

# CellML Simulator - building from source

*Gradually filling in the details to build CellML Simulator from source.*

## Table of contents

|     |   |   |
|-----|---|---|
| 1   | Dependencies.....                           | 2 |
| 1.1 | CellML DOM API.....                         | 2 |
| 1.2 | CVODES (Sundials).....                      | 2 |
| 1.3 | HDF5.....                                   | 2 |
| 1.4 | Redland RDF Libraries.....                  | 3 |
| 1.5 | PLplot - a Scientific Plotting Library..... | 3 |
| 1.6 | YAJL - Yet Another JSON Library.....        | 4 |
| 2   | CMake - cross-platform make.....            | 4 |
| 3   | CellML Simulator.....                       | 5 |

## 1. Dependencies

### 1.1. CellML DOM API

Currently using version 1.4 of the Auckland implementation of the [CellML API](#).

There is little documentation helping to get the Auckland API implementation built, but some documentation is starting to become [available](#). Here are the steps I use on a Fedora 6 (x86\_64) machine...

```
[Download and extract the API source]
> mkdir CellML_DOM_API-build
> cd CellML_DOM_API-build
> ../CellML_DOM_API_1_4/configure --prefix $HOME/std-libs/CellML-1.4
--enable-corba=no --enable-context=no --enable-ccgs --enable-cis=no
--enable-server=no --enable-static --enable-shared=no --enable-annotools
--enable-cuses --enable-cevas --enable-vacss --enable-malaes
> make -j 4
> make install
```

### 1.2. CVODES (Sundials)

I use the CVODES numerical integrator from the [Sundials](#) package — currently using version 2.3.0. I grab the source .tar.gz and build it with the following steps...

```
> tar xvzf sundials-2.3.0.tar.gz
> cd sundials-2.3.0
> ./configure --prefix=$HOME/std-libs/sundials-2.3.0 --disable-shared
--disable-fcmix --disable-mpi
> make
> make install
```

### 1.3. HDF5

CellML Simulator uses [HDF5](#) for high performance data storage. In testing, the best performance was obtained using the PacketTable interface available in the 1.8.0 testing versions. Currently I am using HDF5 version 1.8.1 built with the following steps...

```
> tar xvjf hdf5-1.8.1.tar.bz2
> cd hdf5-1.8.1
> ./configure --prefix=$HOME/std-libs/hdf5-1.8.1 --disable-shared
--enable-static-exec
> make
> make install
```

## 1.4. Redland RDF Libraries

CellML Simulator uses the Redland RDF Library for handling metadata specified using RDF. I am currently using version 1.0.5 with one small patch to fix a bug in that version (should be fixed in newer versions but yet to test 1.0.6). **Yep, all good with 1.0.6.** With that fix in place, I build Redland with the following steps...

```
> tar xvjf redland-1.0.6.tar.gz
> cd redland-1.0.6
> ./configure --prefix=$HOME/std-libs/redland-1.0.6 --disable-digests
--with-bdb=no --with-mysql=no --with-openssl-digests=no
--with-postgresql=no --with-raptor=internal --with-rasqal=internal
--with-sqlite=no --with-threestore=no --with-xml-parser=libxml
--disable-digests --enable-release --disable-shared --enable-static
> make
> make install
```

## 1.5. PLplot - a Scientific Plotting Library

PLplot is used for all the graph drawing performed by CellML Simulator. There are some tricks to getting this all set up and there seem to be issues with the various font options depending on what your build system has installed. It looks like newer versions of PLplot will be much better in this regard, but for now I am using version 5.6.1 built with the following steps...(I had to alter the configure script to ensure the fonts provided to the configure command were used)...

```
> tar xvzf plplot-5.6.1.tar.gz
> cd plplot-5.6.1
> ./configure --prefix=$HOME/std-libs/plplot-5.6.1 \
--disable-shared --disable-cxx --disable-f77 --disable-f95 \
--disable-java --disable-python --disable-octave --disable-tcl \
--disable-itcl --disable-pdl --disable-drivers --enable-png \
--enable-ps --enable-xwin --enable-xfig --with-double \
--disable-dyndrivers --disable-gcw \
--with-freetype-font-dir=/usr/share/fonts/dejavu-lgc \
--with-freetype-sans-font=DejaVuLGCSans.ttf \
--with-freetype-sans-bold-font=DejaVuLGCSans-Bold.ttf \
--with-freetype-sans-oblique-font=DejaVuLGCSans-Oblique.ttf \
--with-freetype-sans-italic-font=DejaVuLGCSans-Oblique.ttf \
--with-freetype-sans-bold-oblique-font=DejaVuLGCSans-BoldOblique.ttf \
\
--with-freetype-sans-bold-italic-font=DejaVuLGCSans-BoldOblique.ttf \
--with-freetype-serif-font=DejaVuLGCSerif.ttf \
--with-freetype-serif-bold-font=DejaVuLGCSerif-Bold.ttf \
--with-freetype-serif-italic-font=DejaVuLGCSerif-Oblique.ttf \
--with-freetype-serif-oblique-font=DejaVuLGCSerif-Oblique.ttf \
--with-freetype-serif-bold-italic-font=DejaVuLGCSerif-BoldOblique.ttf \
\
--with-freetype-serif-bold-oblique-font=DejaVuLGCSerif-BoldOblique.ttf \
\
--with-freetype-mono-font=DejaVuLGCSansMono.ttf \
```

```

--with-freetype-mono-bold-font=DejaVuLGCSSansMono-Bold.ttf \
--with-freetype-mono-oblique-font=DejaVuLGCSSansMono-Oblique.ttf \
--with-freetype-mono-italic-font=DejaVuLGCSSansMono-Oblique.ttf \
--with-freetype-mono-bold-oblique-font=DejaVuLGCSSansMono-BoldOblique.ttf
\
--with-freetype-mono-bold-italic-font=DejaVuLGCSSansMono-BoldOblique.ttf
\
--with-freetype-script-font=DejaVuLGCSSerif.ttf \
--with-freetype-script-bold-font=DejaVuLGCSSerif-Bold.ttf \
--with-freetype-script-oblique-font=DejaVuLGCSSerif-Oblique.ttf \
--with-freetype-script-italic-font=DejaVuLGCSSerif-Oblique.ttf \
--with-freetype-script-bold-oblique-font=DejaVuLGCSSerif-BoldOblique.ttf
\
--with-freetype-script-bold-italic-font=DejaVuLGCSSerif-BoldOblique.ttf
\
--with-freetype-symbol-font=DejaVuLGCSSerif.ttf \
--with-freetype-symbol-bold-font=DejaVuLGCSSerif-Bold.ttf \
--with-freetype-symbol-oblique-font=DejaVuLGCSSerif-Oblique.ttf \
--with-freetype-symbol-italic-font=DejaVuLGCSSerif-Oblique.ttf \
--with-freetype-symbol-bold-oblique-font=DejaVuLGCSSerif-BoldOblique.ttf
\
--with-freetype-symbol-bold-italic-font=DejaVuLGCSSerif-BoldOblique.ttf
> make
> make install

```

Note: now using PLplot 5.9.0 and things are much better...

## 1.6. YAJL - Yet Another JSON Library

YAJL is used for the generation and serialisation of the JSON data used in creating reference descriptions based on annotated CellML models.

## 2. CMake - cross-platform make

CellML Simulator uses the CMake build system, which looks a promising and easy to maintain system for multi-platform build tools (the target for CellML Simulator). It has also been adopted as the primary build system by the K Desktop Environment — pretty strong endorsement! It is also, for me, much simpler to set up and maintain than the traditional autoconf and friends based build environment.

In the CellML Simulator source tree there are modules for finding the required header and library files built above. Unless you have installed everything in the system folders, you need to provide a little bit of help to CMake in order to find all the required files. Here is what I use, based on the install paths from above and using a `csh`...

```

> setenv CMAKE_INCLUDE_PATH
$HOME/std-libs/yajl-0.4.0/include/:$HOME/std-libs/sundials-2.3.0/include/:$HOME/std-
> setenv CMAKE_LIBRARY_PATH
$HOME/std-libs/yajl-0.4.0/lib/:$HOME/std-libs/sundials-2.3.0/lib/:$HOME/std-libs/Cel
> setenv CMAKE_PROGRAM_PATH $HOME/std-libs/redland-1.0.6/bin
> setenv PKG_CONFIG_PATH $HOME/std-libs/plplot-5.9.0/lib/pkgconfig/

```

CMAKE\_INCLUDE\_PATH is used to add extra paths to search for the required header files and CMAKE\_LIBRARY\_PATH the required libraries. Redland config scripts that are made as part of the build process which can be used to define include and library paths. CMAKE\_PROGRAM\_PATH is used to define paths to search for this config script. The standard pkg-config tool is used to define the required PLplot definitions, so we need to use the PKG\_CONFIG\_PATH environment variable to specify where to find it.

### 3. CellML Simulator

The CellML Simulator source is currently only available via the SourceForge.net subversion repository. You can grab the source using:

```
> svn checkout
https://cellml.svn.sourceforge.net/svnroot/cellml/CellMLSimulator/trunk
CellMLSimulator
```

for the current trunk code.

Once you have the source checked out, all the dependencies installed, and set the CMake environment variables then you're all set to try building CellML Simulator. I generally make a build directory inside CellML Simulator, but you should be able to build it from anywhere. The CMAKE\_BUILD\_TYPE is used to control the type of build, with Debug and Release being the two tested types. Something like this...

```
> cd CellMLSimulator
> mkdir -p build/debug
> cd build/debug
> cmake -DCMAKE_BUILD_TYPE=Debug ../..
> make
> make test
> cd ../..
> mkdir -p build/release
> cd build/release
> cmake -DCMAKE_BUILD_TYPE=Release ../..
> make
> make test
```

If you are building somewhere else, simply replace ../.. with the path to the CellML Simulator source directory. Thats pretty much it...