row and column. If the row or column is out of bounds, the data frame will be dynamically resized in order to fit the row or column.

## Parameters

<i>row</i>	The row number.
<i>column</i>	The column name.

## Returns

A [CX\\_DataFrameCell](#) that can be read from or written to.

## 13.11.2.15 CX\_DataFrame &amp; CX\_DataFrame::operator= ( const CX\_DataFrame &amp; df )

Copy the contents of another [CX\\_DataFrame](#) to this data frame. Because this is a copy operation, this may be [Blocking Code](#) if the copied data frame is large enough.

## Parameters

<i>df</i>	The data frame to copy.
-----------	-------------------------

## Returns

A reference to this data frame.

## Note

The contents of this data frame are deleted during the copy.

**13.11.2.16** `std::string CX_DataFrame::print ( std::string delimiter = "\t", bool printRowNumbers = true ) const`

Reduced argument version of [print\(\)](#). Prints all rows and columns.

**13.11.2.17** `std::string CX_DataFrame::print ( const std::set< std::string > & columns, std::string delimiter = "\t", bool printRowNumbers = true ) const`

Reduced argument version of [print\(\)](#). Prints all rows and the selected columns.

**13.11.2.18** `std::string CX_DataFrame::print ( const std::vector< rowIndex_t > & rows, std::string delimiter = "\t", bool printRowNumbers = true ) const`

Reduced argument version of [print\(\)](#). Prints all columns and the selected rows.

**13.11.2.19** `std::string CX_DataFrame::print ( const std::set< std::string > & columns, const std::vector< rowIndex_t > & rows, std::string delimiter = "\t", bool printRowNumbers = true ) const`

Prints the selected rows and columns of the data frame to a string. Each cell of the data frame will be separated with the selected delimiter. Each row of the data frame will be ended with a new line (whatever `std::endl` evaluates to, typically `"\n"`).

## Parameters

<i>columns</i>	Columns to print. Column names not found in the data frame will be ignored with a warning.
<i>rows</i>	Rows to print. Row indices not found in the data frame will be ignored with a warning.
<i>delimiter</i>	Delimiter to be used between cells of the data frame. Using comma or semicolon for the delimiter is not recommended because semicolons are used as element delimiters in the string-encoded vectors stored in the data frame and commas are used for element delimiters within each element of the string-encoded vectors.
<i>printRowNumbers</i>	If true, a column will be printed with the header "rowNumber" with the contents of the column being the selected row indices. If false, no row numbers will be printed.

## Returns

A string containing the printed version of the data frame.

## Note

This function may be [Blocking Code](#) if the data frame is large enough.

**13.11.2.20** `bool CX_DataFrame::printToFile ( std::string filename, std::string delimiter = "\t", bool printRowNumbers = true ) const`

Reduced argument version of [printToFile\(\)](#). Prints all rows and columns.

13.11.2.21 `bool CX_DataFrame::printToFile ( std::string filename, const std::set< std::string > & columns, std::string delimiter = "\t", bool printRowNumbers = true ) const`

Reduced argument version of [printToFile\(\)](#). Prints all rows and the selected columns.

13.11.2.22 `bool CX_DataFrame::printToFile ( std::string filename, const std::vector< rowIndex_t > & rows, std::string delimiter = "\t", bool printRowNumbers = true ) const`

Reduced argument version of [printToFile\(\)](#). Prints all columns and the selected rows.

13.11.2.23 `bool CX_DataFrame::printToFile ( std::string filename, const std::set< std::string > & columns, const std::vector< rowIndex_t > & rows, std::string delimiter = "\t", bool printRowNumbers = true ) const`

This function is equivalent in behavior to [print\(\)](#) except that instead of returning a string containing the printed contents of the data frame, the string is printed to a file. If the file exists, it will be overwritten. All parameters expect for filename behave in the same way as in [print\(\)](#).

#### Parameters

<i>filename</i>	Name of the file to print to. If it is an absolute path, the file will be put there. If it is a local path, the file will be placed relative to the data directory of the project.
-----------------	--

#### Returns

True for success, false if there was some problem writing to the file (insufficient permissions, etc.)

13.11.2.24 `bool CX_DataFrame::readFromFile ( std::string filename, InputOptions iOpt )`

Equivalent to a call to `readFromFile(string, string, string, string)`, except that the last three arguments are taken from `iOpt`.

#### Parameters

<i>filename</i>	The name of the file to read data from. If it is a relative path, the file will be read relative to the data directory.
<i>iOpt</i>	Input options, such as the delimiter between cells in the input file.

13.11.2.25 `bool CX_DataFrame::readFromFile ( std::string filename, std::string cellDelimiter = "\t", std::string vectorEncloser = "\"", std::string vectorElementDelimiter = ";" )`

Reads data from the given file into the data frame. This function assumes that there will be a row of column names as the first row of the file. It does not treat consecutive delimiters as a single delimiter.

#### Parameters

<i>filename</i>	The name of the file to read data from. If it is a relative path, the file will be read relative to the data directory.
<i>cellDelimiter</i>	A string containing the delimiter between cells of data in the input file.
<i>vectorEncloser</i>	A string containing the character(s) that surround cells that contain a vector of data in the input file. By default, vectors are enclosed in double quotes (""). This indicates to most software that it should treat the contents of the quotes "as-is", i.e. if it finds a delimiter within the quotes, it should not split there, but wait until out of the quotes. If <code>vectorEncloser</code> is the empty string, this function will not attempt to read in vectors: everything that looks like a vector will just be treated as a string.

**Returns**

False if an error occurred, true otherwise.

**Note**

The contents of the data frame will be deleted before attempting to read in the file.  
 If the data is read in from a file written with a row numbers column, that column will be read into the data frame.  
 You can remove it using `deleteColumn("rowNumber")`.  
 This function may be [Blocking Code](#) if the read in data frame is large enough.

**13.11.2.26** `bool CX_DataFrame::reorderRows ( const vector< CX_DataFrame::rowIndex_t > & newOrder )`

Re-orders the rows in the data frame.

**Parameters**

<i>newOrder</i>	Vector of row indices. <code>newOrder.size()</code> must equal this-> <code>getRowCount()</code> . <code>newOrder</code> must not contain any out-of-range indices (i.e. they must be < <code>getRowCount()</code> ). Both of these error conditions are checked for in the function call and errors are logged.
-----------------	--

**Returns**

true if all of the conditions of `newOrder` are met, false otherwise.

**13.11.2.27** `void CX_DataFrame::setRowCount ( rowIndex_t rowCount )`

Sets the number of rows in the data frame.

**Parameters**

<i>rowCount</i>	The new number of rows in the data frame.
-----------------	---

**Note**

If the row count is less than the number of rows already in the data frame, it will delete those rows with a warning.

**13.11.2.28** `void CX_DataFrame::shuffleRows ( void )`

Randomly re-orders the rows of the data frame using [CX::Instances::RNG](#) as the random number generator for the shuffling.

**Note**

This function may be [Blocking Code](#) if the data frame is large.

**13.11.2.29** `void CX_DataFrame::shuffleRows ( CX_RandomNumberGenerator & rng )`

Randomly re-orders the rows of the data frame.

**Parameters**

<i>rng</i>	Reference to a <a href="#">CX_RandomNumberGenerator</a> to be used for the shuffling.
------------	---

**Note**

This function may be [Blocking Code](#) if the data frame is large.

The documentation for this class was generated from the following files:

- CX\_DataFrame.h
- CX\_DataFrame.cpp

**13.12 CX::CX\_DataFrameCell Class Reference**

```
#include <CX_DataFrameCell.h>
```

**Public Member Functions**

- [CX\\_DataFrameCell](#) (const char \*c)  
Construct the cell, assigning the value to it.
- template<typename T >  
[CX\\_DataFrameCell](#) (const T &value)  
Construct the cell, assigning the values to it.
- template<typename T >  
[CX\\_DataFrameCell](#) (const std::vector< T > &values)  
Construct the cell, assigning the values to it.
- [CX\\_DataFrameCell](#) & [operator=](#) (const char \*c)
- template<typename T >  
[CX\\_DataFrameCell](#) & [operator=](#) (const T &value)  
Assigns a value to the cell.
- template<typename T >  
[CX\\_DataFrameCell](#) & [operator=](#) (const std::vector< T > &values)  
Assigns a vector of values to the cell.
- template<typename T >  
[operator T](#) (void) const  
Attempts to convert the contents of the cell to T using [to\(\)](#).
- template<typename T >  
[operator std::vector< T >](#) (void) const  
Attempts to convert the contents of the cell to vector<T> using [toVector<T>\(\)](#).
- template<typename T >  
void [store](#) (const T &value)
- template<typename T >  
T [to](#) (void) const
- std::string [toString](#) (void) const
- bool [toBool](#) (void) const  
Returns a copy of the stored data converted to bool. Equivalent to [to<bool>\(\)](#).
- int [toInt](#) (void) const  
Returns a copy of the stored data converted to int. Equivalent to [to<int>\(\)](#).
- double [toDouble](#) (void) const  
Returns a copy of the stored data converted to double. Equivalent to [to<double>\(\)](#).

- `template<typename T >`  
`std::vector< T > toVector (void) const`
- `template<typename T >`  
`void storeVector (std::vector< T > values)`
- `void copyCellTo (CX_DataFrameCell *targetCell) const`
- `std::string getStoredType (void) const`
- `void deleteStoredType (void)`
- `bool isVector (void) const`
- `template<>`  
`std::string to (void) const`
- `template<>`  
`std::vector< std::string > toVector (void) const`

### 13.12.1 Detailed Description

This class manages the contents of a single cell in a [CX\\_DataFrame](#). It handles all of the type conversion nonsense that goes on when data is inserted into or extracted from a data frame. It tracks the type of the data that is inserted or extracted and logs warnings if the inserted type does not match the extracted type, with a few exceptions (see notes).

#### Note

There are a few exceptions to the type tracking. If the inserted type is `const char*`, it is treated as a string. Additionally, you can extract anything as string without a warning. This is because the data is stored as a string internally so extracting the data as a string is a lossless operation.

### 13.12.2 Constructor & Destructor Documentation

#### 13.12.2.1 CX\_DataFrameCell::CX\_DataFrameCell ( const char \* c )

Constructs the cell with a string literal, treating it as a `std::string`.

### 13.12.3 Member Function Documentation

#### 13.12.3.1 void CX\_DataFrameCell::copyCellTo ( CX\_DataFrameCell \* targetCell ) const

Copies the contents of this cell to `targetCell`, including type information.

#### Parameters

<i>targetCell</i>	A pointer to the cell to copy data to.
-------------------	--

#### 13.12.3.2 std::string CX\_DataFrameCell::getStoredType ( void ) const

Gets a string representing the type of data stored within the cell. This string is implementation-defined (which is the C++ standards committee way of saying "It can be anything at all"). It is only guaranteed to be the same for the same type, but not necessarily be different for different types.

#### Returns

A string containing the name of the stored type as given by `typeid(typename).name()`.



**13.12.3.3 bool CX\_DataFrameCell::isVector ( void ) const**

Returns `true` if more than one element is stored in the [CX\\_DataFrameCell](#).

**13.12.3.4 CX\_DataFrameCell & CX\_DataFrameCell::operator= ( const char \* c )**

Assigns a string literal to the cell, treating it as a `std::string`.

**13.12.3.5 template<typename T > void CX::CX\_DataFrameCell::store ( const T & value )**

Stores the given value with the given type. This function is a good way to explicitly state the type of the data you are storing into the cell if, for example, it is a literal.

**Template Parameters**

<code>&lt;T&gt;</code>	The type to store the value as. If T is not specified, this function is essentially equivalent to using <code>operator=</code> .
------------------------	--

**Parameters**

<code>value</code>	The value to store.
--------------------	---------------------

**13.12.3.6 template<typename T > void CX::CX\_DataFrameCell::storeVector ( std::vector< T > values )**

Stores a vector of data in the cell. The data is stored as a string with each element delimited by a semicolon. If the data to be stored are strings containing semicolons, the data will not be extracted properly.

**Parameters**

<code>values</code>	A vector of values to store.
---------------------	------------------------------

**13.12.3.7 template<typename T > T CX::CX\_DataFrameCell::to ( void ) const**

Attempts to convert the contents of the cell to type T. There are a variety of reasons why this conversion can fail and they all center on the user inserting data of one type and then attempting to extract data of a different type. Regardless of whether the conversion is possible, if you try to extract a type that is different from the type that is stored in the cell, a warning will be logged.

**Template Parameters**

<code>&lt;T&gt;</code>	The type to convert to.
------------------------	-------------------------

**Returns**

The data in the cell converted to T.

**13.12.3.8 std::string CX::CX\_DataFrameCell::to ( void ) const**

Equivalent to a call to [toString\(\)](#). This is specialized because it skips the type checks of `to<T>`.

**Returns**

A copy of the stored data encoded as a string.

**13.12.3.9 std::string CX\_DataFrameCell::toString ( void ) const**

Equivalent to a call to [to<string>\(\)](#).

### 13.12.3.10 `template<typename T> std::vector< T> CX::CX_DataFrameCell::toVector ( void ) const`

Returns a copy of the contents of the cell converted to a vector of the given type. If the type of data stored in the cell was not a vector of the given type or the type does match but it was a scalar that is stored, the logs a warning but attempts the conversion anyway.

#### Template Parameters

<code>&lt;T&gt;</code>	The type of the elements of the returned vector.
------------------------	--

#### Returns

A vector containing the converted data.

### 13.12.3.11 `std::vector< std::string> CX::CX_DataFrameCell::toVector ( void ) const`

Converts the contents of the [CX\\_DataFrame](#) cell to a vector of strings.

The documentation for this class was generated from the following files:

- [CX\\_DataFrameCell.h](#)
- [CX\\_DataFrameCell.cpp](#)

## 13.13 CX::CX\_DataFrameColumn Class Reference

### Public Member Functions

- [CX\\_DataFrameCell operator\[\]](#) ([CX\\_DataFrame::rowIndex\\_t](#) row)
- [CX\\_DataFrame::rowIndex\\_t size](#) (void)

### Friends

- class **CX\_DataFrame**

### 13.13.1 Detailed Description

### 13.13.2 Member Function Documentation

#### 13.13.2.1 `CX_DataFrameCell CX_DataFrameColumn::operator[] ( CX_DataFrame::rowIndex_t row )`

Accesses the element in the specified row of the column.

#### 13.13.2.2 `CX_DataFrame::rowIndex_t CX_DataFrameColumn::size ( void )`

Get the number of rows in the column.

The documentation for this class was generated from the following files:

- [CX\\_DataFrame.h](#)
- [CX\\_DataFrame.cpp](#)

## 13.14 CX::CX\_DataFrameRow Class Reference

### Public Member Functions

- [CX\\_DataFrameCell](#) **operator[]** (std::string column)
- vector< std::string > **names** (void)
- void **clear** (void)

### Friends

- class **CX\_DataFrame**

#### 13.14.1 Detailed Description

The documentation for this class was generated from the following files:

- CX\_DataFrame.h
- CX\_DataFrame.cpp

## 13.15 CX::Util::CX\_DegreeToPixelConverter Class Reference

#include <CX\_UnitConversion.h>

Inherits [CX::Util::CX\\_BaseUnitConverter](#).

### Public Member Functions

- [CX\\_DegreeToPixelConverter](#) (float pixelsPerUnit, float viewingDistance, bool roundResult=false)
- void **setup** (float pixelsPerUnit, float viewingDistance, bool roundResult=false)
- float **operator()** (float degrees) override
- float **inverse** (float pixels) override
- bool **configureFromFile** (std::string filename, std::string delimiter="=", bool trimWhitespace=true, std::string commentString="//")

#### 13.15.1 Detailed Description

This simple utility class is used for converting degrees of visual angle to pixels on a monitor. This class uses [CX::Util::degreesToPixels\(\)](#) internally. See also [CX::Util::CX\\_CoordinateConverter](#) for a way to also convert from one coordinate system to another.

Example use:

```
CX_DegreeToPixelConverter d2p(34, 60); //34 pixels per unit length (e.g. cm) on the target monitor, user is
    60 length units from monitor.
ofLine( 200, 100, 200 + d2p(1), 100 + d2p(2) ); //Draw a line from (200, 100) (in pixel coordinates) to 1
    degree
//to the right and 2 degrees below that point.
```

### 13.15.2 Constructor & Destructor Documentation

#### 13.15.2.1 `CX::Util::CX_DegreeToPixelConverter::CX_DegreeToPixelConverter ( float pixelsPerUnit, float viewingDistance, bool roundResult = false )`

Constructs an instance of a [CX\\_DegreeToPixelConverter](#) with the given configuration. See [setup\(\)](#) for the meaning of the parameters.

### 13.15.3 Member Function Documentation

#### 13.15.3.1 `bool CX::Util::CX_DegreeToPixelConverter::configureFromFile ( std::string filename, std::string delimiter = "=", bool trimWhitespace = true, std::string commentString = " / " )`

This function exists to serve a per-computer configuration function that is otherwise difficult to provide due to the fact that C++ programs are compiled to binaries and cannot be easily edited on the computer on which they are running. This function takes the file name of a specially constructed configuration file and reads the key-value pairs in that file in order to configure the [CX\\_DegreeToPixelConverter](#). The format of the file is provided in the example code below.

Sample configuration file:

```
D2PC.pixelsPerUnit = 35
D2PC.viewingDistance = 50
D2PC.roundResult = true
```

All of the configuration keys are used in this example. Note that the "D2PC" prefix allows this configuration to be embedded in a file that also performs other configuration functions.

See [CX\\_DegreeToPixelConverter::setup\(\)](#) for details about the meanings of the configuration options.

Because this function uses [CX::Util::readKeyValueFile\(\)](#) internally, it has the same arguments.

#### Parameters

<i>filename</i>	The name of the file containing configuration data.
<i>delimiter</i>	The string that separates the key from the value. In the example, it is "=", but can be other values.
<i>trimWhitespace</i>	If true, whitespace characters surrounding both the key and value will be removed. This is a good idea to do.
<i>commentString</i>	If <i>commentString</i> is not the empty string (""), everything on a line following the first instance of <i>commentString</i> will be ignored.

#### Returns

`true` if there were no problems reading in the file, `false` otherwise.

#### 13.15.3.2 `float CX::Util::CX_DegreeToPixelConverter::inverse ( float pixels ) [override], [virtual]`

Performs the inverse of the operation performed by [operator\(\)](#), i.e. converts pixels to degrees.

#### Parameters

<i>pixels</i>	The number of pixels to convert to degrees.
---------------	---

#### Returns

The number of degrees of visual angle subtended by the given number of pixels.

Reimplemented from [CX::Util::CX\\_BaseUnitConverter](#).

13.15.3.3 float CX::Util::CX\_DegreeToPixelConverter::operator() ( float *degrees* ) [override],[virtual]

Converts the degrees to pixels based on the settings given during construction.

## Parameters

<i>degrees</i>	The number of degrees of visual angle to convert to pixels.
----------------	---

## Returns

The number of pixels corresponding to the number of degrees of visual angle.

Reimplemented from [CX::Util::CX\\_BaseUnitConverter](#).

13.15.3.4 `void CX::Util::CX_DegreeToPixelConverter::setup ( float pixelsPerUnit, float viewingDistance, bool roundResult = false )`

Sets up a [CX\\_DegreeToPixelConverter](#) with the given configuration.

## Parameters

<i>pixelsPerUnit</i>	The number of pixels within one length unit (e.g. inches, centimeters). This can be measured by drawing an object with a known size on the screen and measuring the length of a side and dividing the number of pixels by the total length measured.
<i>viewingDistance</i>	The distance from the monitor that the participant will be viewing the screen from.
<i>roundResult</i>	If true, the result of conversions will be rounded to the nearest integer (i.e. pixel). For drawing certain kinds of stimuli (especially text) it can be helpful to draw on pixel boundaries.

The documentation for this class was generated from the following files:

- [CX\\_UnitConversion.h](#)
- [CX\\_UnitConversion.cpp](#)

## 13.16 CX::CX\_Display Class Reference

```
#include <CX_Display.h>
```

## Public Member Functions

- void [setup](#) (void)
- void [configureFromFile](#) (std::string filename, std::string delimiter="=", bool trimWhitespace=true, std::string commentString="//")
- void [setFullScreen](#) (bool fullScreen)
- bool [isFullscreen](#) (void)  
*Returns true if the display is in full screen mode, false otherwise.*
- void [useHardwareVSync](#) (bool b)
- void [useSoftwareVSync](#) (bool b)
- void [beginDrawingToBackBuffer](#) (void)
- void [endDrawingToBackBuffer](#) (void)
- void [swapBuffers](#) (void)
- void [swapBuffersInThread](#) (void)
- void [setAutomaticSwapping](#) (bool autoSwap)
- bool [isAutomaticallySwapping](#) (void)
- bool [hasSwappedSinceLastCheck](#) (void)
- [CX\\_Millis](#) [getLastSwapTime](#) (void)
- [CX\\_Millis](#) [estimateNextSwapTime](#) (void)
- uint64\_t [getFrameNumber](#) (void)

- void [estimateFramePeriod](#) (CX\_Millis estimationInterval)
- CX\_Millis [getFramePeriod](#) (void)
- CX\_Millis [getFramePeriodStandardDeviation](#) (void)
- void [setWindowResolution](#) (int width, int height)
- void [setWindowTitle](#) (std::string title)
- ofRectangle [getResolution](#) (void)
- ofPoint [getCenterOfDisplay](#) (void)
- void [waitForOpenGL](#) (void)
- CX\_DataFrame [testBufferSwapping](#) (CX\_Millis desiredTestDuration, bool testSecondaryThread)
- void [copyFboToBackBuffer](#) (ofFbo &fbo)
- void [copyFboToBackBuffer](#) (ofFbo &fbo, ofPoint destination)
- void [copyFboToBackBuffer](#) (ofFbo &fbo, ofRectangle source, ofPoint destination)

### 13.16.1 Detailed Description

This class represents an abstract visual display surface, which is my way of saying that it doesn't necessarily represent a monitor. The display surface can either be a window or, if full screen, the whole monitor. It is also a bit abstract in that it does not draw anything, but only creates an context in which things can be drawn.

### 13.16.2 Member Function Documentation

#### 13.16.2.1 void CX\_Display::beginDrawingToBackBuffer ( void )

Prepares a rendering context for using drawing functions. Must be paired with a call to [endDrawingToBackBuffer\(\)](#).

```
Display.beginDrawingToBackBuffer();
//Draw stuff...
Display.endDrawingToBackBuffer();
```

#### 13.16.2.2 void CX\_Display::configureFromFile ( std::string filename, std::string delimiter = "=", bool trimWhitespace = true, std::string commentString = " //" )

This function exists to serve a per-computer configuration function that is otherwise difficult to provide due to the fact that C++ programs are compiled to binaries and cannot be easily edited on the computer on which they are running. This function takes the file name of a specially constructed configuration file and reads the key-value pairs in that file in order to configure the [CX\\_Display](#). The format of the file is provided in the example code below.

Sample configuration file:

```
display.windowWidth = 600
display.windowHeight = 300
display.windowTitle = My Neat Name
display.fullscreen = false
display.hardwareVSync = true
//display.softwareVSync = false //Commented out: no change
//display.swapAutomatically = false //Commented out: no change
```

All of the configuration keys are used in this example. Configuration options can be omitted, in which case there is no change in the configuration of the [CX\\_Display](#) for that option. Note that the "display" prefix allows this configuration to be embedded in a file that also performs other configuration functions.

Because this function uses [CX::Util::readKeyValueFile\(\)](#) internally, it has the same arguments.

## Parameters

<i>filename</i>	The name of the file containing configuration data.
<i>delimiter</i>	The string that separates the key from the value. In the example, it is "=", but can be other values.
<i>trimWhitespace</i>	If true, whitespace characters surrounding both the key and value will be removed. This is a good idea to do.
<i>commentString</i>	If <code>commentString</code> is not the empty string (""), everything on a line following the first instance of <code>commentString</code> will be ignored.

## 13.16.2.3 void CX\_Display::copyFboToBackBuffer ( ofFbo &amp; fbo )

Copies an `ofFbo` to the back buffer using a potentially very slow but pixel-perfect blitting operation. The slowness of the operation is hardware-dependent, with older hardware often being faster at this operation. Generally, you should just draw the `ofFbo` directly using its `draw()` function.

## Note

This function overwrites the contents of the back buffer, it does not draw over them. For this reason, transparency is ignored.

## Parameters

<i>fbo</i>	The framebuffer to copy. It will be drawn starting from (0, 0) and will be drawn at the full dimensions of the fbo (whatever size was chosen at allocation of the fbo).
------------	---

## 13.16.2.4 void CX\_Display::copyFboToBackBuffer ( ofFbo &amp; fbo, ofPoint destination )

Copies an `ofFbo` to the back buffer using a potentially very slow but pixel-perfect blitting operation. The slowness of the operation is hardware-dependent, with older hardware often being faster at this operation. Generally, you should just draw the `ofFbo` directly using its `draw()` function.

## Note

This function overwrites the contents of the back buffer, it does not draw over them. For this reason, transparency is ignored.

## Parameters

<i>fbo</i>	The framebuffer to copy.
<i>destination</i>	The point on the back buffer where the fbo will be placed.

## 13.16.2.5 void CX\_Display::copyFboToBackBuffer ( ofFbo &amp; fbo, ofRectangle source, ofPoint destination )

Copies an `ofFbo` to the back buffer using a potentially very slow but pixel-perfect blitting operation. The slowness of the operation is hardware-dependent, with older hardware often being faster at this operation. Generally, you should just draw the `ofFbo` directly using its `draw()` function.

## Note

This function overwrites the contents of the back buffer, it does not draw over them. For this reason, transparency is ignored.



## Parameters

<i>fbo</i>	The framebuffer to copy.
<i>source</i>	A rectangle giving an area of the fbo to copy.
<i>destination</i>	The point on the back buffer where the area of the fbo will be placed.

If this function does not provide enough flexibility, you can always draw ofFbo's using the following technique, which allows for transparency:

```
Display.beginDrawingToBackBuffer();
ofSetColor( 255 ); //If the color is not set to white, the fbo will be drawn mixed with whatever the
                  //current color is.
fbo.draw( x, y, width, height ); //Replace these variables with the destination location (x,y) and
                                 //dimensions of the FBO.
Display.endDrawingToBackBuffer();
```

#### 13.16.2.6 void CX\_Display::endDrawingToBackBuffer ( void )

Finish rendering to the back buffer. Must be paired with a call to [beginDrawingToBackBuffer\(\)](#).

#### 13.16.2.7 void CX\_Display::estimateFramePeriod ( CX\_Millis estimationInterval )

This function estimates the typical period of the display refresh. This function blocks for estimationInterval while the swapping thread swaps in the background (see [Blocking Code](#)). This function is called with an argument of 300 ms during construction of this class, so there will always be some information about the frame period. If more precision of the estimate is desired, this function can be called again with a longer wait duration.

## Parameters

<i>estimationInterval</i>	The length of time to spend estimating the frame period.
---------------------------	--

#### 13.16.2.8 CX\_Millis CX\_Display::estimateNextSwapTime ( void )

Get an estimate of the next time the front and back buffers will be swapped. This function depends on the precision of the frame period as estimated using [estimateFramePeriod\(\)](#). If the front and back buffers are not swapped every frame, the result of this function is meaningless because it uses the last buffer swap time as a reference.

## Returns

A time value that can be compared to CX::Instances::Clock.now().

#### 13.16.2.9 ofPoint CX\_Display::getCenterOfDisplay ( void )

Returns an ofPoint representing the center of the display. Works in either windowed or full screen mode.

#### 13.16.2.10 uint64\_t CX\_Display::getFrameNumber ( void )

This function returns the number of the last frame presented, as determined by number of front and back buffer swaps. It tracks buffer swaps that result from 1) the front and back buffer swapping automatically (as a result of [setAutomaticSwapping\(true\)](#)) and 2) manual swaps resulting from a call to [swapBuffers\(\)](#) or [swapBuffersInThread\(\)](#).

## Returns

The number of the last frame. This value can only be compared with other values returned by this function.

#### 13.16.2.11 CX\_Millis CX\_Display::getFramePeriod ( void )

Gets the estimate of the frame period calculated with [estimateFramePeriod\(\)](#).

**13.16.2.12 CX\_Millis CX\_Display::getFramePeriodStandardDeviation ( void )**

Gets the estimate of the standard deviation of the frame period calculated with [estimateFramePeriod\(\)](#).

**13.16.2.13 CX\_Millis CX\_Display::getLastSwapTime ( void )**

Get the last time at which the front and back buffers were swapped.

**Returns**

A time value that can be compared with `CX::Instances::Clock.now()`.

**13.16.2.14 ofRectangle CX\_Display::getResolution ( void )**

Returns the resolution of the current display area. If in windowed mode, this will return the resolution of the window. If in full screen mode, this will return the resolution of the monitor.

**Returns**

An `ofRectangle` containing the resolution. The width in pixels is stored in both the `width` and `x` members and the height in pixels is stored in both the `height` and `y` members, so you can use whichever makes the most sense to you.

**13.16.2.15 bool CX\_Display::hasSwappedSinceLastCheck ( void )**

Check to see if the display has swapped the front and back buffers since the last call to this function. This is generally used in conjunction with automatic swapping of the buffers ([setAutomaticSwapping\(\)](#)) or with an individual threaded swap of the buffers ([swapBuffersInThread\(\)](#)). This technically works with [swapBuffers\(\)](#), but given that that function only returns once the buffers have swapped, using this function to check that the buffers have swapped is redundant.

**Returns**

True if a swap has been made since the last call to this function, false otherwise.

**13.16.2.16 bool CX\_Display::isAutomaticallySwapping ( void )**

Determine whether the display is configured to automatically swap the front and back buffers every frame. See [setAutomaticSwapping](#) for more information.

**13.16.2.17 void CX\_Display::setAutomaticSwapping ( bool autoSwap )**

Set whether the front and buffers of the display will swap automatically every frame or not. You can check to see if a swap has occurred by calling [hasSwappedSinceLastCheck\(\)](#). You can check to see if the display is automatically swapping by calling [isAutomaticallySwapping\(\)](#).

**Parameters**

<i>autoSwap</i>	If true, the front and back buffer will swap automatically every frame.
-----------------	---

**Note**

This function may [block](#) for up to 1 frame to due the requirement that it synchronize with the thread.

**13.16.2.18 void CX\_Display::setFullScreen ( bool fullScreen )**

Set whether the display is full screen or not. If the display is set to full screen, the resolution may not be the same as the resolution of display in windowed mode, and vice versa.

**13.16.2.19 void CX\_Display::setup ( void )**

Set up the display. Must be called for the display to function correctly.

**13.16.2.20 void CX\_Display::setWindowResolution ( int *width*, int *height* )**

Sets the resolution of the window. Has no effect if called while in full screen mode.

**Parameters**

<i>width</i>	The desired width of the window, in pixels.
<i>height</i>	The desired height of the window, in pixels.

**13.16.2.21 void CX\_Display::setWindowTitle ( std::string *title* )**

Sets the title of the experiment window.

**Parameters**

<i>title</i>	The new window title.
--------------	-----------------------

**13.16.2.22 void CX\_Display::swapBuffers ( void )**

This function queues up a swap of the front and back buffers then blocks until the swap occurs. This usually should not be used if `isAutomaticallySwapping() == true`. If it is, a warning will be logged.

**See Also**

[Blocking Code](#)

**13.16.2.23 void CX\_Display::swapBuffersInThread ( void )**

This function cues a swap of the front and back buffers. It avoids blocking (like `swapBuffers()`) by spawning a thread in which the swap is waited for. This does not make it obviously better than `swapBuffers()`, because spawning a thread has a cost and may introduce synchronization problems. Also, because this function does not block, in order to know when the buffer swap took place, you need to check `hasSwappedSinceLastCheck()` in order to know when the buffer swap has taken place.

**13.16.2.24 CX\_DataFrame CX\_Display::testBufferSwapping ( CX\_Millis *desiredTestDuration*, bool *testSecondaryThread* )**

This function tests buffer swapping under various combinations of Vsync setting and whether the swaps are requested in the main thread or in a secondary thread. The tests combine visual inspection and automated time measurement. The visual inspection is important because what the computer is told to put on the screen and what is actually drawn on the screen are not always the same. It is best to run the tests in full screen mode, although that is not enforced. At the end of the tests, the results of the tests are provided to you to interpret based on the guidelines described here. The outcome of the test will usually be that there are some modes that work better than others for the tested computer.

In the resulting `CX_DataFrame`, there are three columns that give the test conditions. "thread" indicates whether the main thread or a secondary thread was used. "hardVSync" and "softVSync" indicate whether hardware or software Vsync were enabled for the test (see `useHardwareVSync()` and `useSoftwareVSync()`). Other columns, giving data from the tests, are explained below. Whatever combination of Vsync works best can be set up for use in experiments using `useHardwareVSync()` and `useSoftwareVSync()` to set the values. The threading mode is primarily determined by `CX_SlidePresenter` with the `CX::CX_SlidePresenter::Configuration::SwappingMode` setting, although some experiments might want to use threaded swaps directly.

### Continuous swapping test

This test examines the case of constantly swapping the front and back buffers. It measures the amount of time between swaps, which should always approximately equal the frame period. The data from this test are found in columns of the data frame beginning with "cs": "csDurations" gives the raw between-swap durations, and "csDurationMean" and "csDurationStdDev" give the mean and standard deviation of the durations. If the swapping durations are not very consistent, which can be determined by examination or by looking at the standard deviation, then there is a problem with the configuration. If the mean duration is different from the monitor's actual refresh period, then there is a serious problem with the configuration.

During this test, you should see the screen very rapidly flickering between black and white. If you see slow flickering or a constant color, that is an error.

### Wait swap test

One case that this function checks for is what happens if a swap is requested after a long period of no swaps being requested. In particular, this function swaps, waits for 2.5 swap periods and then swaps twice in a row. The idea is that there is a long delay between the first swap (the "long" swap) and the second swap (the "short" swap), followed by a standard delay before the third swap (the "normal" swap).

There are graded levels of success in this test. Complete success is when the duration of the first swap is 3P, where P is the standard swap period (i.e. the length of one frame), and the duration of both of the second two swaps is 1P. Partial success is if the duration of the long swap is  $\sim 2.5P$ , the duration of the short swap is  $\sim .5P$ , and the duration of the normal swap is 1P. In this case, the short swap at least gets things back on the right track. Failure occurs if the short swap duration is  $\sim 0P$ . Mega failure occurs if the normal swap duration is  $\sim 0P$ . In this case, it is taking multiple repeated swaps in order to regain vertical synchronization, which is unacceptable behavior.

You can visually check these results. During this test, an attempt is made to draw three bars on the left, middle, and right of the screen. The left bar is drawn for the long duration, the middle bar for the short duration, and the right bar for the normal duration. Complete success results in all three bars flickering (although you still need to check the timing data). Partial success results in only the left and right bars flickering with the middle bar location flat black. For the partial success case, the middle bar is never visible because at the time at which it is swapped in, the screen is in the middle of a refresh cycle. When the next refresh cycle starts, then the middle bar can start to be drawn to the screen. However, before it has a chance to be drawn, the right rectangle is drawn to the back buffer, overwriting the middle bar (or at least, this is my best explanation for why it isn't visible).

The timing data from the wait swap test can be found in columns of the data frame beginning with "ws". "wsLongMean", "wsShortMean", and "wsNormalMean" are the averages of the long, short, and normal swap durations, respectively. "wsTotalMean" is the sum of wsLongMean, wsShortMean, and wsNormalMean. But also be sure to check the raw data in "wsDurations", which goes along with the duration type in "wsType".

The wait swap test is not performed for the secondary thread, because the assumption is that if the secondary thread is used, in that thread the front and back buffers will be swapped constantly so there will be no wait swaps.

### Parameters

<i>desiredTestDuration</i>	An approximate amount of time to spend performing all of the tests, so the time if divided among all of the tests.
<i>testSecondaryThread</i>	If true, buffer swapping from within a secondary thread will be tested. If false, only swapping from within the main thread will be tested.

### Returns

A [CX\\_DataFrame](#) containing timing results from the tests.

### Note

This function blocks for approximately `desiredTestDuration` or more. See [Blocking Code](#).

#### 13.16.2.25 void CX\_Display::useHardwareVSync ( bool *b* )

Sets whether the display is using hardware VSync to control frame presentation. Without some form of Vsync, vertical tearing may occur.

**Parameters**

<i>b</i>	If <code>true</code> , hardware VSync will be enabled in the video card driver. If <code>false</code> , it will be disabled.
----------	--

**Note**

This may not work, depending on your video card settings. Modern video card drivers allow you to control whether Vsync is used for all applications or not, or whether the applications are allowed to choose from themselves whether to use Vsync. If your drivers are set to force Vsync to a particular setting, this function is unlikely to have an effect. Even when the drivers allow applications to choose a Vsync setting, it is still possible that this function will have not have the expected effect. OpenGL seems to struggle with VSync.

**See Also**

See [Framebuffers and Buffer Swapping](#) for information on what VSync is.

**13.16.2.26 void CX\_Display::useSoftwareVSync ( bool b )**

Sets whether the display is using software VSync to control frame presentation. Without some form of Vsync, vertical tearing can occur. Hardware VSync, if available, is generally preferable to software VSync, so see [useHardwareVSync\(\)](#) as well. However, software and hardware VSync are not mutually exclusive, sometimes using both together works better than only using one.

**Parameters**

<i>b</i>	If <code>true</code> , the display will attempt to do VSync in software.
----------	--

**See Also**

See [Framebuffers and Buffer Swapping](#) for information on what Vsync is.

**13.16.2.27 void CX\_Display::waitForOpenGL ( void )**

This function blocks until all OpenGL instructions that were given before this was called to complete. This can be useful if you are trying to determine how long a set of rendering commands takes or need to make sure that all rendering is complete before moving on with other tasks.

**See Also**

[Blocking Code](#)

The documentation for this class was generated from the following files:

- CX\_Display.h
- CX\_Display.cpp

**13.17 CX::CX\_InputManager Class Reference**

```
#include <CX_InputManager.h>
```

**Public Member Functions**

- bool [setup](#) (bool useKeyboard, bool useMouse, int joystickIndex=-1)
- bool [pollEvents](#) (void)
- void [clearAllEvents](#) (bool poll=false)

## Public Attributes

- [CX\\_Keyboard Keyboard](#)  
*An instance of [CX::CX\\_Keyboard](#). Enabled or disabled with [CX::CX\\_InputManager::setup\(\)](#).*
- [CX\\_Mouse Mouse](#)  
*An instance of [CX::CX\\_Mouse](#). Enabled or disabled with [CX::CX\\_InputManager::setup\(\)](#).*
- [CX\\_Joystick Joystick](#)  
*An instance of [CX::CX\\_Joystick](#). Enabled or disabled with [CX::CX\\_InputManager::setup\(\)](#).*

## Friends

- [CX\\_InputManager Private::inputManagerFactory](#) (void)

## 13.17.1 Detailed Description

This class is responsible for managing three basic input devices: a keyboard, mouse, and joystick. You access each of these devices with the corresponding member class: Keyboard, Mouse, and Joystick. See [CX::CX\\_Keyboard](#), [CX::CX\\_Mouse](#), and [CX::CX\\_Joystick](#) for more information about each specific device.

By default, all three input devices are disabled. Call [setup\(\)](#) to enable specific devices. Alternately, you can call [CX\\_Mouse::enable\(\)](#) or [CX\\_Keyboard::enable\(\)](#), if that makes more sense to you.

This class has a private constructor because you should never need more than one of them. If you really, really need more than one, you can use [CX::Private::inputManagerFactory\(\)](#) to make one.

## 13.17.2 Member Function Documentation

## 13.17.2.1 void CX::CX\_InputManager::clearAllEvents ( bool poll = false )

This function clears all events on all input devices.

## Parameters

<i>poll</i>	If true, events are polled before they are cleared, so that events that hadn't yet made it into the device specific queues (e.g. the Keyboard queue) are cleared as well.
-------------	---

## 13.17.2.2 bool CX::CX\_InputManager::pollEvents ( void )

It is not typically necessary for the user to call this function directly, although there is no harm in doing so. This function polls for new events on all of the configured input devices (see [setup\(\)](#)). After a call to this function, new events for the input devices can be found by checking the [availableEvents\(\)](#) function for each device.

## Returns

True if there are any events available for enabled devices, false otherwise. The events do not necessarily need to be new events. If there were events that were already stored in Mouse, Keyboard, or Joystick that had not been processed by user code at the time this function was called, this function will return true.

## 13.17.2.3 bool CX::CX\_InputManager::setup ( bool useKeyboard, bool useMouse, int joystickIndex = -1 )

Setup the input manager to use the requested devices. You may call this function multiple times if you want to change the configuration over the course of the experiment. Every time this function is called, all input device events are cleared.

**Parameters**

<i>useKeyboard</i>	Enable or disable the keyboard.
<i>useMouse</i>	Enable or disable the mouse.
<i>joystickIndex</i>	Optional. If $\geq 0$ , an attempt will be made to set up the joystick at that index. If $< 0$ , no attempt will be made to set up the joystick.

**Returns**

False if the requested joystick could not be set up correctly, true otherwise.

The documentation for this class was generated from the following files:

- CX\_InputManager.h
- CX\_InputManager.cpp

**13.18 CX::CX\_Joystick Class Reference**

```
#include <CX_Joystick.h>
```

**Classes**

- struct [Event](#)

**Public Member Functions**

- bool [setup](#) (int joystickIndex)
- std::string [getJoystickName](#) (void)
- int [getJoystickIndex](#) (void)
- bool [pollEvents](#) (void)
- int [availableEvents](#) (void)
- [CX\\_Joystick::Event](#) [getNextEvent](#) (void)
- void [clearEvents](#) (void)
- std::vector< float > [getAxisPositions](#) (void)
- std::vector< unsigned char > [getButtonStates](#) (void)

**13.18.1 Detailed Description**

This class manages a joystick that is attached to the system (if any). If more than one joystick is needed for the experiment, you can create more instances of [CX\\_Joystick](#) other than the one in [CX::Instances::Input](#). Unlike [CX\\_Keyboard](#) and [CX\\_Mouse](#), [CX\\_Joystick](#) does not need to be in a [CX\\_InputManager](#) to work.

**13.18.2 Member Function Documentation****13.18.2.1 int CX\_Joystick::availableEvents ( void )**

Get the number of new events available for this input device.



**13.18.2.2 void CX\_Joystick::clearEvents ( void )**

Clear (delete) all events from this input device.

**Note**

This function only clears already existing events from the device, which means that responses made between a call to [CX\\_InputManager::pollEvents\(\)](#) and a subsequent call to [clearEvents\(\)](#) will not be removed by calling [clearEvents\(\)](#).

**13.18.2.3 vector< float > CX\_Joystick::getAxisPositions ( void )**

This function is to be used for direct access to the axis positions of the joystick. It does not generate events (i.e. [CX\\_Joystick::Event](#)), nor does it do any timestamping. If timestamps and uncertainties are desired, you MUST use [pollEvents\(\)](#) and the associated event functions (e.g. [getNextEvent\(\)](#)).

**13.18.2.4 vector< unsigned char > CX\_Joystick::getButtonStates ( void )**

This function is to be used for direct access to the button states of the joystick. It does not generate events (i.e. [CX\\_Joystick::Event](#)), nor does it do any timestamping. If timestamps and uncertainties are desired, you MUST use [pollEvents\(\)](#) and the associated event functions (e.g. [getNextEvent\(\)](#)).

**13.18.2.5 std::string CX\_Joystick::getJoystickName ( void )**

Get the name of the joystick, presumably as set by the joystick driver. The name may not be very meaningful.

**13.18.2.6 CX\_Joystick::Event CX\_Joystick::getNextEvent ( void )**

Get the next event available for this input device. This is a destructive operation: the returned event is deleted from the input device.

**13.18.2.7 bool CX\_Joystick::pollEvents ( void )**

Check to see if there are any new joystick events. If there are new events, they can be accessed with [availableEvents\(\)](#) and [getNextEvent\(\)](#).

**Returns**

True if there are new events.

**13.18.2.8 bool CX\_Joystick::setup ( int joystickIndex )**

Set up the joystick by attempting to initialize the joystick at the given index. If the joystick is present on the system, it will be initialized and its name can be accessed by calling [getJoystickName\(\)](#).

If the set up is successful (i.e. if the selected joystick is present on the system), this function will return true. If the joystick is not present, it will return false.

The documentation for this class was generated from the following files:

- CX\_Joystick.h
- CX\_Joystick.cpp

**13.19 CX::CX\_Keyboard Class Reference**

```
#include <CX_Keyboard.h>
```

## Classes

- struct [Event](#)

## Public Types

- enum [KeyboardEventType](#) { [PRESSED](#), [RELEASED](#), [REPEAT](#) }

## Public Member Functions

- void [enable](#) (bool enable)
- bool [enabled](#) (void)
- int [availableEvents](#) (void)
- [CX\\_Keyboard::Event getNextEvent](#) (void)
- void [clearEvents](#) (void)
- bool [isKeyHeld](#) (int key)
- [CX\\_Keyboard::Event waitForKeypress](#) (int key, bool clear=true, bool eraseEvent=false)
- [CX\\_Keyboard::Event waitForKeypress](#) (std::vector< int > keys, bool clear=true, bool eraseEvent=false)

## Friends

- class **[CX\\_InputManager](#)**

## 13.19.1 Detailed Description

This class is responsible for managing the mouse.

## 13.19.2 Member Enumeration Documentation

13.19.2.1 enum **[CX::CX\\_Keyboard::KeyboardEventType](#)**

The type of the keyboard event.

## Enumerator

***PRESSED*** A key has been pressed.

***RELEASED*** A key has been released.

***REPEAT*** A key has been held for some time and automatic key repeat has kicked in, causing multiple keypresses to be rapidly sent. This event is one of the many repeats.

## 13.19.3 Member Function Documentation

13.19.3.1 int **[CX::CX\\_Keyboard::availableEvents](#)** ( void )

Get the number of new events available for this input device.

**13.19.3.2 void CX::CX\_Keyboard::clearEvents ( void )**

Clear (delete) all events from this input device.

**Note**

This function only clears already existing events from the device, which means that responses made between a call to [CX\\_InputManager::pollEvents\(\)](#) and a subsequent call to [clearEvents\(\)](#) will not be removed by calling [clearEvents\(\)](#).

**13.19.3.3 void CX::CX\_Keyboard::enable ( bool *enable* )**

Enable or disable the keyboard.

**Parameters**

<i>enable</i>	If <code>true</code> , the keyboard will be enabled; if <code>false</code> it will be disabled.
---------------	---

**13.19.3.4 bool CX::CX\_Keyboard::enabled ( void )**

Returns `true` if the keyboard is enabled.

**13.19.3.5 CX\_Keyboard::Event CX::CX\_Keyboard::getNextEvent ( void )**

Get the next event available for this input device. This is a destructive operation: the returned event is deleted from the input device.

**13.19.3.6 bool CX::CX\_Keyboard::isKeyHeld ( int *key* )**

This function checks to see if the given key is held, which means a keypress has been received, but not a key release.

**Parameters**

<i>key</i>	The key code or character for the key you are interested in. See the documentation for <a href="#">CX_Keyboard::Event::key</a> for more information about this value.
------------	---

**Returns**

True iff the given key is held.

**13.19.3.7 CX\_Keyboard::Event CX::CX\_Keyboard::waitForKeypress ( int *key*, bool *clear* = `true`, bool *eraseEvent* = `false` )**

Identical to [waitForKeypress\(\)](#) that takes a vector of keys except with a length 1 vector.

**13.19.3.8 CX\_Keyboard::Event CX::CX\_Keyboard::waitForKeypress ( std::vector< int > *keys*, bool *clear* = `true`, bool *eraseEvent* = `false` )**

Wait until the first of the given *keys* is pressed. This specifically checks that a key has been pressed: If it was held at the time this function was called and then released, it will have to be pressed again before this function will return. Returns a [CX\\_Keyboard::Event](#) for the key that was waited on, optionally removing that event from the stored events if *eraseEvent* is `true`.

## Parameters

<i>keys</i>	A vector of key codes for the keys that will be waited on. If any of the codes are -1, any keypress will cause this function to return.
<i>clear</i>	If <code>true</code> , all waiting events will be flushed with <code>CX_InputManager::pollEvents()</code> and then all keyboard events will be cleared both before and after waiting for the keypress. If <code>false</code> and <code>this-&gt;availableEvents() &gt; 0</code> , it is possible that one of the available events will include a keypress for a given key, in which case this function will return immediately.
<i>eraseEvent</i>	If <code>true</code> , the event will be erased from the queue of captured events. The implication of this removal is that the return value of this function is the only opportunity to gain access to the event that caused this function to return. The advantage of this approach is that if, after some given key is pressed, all events in the queue are processed, you are guaranteed to not hit the same event twice (once from the return value of this function, once from processing the queue).

## Returns

A `CX_Keyboard::Event` with information about the keypress that caused this function to return.

## Note

If the keyboard is not enabled at the time this function is called, it will be enabled for the duration of the function and then disabled at the end of the function.

The documentation for this class was generated from the following files:

- `CX_Keyboard.h`
- `CX_Keyboard.cpp`

## 13.20 CX::Util::CX\_LapTimer Class Reference

```
#include <CX_TimeUtilities.h>
```

## Public Member Functions

- `CX_LapTimer` (`CX_Clock *clock`, unsigned int logSamples=0)
- void `setup` (`CX_Clock *clock`, unsigned int logSamples=0)
- void `restart` (void)
- void `takeSample` (void)
- unsigned int `collectedSamples` (void)
- `CX_Millis mean` (void)  
Get the mean value of the stored lap times.
- `CX_Millis min` (void)  
Get the shortest stored lap time.
- `CX_Millis max` (void)  
Get the longest stored lap time.
- `CX_Millis stdDev` (void)  
Get the standard deviation of the stored lap times.
- std::string `getStatString` (void)

### 13.20.1 Detailed Description

This class can be used for profiling loops. It measures the amount of time that elapses between subsequent calls to [takeSample\(\)](#). One possible use is to determine how long it takes between calls to an important function, like [CX\\_InputManager::pollEvents\(\)](#) or [CX\\_Display::swapBuffers\(\)](#)

```
//Set up collection:
CX_LapTimer lt;
lt.setup(&Clock, 1000); //Every 1000 samples, the results of those samples will be automatically
                        //logged.

//In the loop:
while (whatever) {
    //other code...
    lt.takeSample();
    //other code...
}
Log.flush(); //Check the results of the profiling.
```

### 13.20.2 Constructor & Destructor Documentation

#### 13.20.2.1 CX::Util::CX\_LapTimer::CX\_LapTimer ( CX\_Clock \* clock, unsigned int logSamples = 0 )

Construct and set up a [CX\\_LapTimer](#). See [CX\\_LapTimer::setup\(\)](#) for a description of the parameters.

### 13.20.3 Member Function Documentation

#### 13.20.3.1 unsigned int CX::Util::CX\_LapTimer::collectedSamples ( void )

Returns the number of lap durations that have been collected.

#### 13.20.3.2 std::string CX::Util::CX\_LapTimer::getStatString ( void )

Get a string summarizing some basic descriptive statistics for the currently stored lap durations.

Returns

A string containing the minimum, mean, maximum, and standard deviation of the collected samples.

#### 13.20.3.3 void CX::Util::CX\_LapTimer::restart ( void )

Restart data collection. All collected samples are cleared.

#### 13.20.3.4 void CX::Util::CX\_LapTimer::setup ( CX\_Clock \* clock, unsigned int logSamples = 0 )

Set up the [CX\\_LapTimer](#) with the selected clock source and the number of samples to log between each automatic logging of results.

Parameters

<i>clock</i>	The instance of <a href="#">CX_Clock</a> to use.
<i>logSamples</i>	If this is not 0, then every <i>logSamples</i> samples, a string containing information about the last <i>logSamples</i> samples will be logged and then those samples will be cleared.

#### 13.20.3.5 void CX::Util::CX\_LapTimer::takeSample ( void )

Take a single sample of time. If at least one previous sample has been taken, the difference between the current time and the previous time is stored as the duration of that "lap" through the code.

The documentation for this class was generated from the following files:

- CX\_TimeUtilities.h
- CX\_TimeUtilities.cpp

### 13.21 CX::Util::CX\_LengthToPixelConverter Class Reference

#include <CX\_UnitConversion.h>

Inherits [CX::Util::CX\\_BaseUnitConverter](#).

#### Public Member Functions

- [CX\\_LengthToPixelConverter](#) (float pixelsPerUnit, bool roundResult=false)
- void [setup](#) (float pixelsPerUnit, bool roundResult=false)
- float [operator\(\)](#) (float length) override
- float [inverse](#) (float pixels) override
- bool [configureFromFile](#) (std::string filename, std::string delimiter="=", bool trimWhitespace=true, std::string commentString="//")

#### 13.21.1 Detailed Description

This simple utility class is used for converting lengths (perhaps of objects drawn on the monitor) to pixels on a monitor. See also [CX::Util::CX\\_CoordinateConverter](#) for a way to also convert from one coordinate system to another. This assumes that pixels are square, which may not be true, especially if you are using a resolution that is not the native resolution of the monitor.

Example use:

```
CX_LengthToPixelConverter l2p(75); //75 pixels per unit length (e.g. inch) on the target monitor.
ofLine( 200, 100, 200 + l2p(1), 100 + l2p(2) ); //Draw a line from (200, 100) (in pixel coordinates) to
//1 distance unit right and 2 units down from that point.
```

#### 13.21.2 Constructor & Destructor Documentation

##### 13.21.2.1 CX::Util::CX\_LengthToPixelConverter::CX\_LengthToPixelConverter ( float pixelsPerUnit, bool roundResult = false )

Constructs a [CX\\_LengthToPixelConverter](#) with the given configuration. See [setup\(\)](#) for the meaning of the parameters.

#### 13.21.3 Member Function Documentation

##### 13.21.3.1 bool CX::Util::CX\_LengthToPixelConverter::configureFromFile ( std::string filename, std::string delimiter = "=", bool trimWhitespace = true, std::string commentString = "//" )

This function exists to serve a per-computer configuration function that is otherwise difficult to provide due to the fact that C++ programs are compiled to binaries and cannot be easily edited on the computer on which they are running. This function takes the file name of a specially constructed configuration file and reads the key-value pairs in that file in order to configure the [CX\\_LengthToPixelConverter](#). The format of the file is provided in the example code below.

Sample configuration file:

```
L2PC.pixelsPerUnit = 35
L2PC.roundResult = true
```

All of the configuration keys are used in this example. Note that the "L2PC" prefix allows this configuration to be embedded in a file that also performs other configuration functions.

See [CX\\_LengthToPixelConverter::setup\(\)](#) for details about the meanings of the configuration options.

Because this function uses [CX::Util::readKeyValueFile\(\)](#) internally, it has the same arguments.

#### Parameters

<i>filename</i>	The name of the file containing configuration data.
<i>delimiter</i>	The string that separates the key from the value. In the example, it is "=", but can be other values.
<i>trimWhitespace</i>	If true, whitespace characters surrounding both the key and value will be removed. This is a good idea to do.
<i>commentString</i>	If <i>commentString</i> is not the empty string (""), everything on a line following the first instance of <i>commentString</i> will be ignored.

#### Returns

`true` if there were no problems reading in the file, `false` otherwise.

**13.21.3.2** `float CX::Util::CX_LengthToPixelConverter::inverse ( float pixels )` `[override],[virtual]`

Performs to inverse of `operator()`, i.e. converts pixels to length.

#### Parameters

<i>pixels</i>	The number of pixels to convert to a length.
---------------	--

#### Returns

The length of the given number of pixels.

Reimplemented from [CX::Util::CX\\_BaseUnitConverter](#).

**13.21.3.3** `float CX::Util::CX_LengthToPixelConverter::operator() ( float length )` `[override],[virtual]`

Converts the length to pixels based on the settings given during construction.

#### Parameters

<i>length</i>	The length to convert to pixels.
---------------	----------------------------------

#### Returns

The number of pixels corresponding to the length.

Reimplemented from [CX::Util::CX\\_BaseUnitConverter](#).

**13.21.3.4** `void CX::Util::CX_LengthToPixelConverter::setup ( float pixelsPerUnit, bool roundResult = false )`

Sets up a [CX\\_LengthToPixelConverter](#) with the given configuration.

#### Parameters

<i>pixelsPerUnit</i>	The number of pixels per one length unit. This can be measured by drawing a ~100-1000 pixel square on the screen and measuring the length of a side and dividing the number of pixels by the total length measured.
<i>roundResult</i>	If true, the result of conversions will be rounded to the nearest integer (i.e. pixel). For drawing certain kinds of stimuli (especially text) it can be helpful to draw on pixel boundaries.

The documentation for this class was generated from the following files:

- CX\_UnitConversion.h
- CX\_UnitConversion.cpp

## 13.22 CX::CX\_Logger Class Reference

```
#include <CX_Logger.h>
```

### Public Member Functions

- `std::stringstream & log (CX_LogLevel level, std::string module="")`
- `std::stringstream & verbose (std::string module="")`
- `std::stringstream & notice (std::string module="")`
- `std::stringstream & warning (std::string module="")`
- `std::stringstream & error (std::string module="")`
- `std::stringstream & fatalError (std::string module="")`
- `void level (CX_LogLevel level, std::string module="")`
- `void levelForConsole (CX_LogLevel level)`
- `void levelForFile (CX_LogLevel level, std::string filename="CX_LOGGER_DEFAULT")`
- `void levelForAllModules (CX_LogLevel level)`
- `CX_LogLevel getModuleLevel (std::string module)`
- `void flush (void)`
- `void timestamps (bool logTimestamps, std::string format="%H:%M:%S.%i")`
- `void setMessageFlushCallback (std::function< void(CX_MessageFlushData &)> f)`
- `void captureOFLogMessages (void)`

### 13.22.1 Detailed Description

This class is used for logging messages throughout the [CX](#) backend code. It can also be used in user code to log messages. Rather than instantiating your own copy of [CX\\_Logger](#), it is probably better to use the preinstantiated [CX::Instances::Log](#).

There is an example showing a number of the features of [CX\\_Logger](#) named example-logging.

### 13.22.2 Member Function Documentation

#### 13.22.2.1 void CX::CX\_Logger::captureOFLogMessages ( void )

Set this instance of [CX\\_Logger](#) to be the target of any messages created by oF logging functions.

#### 13.22.2.2 std::stringstream & CX::CX\_Logger::error ( std::string module = " " )

This function is equivalent to a call to `log(CX_LogLevel::LOG_ERROR, module)`.



**13.22.2.3** `std::stringstream & CX::CX_Logger::fatalError ( std::string module = " " )`

This function is equivalent to a call to `log(CX_LogLevel::LOG_FATAL_ERROR, module)`.

**13.22.2.4** `void CX::CX_Logger::flush ( void )`

Log all of the messages stored since the last call to `flush()` to the selected logging targets. This is a blocking operation, because it may take quite a while to output all log messages to various targets (see [Blocking Code](#)). This function is not thread-safe: Only call it from the main thread.

**13.22.2.5** `CX_LogLevel CX::CX_Logger::getModuleLevel ( std::string module )`

Gets the log level in use by the given module.

**Parameters**

<i>module</i>	The name of the module.
---------------	-------------------------

**Returns**

The `CX_LogLevel` for `module`.

**13.22.2.6** `void CX::CX_Logger::level ( CX_LogLevel level, std::string module = " " )`

Sets the log level for the given module. Messages from that module that are at a lower level than `level` will be ignored.

**Parameters**

<i>level</i>	See the <a href="#">CX::CX_LogLevel</a> enum for valid values.
<i>module</i>	A string representing one of the modules from which log messages are generated.

**13.22.2.7** `void CX::CX_Logger::levelForAllModules ( CX_LogLevel level )`

Set the log level for all modules. This works both retroactively and proactively: All currently known modules are given the log level and the default log level for new modules as set to the level.

**13.22.2.8** `void CX::CX_Logger::levelForConsole ( CX_LogLevel level )`

Set the log level for messages to be printed to the console.

**13.22.2.9** `void CX::CX_Logger::levelForFile ( CX_LogLevel level, std::string filename = "CX_LOGGER_DEFAULT" )`

Sets the log level for the file with given file name. If the file does not exist, it will be created. If the file does exist, it will be overwritten with a warning logged to cerr.

**Parameters**

<i>level</i>	See the <code>CX_LogLevel</code> enum for valid values.
<i>filename</i>	Optional. If no file name is given, a file with name generated from a date/time from the start time of the experiment will be used.

**13.22.2.10** `std::stringstream & CX::CX_Logger::log ( CX_LogLevel level, std::string module = " " )`

This is the basic logging function for this class. Example use:

```
Log.log(CX_LogLevel::LOG_WARNING, "myModule") << "My message number " << 20;
```

Possible output: "[warning] <myModule> My message number 20"

**Parameters**

<i>level</i>	Log level for this message. This has implications for message filtering. See <a href="#">level()</a> . This should not be LOG_ALL or LOG_NONE, because that would be weird, wouldn't it?
<i>module</i>	Name of the module that this log message is related to. This has implications for message filtering. See <a href="#">level()</a> .

**Returns**

A reference to a `std::stringstream` that the log message data should be streamed into.

**Note**

This function and all of the trivial wrappers of this function ([verbose\(\)](#), [notice\(\)](#), [warning\(\)](#), [error\(\)](#), [fatalError\(\)](#)) are thread-safe.

**13.22.2.11** `std::stringstream & CX::CX_Logger::notice ( std::string module = " " )`

This function is equivalent to a call to `log(CX_LogLevel::LOG_NOTICE, module)`.

**13.22.2.12** `void CX::CX_Logger::setMessageFlushCallback ( std::function< void(CX_MessageFlushData &)> f )`

Sets the user function that will be called on each message flush event. For every message that has been logged, the user function will be called. No filtering is performed: All messages regardless of the module log level will be sent to the user function.

**Parameters**

<i>f</i>	A pointer to a user function that takes a reference to a <a href="#">CX_MessageFlushData</a> struct and returns nothing.
----------	--

**13.22.2.13** `void CX::CX_Logger::timestamps ( bool logTimestamps, std::string format = "%H:%M:%S.%i" )`

Set whether or not to log timestamps and the format for the timestamps.

**Parameters**

<i>logTimestamps</i>	Does what it says.
<i>format</i>	Timestamp format string. See <a href="http://pocoproject.org/docs/Poco.DateTime-Formatter.html#4684">http://pocoproject.org/docs/Poco.DateTime-Formatter.html#4684</a> for documentation of the format. Defaults to H:M:S.i (24-hour clock with milliseconds at the end).

**13.22.2.14** `std::stringstream & CX::CX_Logger::verbose ( std::string module = " " )`

This function is equivalent to a call to `log(CX_LogLevel::LOG_VERBOSE, module)`.

**13.22.2.15** `std::stringstream & CX::CX_Logger::warning ( std::string module = " " )`

This function is equivalent to a call to `log(CX_LogLevel::LOG_WARNING, module)`.

The documentation for this class was generated from the following files:

- CX\_Logger.h
- CX\_Logger.cpp

## 13.23 CX::CX\_MessageFlushData Struct Reference

```
#include <CX_Logger.h>
```

### Public Member Functions

- [CX\\_MessageFlushData](#) (std::string message\_, [CX\\_LogLevel](#) level\_, std::string module\_)

### Public Attributes

- std::string [message](#)  
*A string containing the logged message.*
- [CX\\_LogLevel](#) [level](#)  
*The log level of the message.*
- std::string [module](#)  
*The module associated with the message, usually which created the message.*

#### 13.23.1 Detailed Description

If a user function is listening for flush callbacks by using setMessageFlushCallback(), each time the user function is called, it gets a reference to an instance of this struct with all the information filled in.

#### 13.23.2 Constructor & Destructor Documentation

13.23.2.1 CX::CX\_MessageFlushData::CX\_MessageFlushData ( std::string *message\_*, [CX\\_LogLevel](#) *level\_*, std::string *module\_* )  
[inline]

Convenience constructor which constructs an instance of the struct with the provided values.

The documentation for this struct was generated from the following file:

- CX\_Logger.h

## 13.24 CX::CX\_Mouse Class Reference

```
#include <CX_Mouse.h>
```

### Classes

- struct [Event](#)

### Public Types

- enum [Buttons](#) : int { **LEFT** = OF\_MOUSE\_BUTTON\_LEFT, **MIDDLE** = OF\_MOUSE\_BUTTON\_MIDDLE, **RIGHT** = OF\_MOUSE\_BUTTON\_RIGHT }
- enum [MouseEventType](#) { [MOVED](#), [PRESSED](#), [RELEASED](#), [DRAGGED](#), [SCROLLED](#) }

## Public Member Functions

- void [enable](#) (bool enable)
- bool [enabled](#) (void)
- int [availableEvents](#) (void)
- [CX\\_Mouse::Event getNextEvent](#) (void)
- void [clearEvents](#) (void)
- void [showCursor](#) (bool show)
- void [setCursorPosition](#) (ofPoint pos)
- ofPoint [getCursorPosition](#) (void)

## Friends

- class [CX\\_InputManager](#)

### 13.24.1 Detailed Description

This class is responsible for managing the mouse. It has a private constructor, so you can only use the [CX\\_Mouse](#) that is a part of a [CX\\_InputManager](#), specifically the instance of a [CX\\_InputManager](#) named [CX::Instances::Input](#).

### 13.24.2 Member Enumeration Documentation

#### 13.24.2.1 enum [CX::CX\\_Mouse::Buttons](#) : int [\[strong\]](#)

Names of the mouse buttons corresponding to some of the integer button identifiers.

#### 13.24.2.2 enum [CX::CX\\_Mouse::MouseEventType](#)

## Enumerator

**MOVED** The mouse has been moved without a button being held. [Event::button](#) should be -1 (meaningless).

**PRESSED** A mouse button has been pressed. Check [Event::button](#) for the button index and [Event::x](#) and [Event::y](#) for the location.

**RELEASED** A mouse button has been released. Check [Event::button](#) for the button index and [Event::x](#) and [Event::y](#) for the location.

**DRAGGED** can be changed during a drag, or multiple buttons may be held at once during a drag. The mouse has been moved while at least one button was held. [Event::button](#) may not be meaningful because the held button

**SCROLLED** [Event::x](#) if your mouse has a wheel that can move horizontally. The mouse wheel has been scrolled. Check [Event::y](#) to get the change in the standard mouse wheel direction, or

### 13.24.3 Member Function Documentation

#### 13.24.3.1 int [CX::CX\\_Mouse::availableEvents](#) ( void )

Get the number of new events available for this input device.

**13.24.3.2 void CX::CX\_Mouse::clearEvents ( void )**

Clear (delete) all events from this input device.

**Note**

This function only clears already existing events from the device, which means that responses made between a call to [CX\\_InputManager::pollEvents\(\)](#) and a subsequent call to [clearEvents\(\)](#) will not be removed by calling [clearEvents\(\)](#).

**13.24.3.3 void CX::CX\_Mouse::enable ( bool *enable* )**

Enable or disable the mouse.

**Parameters**

<i>enable</i>	If <code>true</code> , the mouse will be enabled; if <code>false</code> it will be disabled.
---------------	--

**13.24.3.4 bool CX::CX\_Mouse::enabled ( void )**

Returns `true` if the mouse is enabled.

**13.24.3.5 ofPoint CX::CX\_Mouse::getCursorPosition ( void )**

Get the cursor position within the program window. If the mouse has left the window, this will return the last known position of the cursor within the window.

**Returns**

An `ofPoint` with the last cursor position.

**13.24.3.6 CX\_Mouse::Event CX::CX\_Mouse::getNextEvent ( void )**

Get the next event available for this input device. This is a destructive operation: the returned event is deleted from the input device.

**13.24.3.7 void CX::CX\_Mouse::setCursorPosition ( ofPoint *pos* )**

Sets the position of the cursor, relative to the program the window. The window must be focused.

**Parameters**

<i>pos</i>	The location within the window to set the cursor.
------------	---

**13.24.3.8 void CX::CX\_Mouse::showCursor ( bool *show* )**

Show or hide the mouse cursor within the program window. If in windowed mode, the cursor will be visible outside of the window.

**Parameters**

<i>show</i>	If <code>true</code> , the cursor will be shown, if <code>false</code> it will not be shown.
-------------	--

The documentation for this class was generated from the following files:

- CX\_Mouse.h
- CX\_Mouse.cpp

### 13.25 CX::Draw::CX\_PatternProperties\_t Class Reference

```
#include <CX_Draw.h>
```

#### Public Types

- enum { **SINE\_WAVE**, **SQUARE\_WAVE**, **TRIANGLE\_WAVE** }
- enum { **AP\_CIRCLE**, **AP\_RECTANGLE** }

#### Public Attributes

- unsigned char [minValue](#)  
*The minimum value that will be used in the pattern.*
- unsigned char [maxValue](#)  
*The maximum value that will be used in the pattern.*
- float [width](#)
- float [height](#)
- float [angle](#)
- float [period](#)  
*The distance, in pixels, between the center of each wave within the pattern.*
- float [phase](#)  
*The offset, in degrees, of the waves.*
- enum  
CX::Draw::CX\_PatternProperties\_t:: { ... } [maskType](#)
- enum  
CX::Draw::CX\_PatternProperties\_t:: { ... } [apertureType](#)
- float [fallOffPower](#)

#### 13.25.1 Detailed Description

This structure contains settings controlling the creation of greyscale patterns using [CX::Draw::greyscalePatternToPixels\(\)](#). The pattern that is created looks like a simple gabor pattern.

#### 13.25.2 Member Enumeration Documentation

##### 13.25.2.1 anonymous enum

The type of waves that will be used in the pattern.

##### 13.25.2.2 anonymous enum

Because the pattern created with these settings extends to infinity in every direction, an aperture through which it is to be viewed must be specified. The aperture can either be a circle or a rectangle.

#### 13.25.3 Member Data Documentation

##### 13.25.3.1 float CX::Draw::CX\_PatternProperties\_t::angle

The angle at which the waves are oriented.

## 13.25.3.2 enum { ... } CX::Draw::CX\_PatternProperties\_t::apertureType

Because the pattern created with these settings extends to infinity in every direction, an aperture through which it is to be viewed must be specified. The aperture can either be a circle or a rectangle.

## 13.25.3.3 float CX::Draw::CX\_PatternProperties\_t::fallOffPower

The intensity of each pixel is decreased slightly based on how far from the center of the pattern that pixel is, depending on the value of `fallOffPower`. By default, there is no falloff. A value of 1 produces a standard cosine falloff. The falloff is computed as  $(\cos((d/r)^{\text{fallOffPower}} * \text{PI}) + 1) / 2$ , where  $d$  is the distance of the current pixel from the center of the pattern and  $r$  is the radius of the pattern.

## 13.25.3.4 float CX::Draw::CX\_PatternProperties\_t::height

The height of the pattern. ignored if `apertureType` is `AP_CIRCLE`.

## 13.25.3.5 enum { ... } CX::Draw::CX\_PatternProperties\_t::maskType

The type of waves that will be used in the pattern.

## 13.25.3.6 float CX::Draw::CX\_PatternProperties\_t::width

The width of the pattern, or if `apertureType` is `AP_CIRCLE`, the diameter of the circle enclosing the pattern.

The documentation for this class was generated from the following file:

- CX\_Draw.h

## 13.26 CX::CX\_RandomNumberGenerator Class Reference

```
#include <CX_RandomNumberGenerator.h>
```

## Public Member Functions

- [CX\\_RandomNumberGenerator](#) (void)
- void [setSeed](#) (unsigned long seed)
- void [setSeed](#) (const std::string &s)
- unsigned long [getSeed](#) (void)
- [CX\\_RandomInt\\_t](#) [getMinimumRandomInt](#) (void)
- [CX\\_RandomInt\\_t](#) [getMaximumRandomInt](#) (void)
- [CX\\_RandomInt\\_t](#) [randomInt](#) (void)
- [CX\\_RandomInt\\_t](#) [randomInt](#) ([CX\\_RandomInt\\_t](#) rangeLower, [CX\\_RandomInt\\_t](#) rangeUpper)
- double [randomDouble](#) (double lowerBound\_closed, double upperBound\_open)
- template<typename T >  
void [shuffleVector](#) (std::vector< T > \*v)
- template<typename T >  
std::vector< T > [shuffleVector](#) (std::vector< T > v)
- template<typename T >  
T [sample](#) (const std::vector< T > &values)
- template<typename T >  
std::vector< T > [sample](#) (unsigned int count, const std::vector< T > &source, bool withReplacement)
- std::vector< int > [sample](#) (unsigned int count, int lowerBound, int upperBound, bool withReplacement)
- template<typename T >  
T [sampleExclusive](#) (const std::vector< T > &values, const T &exclude)

- `template<typename T >`  
`T sampleExclusive (const std::vector< T > &values, const std::vector< T > &exclude)`
- `template<typename T >`  
`std::vector< T > sampleExclusive (unsigned int count, const std::vector< T > &values, const T &exclude, bool withReplacement)`
- `template<typename T >`  
`std::vector< T > sampleExclusive (unsigned int count, const std::vector< T > &values, const std::vector< T > &exclude, bool withReplacement)`
- `template<typename T >`  
`std::vector< T > sampleBlocks (const std::vector< T > &values, unsigned int blocksToSample)`
- `template<typename stdDist >`  
`std::vector< typename stdDist::result_type > sampleRealizations (unsigned int count, stdDist dist)`
- `template<typename T >`  
`std::vector< T > sampleUniformRealizations (unsigned int count, T lowerBound_closed, T upperBound_open)`
- `template<typename T >`  
`std::vector< T > sampleNormalRealizations (unsigned int count, T mean, T standardDeviation)`
- `std::vector< unsigned int > sampleBinomialRealizations (unsigned int count, unsigned int trials, double prob-Success)`
- `std::mt19937_64 & getGenerator (void)`

### 13.26.1 Detailed Description

This class is used for generating random values from a pseudo-random number generator. It uses a version of the Mersenne Twister algorithm, in particular `std::mt19937_64` (see [http://en.cppreference.com/w/cpp/numeric/random/mersenne\\_twister\\_engine](http://en.cppreference.com/w/cpp/numeric/random/mersenne_twister_engine) for the parameters used with this algorithm).

The monolithic structure of `CX_RandomNumberGenerator` provides a certain important feature that a collection of loose function does not have, which is the ability to trivially track the random seed being used for the random number generator. The function `CX_RandomNumberGenerator::setSeed()` sets the seed for all random number generation tasks performed by this class. Likewise, `CX_RandomNumberGenerator::getSeed()` allows you to easily find the seed that is being used for random number generation. Due to this structure, you can easily save the seed that was used for each participant, which allows you to repeat the exact randomizations used for that participant (unless random number generation varies as a function of the responses given by a participant).

An instance of this class is preinstantiated for you. See `CX::Instances::RNG` for information about the instance with that name.

Because the underlying C++ std library random number generators are not thread safe, `CX_RandomNumberGenerator` is not thread safe. If you want to use a `CX_RandomNumberGenerator` in a thread, that thread should have its own `CX_RandomNumberGenerator`. You may seed the thread's `CX_RandomNumberGenerator` with `CX::Instances::RNG`.

### 13.26.2 Constructor & Destructor Documentation

#### 13.26.2.1 `CX_RandomNumberGenerator::CX_RandomNumberGenerator ( void )`

Constructs an instance of a `CX_RandomNumberGenerator`. Seeds the `CX_RandomNumberGenerator` using a `std::random_device`.

By the C++11 specification, `std::random_device` is supposed to be a non-deterministic (hardware) RNG. However, from [http://en.cppreference.com/w/cpp/numeric/random/random\\_device](http://en.cppreference.com/w/cpp/numeric/random/random_device): "Note that `std::random_device` may be implemented in terms of a pseudo-random number engine if a non-deterministic source (e.g. a hardware device) is not available to the implementation." According to a Stack Overflow comment, Microsoft's implementation of `std::random_device` is based on a ton of stuff, which should result in a fairly ran-



dom result to be used as a seed for our Mersenne Twister. See the comment: <http://stackoverflow.com/questions/9549357/the-implementation-of-random-device-in-vs2010/9575747#9575747> Although this data should have high entropy, it is not a hardware RNG. The random\_device is only used to seed the Mersenne Twister, so as long as the initial value is random enough, it should be fine.

### 13.26.3 Member Function Documentation

#### 13.26.3.1 std::mt19937\_64 & CX\_RandomNumberGenerator::getGenerator ( void )

This function returns a reference to the standard library PRNG used by the [CX\\_RandomNumberGenerator](#). This can be used for various things, including sampling from some of the other distributions provided by the standard library: <http://en.cppreference.com/w/cpp/numeric/random>

```
std::poisson_distribution<int> pois(4);
int deviate = pois(RNG.getGenerator());
```

#### 13.26.3.2 CX\_RandomInt\_t CX\_RandomNumberGenerator::getMaximumRandomInt ( void )

Get the maximum possible value that can be returned by [randomInt\(\)](#).

##### Returns

The maximum value.

#### 13.26.3.3 CX\_RandomInt\_t CX\_RandomNumberGenerator::getMinimumRandomInt ( void )

Get the minimum value that can be returned by [randomInt\(\)](#).

##### Returns

The minimum value.

#### 13.26.3.4 unsigned long CX\_RandomNumberGenerator::getSeed ( void )

Get the seed used to seed the random number generator.

##### Returns

The seed. May have been set by the user with [setSeed\(\)](#) or during construction of the [CX\\_RandomNumberGenerator](#).

#### 13.26.3.5 double CX\_RandomNumberGenerator::randomDouble ( double lowerBound\_closed, double upperBound\_open )

Samples a realization from a uniform distribution with the range [lowerBound\_closed, upperBound\_open).

##### Parameters

<i>lowerBound_closed</i>	The lower bound of the distribution. This bound is closed, meaning that you can observe this value.
--------------------------	---

<i>upperBound_ - open</i>	The upper bound of the distribution. This bound is open, meaning that you cannot observe this value.
---------------------------	--

**Returns**

The realization.

**13.26.3.6 CX\_RandomInt\_t CX\_RandomNumberGenerator::randomInt ( void )**

Get a random integer in the range [getMinimumRandomInt\(\)](#), [getMaximumRandomInt\(\)](#), inclusive.

**Returns**

The int.

**13.26.3.7 CX\_RandomInt\_t CX\_RandomNumberGenerator::randomInt ( CX\_RandomInt\_t min, CX\_RandomInt\_t max )**

This function returns an integer from the range [rangeLower, rangeUpper]. The minimum and maximum values for the int returned from this function are given by [getMinimumRandomInt\(\)](#) and [getMaximumRandomInt\(\)](#).

If rangeLower > rangeUpper, the lower and upper ranges are swapped. If rangeLower == rangeUpper, it returns rangeLower.

**13.26.3.8 template<typename T > T CX::CX\_RandomNumberGenerator::sample ( const std::vector< T > & values )**

Returns a single value sampled randomly from values.

**Returns**

The sampled value.

**Note**

If values.size() == 0, an error will be logged and T() will be returned.

**13.26.3.9 template<typename T > std::vector< T > CX::CX\_RandomNumberGenerator::sample ( unsigned int count, const std::vector< T > & source, bool withReplacement )**

Returns a vector of count values drawn randomly from source, with or without replacement. The returned values are in a random order.

**Parameters**

<i>count</i>	The number of samples to draw.
<i>source</i>	A vector to be sampled from.
<i>withReplacement</i>	Sample with or without replacement.

**Returns**

A vector of the sampled values.

**Note**

If (count > source.size() && withReplacement == false), an empty vector is returned.

13.26.3.10 `std::vector< int > CX_RandomNumberGenerator::sample ( unsigned int count, int lowerBound, int upperBound, bool withReplacement )`

Returns a vector of count integers drawn randomly from the range [lowerBound, upperBound] with or without replacement.

## Parameters

<i>count</i>	The number of samples to draw.
<i>lowerBound</i>	The lower bound of the range to sample from. It is possible to sample this value.
<i>upperBound</i>	The upper bound of the range to sample from. It is possible to sample this value.
<i>withReplacement</i>	Sample with or without replacement.

## Returns

A vector of the samples.

**13.26.3.11** `std::vector< unsigned int > CX_RandomNumberGenerator::sampleBinomialRealizations ( unsigned int count, unsigned int trials, double probSuccess )`

Samples count realizations from a binomial distribution with the given number of trials and probability of success on each trial.

## Parameters

<i>count</i>	The number of deviates to generate.
<i>trials</i>	The number of trials. Must be a non-negative integer.
<i>probSuccess</i>	The probability of a success on a given trial, where a success is the value 1.

## Returns

A vector of the realizations.

**13.26.3.12** `template<typename T> std::vector< T > CX::CX_RandomNumberGenerator::sampleBlocks ( const std::vector< T > & values, unsigned int blocksToSample )`

This function helps with the case where a set of V values must be sampled randomly with the constraint that each block of V samples should have each value in the set. For example, if you want to present a number of trials in four different conditions, where the conditions are intermixed, but you want to observe all four trial types in every block of four trials, you would use this function.

## Parameters

<i>values</i>	The set of values to sample from.
<i>blocksToSample</i>	The number of blocks to sample.

## Returns

A vector with `values.size() * blocksToSample` elements.

**13.26.3.13** `template<typename T> T CX::CX_RandomNumberGenerator::sampleExclusive ( const std::vector< T > & values, const T & exclude )`

Sample a random value from a vector, without the possibility of getting the excluded value.

## Parameters

<i>values</i>	The vectors of values to sample from.
<i>exclude</i>	The value to exclude from sampling.

**Returns**

The sampled value.

**Note**

If all of the values are excluded, an error will be logged and T() will be returned.

**13.26.3.14** `template<typename T> T CX::CX_RandomNumberGenerator::sampleExclusive ( const std::vector< T > & values, const std::vector< T > & exclude )`

Sample a random value from a vector without the possibility of getting any of the excluded values.

**Parameters**

<i>values</i>	The vector of values to sample from.
<i>exclude</i>	The vector of values to exclude from sampling.

**Returns**

The sampled value.

**Note**

If all of the values are excluded, an error will be logged and T() will be returned.

**13.26.3.15** `template<typename T> std::vector< T > CX::CX_RandomNumberGenerator::sampleExclusive ( unsigned int count, const std::vector< T > & values, const T & exclude, bool withReplacement )`

Sample some number of random values, with or without replacement, from a vector without the possibility of getting the excluded value.

**Parameters**

<i>count</i>	The number of values to sample.
<i>values</i>	The vector of values to sample from.
<i>exclude</i>	The vector of values to exclude from sampling.
<i>withReplacement</i>	If true, values will be sampled with replacement (i.e. the same value can be sampled more than once).

**Returns**

The sampled values, of equal number to count, unless an error has occurred.

**Note**

If all of the values are excluded, an error will be logged and an empty vector will be returned.

**13.26.3.16** `template<typename T> std::vector< T > CX::CX_RandomNumberGenerator::sampleExclusive ( unsigned int count, const std::vector< T > & values, const std::vector< T > & exclude, bool withReplacement )`

Sample some number of random values, with or without replacement, from a vector without the possibility of getting any of the excluded values.

**Parameters**

<i>count</i>	The number of values to sample.
<i>values</i>	The vector of values to sample from.
<i>exclude</i>	The vector of values to exclude from sampling.
<i>withReplacement</i>	If true, values will be sampled with replacement (i.e. the same value can be sampled more than once).

**Returns**

The sampled values, of equal number to count, unless an error has occurred.

**Note**

If all of the values are excluded, an error will be logged and an empty vector will be returned.

**13.26.3.17** `template<typename T> std::vector< T> CX::CX_RandomNumberGenerator::sampleNormalRealizations ( unsigned int count, T mean, T standardDeviation )`

Samples count realizations from a normal distribution with the given mean and standard deviation.

**Template Parameters**

<i>T</i>	The precision with which to sample (should be <code>float</code> or <code>double</code> most of the time).
----------	--

**Parameters**

<i>count</i>	The number of deviates to generate.
<i>mean</i>	The mean of the distribution.
<i>standard-Deviation</i>	The standard deviation of the distribution.

**Returns**

A vector of the realizations.

**13.26.3.18** `template<typename stdDist> std::vector< typename stdDist::result_type> CX::CX_RandomNumberGenerator::sampleRealizations ( unsigned int count, stdDist dist )`

Draws count samples from a distribution dist that is provided by the user.

**Parameters**

<i>count</i>	The number of samples to take.
<i>dist</i>	A configured instance of a distribution class that has <code>operator()(Generator&amp; g)</code> , where Generator is a random number generator that has <code>operator()</code> that returns a random value. Basically, just look at this page: <a href="http://en.cppreference.com/w/cpp/numeric/random">http://en.cppreference.com/w/cpp/numeric/random</a> and pick one of the random number distributions.

**Returns**

A vector of `stdDist::result_type`, where `stdDist::result_type` is the type of data that is returned by the distribution (e.g. `int`, `double`, etc.). You can usually set this when creating the distribution object.

```
//Take 100 samples from a poisson distribution with lamda (mean result value) of 4.2.
//stdDist::result_type is unsigned int in this example.
vector<unsigned int> rpois = RNG.sampleFrom(100, std::poisson_distribution<unsigned int>(4.2));
```

13.26.3.19 `template<typename T> std::vector< T> CX::CX_RandomNumberGenerator::sampleUniformRealizations ( unsigned int count, T lowerBound_closed, T upperBound_open )`

Samples count deviates from a uniform distribution with the range [lowerBound\_closed, upperBound\_open).

## Template Parameters

<i>T</i>	The precision with which to sample (should be <code>float</code> or <code>double</code> most of the time).
----------	--

## Parameters

<i>count</i>	The number of deviates to generate.
<i>lowerBound_ - closed</i>	The lower bound of the distribution. This bound is closed, meaning that you can observe deviates with this value.
<i>upperBound_ - open</i>	The upper bound of the distribution. This bound is open, meaning that you cannot observe deviates with this value.

## Returns

A vector of the realizations.

13.26.3.20 `void CX_RandomNumberGenerator::setSeed ( unsigned long seed )`

Set the seed for the random number generator. You can retrieve the seed with [getSeed\(\)](#).

## Parameters

<i>seed</i>	The new seed.
-------------	---------------

13.26.3.21 `void CX_RandomNumberGenerator::setSeed ( const std::string & seedString )`

This function provides a method of setting the seed using an arbitrary string (e.g. date-time and participant number) as the seed. A CRC32 checksum is used to convert the string into an unsigned long, which is then used as the seed for the [CX\\_RandomNumberGenerator](#). You can retrieve the seed with [getSeed\(\)](#).

## Parameters

<i>seedString</i>	The string from which the new seed will be calculated.
-------------------	--

13.26.3.22 `template<typename T> void CX::CX_RandomNumberGenerator::shuffleVector ( std::vector< T > * v )`

Randomizes the order of the given vector.

## Parameters

<i>v</i>	A pointer to the vector to be shuffled.
----------	---

13.26.3.23 `template<typename T> std::vector< T > CX::CX_RandomNumberGenerator::shuffleVector ( std::vector< T > v )`

Makes a copy of the given vector, randomizes the order of its elements, and returns the shuffled copy.

## Parameters

<i>v</i>	The vector to be operated on.
----------	-------------------------------

## Returns

A shuffled copy of *v*.

The documentation for this class was generated from the following files:

- `CX_RandomNumberGenerator.h`
- `CX_RandomNumberGenerator.cpp`



## 13.27 CX::Util::CX\_SegmentProfiler Class Reference

```
#include <CX_TimeUtilities.h>
```

### Public Member Functions

- [CX\\_SegmentProfiler](#) ([CX\\_Clock](#) \*clock, unsigned int logSamples=0)
- void [setup](#) ([CX\\_Clock](#) \*clock, unsigned int logSamples=0)
- void [t1](#) (void)
- void [t2](#) (void)
- unsigned int [collectedSamples](#) (void)
- void [restart](#) (void)
- std::string [getStatString](#) (void)
- [CX\\_Millis](#) [mean](#) (void)  
*Get the mean of the stored segment durations.*
- [CX\\_Millis](#) [min](#) (void)  
*Get the shortest of the stored segment durations.*
- [CX\\_Millis](#) [max](#) (void)  
*Get the longest of the stored segment durations.*
- [CX\\_Millis](#) [stdDev](#) (void)  
*Get the standard deviation of the stored segment durations.*

### 13.27.1 Detailed Description

This class is used for profiling small segments of code embedded within other code.

```
//During setup
CX::Util::CX_SegmentProfiler profiler(&
    CX::Instances::Clock);

//In main code somewhere you have a process that is repeated some number
//of times that has the code of interest embedded in it.
for (int i = 0; i < 100; i++) {
    //Some code you aren't interested in profiling...

    profiler.t1();
    //Code you are interested in profiling.
    profiler.t2();

    //Other code you aren't interested in profiling...
}

//Once the process has been performed some number of times,
//check out the statistics for the code segment that was profiled.
std::cout << profiler.getStatString() << std::endl;
```

### 13.27.2 Constructor & Destructor Documentation

#### 13.27.2.1 CX::Util::CX\_SegmentProfiler::CX\_SegmentProfiler ( [CX\\_Clock](#) \* clock, unsigned int logSamples = 0 )

Set up the [CX\\_SegmentProfiler](#) with the selected clock source and the number of samples to log between each automatic logging of results.

## Parameters

<i>clock</i>	The instance of <a href="#">CX_Clock</a> to use.
<i>logSamples</i>	If this is not 0, then every <code>logSamples</code> samples, a string containing information about the last <code>logSamples</code> samples will be logged and then those samples will be cleared.

## 13.27.3 Member Function Documentation

13.27.3.1 `unsigned int CX::Util::CX_SegmentProfiler::collectedSamples ( void )`

## Returns

The number of collected samples.

13.27.3.2 `std::string CX::Util::CX_SegmentProfiler::getStatString ( void )`

Get a string summarizing some basic descriptive statistics for the currently stored data.

## Returns

A string containing the minimum, mean, maximum, and standard deviation of the stored data.

13.27.3.3 `void CX::Util::CX_SegmentProfiler::restart ( void )`

Restart data collection. All collected samples are cleared.

13.27.3.4 `void CX::Util::CX_SegmentProfiler::setup ( CX_Clock * clock, unsigned int logSamples = 0 )`

Set up the [CX\\_SegmentProfiler](#) with the selected clock source and the number of samples to log between each automatic logging of results.

## Parameters

<i>clock</i>	The instance of <a href="#">CX_Clock</a> to use.
<i>logSamples</i>	If this is not 0, then every <code>logSamples</code> samples, a string containing information about the last <code>logSamples</code> samples will be logged.

13.27.3.5 `void CX::Util::CX_SegmentProfiler::t1 ( void )`

This function takes a timestamp at the current time and will be compared with the timestamp taken with [t2\(\)](#).

13.27.3.6 `void CX::Util::CX_SegmentProfiler::t2 ( void )`

This function stores the difference between the current time and the time captured with [t1\(\)](#). If enough samples have been collected, equal to the value of `logSamples` during [setup\(\)](#), a summary statistics string will be automatically logged.

The documentation for this class was generated from the following files:

- `CX_TimeUtilities.h`
- `CX_TimeUtilities.cpp`

13.28 `CX::CX_SlidePresenter` Class Reference

```
#include <CX_SlidePresenter.h>
```

## Classes

- struct [Configuration](#)
- struct [FinalSlideFunctionArgs](#)
- struct [PresentationErrorInfo](#)
- struct [Slide](#)
- struct [SlideTimingInfo](#)

## Public Types

- enum [ErrorMode](#) { **PROPAGATE\_DELAYS** }
- enum [SwappingMode](#) { **SINGLE\_CORE\_BLOCKING\_SWAPS**, **MULTI\_CORE** }

## Public Member Functions

- bool [setup](#) (CX\_Display \*display)
- bool [setup](#) (const CX\_SlidePresenter::Configuration &config)
- virtual void [update](#) (void)
- void [appendSlide](#) (CX\_SlidePresenter::Slide slide)
- void [appendSlideFunction](#) (std::function< void(void)> drawingFunction, CX\_Millis slideDuration, std::string slideName="")
- void [beginDrawingNextSlide](#) (CX\_Millis slideDuration, std::string slideName="")
- void [endDrawingCurrentSlide](#) (void)
- bool [startSlidePresentation](#) (void)
- void [stopSlidePresentation](#) (void)
- *Stops slide presentation.*
- bool [presentSlides](#) (void)
- bool [isPresentingSlides](#) (void) const
- *Returns true if slide presentation is in progress, even if the first slide has not yet been presented.*
- void [clearSlides](#) (void)
- std::vector< CX\_SlidePresenter::Slide > & [getSlides](#) (void)
- CX\_SlidePresenter::Slide & [getSlideByName](#) (std::string name)
- std::vector< CX\_Millis > [getActualPresentationDurations](#) (void)
- std::vector< unsigned int > [getActualFrameCounts](#) (void)
- CX\_SlidePresenter::PresentationErrorInfo [checkForPresentationErrors](#) (void) const
- std::string [printLastPresentationInformation](#) (void) const

## 13.28.1 Detailed Description

This class is a very useful abstraction that presents slides (typically a full display) of visual stimuli for fixed durations. See the `changeDetection` and `nBack` examples for the usage of this class.

A brief example:

```
CX_SlidePresenter slidePresenter;
slidePresenter.setup(&Display);

slidePresenter.beginDrawingNextSlide(2000 * 1000, "circle");
ofBackground(50);
ofSetColor(ofColor::red);
ofCircle(Display.getCenterOfDisplay(), 40);
```

```

slidePresenter.beginDrawingNextSlide(1000 * 1000, "rectangle");
ofBackground(50);
ofSetColor(ofColor::green);
ofRect(Display.getCenterOfDisplay() - ofPoint(100, 100), 200, 200);

slidePresenter.beginDrawingNextSlide(1, "off");
ofBackground(50);
slidePresenter.endDrawingCurrentSlide();

slidePresenter.startSlidePresentation();

//Update the slide presenter while waiting for slide presentation to complete
while (slidePresenter.isPresentingSlides()) {
    slidePresenter.update(); //You must remember to call update() regularly while slides are being
        presented!
}

```

### 13.28.2 Member Enumeration Documentation

#### 13.28.2.1 enum CX::CX\_SlidePresenter::ErrorMode [strong]

The settings in this enum are related to what a [CX\\_SlidePresenter](#) does when it encounters a timing error. Timing errors are probably almost exclusively related to one slide being presented for too long.

The PROPAGATE\_DELAYS setting causes the slide presenter to handle these errors by moving the start time of all future stimuli back by the amount of extra time (or frames) used to the erroneous slide. This makes the durations of all future stimuli correct, so that there is only an error in the duration of one slide. If a slide's presentation start time is early, the intended start time is used (i.e. only delays, not early arrivals, are propagated).

Other alternatizes are being developed.

#### 13.28.2.2 enum CX::CX\_SlidePresenter::SwappingMode [strong]

The method used by the slide presenter to swap stimuli that have been drawn to the back buffer to the front buffer. MULTI\_CORE is the best method, but only really works properly if you have at least a 2 core CPU. It uses a secondary thread to constantly swap the front and back buffers, which allows each frame to be counted. This results in really good synchronization between the copies of data to the back buffer and the swaps of the front and back buffers. In the SINGLE\_CORE\_BLOCKING\_SWAPS mode, after a stimulus has been copied to the front buffer, the next stimulus is immediately drawn to the back buffer. After the correct amount of time minus [CX\\_SlidePresenter::Configuration::pre-SwapCPUHoggingDuration](#), the buffers are swapped. The main problem with this mode is that the buffer swapping in this mode [blocks](#) in the main thread while waiting for the swap.

### 13.28.3 Member Function Documentation

#### 13.28.3.1 void CX\_SlidePresenter::appendSlide ( CX\_SlidePresenter::Slide slide )

Add a fully configured slide to the end of the list of slides. The user code must configure several components of the slide:

- If the framebuffer will be used, the framebuffer must be allocated and drawn to.
- If the drawing function will be used, a valid function pointer must be given. A check is made that either the drawing function is set or the framebuffer is allocated and an error is logged if neither is configured.
- The intended duration must be set.
- The name may be set (optional).

## Parameters

<i>slide</i>	The slide to append.
--------------	----------------------

**13.28.3.2** void CX\_SlidePresenter::appendSlideFunction ( std::function< void(void)> *drawingFunction*, CX\_Millis *slideDuration*, std::string *slideName* = " " )

Appends a slide to the slide presenter that will call the given drawing function when it comes time to render the slide to the back buffer. This approach has the advantage over using framebuffers that it takes essentially zero time to append a function to the list of slides, whereas a framebuffer must be allocated, which takes time. Additionally, because framebuffers must be allocated, they use video memory, so if you are using a very large number of slides, you could potentially run out of video memory. Also, when it comes time to draw the slide to the back buffer, it may be faster to draw directly to the back buffer than to copy an FBO to the back buffer (although this depends on various factors).

## Parameters

<i>drawingFunction</i>	A pointer to a function that will draw the slide to the back buffer. The contents of the back buffer are not cleared before this function is called, so the function must clear the background to the desired color.
<i>slideDuration</i>	The amount of time to present the slide for. If this is less than or equal to 0, the slide will be ignored.
<i>slideName</i>	The name of the slide. This can be anything and is purely for the user to use to help identify the slide.

## Note

See [Framebuffers and Buffer Swapping](#) for more information about framebuffers.

One of the most tedious parts of using drawing functions is the fact that they can take no arguments. Here are two ways to get around that limitation using std::bind and function objects (functors):

```
#include "CX_EntryPoint.h"

CX_SlidePresenter SlidePresenter;

//This is the function we want to use to draw a stimulus, but it takes two
//arguments. It needs to take 0 arguments in order to be used by the CX_SlidePresenter.
void drawRectangle(ofRectangle r, ofColor col) {
    ofBackground(0);
    ofSetColor(col);
    ofRect(r);
}

//One option is to use a functor to shift around where the arguments to the function come from. With a
//functor, like rectFunctor, below, you can define an operator() that takes no arguments directly, but gets
//its
//data from members of the structure like position and color. Because 'rectFunctor' has operator(), it
//looks
//like a function and can be called like a function, so you can use instances of it as drawing functions.
struct rectFunctor {
    ofRectangle position;
    ofColor color;
    void operator() (void) {
        drawRectangle(position, color);
    }
};

void runExperiment(void) {

    SlidePresenter.setup(&Display);

    //Here we use the functor. We set up the values for position and color and then give the functor to
    appendSlideFunction.
    rectFunctor rf;
    rf.position = ofRectangle(100, 100, 50, 80);
    rf.color = ofColor(0, 255, 0);
}
```

```

SlidePresenter.appendSlideFunction(rf, 2000.0, "functor rect");

//The other method is to use std::bind to "bake in" values for the arguments of drawRectangle. We will
//set up the rectPos and rectColor values to bind to the arguments of drawRectangle.
ofRectangle rectPos(100, 50, 100, 30);
ofColor rectColor(255, 255, 0);

//With the call to std::bind, we bake in the values rectPos and rectColor to their respective
arguments,
//resulting in a function that takes 0 arguments, which we pass into appendSlideFunction().
SlidePresenter.appendSlideFunction(std::bind(drawRectangle, rectPos, rectColor), 2000.0, "bind rect");

SlidePresenter.startSlidePresentation();
while (SlidePresenter.isPresentingSlides()) {
    SlidePresenter.update();
}
}

```

### 13.28.3.3 void CX\_SlidePresenter::beginDrawingNextSlide ( CX\_Millis slideDuration, std::string slideName = " " )

Prepares the framebuffer of the next slide for drawing so that any drawing commands given between a call to [beginDrawingNextSlide\(\)](#) and [endDrawingCurrentSlide\(\)](#) will cause stimuli to be drawn to the framebuffer of the slide.

#### Parameters

<i>slideDuration</i>	The amount of time to present the slide for. If this is less than or equal to 0, the slide will be ignored.
<i>slideName</i>	The name of the slide. This can be anything and is purely for the user to use to help identify the slide.

```

CX_SlidePresenter sp; //Assume that this has been set up.

sp.beginDrawingNextSlide(2000, "circles");
ofBackground(50);
ofSetColor(255, 0, 0);
ofCirlce(100, 100, 30);
ofCircle(210, 50, 20);
sp.endDrawingCurrentSlide();

```

### 13.28.3.4 CX\_SlidePresenter::PresentationErrorInfo CX\_SlidePresenter::checkForPresentationErrors ( void ) const

Checks the timing data from the last presentation of slides for presentation errors. Currently it checks to see if the intended frame count matches the actual frame count of each slide, which indicates if the duration was correct. It also checks to make sure that the framebuffer was copied to the back buffer before the onset of the slide. If not, vertical tearing might have occurred when the back buffer, containing a partially copied slide, was swapped in.

#### Returns

A struct with information about the errors that occurred on the last presentation of slides.

#### Note

If [clearSlides\(\)](#) has been called since the end of the presentation, this does nothing as its data has been cleared. If this function is called during slide presentation, the returned struct will have the presentationErrorsSuccessfullyChecked member set to false and an error will be logged.

### 13.28.3.5 void CX\_SlidePresenter::clearSlides ( void )

Clears (deletes) all of the slides contained in the slide presenter and stops presentation, if it was in progress.

**13.28.3.6 void CX\_SlidePresenter::endDrawingCurrentSlide ( void )**

Ends drawing to the framebuffer of the slide that is currently being drawn to. See [beginDrawingNextSlide\(\)](#).

**13.28.3.7 std::vector< unsigned int > CX\_SlidePresenter::getActualFrameCounts ( void )**

Gets a vector containing the number of frames that each of the slides from the last presentation of slides was presented for. Note that these frame counts may be wrong. If [checkForPresentationErrors\(\)](#) not detect any errors, the frame counts are likely to be right, but there is no guarantee.

**Returns**

A vector containing the frame counts. The frame count corresponding to the first slide added to the slide presenter will be at index 0.

**Note**

The frame count of the last slide is meaningless. As far as the slide presenter is concerned, as soon as the last slide is put on the screen, it is done presenting the slides. Because the slide presenter is not responsible for removing the last slide from the screen, it has no idea about the duration of that slide.

**13.28.3.8 std::vector< CX\_Millis > CX\_SlidePresenter::getActualPresentationDurations ( void )**

Gets a vector containing the durations of the slides from the last presentation of slides. Note that these durations may be wrong. If [checkForPresentationErrors\(\)](#) does not detect any errors, the durations are likely to be right, but there is no guarantee.

**Returns**

A vector containing the durations. The duration corresponding to the first slide added to the slide presenter will be at index 0.

**Note**

The duration of the last slide is meaningless. As far as the slide presenter is concerned, as soon as the last slide is put on the screen, it is done presenting the slides. Because the slide presenter is not responsible for removing the last slide from the screen, it has no idea about the duration of that slide.

**13.28.3.9 CX\_SlidePresenter::Slide & CX\_SlidePresenter::getSlideByName ( std::string name )**

Gets a reference to the slide with the given name, if found. If the named slide is not found, a `std::out_of_range` exception is thrown and an error is logged (although you will never see the log message unless the exception is caught).

**Parameters**

<i>name</i>	The name of the slide to get.
-------------	-------------------------------

**Returns**

A reference to the named slide.

**Note**

Because the user supplies slide names, there is no guarantee that any given slide name will be unique. Because of this, this function simply returns a reference to the first slide for which the name matches.

### 13.28.3.10 `std::vector< CX_SlidePresenter::Slide > & CX_SlidePresenter::getSlides ( void )`

Get a reference to the vector of slides held by the slide presenter. If you modify any of the members of any of the slides, you do so at your own risk. This data is mostly useful in a read-only sort of way (when was that slide presented?).

#### Returns

A reference to the vector of slides.

### 13.28.3.11 `bool CX_SlidePresenter::presentSlides ( void )`

Performs a "standard" slide presentation in a single function call as a convenience. This function calls [startSlidePresentation\(\)](#) to begin the presentation and then calls [update\(\)](#) and `CX::Instances::Input.pollEvents()` continuously as long as [isPresentingSlides\(\)](#) returns true.

#### Returns

`true` if the slide presentation completed successfully or `false` if the slide presentation could not be started.

### 13.28.3.12 `std::string CX_SlidePresenter::printLastPresentationInformation ( void ) const`

This function prints a ton of data relating to the last presentation of slides. It prints the total number of errors and the types of the errors. For each slide, it prints the slide index and name, and various information about the slide presentation timing. All of the printed information can also be accessed programmatically by using [getSlides\(\)](#).

#### Returns

A string containing formatted presentation information.

### 13.28.3.13 `bool CX_SlidePresenter::setup ( CX_Display * display )`

Set up the slide presenter with the given [CX\\_Display](#) as the display.

#### Parameters

<i>display</i>	Pointer to the display to use.
----------------	--------------------------------

#### Returns

False if there was an error during setup, in which case a message will be logged.

### 13.28.3.14 `bool CX_SlidePresenter::setup ( const CX_SlidePresenter::Configuration & config )`

Set up the slide presenter using the given configuration.

#### Parameters

<i>config</i>	The configuration to use.
---------------	---------------------------

#### Returns

False if there was an error during setup, in which case a message will be logged.



**13.28.3.15 bool CX\_SlidePresenter::startSlidePresentation ( void )**

Start presenting the slides that are stored in the slide presenter. After this function is called, calls to [update\(\)](#) will advance the state of the slide presentation. If you do not call [update\(\)](#), nothing will be presented.

**Returns**

False if an error was encountered while starting presentation, in which case messages will be logged, true otherwise.

**13.28.3.16 void CX\_SlidePresenter::update ( void ) [virtual]**

Updates the state of the slide presenter. If the slide presenter is presenting stimuli, [update\(\)](#) must be called very regularly (at least once per millisecond) in order for the slide presenter to function. If slide presentation is stopped, you do not need to call [update\(\)](#)

The documentation for this class was generated from the following files:

- CX\_SlidePresenter.h
- CX\_SlidePresenter.cpp

**13.29 CX::CX\_SoundBuffer Class Reference**

```
#include <CX_SoundBuffer.h>
```

**Public Member Functions**

- bool [loadFile](#) (std::string fileName)
- bool [addSound](#) (std::string fileName, [CX\\_Millis](#) timeOffset)
- bool [addSound](#) ([CX\\_SoundBuffer](#) so, [CX\\_Millis](#) timeOffset)
- bool [setFromVector](#) (const std::vector< float > &data, int channels, float sampleRate)
- void [clear](#) (void)
- bool [isReadyToPlay](#) (void)
- bool [isLoadedSuccessfully](#) (void)
- bool [applyGain](#) (float gain, int channel=-1)
- bool [multiplyAmplitudeBy](#) (float amount, int channel=-1)
- void [normalize](#) (float amount=1.0)
- float [getPositivePeak](#) (void)
- float [getNegativePeak](#) (void)
- void [setLength](#) ([CX\\_Millis](#) length)
- [CX\\_Millis](#) [getLength](#) (void)
- void [stripLeadingSilence](#) (float tolerance)
- void [addSilence](#) ([CX\\_Millis](#) duration, bool atBeginning)
- void [deleteAmount](#) ([CX\\_Millis](#) duration, bool fromBeginning)
- void [reverse](#) (void)
- void [multiplySpeed](#) (float speedMultiplier)
- void [resample](#) (float newSampleRate)
- float [getSampleRate](#) (void)

*Returns the sample rate of the sound data stored in this [CX\\_SoundBuffer](#).*

- bool [setChannelCount](#) (int channels)
- int [getChannelCount](#) (void)

Returns the number of channels in the sound data stored in this [CX\\_SoundBuffer](#).

- uint64\_t [getTotalSampleCount](#) (void)
- uint64\_t [getSampleFrameCount](#) (void)
- std::vector< float > & [getRawDataReference](#) (void)
- bool [writeToFile](#) (std::string path)

#### Public Attributes

- std::string [name](#)

This stores the name of the file from which data was read, if any. It can be set by the user with no side effects.

#### 13.29.1 Detailed Description

This class is a container for a sound. It can load sound files, manipulate the contents of the sound data, add other sounds to an existing sound at specified offsets.

In order to play a [CX\\_SoundBuffer](#), you use a [CX::CX\\_SoundBufferPlayer](#). See the [soundBuffer](#) tutorial for an introduction on how to use this class along with a [CX\\_SoundBufferPlayer](#).

To record from a microphone to a [CX\\_SoundBuffer](#), you use a [CX::CX\\_SoundBufferRecorder](#).

#### Note

Nearly all functions of this class should be considered [Blocking Code](#). Many of the operations can take quite a while to complete because they are performed on a potentially large vector of sound samples.

#### 13.29.2 Member Function Documentation

##### 13.29.2.1 void CX\_SoundBuffer::addSilence ( CX\_Millis duration, bool atBeginning )

Adds the specified amount of silence to the [CX\\_SoundBuffer](#) at either the beginning or end.

##### Parameters

<i>duration</i>	Duration of added silence in microseconds. Dependent on the sample rate of the sound. If the sample rate changes, so does the duration of silence.
<i>atBeginning</i>	If true, silence is added at the beginning of the <a href="#">CX_SoundBuffer</a> . If false, the silence is added at the end.

##### 13.29.2.2 bool CX\_SoundBuffer::addSound ( std::string fileName, CX\_Millis timeOffset )

Uses [loadFile\(string\)](#) and [addSound\(CX\\_SoundBuffer, uint64\\_t\)](#) to add the given file to the current [CX\\_SoundBuffer](#) at the given time offset (in microseconds). See those functions for more information.

##### Parameters

<i>fileName</i>	Name of the sound file to load.
<i>timeOffset</i>	Time at which to add the new sound.

#### Returns

Returns true if the new sound was added successfully, false otherwise.

**13.29.2.3 bool CX\_SoundBuffer::addSound ( CX\_SoundBuffer *nsb*, CX\_Millis *timeOffset* )**

Adds the sound data in *nsb* at the time offset. If the sample rates of the sounds differ, *nsb* will be resampled to the sample rate of this [CX\\_SoundBuffer](#). If the number of channels of *nsb* does not equal the number of channels of this, an attempt will be made to set the number of channels of *nsb* equal to the number of channels of this [CX\\_SoundBuffer](#). The data from *nsb* and this [CX\\_SoundBuffer](#) are merged by adding the amplitudes of the sounds. The result of the addition is clamped between -1 and 1.

**Parameters**

<i>nsb</i>	A <a href="#">CX_SoundBuffer</a> . Must be successfully loaded.
<i>timeOffset</i>	Time at which to add the new sound data in microseconds. Dependent on sample rate.

**Returns**

True if *nsb* was successfully added to this [CX\\_SoundBuffer](#), false otherwise.

**13.29.2.4 bool CX\_SoundBuffer::applyGain ( float *decibels*, int *channel* = -1 )**

Apply gain to the channel in terms of decibels.

**Parameters**

<i>decibels</i>	Gain to apply. 0 does nothing. Positive values increase volume, negative values decrease volume. Negative infinity is essentially mute, although see <a href="#">multiplyAmplitudeBy()</a> for a more obvious way to do that same operation.
<i>channel</i>	The channel that the gain should be applied to. If channel is less than 0, the gain is applied to all channels.

**13.29.2.5 void CX\_SoundBuffer::clear ( void )**

Clears all data stored in the sound buffer and returns it to an uninitialized state.

**13.29.2.6 void CX\_SoundBuffer::deleteAmount ( CX\_Millis *duration*, bool *fromBeginning* )**

Deletes the specified amount of sound from the [CX\\_SoundBuffer](#) from either the beginning or end.

**Parameters**

<i>duration</i>	Duration of removed sound in microseconds. If this is greater than the duration of the sound, the whole sound is deleted.
<i>fromBeginning</i>	If true, sound is deleted from the beginning of the <a href="#">CX_SoundBuffer</a> 's buffer. If false, the sound is deleted from the end, toward the beginning.

**13.29.2.7 CX\_Millis CX\_SoundBuffer::getLength ( void )**

Gets the length, in time, of the data stored in the sound buffer. This depends on the sample rate of the sound.

**Returns**

The length.

**13.29.2.8 float CX\_SoundBuffer::getNegativePeak ( void )**

Finds the minimum amplitude in the sound buffer.

**Returns**

The minimum amplitude.

**Note**

Amplitudes are between -1 and 1, inclusive.

**13.29.2.9 float CX\_SoundBuffer::getPositivePeak ( void )**

Finds the maximum amplitude in the sound buffer.

**Returns**

The maximum amplitude.

**Note**

Amplitudes are between -1 and 1, inclusive.

**13.29.2.10 std::vector<float>& CX::CX\_SoundBuffer::getRawDataReference ( void ) [inline]**

This function returns a reference to the raw data underlying the [CX\\_SoundBuffer](#).

**Returns**

A reference to the data. Modify at your own risk!

**13.29.2.11 uint64\_t CX::CX\_SoundBuffer::getSampleFrameCount ( void ) [inline]**

This function returns the number of sample frames in the sound data held by the [CX\\_SoundBuffer](#), which is equal to the total number of samples divided by the number of channels.

**13.29.2.12 uint64\_t CX::CX\_SoundBuffer::getTotalSampleCount ( void ) [inline]**

This function returns the total number of samples in the sound data held by the [CX\\_SoundBuffer](#), which is equal to the number of sample frames times the number of channels.

**13.29.2.13 bool CX::CX\_SoundBuffer::isLoadedSuccessfully ( void ) [inline]**

Checks to see if sound data has been successfully loaded into this [CX\\_SoundBuffer](#) from a file.

**13.29.2.14 bool CX\_SoundBuffer::isReadyToPlay ( void )**

Checks to see if the [CX\\_SoundBuffer](#) is ready to play. It basically just checks if there is sound data available and that the number of channels is set to a sane value.

**13.29.2.15 bool CX\_SoundBuffer::loadFile ( std::string fileName )**

Loads a sound file with the given file name into the [CX\\_SoundBuffer](#). Any pre-existing data in the [CX\\_SoundBuffer](#) is deleted. Some sound file types are supported. Others are not. In the limited testing, mp3 and wav files seem to work well. If the file cannot be loaded, descriptive error messages will be logged.

## Parameters

<i>fileName</i>	Name of the sound file to load.
-----------------	---------------------------------

## Returns

True if the sound given in the fileName was loaded successfully, false otherwise.

#### 13.29.2.16 bool CX\_SoundBuffer::multiplyAmplitudeBy ( float *amount*, int *channel* = -1 )

Apply gain to the sound. The original value is simply multiplied by the amount and then clamped to be within [-1, 1].

## Parameters

<i>amount</i>	The gain that should be applied. A value of 0 mutes the channel. 1 does nothing. 2 doubles the amplitude. -1 inverts the waveform.
<i>channel</i>	The channel that the given multiplier should be applied to. If channel is less than 0, the amplitude multiplier is applied to all channels.

#### 13.29.2.17 void CX\_SoundBuffer::multiplySpeed ( float *speedMultiplier* )

This function changes the speed of the sound by some multiple.

## Parameters

<i>speedMultiplier</i>	Amount to multiply the speed by. Must be greater than 0.
------------------------	--

## Note

If you would like to use a negative value to reverse the direction of playback, see [reverse\(\)](#).

#### 13.29.2.18 void CX\_SoundBuffer::normalize ( float *amount* = 1.0 )

Normalizes the contents of the sound buffer.

## Parameters

<i>amount</i>	Must be in the interval [0,1]. If 1, normalize will normalize in the standard way: The peak with the greatest magnitude will be set to +/-1 and everything else will be scaled relative to the peak. If amount is less than 1, the greatest peak will be set to that value.
---------------	---

#### 13.29.2.19 void CX\_SoundBuffer::resample ( float *newSampleRate* )

Resamples the audio data stored in the [CX\\_SoundBuffer](#) by linear interpolation. Linear interpolation is not the ideal way to resample audio data; some audio fidelity is lost, more so than with other resampling techniques. It is, however, very fast compared to higher-quality methods both in terms of run time and programming time. It has acceptable results, at least when the new sample rate is similar to the old sample rate.

## Parameters

<i>newSampleRate</i>	The requested sample rate.
----------------------	----------------------------

#### 13.29.2.20 void CX\_SoundBuffer::reverse ( void )

This function reverses the sound data stored in the [CX\\_SoundBuffer](#) so that if it is played, it will play in reverse.

### 13.29.2.21 `bool CX_SoundBuffer::setChannelCount ( int newChannelCount )`

Sets the number of channels of the sound. Depending on the old number of channels (N) and the new number of channels (M), the conversion is performed in different ways. If N == M, nothing happens. If N == 1, each of the M new channels is set equal to the value of the single old channel. If M == 1, the new channel is set equal to the arithmetic average of the N old channels. If (N != 1 && M != 1 && M > N), the first N channels are preserved unchanged and the M - N new channels are set to the arithmetic average of the N old channels. Any other combination of M and N is an error condition.

#### Parameters

<i>newChannelCount</i>	The number of channels the <code>CX_SoundBuffer</code> will have after the conversion.
------------------------	--

#### Returns

True if the conversion was successful, false if the attempted conversion is unsupported.

### 13.29.2.22 `bool CX_SoundBuffer::setFromVector ( const std::vector< float > & data, int channels, float sampleRate )`

Set the contents of the sound buffer from a vector of float data.

#### Parameters

<i>data</i>	A vector of sound samples. These values should go from -1 to 1. This requirement is not checked for. If there is more than once channel of data, the data must be interleaved. This means that if, for example, there are two channels, the ordering of the samples is 12121212... where 1 represents a sample for channel 1 and 2 represents a sample for channel 2. This requirement is not checked for. The number of samples in this vector must be evenly divisible by the number of channels set with the <code>channels</code> argument.
<i>channels</i>	The number of channels worth of data that is stored in <i>data</i> .
<i>sampleRate</i>	The sample rate of the samples. If <i>data</i> contains, for example, a sine wave, that wave was sampled at some rate (e.g. 48000 samples per second of waveform). <i>sampleRate</i> should be that rate. return True in all cases. No checking is done on any of the arguments.

### 13.29.2.23 `void CX_SoundBuffer::setLength ( CX_Millis length )`

Set the length of the sound to the specified length in microseconds. If the new length is longer than the old length, the new data is zeroed (i.e. set to silence).

### 13.29.2.24 `void CX_SoundBuffer::stripLeadingSilence ( float tolerance )`

Removes leading "silence" from the sound, where silence is defined by the given tolerance. It is unlikely that the beginning of a sound, even if perceived as silent relative to the rest of the sound, has an amplitude of 0. Therefore, a tolerance of 0 is unlikely to prove useful. Using `getPositivePeak()` and/or `getNegativePeak()` can help to give a reference amplitude of which some small fraction is perceived as "silent".

#### Parameters

<i>tolerance</i>	All sound data up to and including the first instance of a sample with an amplitude with an absolute value greater than or equal to tolerance is removed from the sound.
------------------	--

### 13.29.2.25 `bool CX_SoundBuffer::writeToFile ( std::string filename )`

Writes the contents of the sound buffer to a file with the given file name. The data will be encoded as 16-bit PCM. The sample rate is determined by the sample rate of the sound buffer.

## Parameters

<i>filename</i>	The name of the file to save the sound data to. <i>filename</i> should have a .wav extension. If it does not, ".wav" will be appended to the file name and a warning will be logged.
-----------------	--

## Returns

False if there was an error while opening the file. If so, an error will be logged.

The documentation for this class was generated from the following files:

- CX\_SoundBuffer.h
- CX\_SoundBuffer.cpp

## 13.30 CX::CX\_SoundBufferPlayer Class Reference

```
#include <CX_SoundBufferPlayer.h>
```

## Public Types

- typedef  
[CX\\_SoundStream::Configuration](#) Configuration  
*This is typedef'ed to [CX::CX\\_SoundStream::Configuration](#).*

## Public Member Functions

- bool [setup](#) ([Configuration](#) config)
- bool [setup](#) ([CX\\_SoundStream](#) \*ss)
- bool [play](#) (void)
- bool [startPlayingAt](#) ([CX\\_Millis](#) experimentTime, [CX\\_Millis](#) offset)
- bool [stop](#) (void)
- bool [isPlaying](#) (void)  
*Check if the sound is currently playing.*
- bool [isQueuedToStart](#) (void)  
*Check if the sound is queued to play.*
- [Configuration](#) [getConfiguration](#) (void)
- bool [setSoundBuffer](#) ([CX\\_SoundBuffer](#) \*sound)
- [CX\\_SoundBuffer](#) \* [getSoundBuffer](#) (void)
- [CX\\_SoundStream](#) \* [getSoundStream](#) (void)
- void [setTime](#) ([CX\\_Millis](#) time)

## 13.30.1 Detailed Description

This class is used for playing CX\_SoundBuffers. See the soundBuffer tutorial for an example of how to use this class.

## 13.30.2 Member Function Documentation

## 13.30.2.1 CX\_SoundBufferPlayer::Configuration CX::CX\_SoundBufferPlayer::getConfiguration ( void )

Returns the configuration used for this [CX\\_SoundBufferPlayer](#).

### 13.30.2.2 `CX_SoundBuffer * CX::CX_SoundBufferPlayer::getSoundBuffer ( void )`

This function provides direct access to the `CX_SoundBuffer` that is in use by the `CX_SoundBufferPlayer`.

### 13.30.2.3 `CX_SoundStream* CX::CX_SoundBufferPlayer::getSoundStream ( void ) [inline]`

This function provides direct access to the `CX_SoundStream` used by the `CX_SoundBufferPlayer`.

### 13.30.2.4 `bool CX::CX_SoundBufferPlayer::play ( void )`

Attempts to start playing the current `CX_SoundBuffer` associated with the player.

#### Returns

`true` if the sound buffer associated with the player isReadyToPlay(), `false` otherwise.

### 13.30.2.5 `bool CX::CX_SoundBufferPlayer::setSoundBuffer ( CX_SoundBuffer * sound )`

This function is potentially blocking because the sample rate and number of channels of sound are changed to those of the currently open stream if they do not already match (see [Blocking Code](#)).

#### Parameters

<i>sound</i>	A pointer to a <code>CX_SoundBuffer</code> that will be set as the current sound for the <code>CX_SoundBufferPlayer</code> . There are a variety of reasons why the sound could fail to be set as the current sound for the player. If sound was not loaded successfully, this function call fails and an error is logged. If it is not possible to convert the number of channels of sound to the number of channels that the <code>CX_SoundBufferPlayer</code> is configured to use, this function call fails and an error is logged.
--------------	---

This function call is not blocking if the same rate and channel count of the `CX_SoundBuffer` are the same as those in use by the `CX_SoundBufferPlayer`. See [Blocking Code](#) for more information.

#### Returns

True if sound was successfully set to be the current sound, false otherwise.

### 13.30.2.6 `void CX::CX_SoundBufferPlayer::setTime ( CX_Millis time )`

Set the current time in the active sound. When playback starts, it will begin from that time in the sound. If the sound buffer is currently playing, this will jump to that point in the sound.

#### Parameters

<i>time</i>	The time in the sound to seek to.
-------------	-----------------------------------

### 13.30.2.7 `bool CX::CX_SoundBufferPlayer::setup ( Configuration config )`

Configures the `CX_SoundBufferPlayer` with the given configuration. A `CX_SoundStream` will be set up within the `CX_SoundBufferPlayer` and the sound stream will be started.

#### Parameters

<i>config</i>	The configuration to use for the <code>CX_SoundBufferPlayer</code> , which is really all about configuring the <code>CX_SoundStream</code> used internally by the <code>CX_SoundBufferPlayer</code> .
---------------	---

### 13.30.2.8 `bool CX::CX_SoundBufferPlayer::setup ( CX_SoundStream * ss )`

Set up the sound buffer recorder from an existing `CX_SoundStream`. The `CX_SoundStream` is not started automatically. The `CX_SoundStream` must exist for the lifetime of the `CX_SoundBufferPlayer`.



## Parameters

ss	A pointer to a fully configured <a href="#">CX_SoundStream</a> .
----	--

## Returns

`true` in all cases.

### 13.30.2.9 bool CX::CX\_SoundBufferPlayer::startPlayingAt ( CX\_Millis *experimentTime*, CX\_Millis *latencyOffset* )

Queue the start time of the sound in experiment time with an offset to account for latency.

## Parameters

<i>experimentTime</i>	The desired experiment time at which the sound should start playing. This time plus the offset should be in the future. If it is not, the sound will start playing immediately.
<i>latencyOffset</i>	An offset that accounts for latency. If, for example, you called this function with an offset of 0 and discovered that the sound played 200 ms later than you were expecting it to, you would set offset to -200 in order to queue the start time 200 ms earlier than the desired experiment time.

## Returns

False if the start time plus the offset is in the past. True otherwise.

## Note

See [setTime\(\)](#) for a way to choose the current time point within the sound.

### 13.30.2.10 bool CX::CX\_SoundBufferPlayer::stop ( void )

Stop the currently playing sound buffer, or, if a playback start was cued, cancel the cued playback.

## Returns

Always returns `true` currently.

The documentation for this class was generated from the following files:

- CX\_SoundBufferPlayer.h
- CX\_SoundBufferPlayer.cpp

## 13.31 CX::CX\_SoundBufferRecorder Class Reference

```
#include <CX_SoundBufferRecorder.h>
```

## Public Types

- typedef  
[CX\\_SoundStream::Configuration](#) Configuration  
*This is typedef'ed to [CX::CX\\_SoundStream::Configuration](#).*

## Public Member Functions

- `bool setup (Configuration &config)`
- `bool setup (CX_SoundStream *ss)`
- `Configuration getConfiguration (void)`
- `CX_SoundStream * getSoundStream (void)`  
*This function provides direct access to the `CX_SoundStream` used by the `CX_SoundBufferRecorder`.*
- `void setSoundBuffer (CX_SoundBuffer *so)`
- `CX_SoundBuffer * getSoundBuffer (void)`
- `void startRecording (bool clearExistingData=false)`
- `void stopRecording (void)`

### 13.31.1 Detailed Description

This class is used for recording audio data from, e.g., a microphone. The recorded data is stored in a `CX_SoundBuffer` for further use.

```
CX_SoundBufferRecorder recorder;

CX_SoundBufferRecorder::Configuration recorderConfig;
recorderConfig.inputChannels = 1;
//You will probably need to configure more than just the number of input channels.
recorder.setup(recorderConfig);

CX_SoundBuffer recording;
recorder.setSoundBuffer(&recording); //Associate a CX_SoundBuffer with the recorder so that the buffer can
    be recorded to.

//Record for 5 seconds
recorder.startRecording();
Clock.sleep(CX_Seconds(5));
recorder.stopRecording();

//Write the recording to a file
recording.writeToFile("recording.wav");
```

### 13.31.2 Member Function Documentation

#### 13.31.2.1 CX\_SoundBufferRecorder::Configuration CX::CX\_SoundBufferRecorder::getConfiguration ( void )

Returns the configuration used for this `CX_SoundBufferRecorder`.

#### 13.31.2.2 CX\_SoundBuffer \* CX::CX\_SoundBufferRecorder::getSoundBuffer ( void )

This function returns a pointer to the `CX_SoundBuffer` that is currently in use by the `CX_SoundBufferRecorder`.

#### 13.31.2.3 void CX::CX\_SoundBufferRecorder::setSoundBuffer ( CX\_SoundBuffer \* so )

This function associates a `CX_SoundBuffer` with the `CX_SoundBufferRecorder`. The `CX_SoundBuffer` will be recorded to when `startRecording()` is called.

#### Parameters

so	The <code>CX_SoundBuffer</code> to associate with the <code>CX_SoundBufferRecorder</code> . The sound buffer will be cleared and it will be configured to have the same number of channels and sample rate that the <code>CX_SoundBufferRecorder</code> was configured to use.
----	--

#### 13.31.2.4 bool CX::CX\_SoundBufferRecorder::setup ( CX\_SoundBufferRecorder::Configuration & config )

This function sets up the `CX_SoundStream` that `CX_SoundBufferRecorder` uses to record audio data.

## Parameters

<i>config</i>	A reference to a <a href="#">CX_SoundBufferRecorder::Configuration</a> struct that will be used to configure an internally-stored <a href="#">CX_SoundStream</a> .
---------------	--

## Returns

`true` if configuration of the [CX\\_SoundStream](#) was successful, `false` otherwise.

13.31.2.5 `bool CX::CX_SoundBufferRecorder::setup ( CX_SoundStream * ss )`

Set up the sound buffer recorder from an existing [CX\\_SoundStream](#). The [CX\\_SoundStream](#) is not started automatically. The [CX\\_SoundStream](#) must remain in scope for the lifetime of the [CX\\_SoundBufferRecorder](#).

## Parameters

<i>ss</i>	A pointer to a fully configured <a href="#">CX_SoundStream</a> .
-----------	--

## Returns

`true` in all cases.

13.31.2.6 `void CX::CX_SoundBufferRecorder::startRecording ( bool clearExistingData = false )`

Begins recording data to the [CX\\_SoundBuffer](#) that was associated with this [CX\\_SoundBufferRecorder](#) with [setSoundBuffer\(\)](#).

## Parameters

<i>clearExistingData</i>	If true, any data in the <a href="#">CX_SoundBuffer</a> will be deleted before recording starts.
--------------------------	--

13.31.2.7 `void CX::CX_SoundBufferRecorder::stopRecording ( void )`

Stop recording sound data.

The documentation for this class was generated from the following files:

- [CX\\_SoundBufferRecorder.h](#)
- [CX\\_SoundBufferRecorder.cpp](#)

## 13.32 CX::CX\_SoundStream Class Reference

```
#include <CX_SoundStream.h>
```

## Classes

- struct [Configuration](#)
- struct [InputEventArgs](#)
- struct [OutputEventArgs](#)

## Public Member Functions

- `bool setup (CX_SoundStream::Configuration &config)`

- bool [closeStream](#) (void)
- bool [start](#) (void)
- bool [stop](#) (void)
- bool [isStreamRunning](#) (void) const
- const  
    [CX\\_SoundStream::Configuration](#) & [getConfiguration](#) (void) const
- uint64\_t [getSampleFrameNumber](#) (void) const
- [CX\\_Millis](#) [getStreamLatency](#) (void)
- bool [hasSwappedSinceLastCheck](#) (void)
- [CX\\_Millis](#) [getLastSwapTime](#) (void)
- [CX\\_Millis](#) [estimateNextSwapTime](#) (void)
- RtAudio \* [getRtAudioInstance](#) (void)

### Static Public Member Functions

- static std::vector< RtAudio::Api > [getCompiledApis](#) (void)
- static std::vector< std::string > [convertApisToStrings](#) (vector< RtAudio::Api > apis)
- static std::string [convertApisToString](#) (vector< RtAudio::Api > apis, std::string delim="\r\n")
- static std::string [convertApiToString](#) (RtAudio::Api api)
- static RtAudio::Api [convertStringToApi](#) (std::string apiString)
- static std::vector< std::string > [formatsToStrings](#) (RtAudioFormat formats)
- static std::string [formatsToString](#) (RtAudioFormat formats, std::string delim="\r\n")
- static std::vector  
    < RtAudio::DeviceInfo > [getDeviceList](#) (RtAudio::Api api)
- static std::string [listDevices](#) (RtAudio::Api api)
- static  
    [CX\\_SoundStream::Configuration](#) [readConfigurationFromFile](#) (std::string filename, std::string delimiter="=", bool trimWhitespace=true, std::string commentStr="//")

### Public Attributes

- ofEvent  
    < [CX\\_SoundStream::OutputEventArgs](#) > [outputEvent](#)  
    *This event is triggered every time the [CX\\_SoundStream](#) needs to feed more data to the output buffer of the sound card.*
- ofEvent  
    < [CX\\_SoundStream::InputEventArgs](#) > [inputEvent](#)  
    *This event is triggered every time the [CX\\_SoundStream](#) has gotten some data from the input buffer of the sound card.*

### 13.32.1 Detailed Description

This class provides a method for directly accessing and manipulating sound data that is sent/received from sound hardware. To use this class, you should set up the stream (see [setup\(\)](#)), set a user function that will be called when either the [outputEvent](#) or [inputEvent](#) is triggered, and start the stream with [start\(\)](#).

If the stream is configured for output, the output event will be triggered whenever the sound card needs more sound data. If the stream is configured for input, the input event will be triggered whenever some amount of sound data has been recorded.

[CX\\_SoundStream](#) uses RtAudio internally, so you are having problems, you might be able to figure out what is going wrong by checking out the page for RtAudio: <http://www.music.mcgill.ca/~gary/rtaudio/index.-html>

## 13.32.2 Member Function Documentation

## 13.32.2.1 bool CX\_SoundStream::closeStream ( void )

Closes the sound stream.

## Returns

`false` if an error was encountered while closing the stream, `true` otherwise.

 13.32.2.2 std::string CX\_SoundStream::convertApisToString ( vector< RtAudio::Api > *apis*, std::string *delim* = "\r\n" )  
 [static]

This helper function converts a vector of RtAudio::Api to a string, with the specified delimiter between API names.

## Parameters

<i>apis</i>	The vector of RtAudio::Api to convert to string.
<i>delim</i>	The delimiter between elements of the string.

## Returns

A string containing the names of the APIs.

 13.32.2.3 std::vector< std::string > CX\_SoundStream::convertApisToStrings ( vector< RtAudio::Api > *apis* ) [static]

This helper function converts a vector of RtAudio::Api to a vector of strings, using [convertApiToString\(\)](#) for the conversion.

## Parameters

<i>apis</i>	A vector of apis to convert to strings.
-------------	---

## Returns

A vector of string names of the apis.

 13.32.2.4 std::string CX\_SoundStream::convertApiToString ( RtAudio::Api *api* ) [static]

This helper function converts an RtAudio::Api to a string.

## Parameters

<i>api</i>	The api to get a string of.
------------	-----------------------------

## Returns

A string of the api name.

 13.32.2.5 RtAudio::Api CX\_SoundStream::convertStringToApi ( std::string *apiString* ) [static]

Converts a string name of an RtAudio API to an RtAudio::Api enum constant.

## Parameters

<i>apiString</i>	The name of the API as a string. Should be one of the following, with no surrounding whitespace: UNSPECIFIED, LINUX_ALSA, LINUX_PULSE, LINUX_OSS, UNIX_JACK, MACOSX_CORE, WINDOWS_ASIO, WINDOWS_DS, RTAUDIO_DUMMY
------------------	---

## Returns

The RtAudio::Api corresponding to the provided string. If the string is not one of the above values, RtAudio::Api::UNSPECIFIED is returned.

### 13.32.2.6 CX\_Millis CX\_SoundStream::estimateNextSwapTime ( void )

Estimate the time at which the next buffer swap will occur.

## Returns

The estimated time of next swap. This value can be compared with the result of CX::Instances::Clock.now().

### 13.32.2.7 std::string CX\_SoundStream::formatsToString ( RtAudioFormat *formats*, std::string *delim* = "\r\n" ) [static]

Converts a bitmask of audio formats to a string, with each format delimited by *delim*.

## Parameters

<i>formats</i>	The bitmask of audio formats.
<i>delim</i>	The delimiter.

## Returns

A string containing string representations of the valid formats in *formats*.

### 13.32.2.8 std::vector< std::string > CX\_SoundStream::formatsToStrings ( RtAudioFormat *formats* ) [static]

Converts a bitmask of audio formats to a vector of strings.

## Parameters

<i>formats</i>	The bitmask of audio formats.
----------------	-------------------------------

## Returns

A vector of strings, one string for each bit set in *formats* for which there is a corresponding valid audio format that RtAudio supports.

### 13.32.2.9 std::vector< RtAudio::Api > CX\_SoundStream::getCompiledApis ( void ) [static]

Get a vector containing a list of all of the APIs for which the RtAudio driver has been compiled to use. If the API you want is not available, you might be able to get it by using a different version of RtAudio.

### 13.32.2.10 const CX\_SoundStream::Configuration& CX::CX\_SoundStream::getConfiguration ( void ) const [inline]

Gets the configuration that was used on the last call to open(). Because some of the configuration options are only suggestions, this function allows you to check what the actual configuration was.

## Returns

A const reference to the configuration struct.

13.32.2.11 `std::vector< RtAudio::DeviceInfo > CX_SoundStream::getDeviceList ( RtAudio::Api api )` `[static]`

For the given `api`, lists all of the devices on the system that support that `api`.

## Parameters

<i>api</i>	Devices that support this API are scanned.
------------	--

## Returns

A machine-readable list of information. See [http://www.music.mcgill.ca/~gary/rtaudio/struct-RtAudio\\_1\\_1DeviceInfo.html](http://www.music.mcgill.ca/~gary/rtaudio/struct-RtAudio_1_1DeviceInfo.html) for information about the members of the `RtAudio::DeviceInfo` struct.

### 13.32.2.12 `CX_Millis CX::CX_SoundStream::getLastSwapTime ( void ) [inline]`

Gets the time at which the last buffer swap occurred.

## Returns

This time value can be compared with the result of `CX::CX_Clock::now()`.

### 13.32.2.13 `RtAudio * CX_SoundStream::getRtAudioInstance ( void )`

This function returns a pointer to the `RtAudio` instance that this `CX_SoundStream` is using. This should not be needed most of the time, but there may be cases in which you need to directly access `RtAudio`. Here is the documentation for `RtAudio`: <https://www.music.mcgill.ca/~gary/rtaudio/>

### 13.32.2.14 `uint64_t CX::CX_SoundStream::getSampleFrameNumber ( void ) const [inline]`

Returns the number of the sample frame that is about to be loaded into the stream buffer on the next buffer swap.

### 13.32.2.15 `CX_Millis CX_SoundStream::getStreamLatency ( void )`

This function gets an estimate of the stream latency. However, it should not be relied on as it is based on what the sound card driver reports, which is often false.

## Returns

The stream latency in microseconds.

### 13.32.2.16 `bool CX_SoundStream::hasSwappedSinceLastCheck ( void )`

This function checks to see if the audio buffers have been swapped since the last time this function was called.

## Returns

True if at least one audio buffer has been swapped out, false if no buffers have been swapped.

### 13.32.2.17 `bool CX_SoundStream::isStreamRunning ( void ) const`

Check whether the sound stream is running.

## Returns

false if the stream is not setup or not running or if `RtAudio` has not been initialized. Returns `true` if the stream is running.

### 13.32.2.18 `std::string CX_SoundStream::listDevices ( RtAudio::Api api ) [static]`

For the given `api`, lists all of the devices on the system that support that `api`. Lots of information about each device is given, like supported sample rates, number of input and output channels, etc.



## Parameters

<i>api</i>	Devices that support this API are scanned.
------------	--

## Returns

A human-readable formatted string containing the scanned information. Can be printed directly to `std::cout` or elsewhere.

### 13.32.2.19 CX\_SoundStream::Configuration CX\_SoundStream::readConfigurationFromFile ( `std::string filename`, `std::string delimiter` = "=", `bool trimWhitespace` = `true`, `std::string commentString` = "//" ) [static]

This function exists to serve a per-computer configuration function that is otherwise difficult to provide due to the fact that C++ programs are compiled to binaries and cannot be easily edited on the computer on which they are running. This function takes the file name of a specially constructed configuration file and reads the key-value pairs in that file in order to fill a [CX\\_SoundStream::Configuration](#) struct. The format of the file is provided in the example code below.

Sample configuration file:

```
ss.api = WINDOWS_DS //See convertStringToApi() for valid API names.
ss.sampleRate = 44100
ss.bufferSize = 512
ss.inputChannels = 0
//ss.inputDeviceId not used in this example because no input channels are used. It would take a
//    non-negative integer.
ss.outputChannels = 2
ss.outputDeviceId = 0
ss.streamOptions.numberOfBuffers = 4
ss.streamOptions.flags = RTAUDIO_SCHEDULE_REALTIME | RTAUDIO_MINIMIZE_LATENCY //The | is not needed,
//but it matches the way these flags are used in code. All flags are supported.
//ss.streamOptions.priority is not used in this example. It would take a positive integer.
```

All of the configuration keys are used in this example. Any values in the [CX\\_SoundStream::Configuration](#) struct that do not have values provided in the configuration file will be left at default values. Note that the "ss" prefix allows this configuration to be embedded in a file that also performs other configuration functions. Note that the names of the data members match the names used in the [CX\\_SoundStream::Configuration](#) struct and have a 1-to-1 relationship with those values.

Because this function uses [CX::Util::readKeyValueFile\(\)](#) internally, it has the same arguments.

## Parameters

<i>filename</i>	The name of the file containing configuration data.
<i>delimiter</i>	The string that separates the key from the value. In the example, it is "=", but can be other values.
<i>trimWhitespace</i>	If true, whitespace characters surrounding both the key and value will be removed. This is a good idea to do.
<i>commentString</i>	If commentString is not the empty string (i.e. ""), everything on a line following the first instance of commentString will be ignored.

### 13.32.2.20 bool CX\_SoundStream::setup ( CX\_SoundStream::Configuration & config )

Opens the sound stream with the specified configuration. If there was an error during configuration, messages will be logged.

## Parameters

<i>config</i>	The configuration settings that are desired. Some of the configuration options are only suggestions, so some of the values that are used may differ from the values that are chosen. In those cases, config is updated based on the actually used settings. You can check the configuration later using <a href="#">getConfiguration()</a> .
---------------	--

**Returns**

True if configuration appeared to be successful, false otherwise.

**Note**

Opening the stream does not start it. See [start\(\)](#).

**13.32.2.21 bool CX\_SoundStream::start ( void )**

Starts the sound stream. The stream must already be have been set up (see [setup\(\)](#)).

**Returns**

*false* if the stream was not started, *true* if the stream was started or if it was already running.

**13.32.2.22 bool CX\_SoundStream::stop ( void )**

Stop the stream, if is running. If there is an error, a message will be logged.

**Returns**

*false* if there was an error, *true* otherwise.

The documentation for this class was generated from the following files:

- CX\_SoundStream.h
- CX\_SoundStream.cpp

**13.33 CX::CX\_Time\_t< T > Class Template Reference**

```
#include <CX_Time_t.h>
```

**Classes**

- struct [PartitionedTime](#)

**Public Member Functions**

- [PartitionedTime getPartitionedTime](#) (void) const
- [CX\\_Time\\_t](#) (void)
- [CX\\_Time\\_t](#) (double t)
- [CX\\_Time\\_t](#) (int t)
- [CX\\_Time\\_t](#) (long long t)
- template<typename tArg >  
[CX\\_Time\\_t](#) (const [CX\\_Time\\_t](#)< tArg > &t)

- double [value](#) (void) const
- double [hours](#) (void) const  
*Get the time stored by this [CX\\_Time\\_t](#) in hours, including fractions of an hour.*
- double [minutes](#) (void) const  
*Get the time stored by this [CX\\_Time\\_t](#) in minutes, including fractions of a minute.*
- double [seconds](#) (void) const  
*Get the time stored by this [CX\\_Time\\_t](#) in seconds, including fractions of a second.*
- double [millis](#) (void) const  
*Get the time stored by this [CX\\_Time\\_t](#) in milliseconds, including fractions of a millisecond.*
- double [micros](#) (void) const  
*Get the time stored by this [CX\\_Time\\_t](#) in microseconds, including fractions of a microsecond.*
- long long [nanos](#) (void) const  
*Get the time stored by this [CX\\_Time\\_t](#) in nanoseconds.*
- template<typename RT >  
[CX\\_Time\\_t](#)< TimeUnit > [operator+](#) (const [CX\\_Time\\_t](#)< RT > &rhs) const  
*Adds together two times.*
- template<typename RT >  
[CX\\_Time\\_t](#)< TimeUnit > [operator-](#) (const [CX\\_Time\\_t](#)< RT > &rhs) const  
*Subtracts two times.*
- template<typename RT >  
double [operator/](#) (const [CX\\_Time\\_t](#)< RT > &rhs) const  
*Divides a [CX\\_Time\\_t](#) by another [CX\\_Time\\_t](#), resulting in a unitless ratio.*
- [CX\\_Time\\_t](#)< TimeUnit > [operator/](#) (double rhs) const  
*Divides a [CX\\_Time\\_t](#) by a unitless value, resulting in a [CX\\_Time\\_t](#) of the same type.*
- [CX\\_Time\\_t](#)< TimeUnit > & [operator\\* =](#) (double rhs)  
*Multiplies a [CX\\_Time\\_t](#) by a unitless value, storing the result in the [CX\\_Time\\_t](#). You cannot multiply a time by another time because that would result in units of time squared.*
- template<typename RT >  
[CX\\_Time\\_t](#)< TimeUnit > & [operator+ =](#) (const [CX\\_Time\\_t](#)< RT > &rhs)  
*Adds a [CX\\_Time\\_t](#) to an existing [CX\\_Time\\_t](#).*
- template<typename RT >  
[CX\\_Time\\_t](#)< TimeUnit > & [operator- =](#) (const [CX\\_Time\\_t](#)< RT > &rhs)  
*Subtracts a [CX\\_Time\\_t](#) from an existing [CX\\_Time\\_t](#).*
- template<typename RT >  
bool [operator<](#) (const [CX\\_Time\\_t](#)< RT > &rhs) const  
*Compares two times in the expected way.*
- template<typename RT >  
bool [operator< =](#) (const [CX\\_Time\\_t](#)< RT > &rhs) const  
*Compares two times in the expected way.*
- template<typename RT >  
bool [operator>](#) (const [CX\\_Time\\_t](#)< RT > &rhs) const  
*Compares two times in the expected way.*
- template<typename RT >  
bool [operator> =](#) (const [CX\\_Time\\_t](#)< RT > &rhs) const  
*Compares two times in the expected way.*
- template<typename RT >  
bool [operator==](#) (const [CX\\_Time\\_t](#)< RT > &rhs) const  
*Compares two times in the expected way.*
- template<typename RT >  
bool [operator!=](#) (const [CX\\_Time\\_t](#)< RT > &rhs) const  
*Compares two times in the expected way.*

## Static Public Member Functions

- static `CX_Time_t< TimeUnit > min` (void)  
*Get the minimum time value that can be represented with this class.*
- static `CX_Time_t< TimeUnit > max` (void)  
*Get the maximum time value that can be represented with this class.*
- static `CX_Time_t< TimeUnit > standardDeviation` (std::vector< `CX_Time_t< TimeUnit >> vals)`

### 13.33.1 Detailed Description

`template<typename T>class CX::CX_Time_t< T >`

This class provides a convenient way to deal with time in various units. The upside of this system is that although all functions in `CX` that take time can take time values in a variety of units. For example, `CX_Clock::wait()` takes `CX_Millis` as the time type so if you were to do

```
Clock.wait(20);
```

it would attempt to wait for 20 milliseconds. However, you could do

```
Clock.wait(CX_Seconds(.5));
```

to wait for half of a second, if units of seconds are easier to think in for the given situation.

`CX_Time_t` has at most nanosecond accuracy. The contents of any of the templated versions of `CX_Time_t` are all stored in nanoseconds, so conversion between time types is lossless.

See this example for a variety of things you can do with this class.

```
CX_Millis mil = 100;
CX_Micros mic = mil; //mic now contains 100000 microseconds == 100 milliseconds.
//Really, they both contain 100,000,000 nanoseconds.

//You can add times together.
CX_Seconds sec = CX_Minutes(.1) + CX_Millis(100); //sec contains 6.1 seconds.

//You can take the ratio of times.
double secondsPerMinute = CX_Seconds(1)/CX_Minutes(1);

//You can compare times using the standard comparison operators (==, !=, <, >, <=, >=).
if (CX_Minutes(60) == CX_Hours(1)) {
    cout << "There are 60 minutes in an hour." << endl;
}

if (CX_Millis(12.3456) == CX_Micros(12345.6)) {
    cout << "Time can be represented as a floating point value with sub-time-unit precision." << endl;
}

//If you want to be explicit about what time unit you want out, you can use the seconds(), millis(), etc.,
//functions:
sec = CX_Seconds(6);
cout << "In " << sec.seconds() << " seconds there are " << sec.millis() << " milliseconds and " << sec.
    minutes() << " minutes." << endl;

//You can alternately do a typecast if you're about to print the result:
cout << "In " << sec << " seconds there are " << (CX_Millis)sec << " milliseconds and " << (
    CX_Minutes)sec << " minutes." << endl;

//The difference between the above examples is the resulting type.
//double minutes = (CX_Minutes)sec; //This does not work: A CX_Minutes cannot be assigned to a double
double minutes = sec.minutes(); //minutes() returns a double.

//You can construct a time with the result of the construction of a time object with a different time unit.
CX_Minutes min = CX_Hours(.05); //3 minutes

//You can get the whole number amounts of different time units.
CX_Seconds longTime = CX_Hours(2) + CX_Minutes(16) +
    CX_Seconds(40) + CX_Millis(123) + CX_Micros(456) + CX_Nanos(1);
CX_Seconds::PartitionedTime parts = longTime.getPartitionedTime();
```

## 13.33.2 Constructor &amp; Destructor Documentation

13.33.2.1 `template<typename T> CX::CX_Time_t< T >::CX_Time_t( void ) [inline]`

Default constructor for [CX\\_Time\\_t](#).

13.33.2.2 `template<typename T> CX::CX_Time_t< T >::CX_Time_t( double t ) [inline]`

Constructs a [CX\\_Time\\_t](#) with the specified time value.

## Parameters

<code>t</code>	A time value with units interpreted depending on the TimeUnit template argument for the instance of the class being constructed.
----------------	--

Example:

```
CX_Minutes quarterHour(15); //Interpreted as 15 minutes
CX_Seconds oneMinute(60); //Interpreted as 60 seconds
```

13.33.2.3 `template<typename T> CX::CX_Time_t< T >::CX_Time_t( int t ) [inline]`

Constructs a [CX\\_Time\\_t](#) with the specified time value.

## Parameters

<code>t</code>	A time value with units interpreted depending on the TimeUnit template argument for the instance of the class being constructed.
----------------	--

Example:

```
CX_Minutes quarterHour(15); //Interpreted as 15 minutes
CX_Seconds oneMinute(60); //Interpreted as 60 seconds
```

13.33.2.4 `template<typename T> CX::CX_Time_t< T >::CX_Time_t( long long t ) [inline]`

Constructs a [CX\\_Time\\_t](#) with the specified time value.

## Parameters

<code>t</code>	A time value with units interpreted depending on the TimeUnit template argument for the instance of the class being constructed.
----------------	--

Example:

```
CX_Minutes quarterHour(15); //Interpreted as 15 minutes
CX_Seconds oneMinute(60); //Interpreted as 60 seconds
```

13.33.2.5 `template<typename T> template<typename tArg> CX::CX_Time_t< T >::CX_Time_t( const CX_Time_t< tArg > & t ) [inline]`

Constructs a [CX\\_Time\\_t](#) based on another instance of a [CX\\_Time\\_t](#). If the TimeUnit template parameter has a different value for `t` than for the [CX\\_Time\\_t](#) being constructed, it does not change the amount of time stored. For example, if `t` is a [CX\\_Time\\_t](#)<std::ratio<60, 1>> (i.e. CX\_Minutes) containing 1 minute, and the [CX\\_Time\\_t](#) that is constructed will contain 1 minute regardless of if that minute is thought of as 1/60 of an hour or 60,000,000 microseconds.

## 13.33.3 Member Function Documentation

13.33.3.1 `template<typename T> PartitionedTime CX::CX_Time_t< T >::getPartitionedTime ( void ) const [inline]`

Partitions a `CX_Time_t` into component parts containing the number of whole time units that are stored in the `CX_Time_t`. This is different from `seconds()`, `millis()`, etc., because those functions return the fractional part (e.g. 5.340 seconds) whereas this returns only whole numbers (e.g. 5 seconds and 340 milliseconds).

#### Returns

A `PartitionedTime` struct containing whole number amounts of the components of the time.

13.33.3.2 `template<typename T> template<typename RT > CX_Time_t<TimeUnit>& CX::CX_Time_t< T >::operator+=( const CX_Time_t< RT > & rhs ) [inline]`

Adds a `CX_Time_t` to an existing `CX_Time_t`.

#### Parameters

<i>rhs</i>	The value to add.
------------	-------------------

13.33.3.3 `template<typename T> template<typename RT > CX_Time_t<TimeUnit>& CX::CX_Time_t< T >::operator-= ( const CX_Time_t< RT > & rhs ) [inline]`

Subtracts a `CX_Time_t` from an existing `CX_Time_t`.

#### Parameters

<i>rhs</i>	The value to subtract.
------------	------------------------

13.33.3.4 `template<typename T> static CX_Time_t<TimeUnit> CX::CX_Time_t< T >::standardDeviation ( std::vector< CX_Time_t< TimeUnit >> vals ) [inline], [static]`

This function calculates the sample standard deviation for a vector of time values.

13.33.3.5 `template<typename T> double CX::CX_Time_t< T >::value ( void ) const [inline]`

Get the numerical value of the time in units of the time type. For example, if you are using an instance of `CX_Seconds`, this will return the time value in seconds, including fractional seconds.

The documentation for this class was generated from the following file:

- `CX_Time_t.h`

## 13.34 CX::Util::CX\_TrialController Class Reference

```
#include <CX_TrialController.h>
```

#### Public Member Functions

- int `update` (void)
- void `start` (void)  
"Arm" the trial controller. Before this is called, `update()` will do nothing.
- void `stop` (void)  
"Disarm" the trial controller. After this is called, `update()` will do nothing.
- bool `isActive` (void)

Check to see if the trial controller is active. See [start\(\)](#) and [stop\(\)](#).

- void [appendFunction](#) (std::function< int(void)> userFunction)
- void [reset](#) (void)
- bool [setCurrentFunction](#) (int currentFunction)
- int [getCurrentFunction](#) (void)

Get the index of the current function (i.e. the function that will be called the next time [update\(\)](#) is called).

- unsigned int [getFunctionCount](#) (void)

Get the number of user functions stored by this trial controller.

#### 13.34.1 Detailed Description

This class is used to help with the fact that most psychology experiments are by nature more or less linear, but that CX works better if code does not block (see [Blocking Code](#)).

The way this works is that segments of user code, each representing one part of a trial, are put into functions. Those functions are added to the trial controller with [appendFunction\(\)](#), which puts the function at the end of the list of functions. User functions take no arguments and return an int.

When you want to use the trial controller, call [start\(\)](#) and it will be "armed". When it is armed and [update\(\)](#) is called, it will call the current function in the list of user functions. If the user function is done with whatever it needs to do, it should return 1. This will cause the trial controller to move on to the next user function. If the user function is not done with its task, it should return 0. If it returns 0, it will be called again the next time [update\(\)](#) is called.

#### 13.34.2 Member Function Documentation

##### 13.34.2.1 void CX\_TrialController::appendFunction ( std::function< int(void)> userFunction )

Adds a user function to the end of the list of functions to be called by the trial controller.

###### Parameters

<i>userFunction</i>	Typically a pointer to a function that takes no arguments and returns an int. Because it is a std::function, it can also be a lamda.
---------------------	--

##### 13.34.2.2 void CX\_TrialController::reset ( void )

Clear the user functions and otherwise reset to default state.

###### Note

This function stops the trial controller ([isActive\(\)](#) will return false).

##### 13.34.2.3 bool CX\_TrialController::setCurrentFunction ( int currentFunction )

Sets the current user function by index, which allows you to skip over functions or go back to a previous function.

###### Parameters

<i>currentFunction</i>	The new current function index. If this is out of range, an error will be logged and the function will return false.
------------------------	--

###### Returns

False if the index was out of range, true otherwise.

**Note**

If this is called from within a user function that has been called from this instance of the [CX\\_TrialController](#), that function should return 0. If it does not, [setCurrentFunction\(\)](#) will set the function index and then that index will be incremented after the user function completes. However, if 0 is returned, the function index is not incremented.

**13.34.2.4 int CX\_TrialController::update ( void )**

Updates the trial controller state. Each time this function is called, the user function at the current function index is called. If that function returns a nonzero value, the trial controller will increment the current function index and that function will be called the next time [update\(\)](#) is called. If the current function index is incremented past the end of the list of functions, it will wrap around to the beginning of the list.

**Returns**

The value returned by the user function that was called.

**Note**

This function should probably be called every time `updateExperiment()` is called, although there are other use cases.

If not [isActive\(\)](#), this function does nothing and returns 0.

The documentation for this class was generated from the following files:

- CX\_TrialController.h
- CX\_TrialController.cpp

**13.35 CX::CX\_WindowConfiguration\_t Struct Reference**

```
#include <CX_EntryPoint.h>
```

**Public Attributes**

- ofWindowMode **mode**
- int **width**
- int **height**
- unsigned int **msaaSampleCount**
- ofPtr< ofBaseGLRenderer > **desiredRenderer**
- Private::CX\_GLVersion **desiredOpenGLVersion**
- std::string **windowTitle**

**13.35.1 Detailed Description**

This structure is used to configure windows opened with [CX::reopenWindow\(\)](#).

The documentation for this struct was generated from the following file:

- CX\_EntryPoint.h



## 13.36 CX::Synth::Envelope Class Reference

```
#include <CX_ModularSynth.h>
```

Inherits [CX::Synth::ModuleBase](#).

### Public Member Functions

- double [getNextSample](#) (void) override
- void [attack](#) (void)  
*Trigger the attack of the [Envelope](#).*
- void [release](#) (void)  
*Trigger the release of the [Envelope](#).*

### Public Attributes

- [ModuleParameter](#) [gateInput](#)
- [ModuleParameter](#) [a](#)  
*The number of seconds it takes, following the attack, for the level to rise from 0 to 1. Should be non-negative.*
- [ModuleParameter](#) [d](#)  
*The number of seconds it takes, once reaching the attack peak, to fall to *s*. Should be non-negative.*
- [ModuleParameter](#) [s](#)  
*The level at which the envelope sustains while waiting for the release. Should be between 0 and 1.*
- [ModuleParameter](#) [r](#)  
*The number of seconds it takes, following the release, for the level to fall to 0 from *s*. Should be non-negative.*

### Additional Inherited Members

#### 13.36.1 Detailed Description

This class is a standard ADSR envelope: [http://en.wikipedia.org/wiki/Synthesizer#ADSR\\_envelope](http://en.wikipedia.org/wiki/Synthesizer#ADSR_envelope). *s* should be in the interval [0,1]. *a*, *d*, and *r* are expressed in seconds. Call [attack\(\)](#) to start the envelope. Once the attack and decay are finished, the envelope will stay at the sustain level until [release\(\)](#) is called.

The output values produced start at 0, rise to 1 during the attack, drop to the sustain level (*s*) during the decay, and drop from *s* to 0 during the release.

#### 13.36.2 Member Function Documentation

##### 13.36.2.1 double Envelope::getNextSample ( void ) [override],[virtual]

This function should be overloaded for any derived class that can be used as the input for another module.

Reimplemented from [CX::Synth::ModuleBase](#).

#### 13.36.3 Member Data Documentation

### 13.36.3.1 ModuleParameter CX::Synth::Envelope::gateInput

This parameter can be used by another module as a way to signal the [Envelope](#). When the output of the module inputting to `gateInput` changes to 1.0, the attack of the envelope is triggered. When it changes to 0, the release is triggered.

The documentation for this class was generated from the following files:

- CX\_ModularSynth.h
- CX\_ModularSynth.cpp

## 13.37 CX::CX\_Keyboard::Event Struct Reference

```
#include <CX_Keyboard.h>
```

### Public Attributes

- [int key](#)
- [CX\\_Millis time](#)  
*The time at which the event was registered. Can be compared to the result of [CX::CX\\_Clock::now\(\)](#).*
- [CX\\_Millis uncertainty](#)  
*The uncertainty in eventTime. The event occurred some time between eventTime and eventTime minus uncertainty.*
- [KeyboardEventType type](#)  
*The type of the event: press, release, or key repeat.*

### 13.37.1 Detailed Description

This struct contains the results of a keyboard event, whether it be a key press or release, or key repeat.

### 13.37.2 Member Data Documentation

#### 13.37.2.1 int CX::CX\_Keyboard::Event::key

The key involved in this event. The value of this can be compared with character literals for many of the standard keyboard keys. For example, you could use `(myKeyEvent.key == 'e')` to test if the key was the E key. Always check for lower case letters, because shift/capslock are ignored when setting the value for `key`.

For special keys, `key` can be compared with the key constant values defined in `ofConstants.h` (e.g. `OF_KEY_ESC`).

Note that the modifier keys (shift, ctrl, alt, and super) are treated a little unusually. For those keys, you can check for a specific key using, for example, the constants defined in "ofConstants.h" of `OF_KEY_RIGHT_CONTROL` or `OF_KEY_LEFT_CONTROL`. However, you can alternately check to see if `key` is either of the control keys by performing a bitwise AND (`&`) with `OF_KEY_CONTROL` and checking that the result of the AND is still `OF_KEY_CONTROL`. For example:

```
if ((myKeyEvent.key & OF_KEY_CONTROL) == OF_KEY_CONTROL) {
//...
}
```

This works the same way for all of the modifier keys.

The documentation for this struct was generated from the following file:

- CX\_Keyboard.h

### 13.38 CX::CX\_Joystick::Event Struct Reference

```
#include <CX_Joystick.h>
```

#### Public Types

- enum [JoystickEventType](#) { [BUTTON\\_PRESS](#), [BUTTON\\_RELEASE](#), [AXIS\\_POSITION\\_CHANGE](#) }

#### Public Attributes

- int [buttonIndex](#)  
*If eventType is [BUTTON\\_PRESS](#) or [BUTTON\\_RELEASE](#), this contains the index of the button that was changed.*
- unsigned char [buttonState](#)  
*If eventType is [BUTTON\\_PRESS](#) or [BUTTON\\_RELEASE](#), this contains the current state of the button.*
- int [axisIndex](#)  
*If eventType is [AXIS\\_POSITION\\_CHANGE](#), this contains the index of the axis which changed.*
- float [axisPosition](#)  
*If eventType is [AXIS\\_POSITION\\_CHANGE](#), this contains the amount by which the axis changed.*
- [CX\\_Millis](#) [eventTime](#)  
*The time at which the event was registered. Can be compared to the result of [CX::CX\\_Clock::now\(\)](#).*
- [CX\\_Millis](#) [uncertainty](#)  
*The uncertainty in eventTime. The event occurred some time between eventTime and eventTime minus uncertainty.*
- enum  
[CX::CX\\_Joystick::Event::JoystickEventType](#) [eventType](#)  
*The type of the event.*

#### 13.38.1 Detailed Description

This struct contains information about joystick events. Joystick events are either a button press or release or a change in the axes of the joystick.

#### 13.38.2 Member Enumeration Documentation

##### 13.38.2.1 enum CX::CX\_Joystick::Event::JoystickEventType

The type of the joystick event.

#### Enumerator

**[BUTTON\\_PRESS](#)** A button on the joystick has been pressed. See [buttonIndex](#) and [buttonState](#) for the event data.

**[BUTTON\\_RELEASE](#)** A button on the joystick has been released. See [buttonIndex](#) and [buttonState](#) for the event data.

**[AXIS\\_POSITION\\_CHANGE](#)** The joystick has been moved in one of its axes. See [axisIndex](#) and [axisPosition](#) for the event data.

The documentation for this struct was generated from the following file:

- [CX\\_Joystick.h](#)

### 13.39 CX::CX\_Mouse::Event Struct Reference

```
#include <CX_Mouse.h>
```

#### Public Attributes

- int [button](#)  
*The relevant mouse button if the event `type` is `PRESSED`, `RELEASED`, or `DRAGGED`. Can be compared with elements of enum [CX\\_Mouse::Buttons](#) to find out about the named buttons.*
- int [x](#)  
*The `x` position of the cursor at the time of the event, or the change in the `x`-axis scroll if the `eventType` is `MouseEventType::SCROLLED`.*
- int [y](#)  
*The `y` position of the cursor at the time of the event, or the change in the `y`-axis scroll if the `eventType` is `SCROLLED`.*
- [CX\\_Millis time](#)  
*The time at which the event was registered. Can be compared to the result of `CX::Clock::now()`.*
- [CX\\_Millis uncertainty](#)  
*The uncertainty in `eventTime`. The event occurred some time between `eventTime` and `eventTime` minus `uncertainty`.*
- [MouseEventType type](#)  
*The type of the event.*

#### 13.39.1 Detailed Description

This struct contains the results of a mouse event, which is any type of interaction with the mouse, be it simply movement, a button press or release, a drag event (mouse button held while mouse is moved), or movement of the scroll wheel.

The documentation for this struct was generated from the following file:

- [CX\\_Mouse.h](#)

### 13.40 CX::Synth::Filter Class Reference

```
#include <CX_ModularSynth.h>
```

Inherits [CX::Synth::ModuleBase](#).

#### Public Types

- enum [FilterType](#) { `LOW_PASS`, `HIGH_PASS`, `BAND_PASS`, `NOTCH` }

#### Public Member Functions

- void [setType](#) ([FilterType](#) type)  
*Set the type of filter to use, from the [Filter::FilterType](#) enum.*
- double [getNextSample](#) (void) override

## Public Attributes

- [ModuleParameter cutoff](#)
- [ModuleParameter bandwidth](#)

## Additional Inherited Members

## 13.40.1 Detailed Description

This class provides a basic way to filter waveforms as part of subtractive synthesis or other audio manipulation.

This class is based on simple IIR filters. They may not be stable at all frequencies. They are computationally very efficient. They are not highly configurable. They may be chained for sharper frequency response. This class is based on this chapter: <http://www.dspguide.com/ch19.htm>.

## 13.40.2 Member Enumeration Documentation

## 13.40.2.1 enum CX::Synth::Filter::FilterType

The type of filter to use.

## 13.40.3 Member Function Documentation

## 13.40.3.1 double Filter::getNextSample ( void ) [override],[virtual]

This function should be overloaded for any derived class that can be used as the input for another module.

Reimplemented from [CX::Synth::ModuleBase](#).

## 13.40.4 Member Data Documentation

## 13.40.4.1 ModuleParameter CX::Synth::Filter::bandwidth

Only used for BAND\_PASS and NOTCH FilterTypes. Sets the width (in frequency domain) of the stop or pass band at which the amplitude is equal to  $\sin(\pi/4)$  (i.e. .707). So, for example, if you wanted the frequencies 100 Hz above and below the breakpoint to be at .707 of the maximum amplitude, set `bandwidth` to 100. Of course, past those frequencies the attenuation continues. Larger values result in a less pointy band.

## 13.40.4.2 ModuleParameter CX::Synth::Filter::cutoff

The cutoff frequency of the filter.

The documentation for this class was generated from the following files:

- CX\_ModularSynth.h
- CX\_ModularSynth.cpp

## 13.41 CX::CX\_SlidePresenter::FinalSlideFunctionArgs Struct Reference

```
#include <CX_SlidePresenter.h>
```

## Public Attributes

- `CX_SlidePresenter * instance`  
*A pointer to the `CX_SlidePresenter` that called the user function.*
- unsigned int `currentSlideIndex`  
*The index of the slide that is currently being presented.*

### 13.41.1 Detailed Description

The final slide function takes a reference to a struct of this type.

The documentation for this struct was generated from the following file:

- `CX_SlidePresenter.h`

## 13.42 CX::Synth::FIRFilter Class Reference

```
#include <CX_ModularSynth.h>
```

Inherits `CX::Synth::ModuleBase`.

## Public Types

- enum `FilterType` { `LOW_PASS`, `HIGH_PASS`, `FIR_USER_DEFINED` }
- enum `WindowType` { `RECTANGULAR`, `HANNING`, `BLACKMAN` }

## Public Member Functions

- void **setup** (`FilterType` filterType, unsigned int coefficientCount)
- void **setup** (std::vector< double > coefficients)
- void **setCutoff** (double cutoff)
- double `getNextSample` (void)

## Additional Inherited Members

### 13.42.1 Detailed Description

This class is a start at implementing a Finite Impulse Response filter ([http://en.wikipedia.org/wiki/Finite\\_impulse\\_response](http://en.wikipedia.org/wiki/Finite_impulse_response)). You can use it as a basic low-pass or high-pass filter, or, if you supply your own coefficients, which cause the filter to do filtering in whatever way you want. See the "signal" package for R for a method of constructing your own coefficients.

### 13.42.2 Member Enumeration Documentation

#### 13.42.2.1 enum `CX::Synth::FIRFilter::FilterType` [strong]

The type of filter to use.

### 13.42.3 Member Function Documentation

#### 13.42.3.1 double CX::Synth::FIRFilter::getNextSample ( void ) [inline],[virtual]

This function should be overloaded for any derived class that can be used as the input for another module.

Reimplemented from [CX::Synth::ModuleBase](#).

The documentation for this class was generated from the following file:

- CX\_ModularSynth.h

## 13.43 CX::Synth::GenericOutput Class Reference

```
#include <CX_ModularSynth.h>
```

Inherits [CX::Synth::ModuleBase](#).

### Public Member Functions

- double [getNextSample](#) (void) override

### Additional Inherited Members

#### 13.43.1 Detailed Description

This class is used within output modules that actually output data. This class serves as an endpoint for data that is then retrieved by the class containing the [GenericOutput](#). See, for example, the [StereoStreamOutput](#) class.

### 13.43.2 Member Function Documentation

#### 13.43.2.1 double CX::Synth::GenericOutput::getNextSample ( void ) [inline],[override],[virtual]

This function should be overloaded for any derived class that can be used as the input for another module.

Reimplemented from [CX::Synth::ModuleBase](#).

The documentation for this class was generated from the following file:

- CX\_ModularSynth.h

## 13.44 CX::CX\_SoundStream::InputEventArgs Struct Reference

```
#include <CX_SoundStream.h>
```

### Public Attributes

- bool [bufferOverflow](#)  
*This is set to true if there was a buffer overflow, which means that the sound hardware recorded data that was not processed.*
- float \* [inputBuffer](#)  
*A pointer to an array of sound data that should be processed by the event handler function.*

- unsigned int [bufferSize](#)  
*The number of sample frames that are in `inputBuffer`. The total number of samples is `bufferSize * inputChannels`.*
- int [inputChannels](#)  
*The number of channels worth of data in `inputBuffer`.*
- [CX\\_SoundStream](#) \* [instance](#)  
*A pointer to the [CX\\_SoundStream](#) instance that notified this input event.*

#### 13.44.1 Detailed Description

The audio input event of the [CX\\_SoundStream](#) sends a copy of this structure with the fields filled out when the event is called.

The documentation for this struct was generated from the following file:

- [CX\\_SoundStream.h](#)

### 13.45 CX::CX\_DataFrame::IoOptions Class Reference

```
#include <CX_DataFrame.h>
```

Inherited by [CX::CX\\_DataFrame::InputOptions](#), and [CX::CX\\_DataFrame::OutputOptions](#).

#### Public Attributes

- std::string [cellDelimiter](#)  
*The delimiter between cells of the data frame when in file format. Defaults to `tab`.*
- std::string [vectorEncloser](#)
- std::string [vectorElementDelimiter](#)

#### 13.45.1 Detailed Description

Options for the format of files that are output to or input from a [CX\\_DataFrame](#).

The documentation for this class was generated from the following file:

- [CX\\_DataFrame.h](#)

### 13.46 CX::Algo::LatinSquare Class Reference

```
#include <CX_Algorithm.h>
```

#### Public Member Functions

- [LatinSquare](#) (void)  
*Construct a [LatinSquare](#) with no contents.*
- [LatinSquare](#) (unsigned int dimensions)
- void [generate](#) (unsigned int dimensions)  
*Construct a [LatinSquare](#) with no contents.*



- void [reorderRight](#) (void)
- void [reorderLeft](#) (void)
- void [reorderUp](#) (void)
- void [reorderDown](#) (void)
- void [reverseColumns](#) (void)
- void [reverseRows](#) (void)
- void [swapColumns](#) (unsigned int c1, unsigned int c2)
- void [swapRows](#) (unsigned int r1, unsigned int r2)
- bool [appendRight](#) (const [LatinSquare](#) &ls)
- bool [appendBelow](#) (const [LatinSquare](#) &ls)
- [LatinSquare](#) & [operator+=](#) (unsigned int value)
- std::string [print](#) (std::string delim=",")
- bool [validate](#) (void) const
- unsigned int [columns](#) (void) const
- unsigned int [rows](#) (void) const
- std::vector< unsigned int > [getColumn](#) (unsigned int col) const
- std::vector< unsigned int > [getRow](#) (unsigned int row) const

#### Public Attributes

- std::vector< std::vector  
< unsigned int > > [square](#)  
*The Latin square.*

#### 13.46.1 Detailed Description

This class provides a way to work with Latin squares in a relatively easy way.

```
Algo::LatinSquare ls(4); //Construct a standard 4x4 LatinSquare.
cout << "This latin square has " << ls.rows() << " rows and " << ls.columns() << " columns." << endl;
cout << ls.print() << endl;

ls.reverseColumns();
cout << "Reverse the columns: " << endl << ls.print() << endl;

ls.swapRows(0, 2);
cout << "Swap rows 0 and 2: " << endl << ls.print() << endl;

if (ls.validate()) {
    cout << "The latin square is still a valid latin square." << endl;
}

cout << "Let's copy, reverse, and append a latin square." << endl;
Algo::LatinSquare sq = ls;
sq.reverseColumns();
ls.appendBelow(sq);

cout << ls.print() << endl;
if (!ls.validate()) {
    cout << "The latin square is no longer valid, but it is still useful (8 counterbalancing conditions,
        both forward and backward ordering)." << endl;
}
```

#### 13.46.2 Constructor & Destructor Documentation

##### 13.46.2.1 [LatinSquare::LatinSquare](#) ( unsigned int *dimensions* )

Construct a [LatinSquare](#) with the given dimensions. The generated square is the basic latin square that, for dimension 3, has {0,1,2} on the first row, {1,2,0} on the middle row, and {2,0,1} on the last row.

### 13.46.3 Member Function Documentation

#### 13.46.3.1 `bool LatinSquare::appendBelow ( const LatinSquare & ls )`

Appends another [LatinSquare](#) (ls) below of this one. If the number of columns of both latin squares is not equal, this has no effect and returns false.

#### 13.46.3.2 `bool LatinSquare::appendRight ( const LatinSquare & ls )`

Appends another [LatinSquare](#) (ls) to the right of this one. If the number of rows of both latin squares is not equal, this has no effect and returns false.

#### 13.46.3.3 `unsigned int LatinSquare::columns ( void ) const`

Returns the number of columns.

#### 13.46.3.4 `void LatinSquare::generate ( unsigned int dimensions )`

Construct a [LatinSquare](#) with no contents.

#### Note

This deletes any previous contents of the latin square.

#### 13.46.3.5 `std::vector< unsigned int > LatinSquare::getColumn ( unsigned int col ) const`

Returns a copy of the given column. Throws `std::out_of_range` if the column is out of range.

#### 13.46.3.6 `std::vector< unsigned int > LatinSquare::getRow ( unsigned int row ) const`

Returns a copy of the given row. Throws `std::out_of_range` if the row is out of range.

#### 13.46.3.7 `LatinSquare & LatinSquare::operator+=( unsigned int value )`

Adds the given value to all of the values in the latin square.

#### 13.46.3.8 `std::string LatinSquare::print ( std::string delim = " , " )`

Prints the contents of the latin square to a string with the given delimiter between elements of the latin square.

#### 13.46.3.9 `void LatinSquare::reorderDown ( void )`

This function moves all of the rows down one place, then moves the bottommost row to the top.

#### 13.46.3.10 `void LatinSquare::reorderLeft ( void )`

This function shifts the columns to the left and the first column is moved to be the last column.

#### 13.46.3.11 `void LatinSquare::reorderRight ( void )`

This function shifts the columns to the right and the last column is moved to be the first column.

#### 13.46.3.12 `void LatinSquare::reorderUp ( void )`

This function moves all of the rows up one place, then moves the topmost row to the bottom.

13.46.3.13 void LatinSquare::reverseColumns ( void )

Reverses the order of the columns in the latin square.

13.46.3.14 void LatinSquare::reverseRows ( void )

Reverses the order of the rows in the latin square.

13.46.3.15 unsigned int LatinSquare::rows ( void ) const

Returns the number of rows.

13.46.3.16 void LatinSquare::swapColumns ( unsigned int *c1*, unsigned int *c2* )

Swap the given columns. If either column is out of range, this function has no effect.

13.46.3.17 void LatinSquare::swapRows ( unsigned int *r1*, unsigned int *r2* )

Swap the given rows. If either row is out of range, this function has no effect.

13.46.3.18 bool LatinSquare::validate ( void ) const

Checks to make sure that the latin square held by this instance is a valid latin square.

The documentation for this class was generated from the following files:

- CX\_Algorithm.h
- CX\_Algorithm.cpp

## 13.47 CX::Synth::Mixer Class Reference

```
#include <CX_ModularSynth.h>
```

Inherits [CX::Synth::ModuleBase](#).

### Public Member Functions

- double [getNextSample](#) (void) override

### Additional Inherited Members

#### 13.47.1 Detailed Description

This class mixes together a number of inputs. It does no mixing in the usual sense of setting levels of the inputs. Use Multipliers on the inputs for that. This class simply adds together all of the inputs with no amplitude correction, so it is possible for the output of the mixer to have very large amplitudes.

This class is special in that it can have more than one input.

#### 13.47.2 Member Function Documentation

13.47.2.1 double Mixer::getNextSample ( void ) `[override], [virtual]`

This function should be overloaded for any derived class that can be used as the input for another module.

Reimplemented from [CX::Synth::ModuleBase](#).

The documentation for this class was generated from the following files:

- [CX\\_ModularSynth.h](#)
- [CX\\_ModularSynth.cpp](#)

### 13.48 CX::Synth::ModuleBase Class Reference

```
#include <CX_ModularSynth.h>
```

Inherited by [CX::Synth::Adder](#), [CX::Synth::AdditiveSynth](#), [CX::Synth::Clamper](#), [CX::Synth::Envelope](#), [CX::Synth::Filter](#), [CX::Synth::FIRFilter](#), [CX::Synth::GenericOutput](#), [CX::Synth::Mixer](#), [CX::Synth::Multiplier](#), [CX::Synth::Oscillator](#), [CX::Synth::SoundBufferInput](#), [CX::Synth::SoundBufferOutput](#), [CX::Synth::Splitter](#), [CX::Synth::StreamOutput](#), and [CX::Synth::TrivialGenerator](#).

#### Public Member Functions

- virtual double [getNextSample](#) (void)
- void [setData](#) (ModuleControlData\_t d)
- ModuleControlData\_t [getData](#) (void)
- void [disconnectInput](#) (ModuleBase \*in)
- void [disconnectOutput](#) (ModuleBase \*out)

#### Protected Member Functions

- virtual void [\\_dataSetEvent](#) (void)
- void [\\_dataSet](#) (ModuleBase \*caller)
- void [\\_setDataIfNotSet](#) (ModuleBase \*target)
- void [\\_registerParameter](#) (ModuleParameter \*p)
- virtual void [\\_assignInput](#) (ModuleBase \*in)
- virtual void [\\_assignOutput](#) (ModuleBase \*out)
- virtual int [\\_maxInputs](#) (void)
- virtual int [\\_maxOutputs](#) (void)
- virtual void [\\_inputAssignedEvent](#) (ModuleBase \*in)
- virtual void [\\_outputAssignedEvent](#) (ModuleBase \*out)

#### Protected Attributes

- vector< [ModuleBase](#) \* > [\\_inputs](#)
- vector< [ModuleBase](#) \* > [\\_outputs](#)
- vector< [ModuleParameter](#) \* > [\\_parameters](#)
- ModuleControlData\_t \* [\\_data](#)

#### Friends

- [ModuleBase](#) & [operator>>](#) ([ModuleBase](#) &l, [ModuleBase](#) &r)
- void [operator>>](#) ([ModuleBase](#) &l, [ModuleParameter](#) &r)

### 13.48.1 Detailed Description

All modules of the modular synth inherit from this class.

### 13.48.2 Member Function Documentation

#### 13.48.2.1 void ModuleBase::\_registerParameter ( ModuleParameter \* p ) [protected]

If you are using a [CX::Synth::ModuleParameter](#) in your module, you must register that [ModuleParameter](#) during construction (or setup) of the module using this function.

```
class MyModule : public ModuleBase {
public:
    MyModule(void) {
        this->_registerParameter (&myParam);
        //...
    }

    ModuleParameter myParam;
    //...
};
```

#### 13.48.2.2 void ModuleBase::disconnectInput ( ModuleBase \* in )

This is a reciprocal operation: This module's input is disconnected and *in*'s output to this module is disconnected.

#### 13.48.2.3 void ModuleBase::disconnectOutput ( ModuleBase \* out )

This is a reciprocal operation: This module's output is disconnected and *out*'s input from this module is disconnected.

#### 13.48.2.4 ModuleControlData\_t ModuleBase::getData ( void )

Gets the data used by the module.

#### 13.48.2.5 virtual double CX::Synth::ModuleBase::getNextSample ( void ) [inline],[virtual]

This function should be overloaded for any derived class that can be used as the input for another module.

Reimplemented in [CX::Synth::FIRFilter](#), [CX::Synth::TrivialGenerator](#), [CX::Synth::SoundBufferInput](#), [CX::Synth::Splitter](#), [CX::Synth::Oscillator](#), [CX::Synth::Multiplier](#), [CX::Synth::Mixer](#), [CX::Synth::GenericOutput](#), [CX::Synth::Filter](#), [CX::Synth::Envelope](#), [CX::Synth::Clamper](#), [CX::Synth::Adder](#), and [CX::Synth::AdditiveSynth](#).

#### 13.48.2.6 void ModuleBase::setData ( ModuleControlData\_t d )

This function sets the data needed by this module in order to function properly. Many modules need this data, specifically the sample rate that the synth using. If several modules are connected together, you will only need to set the data for one module and the change will propagate to the other connected modules automatically.

This function does not usually need to be called directly by the user. If an appropriate input or output is connected, the data will be set from that module. However, there are some cases where a pattern of reconnecting previously used modules may result in inappropriate sample rates being set. For that reason, if you are having a problem with seeing the correct sample rate after reconnecting some modules, try manually calling [setData\(\)](#).

#### Parameters

---

<i>d</i>	The data to set.
----------	------------------

The documentation for this class was generated from the following files:

- CX\_ModularSynth.h
- CX\_ModularSynth.cpp

### 13.49 CX::Synth::ModuleParameter Class Reference

```
#include <CX_ModularSynth.h>
```

#### Public Member Functions

- [ModuleParameter](#) (void)  
*Construct a [ModuleParameter](#) with no value.*
- [ModuleParameter](#) (double d)  
*Construct a [ModuleParameter](#) with the given start value.*
- void [updateValue](#) (void)
- bool [valueUpdated](#) (bool checkForUpdates=true)
- double & [getValue](#) (void)
- [operator double](#) (void)  
*Implicitly converts the parameter to *double*.*
- [ModuleParameter](#) & [operator=](#) (double d)  
*Assign a value to the parameter.*

#### Friends

- class **ModuleBase**
- void **operator>>** ([ModuleBase](#) &l, [ModuleParameter](#) &r)

#### 13.49.1 Detailed Description

This class is used to provide modules with the ability to have their control parameters change as a function of incoming data from other modules. For example, if you want to change the frequency of an oscillator, you can feed an LFO into the frequency parameter of the oscillator.

If you create a module that uses a [ModuleParameter](#), you must perform one setup step in the constructor of the module. You must call [ModuleBase::\\_registerParameter\(\)](#) with the [ModuleParameter](#) as the argument.

#### 13.49.2 Member Function Documentation

##### 13.49.2.1 double & ModuleParameter::getValue ( void )

Gets the current value of the parameter.

##### 13.49.2.2 void ModuleParameter::updateValue ( void )

Update the value of the module parameter. This gets the next sample from the module that is the input for the [ModuleParameter](#), if any.

### 13.49.2.3 bool ModuleParameter::valueUpdated ( bool *checkForUpdates* = true )

Returns `true` if the value of the [ModuleParameter](#) has been updated since the last time this function was called. This should be called right after [updateValue\(\)](#) or with `checkForUpdates = true`. Updates to the value resulting from assignment of a new value with `operator=()` count as updates to the value.

If you don't care whether the value has been updated before using it, don't call this function. Instead, just use [updateValue\(\)](#) and [getValue\(\)](#).

#### Parameters

<i>checkForUpdates</i>	Check for updates before determining whether the value has been updated.
------------------------	--

#### Returns

`true` if the value has been updated since the last check.

The documentation for this class was generated from the following files:

- CX\_ModularSynth.h
- CX\_ModularSynth.cpp

## 13.50 CX::Synth::Multiplier Class Reference

```
#include <CX_ModularSynth.h>
```

Inherits [CX::Synth::ModuleBase](#).

#### Public Member Functions

- double [getNextSample](#) (void) override
- void [setGain](#) (double decibels)

#### Public Attributes

- [ModuleParameter](#) **amount**

#### Additional Inherited Members

### 13.50.1 Detailed Description

This class multiplies an input by an `amount`. You can set the amount in terms of decibels of gain by using the [setGain\(\)](#) function. If there is no input to this module, it behaves as though the input was 0 and consequently outputs 0.

### 13.50.2 Member Function Documentation

#### 13.50.2.1 double Multiplier::getNextSample ( void ) [override], [virtual]

This function should be overloaded for any derived class that can be used as the input for another module.

Reimplemented from [CX::Synth::ModuleBase](#).

### 13.50.2.2 void Multiplier::setGain ( double *decibels* )

Sets the `amount` of the multiplier based on gain in decibels.



## Parameters

<i>decibels</i>	The gain to apply. If greater than 0, <code>amount</code> will be greater than 1. If less than 0, <code>amount</code> will be less than 1. After calling this function, <code>amount</code> will never be negative.
-----------------	---

The documentation for this class was generated from the following files:

- CX\_ModularSynth.h
- CX\_ModularSynth.cpp

## 13.51 CX::Synth::Oscillator Class Reference

```
#include <CX_ModularSynth.h>
```

Inherits [CX::Synth::ModuleBase](#).

## Public Member Functions

- double [getNextSample](#) (void) override
- void [setGeneratorFunction](#) (std::function< double(double)> f)

## Static Public Member Functions

- static double **saw** (double wp)
- static double **sine** (double wp)
- static double **square** (double wp)
- static double **triangle** (double wp)
- static double **whiteNoise** (double wp)

## Public Attributes

- [ModuleParameter](#) **frequency**

## Additional Inherited Members

## 13.51.1 Detailed Description

This class provides one of the simplest ways of generating waveforms. The output from an [Oscillator](#) can be filtered with a [CX::Synth::Filter](#) or used in other ways.

```
using namespace CX::Synth;
//Configure the oscillator to produce a square wave with a fundamental frequency of 200 Hz.
Oscillator osc;
osc.frequency = 200; //200 Hz
osc.setGeneratorFunction(Oscillator::square); //Produce a square wave
```

## 13.51.2 Member Function Documentation

## 13.51.2.1 double Oscillator::getNextSample ( void ) [override],[virtual]

This function should be overloaded for any derived class that can be used as the input for another module.

Reimplemented from [CX::Synth::ModuleBase](#).

### 13.51.2.2 void Oscillator::setGeneratorFunction ( std::function< double(double)> f )

It is very easy to make your own waveform generating functions to be used with an [Oscillator](#). A waveform generating function takes a value that represents the location in the waveform at the current point in time. These values are in the interval [0,1).

The waveform generating function should return a double representing the amplitude of the wave at the given waveform position.

To put this all together, a sine wave generator looks like this:

```
double sineWaveGeneratorFunction(double waveformPosition) {
    return sin(2 * PI * waveformPosition); //The argument for sin() is in radians. 1 cycle is 2*PI radians.
}
```

The documentation for this class was generated from the following files:

- CX\_ModularSynth.h
- CX\_ModularSynth.cpp

## 13.52 CX::CX\_SoundStream::OutputEventArgs Struct Reference

```
#include <CX_SoundStream.h>
```

### Public Attributes

- bool [bufferUnderflow](#)  
*This is set to true if there was a buffer underflow, which means that the sound hardware ran out of data to output.*
- float \* [outputBuffer](#)  
*A pointer to an array that should be filled with sound data.*
- unsigned int [bufferSize](#)  
*The number of sample frames that are in outputBuffer. The total number of samples is bufferSize \* outputChannels.*
- int [outputChannels](#)  
*The number of channels worth of data in outputBuffer.*
- CX\_SoundStream \* [instance](#)  
*A pointer to the CX\_SoundStream instance that notified this output event.*

### 13.52.1 Detailed Description

The audio output event of the [CX\\_SoundStream](#) sends a copy of this structure with the fields filled out when the event is called.

The documentation for this struct was generated from the following file:

- CX\_SoundStream.h

## 13.53 CX::CX\_Time\_t< T >::PartitionedTime Struct Reference

```
#include <CX_Time_t.h>
```

## Public Attributes

- int [hours](#)  
*The hours component of the time.*
- int [minutes](#)  
*The minutes component of the time.*
- int [seconds](#)  
*The seconds component of the time.*
- int [milliseconds](#)  
*The milliseconds component of the time.*
- int [microseconds](#)  
*The microseconds component of the time.*
- int [nanoseconds](#)  
*The nanoseconds component of the time.*

## 13.53.1 Detailed Description

```
template<typename T>struct CX::CX_Time_t< T >::PartitionedTime
```

This struct contains the result of [CX\\_Time\\_t::getPartitionedTime\(\)](#).

The documentation for this struct was generated from the following file:

- [CX\\_Time\\_t.h](#)

## 13.54 CX::CX\_SlidePresenter::PresentationErrorInfo Struct Reference

```
#include <CX_SlidePresenter.h>
```

## Public Member Functions

- unsigned int [totalErrors](#) (void)  
*Returns the sum of the different types of errors that are measured.*

## Public Attributes

- bool [presentationErrorsSuccessfullyChecked](#)  
*True if presentation errors were successfully checked for. This does not mean that there were no presentation errors, but that there were no presentation error checking errors.*
- unsigned int [incorrectFrameCounts](#)
- unsigned int [lateCopiesToBackBuffer](#)  
*The number of slides for which the time at which the slide finished being copied to the back buffer was after the actual start time of the slide.*

## 13.54.1 Detailed Description

This struct contains information about errors that were detected during slide presentation. See [CX\\_SlidePresenter::checkForPresentationErrors\(\)](#).

### 13.54.2 Member Data Documentation

#### 13.54.2.1 unsigned int CX::CX\_SlidePresenter::PresentationErrorInfo::incorrectFrameCounts

The number of slides for which the actual and intended frame counts did not match, indicating that the slide was presented for too many or too few frames.

The documentation for this struct was generated from the following file:

- CX\_SlidePresenter.h

### 13.55 CX::CX\_SlidePresenter::Slide Struct Reference

```
#include <CX_SlidePresenter.h>
```

#### Public Types

- enum {  
**NOT\_STARTED, COPY\_TO\_BACK\_BUFFER\_PENDING, SWAP\_PENDING, IN\_PROGRESS, FINISHED** }

*Status of the current slide vis a vis presentation. This should not be modified by the user.*

#### Public Attributes

- std::string [slideName](#)  
*The name of the slide. Set by the user during slide creation.*
- ofFbo [framebuffer](#)  
*A framebuffer containing image data that will be drawn to the screen during this slide's presentation. If drawingFunction points to a user function, framebuffer will not be drawn.*
- std::function< void(void)> [drawingFunction](#)  
*Pointer to a user function that will be called to draw the slide. If this points to a user function, it overrides `framebuffer`. The drawing function is not required to call `ofBackground()` or otherwise clear the display before drawing, which allows you to do what is essentially single-buffering using the back buffer as the framebuffer. However, if you want a blank framebuffer, you will have to clear it manually.*
- enum  
CX::CX\_SlidePresenter::Slide:: { ... } [slideStatus](#)  
*Status of the current slide vis a vis presentation. This should not be modified by the user.*
- [SlideTimingInfo intended](#)  
*The intended timing parameters (i.e. what should have happened if there were no presentation errors).*
- [SlideTimingInfo actual](#)  
*The actual timing parameters.*
- CX\_Millis [copyToBackBufferCompleteTime](#)  
*The time at which the drawing operations for this slide finished. This is pretty useful to determine if there was an error on the trial (e.g. framebuffer was copied late). If this is greater than `actual.startTime`, the slide may not have been fully drawn at the time the front and back buffers swapped.*

### 13.55.1 Detailed Description

This struct contains information related to slide presentation using [CX\\_SlidePresenter](#).

The documentation for this struct was generated from the following file:

- [CX\\_SlidePresenter.h](#)

## 13.56 CX::CX\_SlidePresenter::SlideTimingInfo Struct Reference

```
#include <CX_SlidePresenter.h>
```

### Public Attributes

- [uint32\\_t startFrame](#)  
*The frame on which the slide started/should have started. Can be compared with the value given by [Display.getFrameNumber\(\)](#).*
- [uint32\\_t frameCount](#)  
*The number of frames the slide was/should be presented for.*
- [CX\\_Millis startTime](#)  
*The time at which the slide was/should have been started. Can be compared with values from [CX::CX\\_Clock::now\(\)](#).*
- [CX\\_Millis duration](#)  
*The amount of time the slide was/should have been presented for.*

### 13.56.1 Detailed Description

Contains information about the presentation timing of the slide.

The documentation for this struct was generated from the following file:

- [CX\\_SlidePresenter.h](#)

## 13.57 CX::Synth::SoundBufferInput Class Reference

```
#include <CX_ModularSynth.h>
```

Inherits [CX::Synth::ModuleBase](#).

### Public Member Functions

- double [getNextSample](#) (void) override
- void [setSoundBuffer](#) ([CX::CX\\_SoundBuffer](#) \*sb, unsigned int channel=0)
- void [setTime](#) ([CX\\_Millis](#) t)
- bool [canPlay](#) (void)

## Additional Inherited Members

### 13.57.1 Detailed Description

This class allows you to use a [CX\\_SoundBuffer](#) as the input for the modular synth. It is strictly monophonic, so when you associate a [CX\\_SoundBuffer](#) with this class, you must pick one channel of the sound to use. You can use multiple [SoundBufferInput](#)s to play multiple channels from the same [CX\\_SoundBuffer](#).

### 13.57.2 Member Function Documentation

#### 13.57.2.1 `bool SoundBufferInput::canPlay ( void )`

Checks to see if the [CX\\_SoundBuffer](#) that is associated with this [SoundBufferInput](#) is able to play. It is unable to play if [CX\\_SoundBuffer::isReadyToPlay\(\)](#) is false or if the whole sound has been played.

#### 13.57.2.2 `double SoundBufferInput::getNextSample ( void ) [override], [virtual]`

This function should be overloaded for any derived class that can be used as the input for another module.

Reimplemented from [CX::Synth::ModuleBase](#).

#### 13.57.2.3 `void SoundBufferInput::setSoundBuffer ( CX::CX_SoundBuffer * sb, unsigned int channel = 0 )`

This function sets the [CX\\_SoundBuffer](#) from which data will be drawn. Because the [SoundBufferInput](#) is monophonic, you must pick one channel of the [CX\\_SoundBuffer](#) to use.

#### Parameters

<i>sb</i>	The <a href="#">CX_SoundBuffer</a> to use. Because this <a href="#">CX_SoundBuffer</a> is taken as a pointer and is not copied, you should make sure that <i>sb</i> remains in existence and unmodified while the <a href="#">SoundBufferInput</a> is in use.
<i>channel</i>	The channel of the <a href="#">CX_SoundBuffer</a> to use.

#### 13.57.2.4 `void SoundBufferInput::setTime ( CX::CX_Millis t )`

Set the playback time of the current [CX\\_SoundBuffer](#). When playback starts, it will start from this time. If playback is in progress, playback will skip to the selected time.

The documentation for this class was generated from the following files:

- [CX\\_ModularSynth.h](#)
- [CX\\_ModularSynth.cpp](#)

## 13.58 CX::Synth::SoundBufferOutput Class Reference

```
#include <CX_ModularSynth.h>
```

Inherits [CX::Synth::ModuleBase](#).

### Public Member Functions

- `void setup` (float sampleRate)
- `void sampleData` (CX\_Millis t)

## Public Attributes

- [CX::CX\\_SoundBuffer](#) **sb**

## Additional Inherited Members

## 13.58.1 Detailed Description

This class provides a method of capturing the output of a modular synth and storing it in a [CX\\_SoundBuffer](#) for later use.

## 13.58.2 Member Function Documentation

13.58.2.1 void SoundBufferOutput::sampleData ( CX::CX\_Millis *t* )

This function samples *t* milliseconds of data at the sample rate given in [setup\(\)](#). The result is stored in the **sb** member of this class. If **sb** is not empty when this function is called, the data is appended to **sb**.

13.58.2.2 void SoundBufferOutput::setup ( float *sampleRate* )

Configure the output to use a particular sample rate. If this function is not called, the sample rate of the modular synth may be undefined.

## Parameters

<i>sampleRate</i>	The sample rate in Hz.
-------------------	------------------------

The documentation for this class was generated from the following files:

- CX\_ModularSynth.h
- CX\_ModularSynth.cpp

## 13.59 CX::Synth::Splitter Class Reference

```
#include <CX_ModularSynth.h>
```

Inherits [CX::Synth::ModuleBase](#).

## Public Member Functions

- double [getNextSample](#) (void) override

## Additional Inherited Members

## 13.59.1 Detailed Description

This class splits a signal and sends that signal to multiple outputs. This can be used for panning effects, for example.

This class is special because it allows multiple outputs.

```
using namespace CX::Synth;
Splitter sp;
Oscillator osc;
```

```
Multiplier m1;
Multiplier m2;
StereoStreamOutput out;

//In runExperiment:
osc >> sp;
sp >> m1 >> out.left;
sp >> m2 >> out.right;
```

### 13.59.2 Member Function Documentation

#### 13.59.2.1 double Splitter::getNextSample ( void ) [override],[virtual]

This function should be overloaded for any derived class that can be used as the input for another module.

Reimplemented from [CX::Synth::ModuleBase](#).

The documentation for this class was generated from the following files:

- [CX\\_ModularSynth.h](#)
- [CX\\_ModularSynth.cpp](#)

## 13.60 CX::Synth::StereoSoundBufferOutput Class Reference

```
#include <CX_ModularSynth.h>
```

### Public Member Functions

- void [setup](#) (float sampleRate)
- void [sampleData](#) ([CX\\_Millis](#) t)

### Public Attributes

- [GenericOutput](#) left  
*The left channel of the buffer.*
- [GenericOutput](#) right  
*The right channel of the buffer.*
- [CX::CX\\_SoundBuffer](#) sb

#### 13.60.1 Detailed Description

This class provides a method of capturing the output of a modular synth and storing it in a [CX\\_SoundBuffer](#) for later use. This captures stereo audio by taking the output of different streams of data into either the `left` or `right` modules that this class has. See the example code.

```
#include "CX_EntryPoint.h"

using namespace CX::Synth;

void runExperiment(void) {
    StereoSoundBufferOutput sout;
    sout.setup(44100);

    Splitter sp;
    Oscillator osc;
    Multiplier leftM;
    Multiplier rightM;
```



```

osc.frequency = 400;
leftM.amount = .1;
rightM.amount = .01;

osc >> sp;
sp >> leftM >> sout.left;
sp >> rightM >> sout.right;

sout.sampleData(CX_Seconds(2)); //Sample 2 seconds worth of data on both channels.
sout.sb.writeToFile("Stereo.wav");
}

```

### 13.60.2 Member Function Documentation

#### 13.60.2.1 void StereoSoundBufferOutput::sampleData ( CX::CX\_Millis *t* )

This function samples *t* milliseconds of data at the sample rate given in [setup\(\)](#). The result is stored in the *sb* member of this class. If *sb* is not empty when this function is called, the data is appended to *sb*.

#### 13.60.2.2 void StereoSoundBufferOutput::setup ( float *sampleRate* )

Configure the output to use a particular sample rate. If this function is not called, the sample rate of the modular synth may be undefined.

##### Parameters

<i>sampleRate</i>	The sample rate in Hz.
-------------------	------------------------

The documentation for this class was generated from the following files:

- CX\_ModularSynth.h
- CX\_ModularSynth.cpp

### 13.61 CX::Synth::StereoStreamOutput Class Reference

```
#include <CX_ModularSynth.h>
```

#### Public Member Functions

- void **setOutputStream** (CX::CX\_SoundStream \*stream)

#### Public Attributes

- [GenericOutput](#) **left**
- [GenericOutput](#) **right**

#### 13.61.1 Detailed Description

This class is much like [StreamOutput](#) except in stereo. This captures stereo audio by taking the output of different streams of data into either the *left* or *right* modules that this class has. See the example code for [CX::Synth::StereoSoundBufferOutput](#) and [CX::Synth::StreamOutput](#) for ideas on how to use this class.

The documentation for this class was generated from the following files:

- CX\_ModularSynth.h
- CX\_ModularSynth.cpp

## 13.62 CX::Synth::StreamOutput Class Reference

```
#include <CX_ModularSynth.h>
```

Inherits [CX::Synth::ModuleBase](#).

### Public Member Functions

- void **setOutputStream** ([CX::CX\\_SoundStream](#) \*stream)

### Additional Inherited Members

#### 13.62.1 Detailed Description

This class provides a method of playing the output of a modular synth using a [CX\\_SoundStream](#). In order to use this class, you need to configure a [CX\\_SoundStream](#) for use. See the `soundBuffer` example and the [CX::CX\\_SoundStream](#) class for more information.

```
using namespace CX::Synth;
//Assume that both osc and ss have been configured and that ss has been started.
CX_SoundStream ss;
Oscillator osc;

Synth::StreamOutput output;
output.setOutputStream(&ss);

osc >> output; //Sound should be playing past this point.
```

The documentation for this class was generated from the following files:

- CX\_ModularSynth.h
- CX\_ModularSynth.cpp

## 13.63 CX::Synth::TrivialGenerator Class Reference

```
#include <CX_ModularSynth.h>
```

Inherits [CX::Synth::ModuleBase](#).

### Public Member Functions

- double **getNextSample** (void) override

### Public Attributes

- [ModuleParameter](#) **value**
- [ModuleParameter](#) **step**

### Additional Inherited Members

#### 13.63.1 Detailed Description

This class is used for numerically, rather than auditorily, testing other modules. It produces samples starting at `value` and increasing by `step`.

### 13.63.2 Member Function Documentation

#### 13.63.2.1 double TrivialGenerator::getNextSample ( void ) [override],[virtual]

This function should be overloaded for any derived class that can be used as the input for another module.

Reimplemented from [CX::Synth::ModuleBase](#).

The documentation for this class was generated from the following files:

- CX\_ModularSynth.h
- CX\_ModularSynth.cpp

## Index

- `_registerParameter`
    - `CX::Synth::ModuleBase`, 165
- `AXIS_POSITION_CHANGE`
  - `CX::CX_Joystick::Event`, 155
- `addColumn`
  - `CX::CX_DataFrame`, 71
- `addSilence`
  - `CX::CX_SoundBuffer`, 130
- `addSound`
  - `CX::CX_SoundBuffer`, 130
- `angle`
  - `CX::Draw::CX_PatternProperties_t`, 110
- `apertureType`
  - `CX::Draw::CX_PatternProperties_t`, 110
- `api`
  - `CX::CX_SoundStream::Configuration`, 61
- `append`
  - `CX::CX_DataFrame`, 71
- `appendBelow`
  - `CX::Algo::LatinSquare`, 162
- `appendFunction`
  - `CX::Util::CX_TrialController`, 151
- `appendRight`
  - `CX::Algo::LatinSquare`, 162
- `appendRow`
  - `CX::CX_DataFrame`, 71
- `appendSlide`
  - `CX::CX_SlidePresenter`, 124
- `appendSlideFunction`
  - `CX::CX_SlidePresenter`, 125
- `applyGain`
  - `CX::CX_SoundBuffer`, 131
- `arc`
  - `CX::Draw`, 29
- `arrayToVector`
  - `CX::Util`, 44
- `arrowToPath`
  - `CX::Draw`, 30
- `at`
  - `CX::CX_DataFrame`, 71
- `availableEvents`
  - `CX::CX_Joystick`, 96
  - `CX::CX_Keyboard`, 98
  - `CX::CX_Mouse`, 108
- `BUTTON_PRESS`
  - `CX::CX_Joystick::Event`, 155
- `BUTTON_RELEASE`
  - `CX::CX_Joystick::Event`, 155
- `bandwidth`
  - `CX::Synth::Filter`, 157
- `beginDrawingNextSlide`
  - `CX::CX_SlidePresenter`, 126
- `beginDrawingToBackBuffer`
  - `CX::CX_Display`, 87
- `bezier`
  - `CX::Draw`, 30
- `BlockSampler`
  - `CX::Algo::BlockSampler`, 58
- `bufferSize`
  - `CX::CX_SoundStream::Configuration`, 61
- `Buttons`
  - `CX::CX_Mouse`, 108
- `CX::CX_Joystick::Event`
  - `AXIS_POSITION_CHANGE`, 155
  - `BUTTON_PRESS`, 155
  - `BUTTON_RELEASE`, 155
- `CX::CX_Keyboard`
  - `PRESSED`, 98
  - `RELEASED`, 98
  - `REPEAT`, 98
- `CX::CX_Mouse`
  - `DRAGGED`, 108
  - `MOVED`, 108
  - `PRESSED`, 108
  - `RELEASED`, 108
  - `SCROLLED`, 108
- `CX::Util`
  - `ROUND_DOWN`, 43
  - `ROUND_TO_NEAREST`, 43
  - `ROUND_TOWARD_ZERO`, 43
  - `ROUND_UP`, 43
- `CX`, 23
  - `operator<<`, 24
  - `operator>>`, 25
  - `reopenWindow`, 25
- `CX::Algo`, 25
  - `fullyCross`, 26
  - `generateSeparatedValues`, 27
- `CX::Algo::BlockSampler`
  - `BlockSampler`, 58
  - `getBlockNumber`, 58
  - `getBlockPosition`, 58
  - `getNextValue`, 59
  - `restartSampling`, 59
  - `setup`, 59
- `CX::Algo::BlockSampler< T >`, 58
- `CX::Algo::LatinSquare`, 160
  - `appendBelow`, 162
  - `appendRight`, 162
  - `columns`, 162

- generate, 162
- getColumn, 162
- getRow, 162
- LatinSquare, 161
- operator+=, 162
- print, 162
- reorderDown, 162
- reorderLeft, 162
- reorderRight, 162
- reorderUp, 162
- reverseColumns, 162
- reverseRows, 163
- rows, 163
- swapColumns, 163
- swapRows, 163
- validate, 163
- CX::CX\_BaseClockImplementation, 62
- CX::CX\_Clock, 63
  - delay, 63
  - getDateTimeString, 63
  - getExperimentStartDateTimeString, 65
  - now, 65
  - precisionTest, 65
  - resetExperimentStartTime, 65
  - setImplementation, 65
  - sleep, 66
- CX::CX\_DataFrame, 69
  - addColumn, 71
  - append, 71
  - appendRow, 71
  - at, 71
  - clear, 73
  - convertVectorColumnToColumns, 73
  - copyColumn, 73
  - copyColumns, 74
  - copyRows, 74
  - deleteColumn, 74
  - deleteRow, 75
  - getColumnNames, 75
  - getRowCount, 75
  - operator(), 75
  - operator=, 75
  - print, 76
  - printToFile, 76, 77
  - readFromFile, 77
  - reorderRows, 78
  - setRowCount, 78
  - shuffleRows, 78
- CX::CX\_DataFrame::loOptions, 160
- CX::CX\_DataFrameCell, 79
  - CX\_DataFrameCell, 80
  - copyCellTo, 80
  - getStoredType, 80
  - isVector, 80
- operator=, 81
- store, 81
- storeVector, 81
- to, 81
- toString, 81
- toVector, 81, 82
- CX::CX\_DataFrameColumn, 82
  - size, 82
- CX::CX\_DataFrameRow, 83
- CX::CX\_Display, 86
  - beginDrawingToBackBuffer, 87
  - configureFromFile, 87
  - copyFboToBackBuffer, 88
  - endDrawingToBackBuffer, 89
  - estimateFramePeriod, 89
  - estimateNextSwapTime, 89
  - getCenterOfDisplay, 89
  - getFrameNumber, 89
  - getFramePeriod, 89
  - getFramePeriodStandardDeviation, 89
  - getLastSwapTime, 90
  - getResolution, 90
  - hasSwappedSinceLastCheck, 90
  - isAutomaticallySwapping, 90
  - setAutomaticSwapping, 90
  - setFullScreen, 90
  - setWindowResolution, 91
  - setWindowTitle, 91
  - setup, 90
  - swapBuffers, 91
  - swapBuffersInThread, 91
  - testBufferSwapping, 91
  - useHardwareVSync, 92
  - useSoftwareVSync, 94
  - waitForOpenGL, 94
- CX::CX\_InputManager, 94
  - clearAllEvents, 95
  - pollEvents, 95
  - setup, 95
- CX::CX\_Joystick, 96
  - availableEvents, 96
  - clearEvents, 96
  - getAxisPositions, 97
  - getButtonStates, 97
  - getJoystickName, 97
  - getNextEvent, 97
  - pollEvents, 97
  - setup, 97
- CX::CX\_Joystick::Event, 155
  - JoystickEventType, 155
- CX::CX\_Keyboard, 97
  - availableEvents, 98
  - clearEvents, 98
  - enable, 99

- enabled, 99
- getNextEvent, 99
- isKeyHeld, 99
- KeyboardEventType, 98
- waitForKeypress, 99
- CX::CX\_Keyboard::Event, 154
  - key, 154
- CX::CX\_Logger, 104
  - captureOfLogMessages, 104
  - error, 104
  - fatalError, 104
  - flush, 105
  - getModuleLevel, 105
  - level, 105
  - levelForAllModules, 105
  - levelForConsole, 105
  - levelForFile, 105
  - log, 105
  - notice, 106
  - setMessageFlushCallback, 106
  - timestamps, 106
  - verbose, 106
  - warning, 106
- CX::CX\_MessageFlushData, 107
  - CX\_MessageFlushData, 107
- CX::CX\_Mouse, 107
  - availableEvents, 108
  - Buttons, 108
  - clearEvents, 108
  - enable, 109
  - enabled, 109
  - getCursorPosition, 109
  - getNextEvent, 109
  - MouseEventType, 108
  - setCursorPosition, 109
  - showCursor, 109
- CX::CX\_Mouse::Event, 156
- CX::CX\_RandomNumberGenerator, 111
  - CX\_RandomNumberGenerator, 112
  - getGenerator, 113
  - getMaximumRandomInt, 113
  - getMinimumRandomInt, 113
  - getSeed, 113
  - randomDouble, 113
  - randomInt, 114
  - sample, 114
  - sampleBinomialRealizations, 116
  - sampleBlocks, 116
  - sampleExclusive, 116, 117
  - sampleNormalRealizations, 118
  - sampleRealizations, 118
  - sampleUniformRealizations, 118
  - setSeed, 120
  - shuffleVector, 120
- CX::CX\_SlidePresenter, 122
  - appendSlide, 124
  - appendSlideFunction, 125
  - beginDrawingNextSlide, 126
  - checkForPresentationErrors, 126
  - clearSlides, 126
  - endDrawingCurrentSlide, 126
  - ErrorMode, 124
  - getActualFrameCounts, 127
  - getActualPresentationDurations, 127
  - getSlideByName, 127
  - getSlides, 127
  - presentSlides, 128
  - printLastPresentationInformation, 128
  - setup, 128
  - startSlidePresentation, 128
  - SwappingMode, 124
  - update, 129
- CX::CX\_SlidePresenter::Configuration, 60
- CX::CX\_SlidePresenter::FinalSlideFunctionArgs, 157
- CX::CX\_SlidePresenter::PresentationErrorInfo, 171
  - incorrectFrameCounts, 172
- CX::CX\_SlidePresenter::Slide, 172
- CX::CX\_SlidePresenter::SlideTimingInfo, 173
- CX::CX\_SoundBuffer, 129
  - addSilence, 130
  - addSound, 130
  - applyGain, 131
  - clear, 131
  - deleteAmount, 131
  - getLength, 131
  - getNegativePeak, 131
  - getPositivePeak, 132
  - getRawDataReference, 132
  - getSampleFrameCount, 132
  - getTotalSampleCount, 132
  - isLoadedSuccessfully, 132
  - isReadyToPlay, 132
  - loadFile, 132
  - multiplyAmplitudeBy, 133
  - multiplySpeed, 133
  - normalize, 133
  - resample, 133
  - reverse, 133
  - setChannelCount, 133
  - setFromVector, 134
  - setLength, 134
  - stripLeadingSilence, 134
  - writeToFile, 134
- CX::CX\_SoundBufferPlayer, 135
  - getConfiguration, 135
  - getSoundBuffer, 135
  - getSoundStream, 136
  - play, 136

- setSoundBuffer, [136](#)
  - setTime, [136](#)
  - setup, [136](#)
  - startPlayingAt, [137](#)
  - stop, [137](#)
- CX::CX\_SoundBufferRecorder, [137](#)
  - getConfiguration, [138](#)
  - getSoundBuffer, [138](#)
  - setSoundBuffer, [138](#)
  - setup, [138](#), [139](#)
  - startRecording, [139](#)
  - stopRecording, [139](#)
- CX::CX\_SoundStream, [139](#)
  - closeStream, [141](#)
  - convertApiToString, [141](#)
  - convertApisToString, [141](#)
  - convertApisToStrings, [141](#)
  - convertStringToApi, [141](#)
  - estimateNextSwapTime, [142](#)
  - formatsToString, [142](#)
  - formatsToStrings, [142](#)
  - getCompiledApis, [142](#)
  - getConfiguration, [142](#)
  - getDeviceList, [142](#)
  - getLastSwapTime, [144](#)
  - getRtAudioInstance, [144](#)
  - getSampleFrameNumber, [144](#)
  - getStreamLatency, [144](#)
  - hasSwappedSinceLastCheck, [144](#)
  - isStreamRunning, [144](#)
  - listDevices, [144](#)
  - readConfigurationFromFile, [145](#)
  - setup, [145](#)
  - start, [146](#)
  - stop, [146](#)
- CX::CX\_SoundStream::Configuration, [61](#)
  - api, [61](#)
  - bufferSize, [61](#)
  - sampleRate, [61](#)
  - streamOptions, [61](#)
- CX::CX\_SoundStream::InputEventArgs, [159](#)
- CX::CX\_SoundStream::OutputEventArgs, [170](#)
- CX::CX\_Time\_t
  - CX\_Time\_t, [149](#)
  - getPartitionedTime, [149](#)
  - operator+=, [150](#)
  - operator-=, [150](#)
  - standardDeviation, [150](#)
  - value, [150](#)
- CX::CX\_Time\_t< T >, [146](#)
- CX::CX\_Time\_t< T >::PartitionedTime, [170](#)
- CX::CX\_WindowConfiguration\_t, [152](#)
- CX::Draw, [28](#)
  - arc, [29](#)
  - arrowToPath, [30](#)
  - bezier, [30](#)
  - centeredString, [30](#)
  - colorArc, [31](#)
  - colorArcToVbo, [31](#)
  - colorWheel, [31](#)
  - colorWheelToVbo, [32](#)
  - convertColors, [32](#)
  - convertToRGB, [33](#)
  - fixationCross, [33](#)
  - fixationCrossToPath, [33](#)
  - gabor, [33](#)
  - getBezierVertices, [34](#)
  - getFixationCrossVertices, [34](#)
  - getRGBSpectrum, [34](#)
  - getStarVertices, [34](#)
  - greyscalePatternToPixels, [35](#)
  - line, [35](#)
  - lines, [35](#)
  - patternMask, [35](#)
  - ring, [37](#)
  - saveFboToFile, [37](#)
  - squircleToPath, [37](#)
  - star, [37](#)
  - starToPath, [39](#)
- CX::Draw::CX\_PatternProperties\_t, [110](#)
  - angle, [110](#)
  - apertureType, [110](#)
  - fallOffPower, [111](#)
  - height, [111](#)
  - maskType, [111](#)
  - width, [111](#)
- CX::Instances, [39](#)
- CX::Synth, [39](#)
  - operator>>, [40](#)
  - relativeFrequency, [40](#)
  - sinc, [42](#)
- CX::Synth::Adder, [54](#)
  - getNextSample, [54](#)
- CX::Synth::AdditiveSynth, [54](#)
  - calculateAmplitudes, [55](#)
  - getNextSample, [56](#)
  - HarmonicAmplitudeType, [55](#)
  - pruneLowAmplitudeHarmonics, [56](#)
  - setAmplitudes, [56](#)
  - setFundamentalFrequency, [57](#)
  - setHarmonicSeries, [57](#)
  - setStandardHarmonicSeries, [57](#)
- CX::Synth::Clamper, [59](#)
  - getNextSample, [60](#)
- CX::Synth::Envelope, [153](#)
  - gateInput, [153](#)
  - getNextSample, [153](#)
- CX::Synth::FIRFilter, [158](#)

- FilterType, 158
- getNextSample, 159
- CX::Synth::Filter, 156
  - bandwidth, 157
  - cutoff, 157
  - FilterType, 157
  - getNextSample, 157
- CX::Synth::GenericOutput, 159
  - getNextSample, 159
- CX::Synth::Mixer, 163
  - getNextSample, 163
- CX::Synth::ModuleBase, 164
  - \_registerParameter, 165
  - disconnectInput, 165
  - disconnectOutput, 165
  - getData, 165
  - getNextSample, 165
  - setData, 165
- CX::Synth::ModuleParameter, 166
  - getValue, 166
  - updateValue, 166
  - valueUpdated, 166
- CX::Synth::Multiplier, 167
  - getNextSample, 167
  - setGain, 167
- CX::Synth::Oscillator, 169
  - getNextSample, 169
  - setGeneratorFunction, 169
- CX::Synth::SoundBufferInput, 173
  - canPlay, 174
  - getNextSample, 174
  - setSoundBuffer, 174
  - setTime, 174
- CX::Synth::SoundBufferOutput, 174
  - sampleData, 175
  - setup, 175
- CX::Synth::Splitter, 175
  - getNextSample, 176
- CX::Synth::StereoSoundBufferOutput, 176
  - sampleData, 177
  - setup, 177
- CX::Synth::StereoStreamOutput, 177
- CX::Synth::StreamOutput, 178
- CX::Synth::TrivialGenerator, 178
  - getNextSample, 179
- CX::Util, 42
  - arrayToVector, 44
  - CX\_RoundingConfiguration, 43
  - checkOFVersion, 44
  - clamp, 44
  - concatenate, 45
  - degreesToPixels, 45
  - exclude, 45
  - getAngleBetweenPoints, 45
  - getMsaSampleCount, 46
  - getRelativePointFromDistanceAndAngle, 46
  - intVector, 46
  - max, 46
  - mean, 46, 48
  - min, 48
  - pixelsToDegrees, 48
  - readKeyValueFile, 48
  - repeat, 50
  - round, 51
  - sequence, 51
  - sequenceAlong, 51
  - sequenceSteps, 51
  - stringToVector, 52
  - unique, 52
  - var, 52, 53
  - vectorToString, 53
  - writeToFile, 53
- CX::Util::CX\_BaseUnitConverter, 62
  - inverse, 63
  - operator(), 63
- CX::Util::CX\_CoordinateConverter, 66
  - CX\_CoordinateConverter, 67
  - inverse, 67
  - operator(), 67, 68
  - setAxisInversion, 68
  - setMultiplier, 68
  - setOrigin, 68
  - setUnitConverter, 68
- CX::Util::CX\_DegreeToPixelConverter, 83
  - CX\_DegreeToPixelConverter, 84
  - configureFromFile, 84
  - inverse, 84
  - operator(), 84
  - setup, 86
- CX::Util::CX\_LapTimer, 100
  - CX\_LapTimer, 101
  - collectedSamples, 101
  - getStatString, 101
  - restart, 101
  - setup, 101
  - takeSample, 101
- CX::Util::CX\_LengthToPixelConverter, 102
  - CX\_LengthToPixelConverter, 102
  - configureFromFile, 102
  - inverse, 103
  - operator(), 103
  - setup, 103
- CX::Util::CX\_SegmentProfiler, 121
  - CX\_SegmentProfiler, 121
  - collectedSamples, 122
  - getStatString, 122
  - restart, 122
  - setup, 122



- t1, [122](#)
- t2, [122](#)
- CX::Util::CX\_TrialController, [150](#)
  - appendFunction, [151](#)
  - reset, [151](#)
  - setCurrentFunction, [151](#)
  - update, [152](#)
- CX\_CoordinateConverter
  - CX::Util::CX\_CoordinateConverter, [67](#)
- CX\_DataFrameCell
  - CX::CX\_DataFrameCell, [80](#)
- CX\_DegreeToPixelConverter
  - CX::Util::CX\_DegreeToPixelConverter, [84](#)
- CX\_LapTimer
  - CX::Util::CX\_LapTimer, [101](#)
- CX\_LengthToPixelConverter
  - CX::Util::CX\_LengthToPixelConverter, [102](#)
- CX\_LogLevel
  - Message Logging, [17](#)
- CX\_MessageFlushData
  - CX::CX\_MessageFlushData, [107](#)
- CX\_RandomNumberGenerator
  - CX::CX\_RandomNumberGenerator, [112](#)
- CX\_RoundingConfiguration
  - CX::Util, [43](#)
- CX\_SegmentProfiler
  - CX::Util::CX\_SegmentProfiler, [121](#)
- CX\_Time\_t
  - CX::CX\_Time\_t, [149](#)
- calculateAmplitudes
  - CX::Synth::AdditiveSynth, [55](#)
- canPlay
  - CX::Synth::SoundBufferInput, [174](#)
- captureOFLogMessages
  - CX::CX\_Logger, [104](#)
- centeredString
  - CX::Draw, [30](#)
- checkForPresentationErrors
  - CX::CX\_SlidePresenter, [126](#)
- checkOFVersion
  - CX::Util, [44](#)
- clamp
  - CX::Util, [44](#)
- clear
  - CX::CX\_DataFrame, [73](#)
  - CX::CX\_SoundBuffer, [131](#)
- clearAllEvents
  - CX::CX\_InputManager, [95](#)
- clearEvents
  - CX::CX\_Joystick, [96](#)
  - CX::CX\_Keyboard, [98](#)
  - CX::CX\_Mouse, [108](#)
- clearSlides
  - CX::CX\_SlidePresenter, [126](#)
- Clock
  - Timing, [20](#)
- closeStream
  - CX::CX\_SoundStream, [141](#)
- collectedSamples
  - CX::Util::CX\_LapTimer, [101](#)
  - CX::Util::CX\_SegmentProfiler, [122](#)
- colorArc
  - CX::Draw, [31](#)
- colorArcToVbo
  - CX::Draw, [31](#)
- colorWheel
  - CX::Draw, [31](#)
- colorWheelToVbo
  - CX::Draw, [32](#)
- columns
  - CX::Algo::LatinSquare, [162](#)
- concatenate
  - CX::Util, [45](#)
- configureFromFile
  - CX::CX\_Display, [87](#)
  - CX::Util::CX\_DegreeToPixelConverter, [84](#)
  - CX::Util::CX\_LengthToPixelConverter, [102](#)
- convertApiToString
  - CX::CX\_SoundStream, [141](#)
- convertApisToString
  - CX::CX\_SoundStream, [141](#)
- convertApisToStrings
  - CX::CX\_SoundStream, [141](#)
- convertColors
  - CX::Draw, [32](#)
- convertStringToApi
  - CX::CX\_SoundStream, [141](#)
- convertToRGB
  - CX::Draw, [33](#)
- convertVectorColumnToColumns
  - CX::CX\_DataFrame, [73](#)
- copyCellTo
  - CX::CX\_DataFrameCell, [80](#)
- copyColumn
  - CX::CX\_DataFrame, [73](#)
- copyColumns
  - CX::CX\_DataFrame, [74](#)
- copyFboToBackBuffer
  - CX::CX\_Display, [88](#)
- copyRows
  - CX::CX\_DataFrame, [74](#)
- cutoff
  - CX::Synth::Filter, [157](#)
- DRAGGED
  - CX::CX\_Mouse, [108](#)
- Data, [14](#)
- degreesToPixels

- CX::Util, 45
- delay
  - CX::CX\_Clock, 63
- deleteAmount
  - CX::CX\_SoundBuffer, 131
- deleteColumn
  - CX::CX\_DataFrame, 74
- deleteRow
  - CX::CX\_DataFrame, 75
- disconnectInput
  - CX::Synth::ModuleBase, 165
- disconnectOutput
  - CX::Synth::ModuleBase, 165
- Display
  - Entry Point, 15
- enable
  - CX::CX\_Keyboard, 99
  - CX::CX\_Mouse, 109
- enabled
  - CX::CX\_Keyboard, 99
  - CX::CX\_Mouse, 109
- endDrawingCurrentSlide
  - CX::CX\_SlidePresenter, 126
- endDrawingToBackBuffer
  - CX::CX\_Display, 89
- Entry Point, 15
  - Display, 15
  - Input, 15
  - Log, 15
  - RNG, 15
  - runExperiment, 15
- error
  - CX::CX\_Logger, 104
- ErrorMode
  - CX::CX\_SlidePresenter, 124
- estimateFramePeriod
  - CX::CX\_Display, 89
- estimateNextSwapTime
  - CX::CX\_Display, 89
  - CX::CX\_SoundStream, 142
- exclude
  - CX::Util, 45
- fallOffPower
  - CX::Draw::CX\_PatternProperties\_t, 111
- fatalError
  - CX::CX\_Logger, 104
- FilterType
  - CX::Synth::Filter, 157
  - CX::Synth::FIRFilter, 158
- fixationCross
  - CX::Draw, 33
- fixationCrossToPath
  - CX::Draw, 33
- flush
  - CX::CX\_Logger, 105
- formatsToString
  - CX::CX\_SoundStream, 142
- formatsToStrings
  - CX::CX\_SoundStream, 142
- fullyCross
  - CX::Algo, 26
- gabor
  - CX::Draw, 33
- gateInput
  - CX::Synth::Envelope, 153
- generate
  - CX::Algo::LatinSquare, 162
- generateSeparatedValues
  - CX::Algo, 27
- getActualFrameCounts
  - CX::CX\_SlidePresenter, 127
- getActualPresentationDurations
  - CX::CX\_SlidePresenter, 127
- getAngleBetweenPoints
  - CX::Util, 45
- getAxisPositions
  - CX::CX\_Joystick, 97
- getBezierVertices
  - CX::Draw, 34
- getBlockNumber
  - CX::Algo::BlockSampler, 58
- getBlockPosition
  - CX::Algo::BlockSampler, 58
- getButtonStates
  - CX::CX\_Joystick, 97
- getCenterOfDisplay
  - CX::CX\_Display, 89
- getColumn
  - CX::Algo::LatinSquare, 162
- getColumnNames
  - CX::CX\_DataFrame, 75
- getCompiledApis
  - CX::CX\_SoundStream, 142
- getConfiguration
  - CX::CX\_SoundBufferPlayer, 135
  - CX::CX\_SoundBufferRecorder, 138
  - CX::CX\_SoundStream, 142
- getCursorPosition
  - CX::CX\_Mouse, 109
- getData
  - CX::Synth::ModuleBase, 165
- getDateTimeString
  - CX::CX\_Clock, 63
- getDeviceList
  - CX::CX\_SoundStream, 142
- getExperimentStartDateTimeString

- CX::CX\_Clock, 65
- getFixationCrossVertices
  - CX::Draw, 34
- getFrameNumber
  - CX::CX\_Display, 89
- getFramePeriod
  - CX::CX\_Display, 89
- getFramePeriodStandardDeviation
  - CX::CX\_Display, 89
- getGenerator
  - CX::CX\_RandomNumberGenerator, 113
- getJoystickName
  - CX::CX\_Joystick, 97
- getLastSwapTime
  - CX::CX\_Display, 90
  - CX::CX\_SoundStream, 144
- getLength
  - CX::CX\_SoundBuffer, 131
- getMaximumRandomInt
  - CX::CX\_RandomNumberGenerator, 113
- getMinimumRandomInt
  - CX::CX\_RandomNumberGenerator, 113
- getModuleLevel
  - CX::CX\_Logger, 105
- getMsaaSampleCount
  - CX::Util, 46
- getNegativePeak
  - CX::CX\_SoundBuffer, 131
- getNextEvent
  - CX::CX\_Joystick, 97
  - CX::CX\_Keyboard, 99
  - CX::CX\_Mouse, 109
- getNextSample
  - CX::Synth::Adder, 54
  - CX::Synth::AdditiveSynth, 56
  - CX::Synth::Clamper, 60
  - CX::Synth::Envelope, 153
  - CX::Synth::Filter, 157
  - CX::Synth::FIRFilter, 159
  - CX::Synth::GenericOutput, 159
  - CX::Synth::Mixer, 163
  - CX::Synth::ModuleBase, 165
  - CX::Synth::Multiplier, 167
  - CX::Synth::Oscillator, 169
  - CX::Synth::SoundBufferInput, 174
  - CX::Synth::Splitter, 176
  - CX::Synth::TrivialGenerator, 179
- getNextValue
  - CX::Algo::BlockSampler, 59
- getPartitionedTime
  - CX::CX\_Time\_t, 149
- getPositivePeak
  - CX::CX\_SoundBuffer, 132
- getRGBSpectrum
  - CX::Draw, 34
- getRawDataReference
  - CX::CX\_SoundBuffer, 132
- getRelativePointFromDistanceAndAngle
  - CX::Util, 46
- getResolution
  - CX::CX\_Display, 90
- getRow
  - CX::Algo::LatinSquare, 162
- getRowCount
  - CX::CX\_DataFrame, 75
- getRtAudioInstance
  - CX::CX\_SoundStream, 144
- getSampleFrameCount
  - CX::CX\_SoundBuffer, 132
- getSampleFrameNumber
  - CX::CX\_SoundStream, 144
- getSeed
  - CX::CX\_RandomNumberGenerator, 113
- getSlideByName
  - CX::CX\_SlidePresenter, 127
- getSlides
  - CX::CX\_SlidePresenter, 127
- getSoundBuffer
  - CX::CX\_SoundBufferPlayer, 135
  - CX::CX\_SoundBufferRecorder, 138
- getSoundStream
  - CX::CX\_SoundBufferPlayer, 136
- getStarVertices
  - CX::Draw, 34
- getStatString
  - CX::Util::CX\_LapTimer, 101
  - CX::Util::CX\_SegmentProfiler, 122
- getStoredType
  - CX::CX\_DataFrameCell, 80
- getStreamLatency
  - CX::CX\_SoundStream, 144
- getTotalSampleCount
  - CX::CX\_SoundBuffer, 132
- getValue
  - CX::Synth::ModuleParameter, 166
- greyscalePatternToPixels
  - CX::Draw, 35
- HarmonicAmplitudeType
  - CX::Synth::AdditiveSynth, 55
- hasSwappedSinceLastCheck
  - CX::CX\_Display, 90
  - CX::CX\_SoundStream, 144
- height
  - CX::Draw::CX\_PatternProperties\_t, 111
- incorrectFrameCounts
  - CX::CX\_SlidePresenter::PresentationErrorInfo, 172
- Input

- Entry Point, 15
- Input Devices, 16
- intVector
  - CX::Util, 46
- inverse
  - CX::Util::CX\_BaseUnitConverter, 63
  - CX::Util::CX\_CoordinateConverter, 67
  - CX::Util::CX\_DegreeToPixelConverter, 84
  - CX::Util::CX\_LengthToPixelConverter, 103
- isAutomaticallySwapping
  - CX::CX\_Display, 90
- isKeyHeld
  - CX::CX\_Keyboard, 99
- isLoadedSuccessfully
  - CX::CX\_SoundBuffer, 132
- isReadyToPlay
  - CX::CX\_SoundBuffer, 132
- isStreamRunning
  - CX::CX\_SoundStream, 144
- isVector
  - CX::CX\_DataFrameCell, 80
- JoystickEventType
  - CX::CX\_Joystick::Event, 155
- key
  - CX::CX\_Keyboard::Event, 154
- KeyboardEventType
  - CX::CX\_Keyboard, 98
- LatinSquare
  - CX::Algo::LatinSquare, 161
- level
  - CX::CX\_Logger, 105
- levelForAllModules
  - CX::CX\_Logger, 105
- levelForConsole
  - CX::CX\_Logger, 105
- levelForFile
  - CX::CX\_Logger, 105
- line
  - CX::Draw, 35
- lines
  - CX::Draw, 35
- listDevices
  - CX::CX\_SoundStream, 144
- loadFile
  - CX::CX\_SoundBuffer, 132
- Log
  - Entry Point, 15
- log
  - CX::CX\_Logger, 105
- MOVED
  - CX::CX\_Mouse, 108
- maskType
  - CX::Draw::CX\_PatternProperties\_t, 111
- max
  - CX::Util, 46
- mean
  - CX::Util, 46, 48
- Message Logging, 17
  - CX\_LogLevel, 17
- min
  - CX::Util, 48
- MouseEventType
  - CX::CX\_Mouse, 108
- multiplyAmplitudeBy
  - CX::CX\_SoundBuffer, 133
- multiplySpeed
  - CX::CX\_SoundBuffer, 133
- normalize
  - CX::CX\_SoundBuffer, 133
- notice
  - CX::CX\_Logger, 106
- now
  - CX::CX\_Clock, 65
- operator<<
  - CX, 24
- operator>>
  - CX, 25
  - CX::Synth, 40
- operator()
  - CX::CX\_DataFrame, 75
  - CX::Util::CX\_BaseUnitConverter, 63
  - CX::Util::CX\_CoordinateConverter, 67, 68
  - CX::Util::CX\_DegreeToPixelConverter, 84
  - CX::Util::CX\_LengthToPixelConverter, 103
- operator+=
  - CX::Algo::LatinSquare, 162
  - CX::CX\_Time\_t, 150
- operator-=
  - CX::CX\_Time\_t, 150
- operator=
  - CX::CX\_DataFrame, 75
  - CX::CX\_DataFrameCell, 81
- PRESSED
  - CX::CX\_Keyboard, 98
  - CX::CX\_Mouse, 108
- patternMask
  - CX::Draw, 35
- pixelsToDegrees
  - CX::Util, 48
- play
  - CX::CX\_SoundBufferPlayer, 136
- pollEvents
  - CX::CX\_InputManager, 95

- CX::CX\_Joystick, 97
- precisionTest
  - CX::CX\_Clock, 65
- presentSlides
  - CX::CX\_SlidePresenter, 128
- print
  - CX::Algo::LatinSquare, 162
  - CX::CX\_DataFrame, 76
- printLastPresentationInformation
  - CX::CX\_SlidePresenter, 128
- printToFile
  - CX::CX\_DataFrame, 76, 77
- pruneLowAmplitudeHarmonics
  - CX::Synth::AdditiveSynth, 56
- RELEASED
  - CX::CX\_Keyboard, 98
  - CX::CX\_Mouse, 108
- REPEAT
  - CX::CX\_Keyboard, 98
- ROUND\_DOWN
  - CX::Util, 43
- ROUND\_TO\_NEAREST
  - CX::Util, 43
- ROUND\_TOWARD\_ZERO
  - CX::Util, 43
- ROUND\_UP
  - CX::Util, 43
- RNG
  - Entry Point, 15
- randomDouble
  - CX::CX\_RandomNumberGenerator, 113
- randomInt
  - CX::CX\_RandomNumberGenerator, 114
- Randomization, 18
- readConfigurationFromFile
  - CX::CX\_SoundStream, 145
- readFromFile
  - CX::CX\_DataFrame, 77
- readKeyValueFile
  - CX::Util, 48
- relativeFrequency
  - CX::Synth, 40
- reopenWindow
  - CX, 25
- reorderDown
  - CX::Algo::LatinSquare, 162
- reorderLeft
  - CX::Algo::LatinSquare, 162
- reorderRight
  - CX::Algo::LatinSquare, 162
- reorderRows
  - CX::CX\_DataFrame, 78
- reorderUp
  - CX::Algo::LatinSquare, 162
- repeat
  - CX::Util, 50
- resample
  - CX::CX\_SoundBuffer, 133
- reset
  - CX::Util::CX\_TrialController, 151
- resetExperimentStartTime
  - CX::CX\_Clock, 65
- restart
  - CX::Util::CX\_LapTimer, 101
  - CX::Util::CX\_SegmentProfiler, 122
- restartSampling
  - CX::Algo::BlockSampler, 59
- reverse
  - CX::CX\_SoundBuffer, 133
- reverseColumns
  - CX::Algo::LatinSquare, 162
- reverseRows
  - CX::Algo::LatinSquare, 163
- ring
  - CX::Draw, 37
- round
  - CX::Util, 51
- rows
  - CX::Algo::LatinSquare, 163
- runExperiment
  - Entry Point, 15
- SCROLLED
  - CX::CX\_Mouse, 108
- sample
  - CX::CX\_RandomNumberGenerator, 114
- sampleBinomialRealizations
  - CX::CX\_RandomNumberGenerator, 116
- sampleBlocks
  - CX::CX\_RandomNumberGenerator, 116
- sampleData
  - CX::Synth::SoundBufferOutput, 175
  - CX::Synth::StereoSoundBufferOutput, 177
- sampleExclusive
  - CX::CX\_RandomNumberGenerator, 116, 117
- sampleNormalRealizations
  - CX::CX\_RandomNumberGenerator, 118
- sampleRate
  - CX::CX\_SoundStream::Configuration, 61
- sampleRealizations
  - CX::CX\_RandomNumberGenerator, 118
- sampleUniformRealizations
  - CX::CX\_RandomNumberGenerator, 118
- saveFboToFile
  - CX::Draw, 37
- sequence
  - CX::Util, 51

- sequenceAlong
  - CX::Util, [51](#)
- sequenceSteps
  - CX::Util, [51](#)
- setAmplitudes
  - CX::Synth::AdditiveSynth, [56](#)
- setAutomaticSwapping
  - CX::CX\_Display, [90](#)
- setAxisInversion
  - CX::Util::CX\_CoordinateConverter, [68](#)
- setChannelCount
  - CX::CX\_SoundBuffer, [133](#)
- setCurrentFunction
  - CX::Util::CX\_TrialController, [151](#)
- setCursorPosition
  - CX::CX\_Mouse, [109](#)
- setData
  - CX::Synth::ModuleBase, [165](#)
- setFromVector
  - CX::CX\_SoundBuffer, [134](#)
- setFullScreen
  - CX::CX\_Display, [90](#)
- setFundamentalFrequency
  - CX::Synth::AdditiveSynth, [57](#)
- setGain
  - CX::Synth::Multiplier, [167](#)
- setGeneratorFunction
  - CX::Synth::Oscillator, [169](#)
- setHarmonicSeries
  - CX::Synth::AdditiveSynth, [57](#)
- setImplementation
  - CX::CX\_Clock, [65](#)
- setLength
  - CX::CX\_SoundBuffer, [134](#)
- setMessageFlushCallback
  - CX::CX\_Logger, [106](#)
- setMultiplier
  - CX::Util::CX\_CoordinateConverter, [68](#)
- setOrigin
  - CX::Util::CX\_CoordinateConverter, [68](#)
- setRowCount
  - CX::CX\_DataFrame, [78](#)
- setSeed
  - CX::CX\_RandomNumberGenerator, [120](#)
- setSoundBuffer
  - CX::CX\_SoundBufferPlayer, [136](#)
  - CX::CX\_SoundBufferRecorder, [138](#)
  - CX::Synth::SoundBufferInput, [174](#)
- setStandardHarmonicSeries
  - CX::Synth::AdditiveSynth, [57](#)
- setTime
  - CX::CX\_SoundBufferPlayer, [136](#)
  - CX::Synth::SoundBufferInput, [174](#)
- setUnitConverter
  - CX::Util::CX\_CoordinateConverter, [68](#)
- setWindowResolution
  - CX::CX\_Display, [91](#)
- setWindowTitle
  - CX::CX\_Display, [91](#)
- setup
  - CX::Algo::BlockSampler, [59](#)
  - CX::CX\_Display, [90](#)
  - CX::CX\_InputManager, [95](#)
  - CX::CX\_Joystick, [97](#)
  - CX::CX\_SlidePresenter, [128](#)
  - CX::CX\_SoundBufferPlayer, [136](#)
  - CX::CX\_SoundBufferRecorder, [138](#), [139](#)
  - CX::CX\_SoundStream, [145](#)
  - CX::Synth::SoundBufferOutput, [175](#)
  - CX::Synth::StereoSoundBufferOutput, [177](#)
  - CX::Util::CX\_DegreeToPixelConverter, [86](#)
  - CX::Util::CX\_LapTimer, [101](#)
  - CX::Util::CX\_LengthToPixelConverter, [103](#)
  - CX::Util::CX\_SegmentProfiler, [122](#)
- showCursor
  - CX::CX\_Mouse, [109](#)
- shuffleRows
  - CX::CX\_DataFrame, [78](#)
- shuffleVector
  - CX::CX\_RandomNumberGenerator, [120](#)
- sinc
  - CX::Synth, [42](#)
- size
  - CX::CX\_DataFrameColumn, [82](#)
- sleep
  - CX::CX\_Clock, [66](#)
- Sound, [19](#)
- squircleToPath
  - CX::Draw, [37](#)
- standardDeviation
  - CX::CX\_Time\_t, [150](#)
- star
  - CX::Draw, [37](#)
- starToPath
  - CX::Draw, [39](#)
- start
  - CX::CX\_SoundStream, [146](#)
- startPlayingAt
  - CX::CX\_SoundBufferPlayer, [137](#)
- startRecording
  - CX::CX\_SoundBufferRecorder, [139](#)
- startSlidePresentation
  - CX::CX\_SlidePresenter, [128](#)
- stop
  - CX::CX\_SoundBufferPlayer, [137](#)
  - CX::CX\_SoundStream, [146](#)
- stopRecording
  - CX::CX\_SoundBufferRecorder, [139](#)

- store
  - CX::CX\_DataFrameCell, [81](#)
- storeVector
  - CX::CX\_DataFrameCell, [81](#)
- streamOptions
  - CX::CX\_SoundStream::Configuration, [61](#)
- stringToVector
  - CX::Util, [52](#)
- stripLeadingSilence
  - CX::CX\_SoundBuffer, [134](#)
- swapBuffers
  - CX::CX\_Display, [91](#)
- swapBuffersInThread
  - CX::CX\_Display, [91](#)
- swapColumns
  - CX::Algo::LatinSquare, [163](#)
- swapRows
  - CX::Algo::LatinSquare, [163](#)
- SwappingMode
  - CX::CX\_SlidePresenter, [124](#)
- t1
  - CX::Util::CX\_SegmentProfiler, [122](#)
- t2
  - CX::Util::CX\_SegmentProfiler, [122](#)
- takeSample
  - CX::Util::CX\_LapTimer, [101](#)
- testBufferSwapping
  - CX::CX\_Display, [91](#)
- timestamps
  - CX::CX\_Logger, [106](#)
- Timing, [20](#)
  - Clock, [20](#)
- to
  - CX::CX\_DataFrameCell, [81](#)
- toString
  - CX::CX\_DataFrameCell, [81](#)
- toVector
  - CX::CX\_DataFrameCell, [81](#), [82](#)
- unique
  - CX::Util, [52](#)
- update
  - CX::CX\_SlidePresenter, [129](#)
  - CX::Util::CX\_TrialController, [152](#)
- updateValue
  - CX::Synth::ModuleParameter, [166](#)
- useHardwareVSync
  - CX::CX\_Display, [92](#)
- useSoftwareVSync
  - CX::CX\_Display, [94](#)
- Utility, [21](#)
- validate
  - CX::Algo::LatinSquare, [163](#)
- value
  - CX::CX\_Time\_t, [150](#)
- valueUpdated
  - CX::Synth::ModuleParameter, [166](#)
- var
  - CX::Util, [52](#), [53](#)
- vectorToString
  - CX::Util, [53](#)
- verbose
  - CX::CX\_Logger, [106](#)
- Video, [22](#)
- waitForKeypress
  - CX::CX\_Keyboard, [99](#)
- waitForOpenGL
  - CX::CX\_Display, [94](#)
- warning
  - CX::CX\_Logger, [106](#)
- width
  - CX::Draw::CX\_PatternProperties\_t, [111](#)
- writeToFile
  - CX::CX\_SoundBuffer, [134](#)
  - CX::Util, [53](#)