Building CompuCell3D on Mac OS X

Compiling CompuCell3D from source code requires several 3rd-party software libraries. Once all software dependencies have been satisfied, building CompuCell3D from source on Mac OS X systems is fairly straightforward.

Prerequisites

Software

gcc

Default compilers on Mac OS X are Clang and LLVM. Recent Xcode dev tools don't include standard gcc compilers.

To rely on a single C/C++ compiler, CC3D is compiled using standard *gcc* on Mac OS X as well. Various precompiled *gcc* packages for Mac OS X are available at http://hpc.sourceforge.net.

CC3D 3.6.2 source code is currently compiled with *gcc 4.6*. To compile CC3D with *gcc 4.7* or newer (on any platform), there are porting issues caused by name lookup changes in gcc. Adding the "-fpermissive" flag when invoking *gcc* allows *gcc 4.7* to compile CC3D.

cmake

CC3D is built using *cmake*. Binary distributions can be obtained from the Kitware cmake website .

Software Dependencies

Both the *gcc* binary distribution for OS X, as well as other required 3rd-party software libraries, can be placed in any directory. A subdirectory of `/Users/Shared/` can be used for system-wide access from any username.

A known working set of required software libraries and versions for compiling CompuCell3D on OS X 10.8:

- Qt - 4.8.3
- sip - 4.13.3
- PyQt - 4.9.4
- PyQwt - 5.2.0
- QScintilla - 2.6.2
- VTK - 5.8.0

For more details about building all the required 3rd-party software libraries from source code on Mac OS X, the Building CompuCell3D from source code on Mac OS X document is available on the CompuCell 3D Developer Site on github.iu.edu .

Hardware

CompuCell3D builds and runs on any Mac hardware capable of running OS X 10.8 "Mountain Lion".

Source Code

Once all 3rd-party software dependencies have been satisfied, obtain the source code from our SVN repository using the following command:

```
mkdir CC3D_GIT
cd CC3D_GIT
git clone -b 3.7.0 https://github.com/CompuCell3D/CompuCell3D.git .
```

This will download the source code into CC3D_GIT.

Compiling CC3D

A complete CompuCell3D distribution can be built with the following Python script. Set the PATH strings in the Python script to where *gcc* and other 3rd-party libraries have been installed. Running the following script will produce a complete CC3D 3.6.2 binary distribution on Mac OS X 10.8.x :

{{{#!/usr/bin/env python # <--- the above line asks the 'env' system command to find the python executable, and then executes it. # everything below is in python

this script builds CompuCell3D 3.6.2 on Mac OS X 10.8.x # #

assuming that you already have installed Qt on your system, and # # downloaded and built VTK, SIP, PyQt, QScintilla, etc... # # # # this script also assumes that you have downloaded and built # # gcc 4.7.1 (or newer) including its OpenMP support libraries # #

2010-2013 compucell3d.org # # edited by Mitja Hmeljak, # # based on 2009 script by Benjamin Zaitlen # #

*** Please note *** # # this file will start working *only* after you manually set: # the PATH_TO_WORK_DIR string to your OWN path where you want to build CC3D, # as for example "/Users/Shared/CC3Ddev/CC3D362build" # the QT_LIB_DIR to your OWN path where you have installed Qt libraries, # as for example "/Users/Shared/CC3Ddev/Qt-4.8.3/lib" # the PYQT_SITE_PACKAGES_DIR to your OWN path where you have installed PyQt, # as for example "/Library/Python/2.6/site-packages" # the QWT_LIB_DIR to your OWN path where you have installed qwt, # as for example "/Users/Shared/CC3Ddev/PyQwt-5.2.0/PyQwt-5.2.0/qwt-5.2/lib" # the VTK_BIN_AND_BUILD_DIR to your OWN path where you have installed VTK, # as for example "/Users/Shared/CC3Ddev/VTK_5.8.0_bin_and_build" # the GCC_DIR string to your OWN path where you have the GCC 4.7.1 distribution: # as for example "/Users/Shared/CC3Ddev/gcc_4.7.1" # the CC3D_LOCAL_GIT string to your OWN path where you keep the local CC3D 3.6.2 git repository: # as for example "/Users/Shared/CC3Ddev/git_CC3D/CompuCell3D/CompuCell3D" # the BIONET_LOCAL_GIT string to your OWN path where you keep the BIONET 0.0.6 git repository: # as for example "/Users/Shared/CC3Ddev/git_CC3D/CompuCell3D/BionetSolver/0.0.6" # the CELLDRAW_LOCAL_GIT string to your OWN path where you keep the CELLDRAW git repository: # as for example "/Users/Shared/CC3Ddev/git_CC3D/CompuCell3D/CellDraw/1.5.1/src"

PATH_TO_WORK_DIR = "/Users/Shared/CC3Ddev/CC3D362build" QT_LIB_DIR = "/Users/Shared/CC3Ddev/Qt-4.8.3/lib" PYQT_SITE_PACKAGES_DIR = "/Library/Python/2.6/site-packages" QWT_LIB_DIR = "/usr/local/qwt-5.2.1-svn/lib" VTK_BIN_AND_BUILD_DIR = "/Users/Shared/CC3Ddev/VTK_5.8.0_bin_and_build" GCC_DIR = "/Users/Shared/CC3Ddev/gcc_4.7.1" CC3D_LOCAL_GIT = "/Users/Shared/CC3Ddev/git_CC3D/CompuCell3D/CompuCell3D" BIONET_LOCAL_GIT = "/Users/Shared/CC3Ddev/git_CC3D/CompuCell3D/BionetSolver/0.0.6" CELLDRAW_LOCAL_GIT = "/Users/Shared/CC3Ddev/git_CC3D/CompuCell3D/CellDraw/1.5.1/src"

after you've set the above PATH strings with your own setting, run this script in Terminal.app

import sys, shutil, os from subprocess import call

print "##" print "building CompuCell3D 3.6.2 on Mac OS X 10.8" print "##"

if (PATH_TO_WORK_DIR == "") or (QT_LIB_DIR == "") or (PYQT_SITE_PACKAGES_DIR == "") or (VTK_BIN_AND_BUILD_DIR == "") or (GCC_DIR == "") or (CC3D_LOCAL_GIT == "") or (BIONET_LOCAL_GIT == "") or (CELLDRAW_LOCAL_GIT == ""):

> print "This file will start working *only* after you manually..." print "... set the PATH_TO_WORK_DIR string to your OWN path where you want to build CC3D," print "... set the QT_LIB_DIR string to your OWN path where you have installed Qt libraries,"
> print "... set the PYQT_SITE_PACKAGES_DIR string to your OWN path where you have installed PyQt," print "... set the QWT_LIB_DIR string to your OWN path where you have installed Qwt," print "... set the VTK_BIN_AND_BUILD_DIR to your OWN path where you have installed VTK," print "... set the GCC_DIR string to your OWN path where you have installed GCC 4.7.1," print "... set the CC3D_LOCAL_GIT string to your OWN path to your local CC3D 3.6.2 git copy." print "... set the BIONET_LOCAL_GIT string to your OWN path to your local BIONET 0.0.6 git copy." print "... and the CELLDRAW_LOCAL_GIT string to your OWN path to your local CellDraw git copy." print "" print "Please open this script in a text editor and read code comments for more information." print "" sys.exit(1)

else:

> print "=====>=====> PATH_TO_WORK_DIR = " + PATH_TO_WORK_DIR print "=====>=====> QT_LIB_DIR = " + QT_LIB_DIR print "=====>=====> PYQT_SITE_PACKAGES_DIR " + PYQT_SITE_PACKAGES_DIR print "=====>=====> QWT_LIB_DIR " + QWT_LIB_DIR print "=====>=====> VTK_BIN_AND_BUILD_DIR = " + VTK_BIN_AND_BUILD_DIR print "=====>=====> GCC_DIR = " + GCC_DIR print "=====>=====> CC3D_LOCAL_GIT = " + CC3D_LOCAL_GIT print "=====>=====> BIONET_LOCAL_GIT = " + BIONET_LOCAL_GIT print

"=====>=====> CELLDRAW_LOCAL_GIT = " + CELLDRAW_LOCAL_GIT print "... CC3D build starting ..."

set, clear and create directories # # for building CC3D using CMAKE, # # 3rd party supporting libraries, # # distribution binaries # #

print "=====>=====> the previous umask was =", os.umask(022) print "=====>=====> now the umask is =", os.umask(022)

CC3D_ENDUSER_DIR_NAME = 'CC3D_3.6.2_MacOSX_10.8' CC3D_ENDUSER_DIR_NAME_TMP = 'CC3D_3.6.2_MacOSX_10.8_tmp' CC3D_ARCHIVE = 'CC3D_3.6.2_MacOSX_10.8.zip'

WHY are in some "os.path.join()" calls passed *two* arguments to join directory paths, when the 2nd argument is an absolute path anyway?

# BUILD_DIR_FOR_CMAKE # #

# BUILD_DIR_FOR_CMAKE is the directory into which CMAKE will conduct the entire build: # # (this directory is NOT the place where we run the scripts to prepare a CC3D distribution) # BUILD_DIR_FOR_CMAKE=os.path.join(PATH_TO_WORK_DIR, 'build_dir_aka_cmake_install_dir') print "=====>=====> BUILD_DIR_FOR_CMAKE = " + BUILD_DIR_FOR_CMAKE

# first clean, then create the directory - if os.path.isdir(BUILD_DIR_FOR_CMAKE):

> print "=====>=====> Build directory (for CMAKE doing CC3D) exists... Removing ", BUILD_DIR_FOR_CMAKE, " and creating new directory." shutil.rmtree(BUILD_DIR_FOR_CMAKE)

# create the directory used during CMAKE's build and its own installation procedure: os.mkdir(BUILD_DIR_FOR_CMAKE)

DEP_DIR creation # #

DEP_DIR is the directory into which we place ALL 3rd party dependency libraries, for building # DEP_DIR=os.path.join(PATH_TO_WORK_DIR, 'third_party_support_libs_dependencies') print "=====>=====> DEP_DIR = " + DEP_DIR

# For CC3D 3.6.2, it is necessary to manually include two libraries to support OpenMP: OPENMP_LIBS_DEP_DIR=os.path.join(DEP_DIR,'OpenMPlib') print "=====>=====> OPENMP_LIBS_DEP_DIR = " + OPENMP_LIBS_DEP_DIR

# For CC3D 3.6.2, it is necessary to manually include QScintilla libraries for twedit++: QSCINTILLA_LIBS_DEP_DIR=os.path.join(DEP_DIR,'QScintillalib') print "=====>=====> QSCINTILLA_LIBS_DEP_DIR = " + QSCINTILLA_LIBS_DEP_DIR

# For CC3D 3.6.2, only Qt libraries go in the DEP_DIR directory. # (in CC3D 3.4.1 and previous, versions of VTK, Qt, etc. were placed in other subdirectories) # The Qt binaries copied from within the Qt installation frameworks go in here: QT_DEPS_DEP_DIR=os.path.join(DEP_DIR,'Deps') print "=====>=====> QT_DEPS_DEP_DIR = " + QT_DEPS_DEP_DIR

# For CC3D 3.6.2, almost all dependencies are placed temporarily in this directory: FOR_PLAYER_DEP_DIR=os.path.join(DEP_DIR,'ForPlayerDir') print "=====>=====> FOR_PLAYER_DEP_DIR = " + FOR_PLAYER_DEP_DIR # # Subdirectories have to be created (inside what will become "player/" in the final distribution) into which various libraries will be copied: # VTKLibs/ <--- all .dylib library files from the VTK distribution's "lib/vtk-5.8/" directory # vtk/ <--- all files from the VTK distribution's "Wrapping/Python/vtk/" for Python AND ALSO all files ending in .so from VTK's build/bin directory # PyQt4/ <--- the "PyQt4" directory from the system-wide "site-packages/" PyQt distribution # no subdirectory for sip* files from the system-wide "site-packages/" PyQt distribution # so the above directory for 3rd party support libraries has to contain the following: DEP_LIBVTK_DIR=os.path.join(FOR_PLAYER_DEP_DIR,'VTKLibs') print "=====>=====> DEP_LIBVTK_DIR = " + DEP_LIBVTK_DIR DEP_VTK_DIR=os.path.join(FOR_PLAYER_DEP_DIR,'vtk') print "=====>=====> DEP_VTK_DIR = " + DEP_VTK_DIR DEP_PYQT_DIR=os.path.join(FOR_PLAYER_DEP_DIR,'PyQt4') print "=====>=====> DEP_PYQT_DIR = " + DEP_PYQT_DIR DEP_PYQWT_DIR=os.path.join(DEP_PYQT_DIR,'Qwt5')

if os.path.isdir(DEP_DIR):

> print "=====>=====> 3rd party dependency libraries directory exists... Removing ", DEP_DIR, " and creating new 3rd party directory." shutil.rmtree(DEP_DIR)

print "=====>=====> Creating 3rd party dependency libraries directories, DEP_DIR = ", DEP_DIR os.mkdir(DEP_DIR) print "=====>=====> Creating 3rd party dependency libraries directories, OPENMP_LIBS_DEP_DIR = ", OPENMP_LIBS_DEP_DIR os.mkdir(OPENMP_LIBS_DEP_DIR) print "=====>=====> Creating 3rd party dependency libraries directories, QSCINTILLA_LIBS_DEP_DIR = ", QSCINTILLA_LIBS_DEP_DIR os.mkdir(QSCINTILLA_LIBS_DEP_DIR) print "=====>=====> Creating 3rd party dependency libraries directories, QT_DEPS_DEP_DIR = ", QT_DEPS_DEP_DIR os.mkdir(QT_DEPS_DEP_DIR) print "=====>=====> Creating 3rd party dependency libraries directories, FOR_PLAYER_DEP_DIR = ", FOR_PLAYER_DEP_DIR os.mkdir(FOR_PLAYER_DEP_DIR) print "=====>=====> Creating 3rd party dependency libraries directories, DEP_LIBVTK_DIR = ", DEP_LIBVTK_DIR os.mkdir(DEP_LIBVTK_DIR) print "=====>=====> Creating 3rd party dependency libraries directories, DEP_VTK_DIR = ", DEP_VTK_DIR os.mkdir(DEP_VTK_DIR) print "=====>=====> Creating 3rd party dependency libraries directories, DEP_PYQT_DIR = ", DEP_PYQT_DIR os.mkdir(DEP_PYQT_DIR)

DEP_DIR file copy # #

print "=====>=====> now getting all 3rd party (previously compiled) dependency libraries into one place <=====<====="

os.chdir(OPENMP_LIBS_DEP_DIR) call('pwd') # in here we copy: three library files grabbed from the GCC 4.7.1 distribution we have on our system: # (if you download or build GCC 4.7.1 separately, modify the GCC_DIR path at the top of this script) call('rsync -rlPpa '+GCC_DIR+'/lib/libgomp.1.dylib . ',shell=True) call('rsync -rlPpa '+GCC_DIR+'/lib/libgcc_s.1.dylib . ',shell=True) call('rsync -rlPpa '+GCC_DIR+'/lib/libstdc++.6.dylib . ',shell=True)

os.chdir(QSCINTILLA_LIBS_DEP_DIR) call('pwd') # in here we copy: a library file grabbed from the QScintilla distribution, and create the necessary symlinks: call('rsync -rlPpa '+QT_LIB_DIR+'/libqscintilla2.8.0.2.dylib . ',shell=True) call('ln -f -s ./libqscintilla2.8.0.2.dylib ./libqscintilla2.dylib',shell=True) call('ln -f -s ./libqscintilla2.8.0.2.dylib ./libqscintilla2.8.dylib',shell=True) call('ln -f -s ./libqscintilla2.8.0.2.dylib ./libqscintilla2.8.0.dylib',shell=True)

os.chdir(QT_DEPS_DEP_DIR) call('pwd') # in here we copy: four binary files grabbed from WITHIN the frameworks that are part of the main Qt distribution: # (this could be done cleaner on Mac OS X - read developer.apple.com about it) call('rsync -rlPpa '+QT_LIB_DIR+'/QtGui.framework/Versions/Current/QtGui . ',shell=True) call('rsync -rlPpa '+QT_LIB_DIR+'/QtCore.framework/Versions/Current/QtCore . ',shell=True) call('rsync -rlPpa '+QT_LIB_DIR+'/QtNetwork.framework/Versions/Current/QtNetwork . ',shell=True) call('rsync -rlPpa '+QT_LIB_DIR+'/QtOpenGL.framework/Versions/Current/QtOpenGL . ',shell=True) call('rsync -rlPpa '+QT_LIB_DIR+'/QtXml.framework/Versions/Current/QtXml . ',shell=True) # in here also copy: the QtSvg library, even if it isn't used by any CC3D code, but qwt links to it: call('rsync -rlPpa '+QT_LIB_DIR+'/QtSvg.framework/Versions/Current/QtSvg . ',shell=True) # in here we copy: ALSO a .nib directory (although it might be cleaner to provide the entire framework) call('rsync -rlPpa '+QT_LIB_DIR+'/QtGui.framework/Versions/Current/Resources/qt_menu.nib . ',shell=True)

# 2011: include QScintilla libraries in the install: # 2012: unnecessary, because they're already included in the QSCINTILLA_LIBS_DEP_DIR and subsequently moved into the lib/ directory. # call('rsync -rlPpa '+QT_LIB_DIR+'/libqscintilla2.8.0.2.dylib . ',shell=True) # call('ln -f -s ./libqscintilla2.8.0.2.dylib ./libqscintilla2.dylib',shell=True) # call('ln -f -s ./libqscintilla2.8.0.2.dylib ./libqscintilla2.8.dylib',shell=True) # call('ln -f -s ./libqscintilla2.8.0.2.dylib ./libqscintilla2.8.0.dylib',shell=True)

os.chdir(DEP_LIBVTK_DIR) call('pwd') # in here we copy: all .dylib files (including all the symbolic links to them in the same dir) call('rsync -rlPpa '+VTK_BIN_AND_BUILD_DIR+'/lib/vtk-5.8/*dylib . ',shell=True)

a useful note from rsync's man page: # # A trailing slash on the source changes this behavior to avoid creating an additional directory level at the destination. # You can think of a trailing / on a source as meaning "copy the contents of this directory" as opposed to "copy the directory by name", # but in both cases the attributes of the containing directory are transferred to the containing directory on the destination. # In other words, each of the following commands copies the files in the same way, # including their setting of the attributes of /dest/foo: # # rsync -av /src/foo /dest # rsync -av /src/foo/ /dest/foo # os.chdir(DEP_VTK_DIR) call('pwd') # in here we copy: all files from VTK's Wrapping directory for Python: call('rsync -rlPpa '+VTK_BIN_AND_BUILD_DIR+'/Wrapping/Python/vtk/ . ',shell=True) # in here we ALSO copy: all files ending in .so from VTK's build/bin directory: call('rsync -rlPpa '+VTK_BIN_AND_BUILD_DIR+'/bin/*.so . ',shell=True)

os.chdir(DEP_PYQT_DIR) call('pwd') # in here we copy: the PyQt4 directory as created by the original (system-wide) PyQt install procedure) call('rsync -rlPpa '+PYQT_SITE_PACKAGES_DIR+'/PyQt4/ . ',shell=True)

os.chdir(DEP_PYQWT_DIR) call('pwd') # in here we copy: a library file grabbed from the qwt build, and create the necessary symlinks: call('rsync -rlPpa '+QWT_LIB_DIR+'/libqwt.5.2.1.dylib . ',shell=True) call('ln -f -s ./libqwt.5.2.1.dylib ./libqwt.dylib',shell=True) call('ln -f -s ./libqwt.5.2.1.dylib ./libqwt.5.dylib',shell=True) call('ln -f -s ./libqwt.5.2.1.dylib ./libqwt.5.2.dylib',shell=True)

# PyQwt and qwt fix - change the id and the paths so that the loader can find libqwt: call('install_name_tool -id @loader_path/libqwt.5.dylib libqwt.5.2.1.dylib',shell=True) call('otool -L libqwt.5.2.1.dylib',shell=True) call('install_name_tool -change /usr/local/qwt-5.2.1-svn/lib/libqwt.5.dylib @loader_path/libqwt.5.dylib Qwt.so',shell=True) call('otool -L Qwt.so',shell=True) call('install_name_tool -change /usr/local/qwt-5.2.1-svn/lib/libqwt.5.dylib @loader_path/libqwt.5.dylib _iqt.so',shell=True) call('otool -L _iqt.so',shell=True)

os.chdir(FOR_PLAYER_DEP_DIR) call('pwd') # in here we copy: some files from Python's directory into FOR_PLAYER_DEP_DIR: call('rsync -rlPpa '+PYQT_SITE_PACKAGES_DIR+'/sip.so . ',shell=True) call('rsync -rlPpa '+PYQT_SITE_PACKAGES_DIR+'/sipconfig.py . ',shell=True) call('rsync -rlPpa '+PYQT_SITE_PACKAGES_DIR+'/sipdistutils.py . ',shell=True)

CUR_DIR # #

# other references: CUR_DIR=os.path.join(os.getcwd(), PATH_TO_WORK_DIR) print "=====>=====> CUR_DIR = " + CUR_DIR

BIN_DIR # #

# here will go the resulting complete distribution archive: BIN_DIR=os.path.join(CUR_DIR,

CC3D_ENDUSER_DIR_NAME) print "=====>=====> BIN_DIR = " + BIN_DIR

# first clean, then create - if os.path.isdir(BIN_DIR):

> print "=====>=====> Binary distribution directory exists... Removing", BIN_DIR, " and creating new binary directory." shutil.rmtree(BIN_DIR)

# create a directory where the complete cc3d binary+supporting libraries will be placed for distribution os.mkdir(BIN_DIR)

SRC_DIR # #

# this MUST be a directory called "CompuCell3D" because we grab the "CompuCell3D" directory from the git repository, it's hardcoded in this script: SRC_DIR=os.path.join(PATH_TO_WORK_DIR, 'CompuCell3D') print "=====>=====> SRC_DIR = " + SRC_DIR

# assure that the source directory is cleared, we're going to build cc3d from it. # the following has to be removed but NOT recreated right away, because it'll be downloaded below using git: if os.path.isdir(SRC_DIR):

> print "=====>=====> Source directory exists... Removing", SRC_DIR, " <=====<=====" shutil.rmtree(SRC_DIR)

build CC3D # #

print "=====>=====> now finally BUILD CompuCell 3D <=====<====="

print "=====>=====> obtaining the latest CC3D source code from our OWN local copy of the CC3D 3.6.2 git repository:" # cd to the directory we're using to hold it all, and grab the latest source code from our OWN local copy of the CC3D 3.6.2 git repository: os.chdir(CUR_DIR) call('pwd') # call('git ...something... http://code.compucell3d.org/svn/cc3d/branch/3.6.2',shell=True) call('rsync -rlPpa '+CC3D_LOCAL_GIT+' . ', shell=True)

print "=====>=====> building CC3D:" # cd to the directory holding all the cc3d source code, and compile it: os.chdir(SRC_DIR) call('pwd')

print "=====>=====> prepare all make files using cmake with a few command-line settings/options:"

# this setting would build "fat" 32bit+64bit code, but the OpenMP libraries we use are 64 bit only anyway: # call('cmake -DCMAKE_INSTALL_PREFIX:PATH='+BUILD_DIR_FOR_CMAKE+ " -DCMAKE_OSX_DEPLOYMENT_TARGET:STRING=10.6 -DVTK_DIR:PATH="+VTK_BIN_AND_BUILD_DIR+"/lib/vtk-5.8 -DCMAKE_OSX_ARCHITECTURES:STRING=i386\;x86_64" ,shell=True)

# this setting would build CC3D 3.6.2 (64bit code only) using gcc 4.2.0 as included with the Xcode distribution : #call('cmake -DCMAKE_INSTALL_PREFIX:PATH='+BUILD_DIR_FOR_CMAKE+ " -DCMAKE_OSX_DEPLOYMENT_TARGET:STRING=10.6 -DVTK_DIR:PATH="+VTK_BIN_AND_BUILD_DIR+"/lib/vtk-5.8 -DCMAKE_OSX_ARCHITECTURES:STRING=x86_64" ,shell=True)

# this setting builds CC3D 3.6.2 (64bit code only) using gcc 4.7.1 from its own separate compiler distribution directory: # add the -ftree-vectorizer-verbose=1 (or other levels) flag, to see what functions the vectorizer addresses # add the -Q flag, to list print out each function name as it is compiled, and print some statistics about each pass when it finishes # add the -ftime-report -fmem-report, to list detailed reports about time and memory used by gcc # add the -O3 flag, to add maximum GCC optimization levels # add the -fpermissive (required!) to compile CC3D using gcc 4.7.1, because of name lookup changes. See http://gcc.gnu.org/gcc-4.7/porting_to.html

# call('cmake -DCMAKE_C_COMPILER='+GCC_DIR+'/bin/gcc -DCMAKE_CXX_COMPILER='+GCC_DIR+'/bin/g++ -DCMAKE_CXX_FLAGS="-O3 -g -ftree-vectorizer-verbose=1 -time -m64" -DCMAKE_C_FLAGS="-O3 -g -ftree-vectorizer-verbose=1 -time -m64" -DCMAKE_INSTALL_PREFIX:PATH='+BUILD_DIR_FOR_CMAKE+ " -DVTK_DIR:PATH="+VTK_BIN_AND_BUILD_DIR+"/lib/vtk-5.8" , shell=True)

# Randy's cmake call, to which we add the VTK_DIR definition: # call('cmake -DCMAKE_OSX_DEPLOYMENT_TARGET:STRING=10.6 -DCMAKE_INSTALL_PREFIX:PATH='+BUILD_DIR_FOR_CMAKE+' -DCMAKE_CXX_COMPILER:FILEPATH='+GCC_DIR+'/bin/g++ -DCMAKE_C_COMPILER:FILEPATH='+GCC_DIR+'/bin/gcc -DVTK_DIR:PATH='+VTK_BIN_AND_BUILD_DIR+'/lib/vtk-5.8 -DCMAKE_CXX_FLAGS="-mmacosx-version-min=10.6 -O3 -g -ftree-vectorizer-verbose=1 -time -m64" -DCMAKE_C_FLAGS="-mmacosx-version-min=10.6 -O3 -g -ftree-vectorizer-verbose=1 -time -m64"', shell=True)

# cmake call for 10.6 backwards-compatible compile: # call('cmake -DCMAKE_OSX_DEPLOYMENT_TARGET:STRING=10.6 -DCMAKE_C_COMPILER:FILEPATH='+GCC_DIR+'/bin/gcc -DCMAKE_CXX_COMPILER:FILEPATH='+GCC_DIR+'/bin/g++ -DCMAKE_INSTALL_PREFIX:PATH='+BUILD_DIR_FOR_CMAKE+ ' -DVTK_DIR:PATH='+VTK_BIN_AND_BUILD_DIR+'/lib/vtk-5.8'+ ' -DCMAKE_CXX_FLAGS="-mmacosx-version-min=10.6 -O3 -g -ftree-vectorizer-verbose=1 -time -m64" -DCMAKE_C_FLAGS="-mmacosx-version-min=10.6 -O3 -g -ftree-vectorizer-verbose=1 -time -m64"' , shell=True)

# cmake call without 10.6 backwards-compatible compile flags: # -DCMAKE_OSX_DEPLOYMENT_TARGET:STRING=10.6 call('cmake -DPYTHON_EXECUTABLE:FILEPATH=/usr/bin/python2.6 -DPYTHON_INCLUDE_DIR:PATH=/System/Library/Frameworks/Python.framework/Versions/2.6/Headers -

DPYTHON_LIBRARY:FILEPATH=/usr/lib/libpython2.6.dylib ' + ' -DCMAKE_C_COMPILER:FILEPATH='+GCC_DIR+'/bin/gcc -DCMAKE_CXX_COMPILER:FILEPATH='+GCC_DIR+'/bin/g++ -DCMAKE_INSTALL_PREFIX:PATH='+BUILD_DIR_FOR_CMAKE+ ' -DVTK_DIR:PATH='+VTK_BIN_AND_BUILD_DIR+'/lib/vtk-5.8'+ ' -DCMAKE_CXX_FLAGS="-O3 -g -fpermissive -m64" -DCMAKE_C_FLAGS="-O3 -g -fpermissive -m64"' , shell=True)

print "=====>=====> make CC3D:" # avoid parallel make until all works" # call('make install',shell=True) call('make install -j16',shell=True) os.chdir(CUR_DIR) call('pwd')

copy files into CC3D distribution directory # #

print "=====>=====> copy all the 3rd party supporting libraries into one directory for distribution" # now copy all the supporting libraries that are necessary for the cc3d player into the directory for distribution:

call('rsync -rlPp '+OPENMP_LIBS_DEP_DIR+'/ '+BIN_DIR+'/lib',shell=True)

call('rsync -rlPpa '+QSCINTILLA_LIBS_DEP_DIR+'/ '+BIN_DIR+'/lib',shell=True)

call('rsync -rlPp '+FOR_PLAYER_DEP_DIR+'/ '+BIN_DIR+'/player',shell=True)

call('rsync -rlPp '+QT_DEPS_DEP_DIR+' ' + BIN_DIR,shell=True)

print "=====>=====> copy the compiled CC3D from the install directory used by cmake, and various shell files"

# new 2010 style: call('chmod ugo+rx compucell3d.command compucell3d.sh runScript.command runScript.sh twedit++.command twedit++.sh celldraw.command celldraw.sh ', shell=True) call('rsync -rPlp '+BUILD_DIR_FOR_CMAKE+'/ compucell3d.command runScript.command twedit++.command celldraw.command '+BIN_DIR,shell=True)

# remove twedit++.sh from the distribution directory and replace it with our newer version, this could be fixed in CMake files <

TODO if os.path.isfile(BIN_DIR+'/twedit++.sh'):

    os.remove(BIN_DIR+'/twedit++.sh')

call('rsync -rPlp twedit++.sh '+BIN_DIR,shell=True)

# remove compucell3d.sh from the distribution directory and replace it with our newer version, this could be fixed in CMake files <

TODO if os.path.isfile(BIN_DIR+'/compucell3d.sh'):

    os.remove(BIN_DIR+'/compucell3d.sh')

call('rsync -rPlp compucell3d.sh '+BIN_DIR,shell=True)

# remove celldraw.sh from the distribution directory and replace it with our newer version, this could be fixed in CMake files <

TODO if os.path.isfile(BIN_DIR+'/celldraw.sh'):

    os.remove(BIN_DIR+'/celldraw.sh')

call('rsync -rPlp celldraw.sh '+BIN_DIR,shell=True)

# rename old runScript.sh file in the distribution directory: if os.path.isfile(BIN_DIR+'/runScript.sh'):

    os.rename(BIN_DIR+'/runScript.sh', BIN_DIR+'/runScript_older.sh')

new 2011.07 include BionetSolver in the binary build: # #

# this MUST be a directory called "0.0.6" because we grab the "0.0.6" version from the git repository, it's hardcoded in this script: BIONET_SRC_DIR=os.path.join(PATH_TO_WORK_DIR, '0.0.6') print "=====>=====> BIONET_SRC_DIR = " + BIONET_SRC_DIR

# assure that the source directory is cleared, we're going to build BionetSolver from it. # the diretory has to be removed but NOT recreated right away, because it'll be downloaded using git: if os.path.isdir(BIONET_SRC_DIR):

    print "=====>=====> BionetSolver directory exists... Removing", BIONET_SRC_DIR, " <=====<=====" shutil.rmtree(BIONET_SRC_DIR)

# BIONET_BUILD_DIR_FOR_CMAKE is the directory into which CMAKE will conduct the entire build: BIONET_BUILD_DIR_FOR_CMAKE=os.path.join(PATH_TO_WORK_DIR, 'bionet_build_dir') print "=====>=====> BIONET_BUILD_DIR_FOR_CMAKE = " + BIONET_BUILD_DIR_FOR_CMAKE

# first clean, then create the directory - if os.path.isdir(BIONET_BUILD_DIR_FOR_CMAKE):

print "=====>=====> Build directory (for CMAKE doing BionetSolver) exists... Removing ", BIONET_BUILD_DIR_FOR_CMAKE, " and creating new directory."
shutil.rmtree(BIONET_BUILD_DIR_FOR_CMAKE)

# create the directory used during CMAKE's build and its own installation procedure: os.mkdir(BIONET_BUILD_DIR_FOR_CMAKE)

print "=====>=====> now BUILD BionetSolver <=====<====="

print "=====>=====> obtaining the latest BionetSolver source code from our OWN local copy of the BIONET 0.0.6 git repository:" # cd to the directory we're using to hold it all, and grab the latest source code from our OWN local copy of the BIONET 0.0.6 git repository: os.chdir(CUR_DIR) call('pwd') #call('git ...something... export http://code.compucell3d.org/svn/cc3d/branch/BionetSolver/0.0.6',shell=True) call('rsync -rlPpa '+BIONET_LOCAL_GIT+' . ', shell=True)

print "=====>=====> building BionetSolver:" # cd to the directory holding all the cc3d source code, and compile it: os.chdir(BIONET_SRC_DIR) call('pwd')

print "=====>=====> prepare all make files using cmake with a few command-line settings/options:" # this setting builds bionetsolver (64bit code only) using gcc 4.7.1 from its own separate compiler distribution directory: # add the -ftree-vectorizer-verbose=1 (or other levels) flag, to see what functions the vectorizer addresses # add the -Q flag, to list print out each function name as it is compiled, and print some statistics about each pass when it finishes # add the -ftime-report -fmem-report, to list detailed reports about time and memory used by gcc # add the -O3 flag, to add maximum GCC optimization levels # add the -fpermissive (required!) to compile CC3D using gcc 4.7.1, because of name lookup changes. See http://gcc.gnu.org/gcc-4.7/porting_to.html

# cmake call for 10.6 backwards-compatible compile: # call('cmake -DCMAKE_OSX_DEPLOYMENT_TARGET:STRING=10.6 -DCMAKE_C_COMPILER='+GCC_DIR+'/bin/gcc -DCMAKE_CXX_COMPILER='+GCC_DIR+'/bin/g++ -DCMAKE_CXX_FLAGS="-mmacosx-version-min=10.6 -O3 -g -ftree-vectorizer-verbose=1 -time -m64" -DCMAKE_C_FLAGS="-mmacosx-version-min=10.6 -O3 -g -ftree-vectorizer-verbose=1 -time -m64" -DCMAKE_INSTALL_PREFIX:PATH='+BIONET_BUILD_DIR_FOR_CMAKE , shell=True)

# cmake call without 10.6 backwards-compatible compile flags: # -DCMAKE_OSX_DEPLOYMENT_TARGET:STRING=10.6 call('cmake -DPYTHON_EXECUTABLE:FILEPATH=/usr/bin/python2.6 -DPYTHON_INCLUDE_DIR:PATH=/System/Library/Frameworks/Python.framework/Versions/2.6/Headers -DPYTHON_LIBRARY:FILEPATH=/usr/lib/libpython2.6.dylib ' + ' -DCMAKE_C_COMPILER='+GCC_DIR+'/bin/gcc -DCMAKE_CXX_COMPILER='+GCC_DIR+'/bin/g++ -DCMAKE_CXX_FLAGS="-O3 -g -fpermissive -m64" -DCMAKE_C_FLAGS="-O3 -g -fpermissive -m64" -DCMAKE_INSTALL_PREFIX:PATH='+BIONET_BUILD_DIR_FOR_CMAKE , shell=True)

# the following would build bionetsolver with the gcc compiler provided with Mac OS X 10.8: # cmake -DCMAKE_INSTALL_PREFIX:PATH=//Users/Shared/CC3Ddev/BionetworkSolver/fsht/BionetSolver/build_test_20110718 -DCMAKE_OSX_DEPLOYMENT_TARGET:STRING=10.6 -DCMAKE_OSX_ARCHITECTURES:STRING=x86_64

print "=====>=====> make BionetSolver:" # avoid parallel make until all works" # call('make install',shell=True) call('make install -j16',shell=True) os.chdir(BIONET_BUILD_DIR_FOR_CMAKE) call('pwd') call('find . -exec setlabel -s Orange {} \;',shell=True)

# copy BionetSolver files into CC3D distribution directory

call('rsync -rlPp '+BIONET_BUILD_DIR_FOR_CMAKE+'/Demos/BionetSolverExamples '+BIN_DIR+'/Demos',shell=True)

# reminder from rsync's man page: # a trailing / on a source as meaning "copy the contents of this directory" as opposed to "copy the directory by name" : call('rsync -rlPp '+BIONET_BUILD_DIR_FOR_CMAKE+'/include/ '+BIN_DIR+'/include ',shell=True)

call('rsync -rlPp '+BIONET_BUILD_DIR_FOR_CMAKE+'/lib/python/ '+BIN_DIR+'/lib/python',shell=True)

call('rsync -rlPp '+BIONET_BUILD_DIR_FOR_CMAKE+'/lib/pkgconfig '+BIN_DIR+'/lib',shell=True)

call('rsync -rlPp '+BIONET_BUILD_DIR_FOR_CMAKE+'/lib/*.a '+BIN_DIR+'/lib',shell=True)

call('rsync -rlPp '+BIONET_BUILD_DIR_FOR_CMAKE+'/lib/*.dylib '+BIN_DIR+'/lib',shell=True)

call('rsync -rlPp '+BIONET_BUILD_DIR_FOR_CMAKE+'/pythonSetupScripts/ '+BIN_DIR+'/pythonSetupScripts ',shell=True)

os.chdir(BIN_DIR)

# BionetSolver build and packaging completed #

new 2012.02 include CellDraw in the binary build script: # #

print "=====>=====> include CellDraw:" os.chdir(CUR_DIR) call('pwd')

CELLDRAW_SRC_DIR=os.path.join(PATH_TO_WORK_DIR, 'build_CellDraw_src_dir') print "=====>=====> CELLDRAW_SRC_DIR = " + CELLDRAW_SRC_DIR

# assure that the source directory is cleared, we're going to build CellDraw from it. # the diretory has to be removed and recreated right away, because we'll place CellDraw's source code dir into it from git: if os.path.isdir(CELLDRAW_SRC_DIR):

> print "=====>=====> CellDraw directory exists... Removing", CELLDRAW_SRC_DIR, " and creating new directory." shutil.rmtree(CELLDRAW_SRC_DIR)

# create the directory used during CellDraw installation procedure: os.mkdir(CELLDRAW_SRC_DIR)

print "=====>=====> now BUILD CellDraw <=====<====="

print "=====>=====> obtaining the latest CellDraw source code from our OWN local copy of the CellDraw git repository:" # cd to the directory we're using to hold it all, and grab the latest source code from our OWN local copy of the CellDraw git repository: os.chdir(CELLDRAW_SRC_DIR) call('pwd') #call('git ...something... http://code.compucell3d.org/svn/cc3d/branch/CellDraw/1.5.1/src',shell=True) call('rsync -rlPpa '+CELLDRAW_LOCAL_GIT+' . ', shell=True) call('\mv -f ./src '+BIN_DIR+'/CellDraw', shell=True)

os.chdir(CUR_DIR) call('pwd')

# CellDraw build and packaging completed #

print "=====>=====> clearing all the *.pyc, .DS_Store, .git/ and .svn/ from the CC3D directory:"

call("pwd", shell=True) # remove any precompiled python code, and any Mac OS X Finder-specific files: # call("find . -type f -iname \'*.pyc\' -exec ls -lad {} \;", shell=True) call("find . -type f -iname \'*.pyc\' -exec rm -f {} \;", shell=True) # call("find . -type f -iname \'*.pyc\' -exec ls -lad {} \;", shell=True)

# call("find . -type f -iname \'.DS_Store\' -exec ls -lad {} \;", shell=True) call("find . -type f -iname \'.DS_Store\' -exec rm -f {} \;", shell=True) # call("find . -type f -iname \'.DS_Store\' -exec ls -lad {} \;", shell=True)

# also remove any .svn directories from the distributed archive: # call("find . -type d -iname \'.svn\' -exec ls -lad {} \;", shell=True) call("find . -type d -iname \'.svn\' -exec rm -rf {} \;", shell=True) # call("find . -type d -iname \'.svn\' -exec ls -lad {} \;", shell=True)

# also remove any .git directories from the distributed archive: # call("find . -type d -iname \'.git\' -exec ls -lad {} \;", shell=True) call("find . -type d -iname \'.git\' -exec rm -rf {} \;", shell=True) # call("find . -type d -iname \'.git\' -exec ls -lad {} \;", shell=True)

# create the .zip archive for distribution: print "=====>=====> zipping the CC3D directory using \'ditto\':" os.chdir(CUR_DIR) call("rm -f " + CC3D_ARCHIVE, shell=True) call('ditto -rsrcFork -arch x86_64 ' + CC3D_ENDUSER_DIR_NAME + ' ' + CC3D_ENDUSER_DIR_NAME_TMP, shell=True) call("rm -rf " + CC3D_ENDUSER_DIR_NAME, shell=True) call('mv ' + CC3D_ENDUSER_DIR_NAME_TMP + ' ' + CC3D_ENDUSER_DIR_NAME, shell=True) call('ditto -c -k --keepParent -rsrcFork ' + CC3D_ENDUSER_DIR_NAME + ' ' + CC3D_ARCHIVE, shell=True) call("ls -lad " + CC3D_ARCHIVE, shell=True)

print "=====>=====>================<=====<=====" print "===> CC3D 3.6.2 distribution ready!! <===" print "=====>=====>================<=====<=====" }}}