University of Washington

LibStructural / LibLA Source Package

Build instructions for the library and language bindings

Frank Bergmann
3/1/2009

LibStructural / LibLA Source Package

This document describes how to build and install the libStructural / libLA libraries. These libraries provide basic linear algebra functionality and analysis methods related to the stoichiometry matrix, including moiety conservation and flux balance information. For more information on the functionality of these libraries see:

http://sys-bio.org/fbergman/libstructural/

The Structure of the Source Package

The source package uses the following directory structure:

```
/
        - doc
        - dependencies
                - lib
                - include
        - include
        - examples
        - LibLA
        - LibStructural
        - LibStructuralCSharp
        - TestProgram
        - SWIG
```

The doc directory contains the API documentation (as generated by doxygen) for the libraries. In the examples directory you will find examples on how to use the library from C / C++ as well as from generated language bindings. The include directory contains all include files needed to use the libStructural library.

Of special interest is the dependencies directory. The source distribution contains pre-build libraries for the CLAPACK library for windows systems which also includes a copy of the libSBML lib-file in the dependencies/lib directory, and the libSBML include directory “sbml” in the dependencies/include.

The directory ‘TestProgram’ contains a basic test program that uses the libStructural library and runs through a directory of SBML files in order to validate all structural matrices.

Finally the SWIG directory contains the files needed to build language bindings for other languages as Java / C# / Python / Ruby or R.

Building the libraries

Dependencies

Prior to building the libraries please make sure that you have the CLAPACK libraries installed on your system:

- On OS X Systems: These libraries are installed with the operating system, with the vecLib framework. There is nothing further to be done for CLAPACK under OS X.
- On Linux Systems: The needed libraries are in various packages dependent on the linux distribution you have installed. You need to install packages named ‘gfortran’, ‘blas’, ‘lapack’ or similar ones.
- On Windows Systems: The CLAPACK libraries are already available with distribution in the dependencies/lib directory.

The libStructural library is able to operate on SBML files, however for this to work it needs the libSBML library. Thus this library will have to be installed first:

- On OSX and Linux Systems: Download the libSBML source package from:

  https://sourceforge.net/project/showfiles.php?group_id=71971&package_id=71670

  unpack it: unzip libSBML-<version>-src.zip
  configure: ./configure <any arguments you might want to have, or none to take defaults>
  compile: make
  and install: sudo make install

  if you don’t install it into the default directory /usr/local you might have to slightly alter the configuration files for libStructural.

- On Windows systems: Download and install the libSBML windows package:

  https://sourceforge.net/project/showfiles.php?group_id=71971&package_id=71670

  i.e. download libSBML-3.3.1-expat.exe

  install the library by running the exe program, and note the directory where libSBML is installed to. Next copy the lib-files from the directory and copy it into the libStructural/dependencies/lib directory, and the include/sbml directory into the libStructural/dependencies/include directory

These are all dependencies needed for libStructural. With these in place you are now ready to build the library.

Disabling Support for SBML

In case SBML support is not desired, it is also possible to disable all calls to libSBML by defining the Pre-processor Macro:

    NO_SBML

This can be done in many different places depending on how you build the libraries:

- The call to qmake, by adding the option “DEFINES+=NO_SBML”
- When using Scons, just add –DNO_SBML in the SConstruct file to the ‘compileFlags’ variable
- Or in Visual Studio to the Project Properties\Configuration Properties\C++\Prepocessor\Prepocessor Definitions. This will have to be repeated for the project configurations “release” and “release-dll” in the libstructural project.

Compiling the libraries on Linux / OS X

In order to build the library on Linux and OS X operating systems we include build scripts for the common build tools scons (http://scons.org) or qmake (as included with all Qt-versions http://trolltech.com/). You will need to install them first. After that building the libraries will be completed with one command:

- For scons: just run ‘scons’ in the libStructural directory, and after that you will find the library in the ./lib directory. NOTE: by default scons will link against bz2, assuming you compiled libSBML with bzip2 support. In the case you don’t want to link against it use the option BZIP=no:

  ```
  scons BZIP=no
  ```

  further options: for OSX you can disable building Universal Binaries by adding UNIVERSAL=no. Finally you can build an example by specifying EXAMPLE=yes (this will build the example below).
- For qmake: run ‘qmake’ in the libStructural directory, and after that ‘make’ you will find the libraries in the ./build directory.

Now you are ready to use the libraries and can test them by compiling one of the example files.

For example:

```
cd examples/c

gcc loadstoichiometry.c –I../../include –L../../lib –lLibStructural –o loadstoichiometry
```

Compiling the libraries under Windows operating systems

On windows we provide Visual Studio project files, compatible with Visual Studio (http://microsoft.com/vstudio), version 2005, 2008 (and the corresponding free Express editions). All that is needed is to double-click on the solution files and build them. After building the libraries they will be in the ./bin folder.

It is also possible to use the qmake build system as provided with any Qt installation. This allows building the library using a non-microsoft compiler. For that you will need to open a command line, make sure the qt environment variables and your compiler environment variables are set and change into the libStructural directory, running ‘qmake’ and after that the make command of your compiler.

Compiling with Visual Studio

The Visual Studio Project file contains a number of sub projects and project configuration. Usually two build runs are necessary. The first one compiles needed static libraries, and the second one generates the shared libraries.

After all dependencies are set up, the next step is to start the Visual Studio Command Line Prompt (found in the Visual Studio Program Group under Visual Studio Tools), or by opening a command prompt and starting:

```
"c:\Program Files\Microsoft Visual Studio 9.0\VC\bin\vcvars32.bat"
```

This sets up the compiler environment variables. Now the project can be compiled by changing into the libStructural-src directory and typing:

```
devenv libstructural-vs2008.sln /build release
```

which generated the C# bindings and static libraries. Followed by:

```
devenv libstructural-vs2008.sln /build "release dll"
```

which completes the compilation process, with all needed shared libraries in the `./bin` folder.

Testing the compiled libraries

After building the libStructural library you can use the TestProgram in order to validate structural matrices calculated for SBML files.

- On Linux you would first set the LD_LIBRARY_PATH to include the directory where LibStructural can be found.
- On OS X you would set the DYLD_LIBRARY_PATH variable to include the directory where LibStructural is situated.
- On Windows systems you would copy the generated LibStructural.dll into the TestProgram folder.

Now you are ready to execute the TestProgram from the command line, by first changing into the TestProgram folder and then invoking:

- TestLibStructural.exe –d <directory containing SBML files>

On Linux and OSX the test program requires the mono framework (http://mono-project.com) and the command to invoke the test program on these platforms would be "mono TestLibStructural.exe -d <directory containing SBML files>"

Building Language Bindings

We now also have experimental SWIG bindings. The Simplified Wrapper and Interface Generator (SWIG) can create language bindings for a variety of programming languages. We have tested LibStructural wrappers generated for the following languages: Java, C#, Python, Ruby and R (version 2.5.1 as SWIG support of R 2.6 and higher was broken at the time of this writing). Examples on how to use the LibStructural from any of these languages can be found in the example folder. You will find examples showing how to load and analyze SBML models as well as Stoichiometry matrices directly.

The SWIG directory contains the SWIG interface files as well as some rudimentary scripts for creating the language bindings. They will be improved in the next version. The build process will be described exemplary on the Java bindings. As the built process on Linux and OS X is in this case exactly the same we only look at the case of OS X and Windows for building the bindings.

Building Language Bindings on Windows

- Download and install a recent version of SWIGWIN as available from: https://sourceforge.net/project/showfiles.php?group_id=1645&package_id=8156 (as the time of this writing this is version 1.3.36, which we installed into \swigwin-1.3.36)
- Download and install a recent version of the Java JDK as available from: http://java.sun.com/javase/downloads/index.jsp (we used JDK 1.4.2 for maximum compatibility of the bindings, which we installed into c:\Program Files\j2sdk1.4.2_04)

Once these pre-requisites are installed, open a command line prompt and change to the LibStructural\SWIG directory. Here all steps are captured in a batch script called createJava.bat. But let us look at the steps in turn:

```
\swigwin-1.3.36\swig.exe -java -c++ -module structural -DBEGIN_C_DECLS
-DEND_C_DECLS -DLIB_EXTERN -I../LibStructural/include -
I../LibLA/include -I\swigwin-1.3.36\lib\java -I\swigwin-1.3.36\lib -
outdir java -o libstructural_wrap_j.cpp java.i
```

This will generate the C wrapper libStructural_wrap_j.cpp as well as a series of Java interface files in the .\java directory.

```
set JAVA_HOME=c:\Program Files\j2sdk1.4.2_04
set OUT_DIR=c:\JavaWrapper
```

Here we set Environment variables. Specifically the JDK to be used, and the place to copy the generated Java Wrapper to.

The next step is to compile the C wrapper interface. For this we first set up the Visual Studio Command line variables, by calling:

```
"c:\Program Files\Microsoft Visual Studio 9.0\VC\bin\vcvars32.bat"
```

or a similar command depending on your Windows Locale / Visual Studio version. Once this is done we can compile the wrapper using:

```
devenv ..\LibStructural\libstructuralJava.vcproj /build "release dll"
```

this will create a LibStructuralj.dll in the ../bin folder. Which we then copy into the output directory using:

```
mkdir "%OUT_DIR%"
copy /y ..\bin\libstructuralj.dll "%OUT_DIR%"
copy /y ..\bin\libstructuralj.dll "%OUT_DIR%\structuralj.dll"
```

You might also want to copy libexpat.dll and libsbml3.dll as the C wrapper will depend on these files. Now all that is left to do is to is to compile the Java interface files. This is done by:

```
cd java
"%JAVA_HOME%\bin\javac.exe" *.java
"%JAVA_HOME%\bin\jar.exe" cf ..\libstructuralj.jar *.class
```

And finally we copy the jar file into the output directory

```
copy /y ..\libstructuralj.jar "%OUT_DIR%"
```

With that the Java bindings are built and can be tested by running:

```
"%JAVA_HOME%\bin\javac.exe" -classpath "%OUT_DIR%\libstructuralj.jar"
..\examples\java\Program.java
```

This compiles the example file. To run it type:

```
SET PATH=%OUT_DIR%;%PATH%
"%JAVA_HOME%\bin\java.exe" -classpath "%OUT_DIR%\
libstructuralj.jar";..\examples\java Program <sbmlfileName>
```

Building Language Bindings on OS X / Linux

Building the language bindings on a non-windows operating system requires the same steps:

- First download SWIG as available from: https://sourceforge.net/project/showfiles.php?group_id=1645&package_id=1608 (as the time of this writing: swig-1.3.36)
- Install SWIG (into /usr/local/)
  ```
  tar zxf swig-1.3.36.tar.gz
  cd swig-1.3.36
  ./configure
  make
  sudo make install
  ```
- Make sure you have a JDK installed on your System (on OSX a JDK is provided with the Operating System), on linux you might want to install openJDK with your favorite package manager (.i.e sudo aptitude install openjdk)
- Download and install Scons http://scons.org

Next change into the LibStructural/SWIG directory. Again you will find a script ./createWrapper.sh which performs the necessary steps:

```
/usr/local/bin/swig -java -c++ -module structural -DBEGIN_C_DECLS -
DEND_C_DECLS -DLIB_EXTERN -I../LibStructural/include -
I../LibLA/include -I/usr/local/share/swig/1.3.33/java -
I/usr/local/share/swig/1.3.33 -cpperraswarn -outdir java -o libs
tructural_wrap_j.cpp java.i
```

This will generate the C wrapper libStructural_wrap_j.cpp as well as a series of Java interface files in the .\java directory. To build the c interface file and jar file run:

```
scons JAVA=
/System/Library/Frameworks/JavaVM.framework/Versions/1.5.0/Headers/
```

on OS X. On Linux substitute the directory with the include directory of your JDK. This will compile the c interface file and generate the jar file in ./bin. On OSX to install the library, copy the resulting ./bin/libstructuralj.dylib into ~/Library/Java/Extensions (on OSX prior to 10.5 you might also want to rename the dylib file to jnilib).

After that you can test the library by running:

```
javac –classpath bin/libstructuralj.jar:../examples/java
../examples/java/Program.java
java –classpath bin/libstructuralj.jar:../examples/java Program
<sbmlFile>
```

Language bindings for other languages are created similarly. The createWrapper script as well as the SConstruct file also contain instructions for building C#, ruby, python and r bindings as well.

Further information

Help files containing information on the C/C++/C# API and coding examples can be found either online (see http://sys-bio.org/fbergman/libstructural/) or more conveniently in the doc subdirectory.

For more up to date information see http://sys-bio.org/fbergman/libstructural/ or contact sbwteam@gmail.com