

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
2	CMake - cross-platform make.....	4
3	CellML Simulator.....	5

1. Dependencies

1.1. CellML DOM API

Currently requires the subversion trunk version 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...

```
> svn checkout
https://svn.physiomeproject.org/svn/physiome/CellML_DOM_API/trunk/
CellML_DOM_API
> mkdir CellML_DOM_API-build
> cd CellML_DOM_API
> aclocal && autoconf && automake
> cd ../CellML_DOM_API-build
> ../CellML_DOM_API/configure --prefix $HOME/std-libs/CellML-new-CCGS
--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 1.8.0-alpha5 built with the following steps...

```
> tar xvjf hdf5-1.8.0-alpha5.tar.bz2
> cd hdf5-1.8.0-alpha5
> ./configure --prefix=$HOME/std-libs/hdf5-1.8.0-alpha5 --disable-shared
```



```

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

```

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
$HOME/std-libs/sundials-2.3.0/include:$HOME/std-libs/CellML-new-CCGS/include:$HOME/s
> setenv CMAKE_LIBRARY_PATH
$HOME/std-libs/sundials-2.3.0/lib:$HOME/std-libs/CellML-new-CCGS/lib:$HOME/std-libs/
> setenv CMAKE_PROGRAM_PATH
$HOME/std-libs/redland-1.0.5/bin:/path/to/CellMLSimulator

```

`CMAKE_INCLUDE_PATH` is used to add extra paths to search for the required header files and `CMAKE_LIBRARY_PATH` the required libraries. Redland and PLplot standard

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 these config scripts. I had some trouble with the PLplot config script, so in the CellMLSimulator source directory there is a wrapper script which hardcodes the path to the actual `plplot-config` script — you'll need to change that to your local PLplot installation.

3. CellMLSimulator

The CellMLSimulator 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 CellMLSimulator. I generally make a build directory inside CellMLSimulator, 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
> cd ../..
> mkdir -p build/release
> cd build/release
> cmake -DCMAKE_BUILD_TYPE=Release ../..
> make
```

If you are building somewhere else, simply replace `../..` with the path to the CellMLSimulator source directory. That's pretty much it...