ODE Integrators (COPASI)

# Parse command line & config files

The boost library contains a convenient way to parse the command line options and config files in simple formats and ini formats. Do not reinvent the wheel.

## Install Boost library

Used for command line parsing. This has to be available in the folder /usr/include/ to link the CopasiModelRunner against it

sudo apt-get install libboost-all-dev

For the linking the following boost libraries are necessary

boost\_system

boost\_program\_options

The libraries can be found in

/usr/include/boost

See the cpp projects for examples of the usage.

# COPASI

Run the

./buildCopasi.sh

which should do most of the installation automatically. This builds the SE und GUI version. Additional information can be found at

<https://github.com/copasi/copasi-dependencies/blob/master/readme_linux.md>

Most of the installation is now already automated. It is only necessary to set the path variables to the project files and the result files

export MULTISCALE\_GALACTOSE=/home/mkoenig/multiscale-galactose

export MULTISCALE\_GALACTOSE\_RESULTS=/home/mkoenig/multiscale-galactose-results

mkdir $MULTISCALE\_GALACTOSE

mkdir $MULTISCALE\_GALACTOSE\_RESULTS

git clone <https://github.com/matthiaskoenig/multiscale-galactose.git>

$MULISCALE\_GALACTOSE

cd $MULISCALE\_GALACTOSE

./buildCopasi.sh

Test installation

CopasiSE

cd $MULTISCALE\_GALACTOSE

./startSimulation.sh

Eclipse C++ Copasi project

**Install Eclipse C++**

Install the C++ version in

~/Programs/eclipse-cpp

<http://www.eclipse.org/downloads/>

## Build Copasi C++ Project

**Create eclipse cpp project with same name as folder and select folder.**

File -> New -> C++ Project

(Empty Project, Cross GCC Toolchain)

Copy the build libraries in copasi-dependencies/lib to the project

The important part is the linking of the static libraries from the copasi build and the copasi-dependencies of the machine.

These are located in the folder

~/copasi

# copy the static libraries from these build folders to the cpp project

export COPASI\_PROJECT=~/multiscale-galactose/cpp/copasi/CopasiModelRunner/

echo $COPASI\_PROJECT

rm $COPASI\_PROJECT/\*.a

rm $COPASI\_PROJECT/\*.la

cp ~/copasi/build\_copasi/copasi/libCOPASISE.a $COPASI\_PROJECT

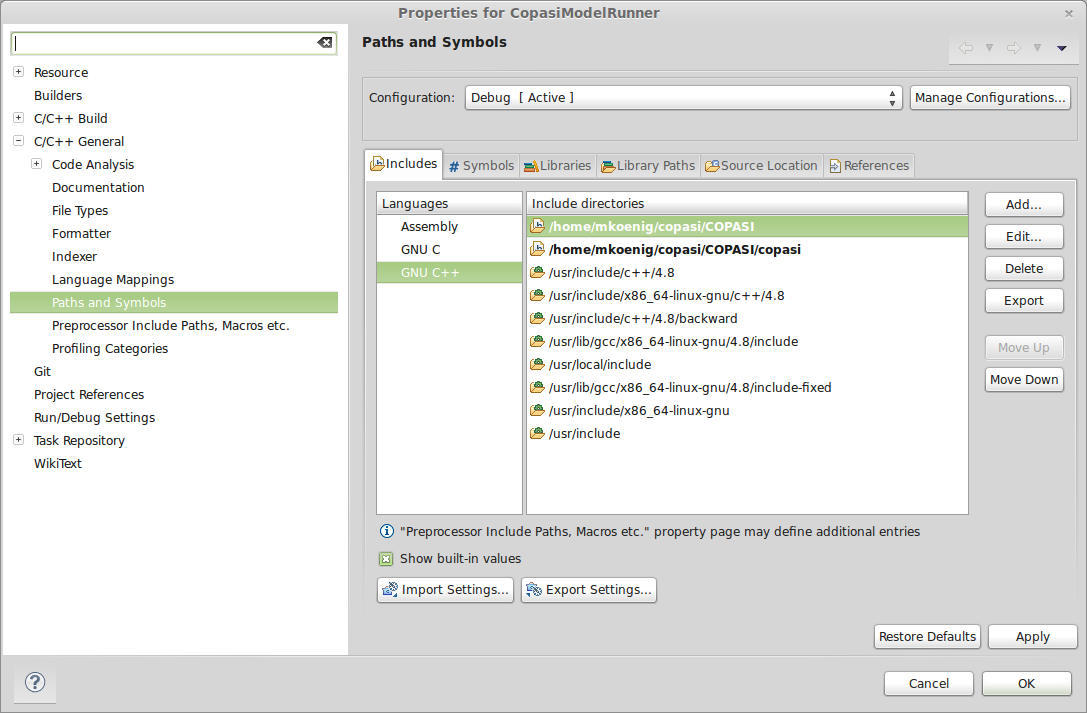
cp ~/copasi/copasi-dependencies/bin/lib/\*.a $COPASI\_PROJECT

cp ~/copasi/copasi-dependencies/bin/lib/\*.la $COPASI\_PROJECT

**Link the copasi souce folder** in ~/copasi/COPASI & ~/copasi/COPASI/copasi

Project -> Properties -> C/C++ General -> Paths and Symbols

The source folder is the folder the copasi libraries were build with.



The include errors should vanish after source linking.

**Link the libraries** (should work directly if copied to the project)

They have to be in the folder with the cpp code and have to be linked without the ‘lib’ and the ‘a, la’ ending.

**libraries to add** (**! The library include order is crucial !)**

boost\_system

boost\_program\_options

COPASISE

SBW-static

sbml-static

sedml-static

raptor

lapack

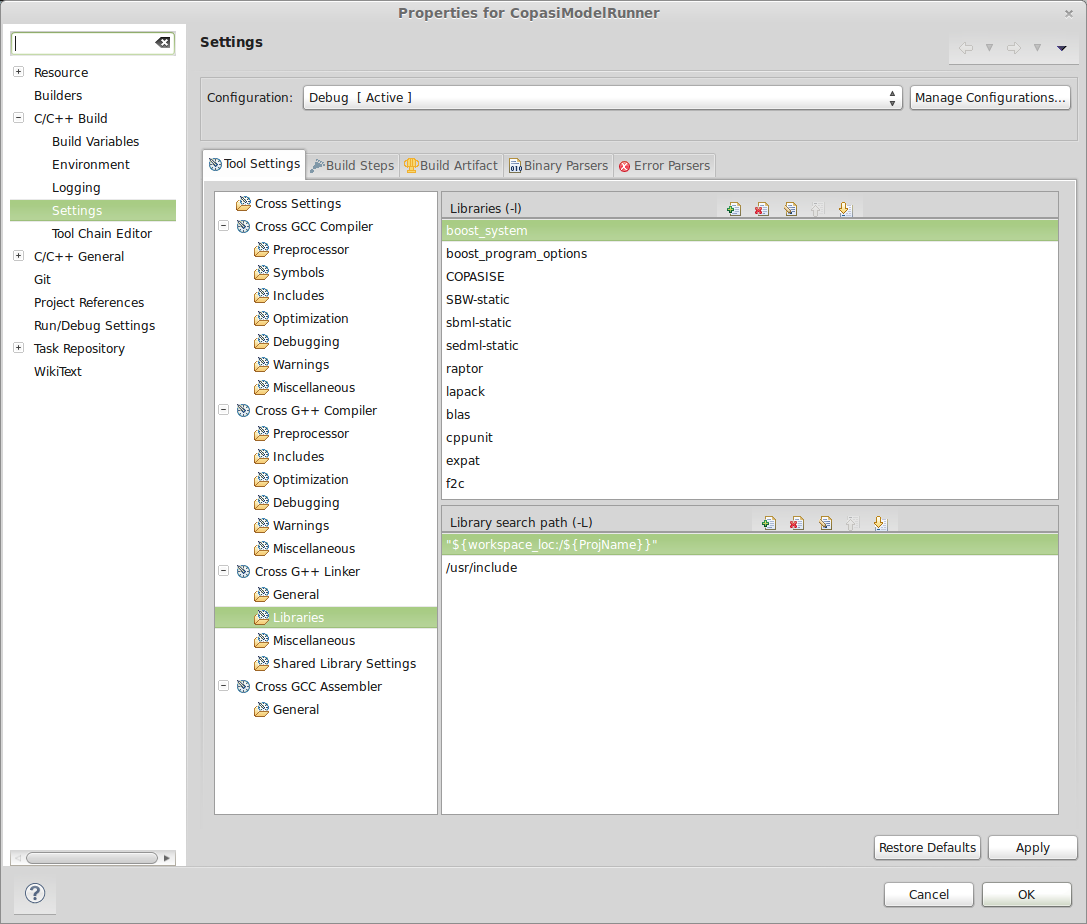
blas

cppunit

expat

f2c

qwt



After linking and including the source, the build should be possible from eclipse.

make all

Building file: ../src/GalactoseTest.cpp

Invoking: Cross G++ Compiler

g++ -I/home/mkoenig/copasi/COPASI -I/home/mkoenig/copasi/COPASI/copasi -O0 -g3 -Wall -c -fmessage-length=0 -MMD -MP -MF"src/GalactoseTest.d" -MT"src/GalactoseTest.d" -o "src/GalactoseTest.o" "../src/GalactoseTest.cpp"

Finished building: ../src/GalactoseTest.cpp

Building target: GalactoseTest

Invoking: Cross G++ Linker

g++ -L"/home/mkoenig/work/multiscale-galactose/cpp/copasi/GalactoseTest" -o "GalactoseTest" ./src/GalactoseTest.o -lCOPASISE -lSBW-static -lsedml-static -lsbml-static -lraptor -llapack -lblas -lcppunit -lexpat -lf2c -lmml -lqwt

Finished building target: GalactoseTest

For the terminal version without GUI even less libraries have to be imported

-lCOPASISE -lSBW-static -lsbml-static -lraptor -llapack -lblas -lcppunit -lexpat -lf2c -lqwt

## 

## Java setup ModelCreator

The modelcreator contains all code related to the SBML handling and generation in Java. It automatically generates the SBML files for different layouts.

clone/checkout project

cd ~/workspace/java

git clone https://matthiaskoenig@git.code.sf.net/p/modelcreator/code modelcreator

Now create a new project in eclipse called **modelcreator** with use default location.

The necessary eclipse files will be added to the project.

Be careful with the eclipse project and workspace files which can generate problems.

Only standard eclipse for Java is necessary to run this project. No special plugins necessary.

All git commits and command should be handled via the command line.

# Multiprocessing & Parallel computing in python

<http://docs.python.org/2/library/multiprocessing.html>

The [multiprocessing](http://docs.python.org/2/library/multiprocessing.html#module-multiprocessing) package offers both local and remote concurrency, effectively side-stepping the [*Global Interpreter Lock*](http://docs.python.org/2/glossary.html#term-global-interpreter-lock) by using subprocesses instead of threads. Due to this, the [multiprocessing](http://docs.python.org/2/library/multiprocessing.html#module-multiprocessing) module allows the programmer to **fully leverage multiple processors on a given machine**.

[multiprocessing](http://docs.python.org/2/library/multiprocessing.html#module-multiprocessing) supports two types of communication channel between processes:

queues

pipes

[multiprocessing](http://docs.python.org/2/library/multiprocessing.html#module-multiprocessing) contains equivalents of all the synchronization primitives from [threading](http://docs.python.org/2/library/threading.html#module-threading). For instance one can use a lock to ensure that only one process prints to standard output at a time.

**The Pool class represents a pool of worker processes**. It has methods which allows tasks to be offloaded to the worker processes in a few different ways.

**Monitoring the cores in linux**

sudo apt-get install sysstat

*watch* is used to run any designated [command](http://www.linfo.org/command.html) at regular intervals.

watch -d "mpstat -P ALL 1 1 | head -n 12”

**Calling system commands from python**

Look at the [subprocess module](http://docs.python.org/library/subprocess.html) in the stdlib:

from subprocess import call

call(["ls", "-l"])

The advantage of subprocess vs system is that it is more flexible (you can get the stdout, stderr, the "real" status code, better error handling, etc...). I think os.system is deprecated, too, or will be

The [subprocess](http://docs.python.org/2/library/subprocess.html#module-subprocess) module allows you to spawn new processes, connect to their input/output/error pipes, and obtain their return codes.

<http://docs.python.org/2/library/subprocess.html>