libcoyote Coyote Playback Server SDK

By Daniel Hopson

libcoyote is a C++ and Python 3 SDK for controlling and manipulating Coyote playback servers.

The SDK is:

- Portable – should run on any multithreaded platform that Qt5WebSockets supports.
- Free and open – Apache 2.0 licensed
- Powerful – with libcoyote, you can create rich, fast applications that manipulate the Coyote.
- Efficient – Performance was a target with libcoyote, and as such, all Coyotes run a native, optimized server specifically dedicated to receive libcoyote commands. The latency can be considerably better than JSON TCP or HTTP.

Languages

libcoyote supports C++11 and up, Python 3, and to a very limited extent, C.

Compiling

To compile libcoyote, you will need CMake, Qt5WebSockets, msgpack-c, and a (preferably GNU or Clang based) C++14 compliant C++ compiler. libcoyote should be fine used with C++11 projects, but it internally utilizes some C++14 features, so it's required for the build process.

It is theoretically possible to compile libcoyote with MSVC, but this is somewhat discouraged as there are no build scripts for it and it is not well tested. That said, effort has been made to ensure that libcoyote will compile for Visual Studio if the correct build scripts and headers are provided.

GNU-specific compiler extensions were deliberately avoided in writing libcoyote, so the C++ should be portable.

Make sure to recursively clone this repository, as libcoyote pulls in a couple of small dependencies as git submodules.

Windows

You will need MSYS2, which provides a modern GNU-based toolchain for 32 and 64 bit Windows. It can be acquired at msys2.org. You should install Qt5WebSockets via MSYS2's "pacman" package manager.

For Python support, it is strongly advised that you also have a working Python 3 installation, separate from MSYS2. You will also need pkg-config to compile Python support. It can be installed with MSYS2.

Point CMake to your Python 3 installation directory, as demonstrated in the example MSYS2 command line below:

```
cd pycoyote
```

```
mkdir build
cd build
cmake .. -G"MSYS Makefiles" -DPYLIBS=/c/Python37 -DPYHEADERS=/c/Python37/include
make -j
```

This will generate pycoyote.pyd in the current directory, and libcoyote.dll in "`../../build`".

You must keep pycoyote.pyd and libcoyote.dll and their dependencies in the same directory, or you will get missing DLL errors.

It is recommended that you ensure your Python installation directory's path does not contain whitespace. Some MSYS2 tools can be temperamental about whitespace in paths.

Linux/BSD

On Linux and likely other BSD systems, the procedure is more simple.
Simply install the -devel package for Qt5WebSockets, msgpack-c, and Python 3,
and run:

```
cd pycoyote
mkdir build
cd build
cmake ..
make -j
```

macOS

libcoyote is not well tested on macOS, but it should not be difficult to compile. You will again need all the dependencies required for Linux/BSD, which can likely be installed via MacPorts.

Other platforms

It should be easy enough to port libcoyote to other compilers and operating systems, so please, feel free to experiment! If you find a problem with the compilation scripts for a particular platform, please let us know, or even submit a pull request.

Without Python

You don't need to compile Python support if you don't want to. The C++ API is quite feature-complete.

Instead of changing to the pycoyote directory, just create a build directory in the libcoyote directory, and replace "`cmake ..`" with "`cmake ../src`".

Quirks to be aware of

- Because of (currently partially implemented) C support, libcoyote uses a standard layout string class called CoyoteString in C++, which is cast to a character pointer in C. A static assertion checks that it is equal to the size of a character pointer. This is most noticeable in pycoyote, where string members of data structure classes must be specified as e.g.:
  "`p.Name = pycoyote.CoyoteString('my string here')`"

- Passing a zero to Take, End, Pause, SeekTo, etc, will perform the operation on the currently selected preset, whichever that happens to be.
- To create an entirely new preset, make sure the PK field is zero, or the Coyote unit will attempt to create a new preset with that PK. If such a preset already exists, the operation will fail.
- Via JSON and internally, the preset layouts are stored as a string, but they are stored as an enum in libcoyote and translated back and forth. This makes it easier to understand and manipulate the preset configuration, but is an implementation detail you should be aware of.

API documentation

The best way to learn about how to use libcoyote is to inspect pycoyote.cpp and session.h's exposed methods, and datastructures_c.h for the members of various classes used by libcoyote.
The API is fairly self explanatory.

There is a direct one-to-one mapping of libcoyote session method names and JSON API command names, though some argument types can differ slightly. This is also mostly true for data structures in libcoyote.

Compatibility with Coyote software

libcoyote uses the same "CoyoteAPIVersion" field as JSON commands, and this is used to determine compatibility. As long as `COYOTE_API_VERSION` in macros.h matches, libcoyote should work. It will be bumped when breaking changes are necessary. Currently all recent releases of Coyote software use API version "0.3".

Contributing

Sonoran Video Systems welcomes any code contributions to libcoyote, provided they are of sufficient quality. We want libcoyote to be as useful and reliable as possible, and any work towards those goals will be appreciated. Not a programmer? No problem. You can submit issues and bug reports to us on GitHub.

Download

You can find the repository for libcoyote at:

https://github.com/Sonoranvideo/libcoyote