Integrating a package with the libSBML source tree

This document describes the mechanism that integrates a new libSBML package with the libSBML 5 CMake build system. This integration will allow to:

- Build the package for testing purposes against a prior build of libsbml5
- Integrate the package sources with the libSBML5 source tree.
- Remove the package sources from the libSBML5 source tree.

This document does not describe how to write LibSBML 5 packages. For information on this please see Akiya's documentation here:

http://sbml.svn.sourceforge.net/viewvc/sbml/branches/libsbml-5/docs/

The directory structure

Packages should use the following directory structure:

```
<package root>
    |-examples
    |---c++
    |---csharp
    |---java
    |---perl
    |---python
    |---ruby
    |-src
    |---bindings
    |-----csharp
    |-----java
    |-----perl
    |-----python
    |-----ruby
    |-----swig
    |---common
    |-----test
    |---extension
    |-----test
    |---sbml
    |-----test
    |---util
    |-----test
```

This directory structure separates examples that use the package, from the implementation (which is found in the 'src' folder). The implementation itself is separated into code necessary for the creation of language bindings (through the use of SWIG) and the actual package code (in the folders common, extension, sbml and test). The 'common' folder is the place for holding forward declarations for all package classes, the 'extension' folder holds the code for all extension points the package defines. In the 'sbml' folder all package specific SBML classes are defined. Finally the 'util' folder allows placing any miscellaneous code. Each of these folders may optionally contain 'test' folders, which may contain unit tests to test the package code.

There are several packages available that can be used for reference; they are kept in the SVN branch:

http://sbml.svn.sourceforge.net/viewvc/sbml/branches/libsbml-packages/

The files

The following files are necessary for integrating a package with the build system. In the explanation below we detail the integration based on the ‘layout’ package. In order to transfer the information to another package, simply replace ‘layout’ with the package name of your package (i.e.: ‘groups’, ‘spatial’, ‘req’ …).

Two types of files are used:

1. Package integration files: that is files that perform the actual integration, allow the removal of the package from the source tree or allow testing of the package.
2. Package build files: these are the files that when integrated with the libSBML source tree provide build options to build and test the package with libSBML.

Package integration files

There are two package integration files:

- CMakeLists.txt in the package root directory
- CMakeLists.txt in the package src directory

The first sets up the integration process and allows package developers to add a compilation and test step for the package. The second contains the copy / removal steps that enable integration of the package with the libSBML source tree, or removal of the files.

CMakeLists.txt in package root

Below is the listing for the CMakeLists.txt file of the package root directory. There is one MODE option defined. This option can take the values ‘integrate’ (to integrate with the source tree), ‘compile’ (to test the package code) and ‘remove’ (to remove the package from the source tree).

The LIBSBML_SOURCE variable is set to the location of the libSBML 5 source with which to integrate the package.

The rest of the file is taken up with the compilation of the package code for testing purposes against an installed version of libSBML 5. For this it is only necessary to locate the existing libsbml5 library as passed to LIBSBML_LIBS. Optionally not only the library is created but also an example is compiled that uses the library (WITH_EXAMPLE option).

```
#
# This CMake Package integrates the SBML Layout Extension
# with libsbml 5
#

cmake_minimum_required(VERSION 2.8)

# the project name should be the same name as the SBML package
project(layout)

set(MODE "integrate" CACHE STRING
   "The operation to perform, valid options are
    integrate|compile|remove.")
set(LIBSBML_SOURCE "$ENV{HOME}/Development/libsbml-5/" CACHE PATH
   "Path to the libsbml source distribution.")
set(EXTRA_LIBS "xml2;bz2;z" CACHE STRING
   "List of Libraries to link against." )

if(MODE STREQUAL "compile")
      # compile the package and link it against an existing
      # libsbml-5 version
      set(SOURCE_FILES
            "src/extension/LayoutExtension.cpp"
            "src/extension/LayoutModelPlugin.cpp"
            "src/extension/LayoutSpeciesReferencePlugin.cpp"
            "src/sbml/BoundingBox.cpp"
            "src/sbml/CompartmentGlyph.cpp"
            "src/sbml/CubicBezier.cpp"
            "src/sbml/Curve.cpp"
            "src/sbml/Dimensions.cpp"
            "src/sbml/GraphicalObject.cpp"
            "src/sbml/Layout.cpp"
            "src/sbml/LineSegment.cpp"
            "src/sbml/Point.cpp"
            "src/sbml/ReactionGlyph.cpp"
            "src/sbml/SpeciesGlyph.cpp"
            "src/sbml/SpeciesReferenceGlyph.cpp"
            "src/sbml/TextGlyph.cpp"
            "src/sbml/Layout.cpp"
            "src/util/LayoutAnnotation.cpp"
            "src/util/LayoutUtilities.cpp"
            )

      include_directories(${LIBSBML_SOURCE}/include)

      find_library(LIBSBML_LIBS
            NAMES libsbml.lib sbml
            PATHS ${LIBSBML_SOURCE}
                    ${LIBSBML_SOURCE/lib}
                    ${LIBSBML_SOURCE/src/.libs}
                    /usr/lib /usr/local/lib
                    ${CMAKE_SOURCE_DIR}
                    ${CMAKE_SOURCE_DIR}/dependencies/lib
                      )

      make_directory(${CMAKE_CURRENT_BINARY_DIR}/include/sbml/layout)
      # copy header files to facilitate build
      foreach(dir common extension sbml util)

                  # copy files
            file(COPY ${CMAKE_CURRENT_SOURCE_DIR}/src/${dir}/
                 DESTINATION
                 ${CMAKE_CURRENT_BINARY_DIR}/include/sbml/layout
                 PATTERN ${CMAKE_CURRENT_SOURCE_DIR}/src/${dir}/*.h)
```

```
        endforeach()

        if (NOT UNIX)
              add_definitions(-DWIN32 -DLIBSBML_EXPORTS -DLIBLAX_EXPORTS)
        endif()


        include_directories(${CMAKE_CURRENT_BINARY_DIR}/include)

        include_directories("src/common")
        include_directories("src/extension")
        include_directories("src/sbml")
        include_directories("src/util")

        add_library(layout STATIC ${SOURCE_FILES} )
        target_link_libraries(layout ${LIBSBML_LIBS})

        option(WITH_EXAMPLE "Compile Example File" ON)

        if(WITH_EXAMPLE)

              set(EXAMPLE_SOURCE examples/c++/example1-L3.cpp)
              add_executable(layout_example ${EXAMPLE_SOURCE})
              target_link_libraries(layout_example layout ${EXTRA_LIBS})

        endif()

else()
        add_subdirectory(src)
endif()
```

CMakeLists.txt in package src

The CMakeLists.txt file in the package src directory is all about integrating the package source with the libsbml5 source tree. It handles the two cases ‘integrate’ and ‘remove’. Below is a listing of the file. At the beginning of the file it includes a ‘common.cmake’ file from the libSBML source tree, which contains a series of utility functions that make it easy to copy the package files. Next two variable lists are populated; the first one PACKAGE_FILES contains a list of all the files of the package implementation, the second BINDING_FILES a list of all files that enable the use of the package in the libSBML language bindings.

Then a check is made as to the value of the MODE variable. Either the package sources and binding file are copied to the libSBML 5 source tree, where they will be picked up by the libSBML build system, or the files will be removed from the source tree.

```
#
# This CMake file integrates the binding source with the libsbml
# source tree
#

# include common functions (used for copying / removing files)
if(NOT EXISTS ${LIBSBML_SOURCE}/common.cmake)
      message(FATAL_ERROR "Invalid libsbml source directory")
endif()

include(${LIBSBML_SOURCE}/common.cmake)


# specify the package files
set(PACKAGE_FILES

            # forward declaractions
            "common/layoutfwd.h"
            "common/LayoutExtensionTypes.h"

            # extension points
            "extension/LayoutExtension.cpp"
            "extension/LayoutExtension.h"
            "extension/LayoutModelPlugin.cpp"
            "extension/LayoutModelPlugin.h"
            "extension/LayoutSpeciesReferencePlugin.cpp"
            "extension/LayoutSpeciesReferencePlugin.cpp"

            # new SBML classes
            "sbml/BoundingBox.cpp"
            "sbml/BoundingBox.h"
            "sbml/CompartmentGlyph.cpp"
            "sbml/CompartmentGlyph.h"
            "sbml/CubicBezier.cpp"
            "sbml/CubicBezier.h"
            "sbml/Curve.cpp"
            "sbml/Curve.h"
            "sbml/Dimensions.cpp"
            "sbml/Dimensions.h"
            "sbml/GraphicalObject.cpp"
            "sbml/GraphicalObject.h"
            "sbml/Layout.cpp"
            "sbml/Layout.h"
            "sbml/LineSegment.cpp"
            "sbml/LineSegment.h"
            "sbml/Point.cpp"
            "sbml/Point.h"
            "sbml/ReactionGlyph.cpp"
            "sbml/ReactionGlyph.h"
            "sbml/SpeciesGlyph.cpp"
            "sbml/SpeciesGlyph.h"
            "sbml/SpeciesReferenceGlyph.cpp"
            "sbml/SpeciesReferenceGlyph.h"
            "sbml/SpeciesReferenceRole.h"
            "sbml/TextGlyph.cpp"
            "sbml/TextGlyph.h"

            # test cases
            "sbml/test/CMakeLists.txt"
            "sbml/test/TestBoundingBox.cpp"
            "sbml/test/TestCompartmentGlyph.cpp"
            "sbml/test/TestCubicBezier.cpp"
            "sbml/test/TestCurve.cpp"
            "sbml/test/TestDimensions.cpp"
            "sbml/test/TestGraphicalObject.cpp"
            "sbml/test/TestLayout.cpp"
            "sbml/test/TestLayoutCreation.cpp"
            "sbml/test/TestLayoutFormatter.cpp"
            "sbml/test/TestLayoutWriting.cpp"
            "sbml/test/TestLineSegment.cpp"
            "sbml/test/TestPoint.cpp"
            "sbml/test/TestReactionGlyph.cpp"
            "sbml/test/TestRunner.c"
            "sbml/test/TestSBMLHandler.cpp"
            "sbml/test/TestSpeciesGlyph.cpp"
            "sbml/test/TestSpeciesReferenceGlyph.cpp"
            "sbml/test/TestTextGlyph.cpp"
            "sbml/test/utility.cpp"
            "sbml/test/utility.h"

            # utility functions
            "util/LayoutAnnotation.cpp"
            "util/LayoutAnnotation.h"
            "util/LayoutUtilities.cpp"
            "util/LayoutUtilities.h"
      )

# specify the files for the language bindings
set(BINDING_FILES

            # C# bindings
            "bindings/csharp/local-downcast-extension-layout.i"
            "bindings/csharp/local-downcast-namespaces-layout.i"
            "bindings/csharp/local-packages-layout.i"

            # java bindings
            "bindings/java/local-downcast-extension-layout.i"
            "bindings/java/local-downcast-namespaces-layout.i"
            "bindings/java/local-packages-layout.i"

            # perl bindings
            "bindings/perl/local-downcast-extension-layout.cpp"
            "bindings/perl/local-downcast-packages-layout.cpp"
            "bindings/perl/local-downcast-namespaces-layout.cpp"
            "bindings/perl/local-downcast-plugins-layout.cpp"

            # python bindings
            "bindings/python/local-downcast-extension-layout.cpp"
            "bindings/python/local-downcast-packages-layout.cpp"
            "bindings/python/local-downcast-namespaces-layout.cpp"
            "bindings/python/local-downcast-plugins-layout.cpp"
            "bindings/python/local-layout.i"

            # ruby bindings
            "bindings/ruby/local-downcast-extension-layout.cpp"
            "bindings/ruby/local-downcast-packages-layout.cpp"
            "bindings/ruby/local-downcast-namespaces-layout.cpp"
            "bindings/ruby/local-downcast-plugins-layout.cpp"
            "bindings/ruby/local-layout.i"

            # generic swig bindings
            "bindings/swig/layout-package.h"
            "bindings/swig/layout-package.i"
      )

if(MODE STREQUAL "integrate")
      # integrate the package with the specified libsbml source
directory
```

```
      # copy the CMake script that integrates the source files with
libsbml-5
      copy_file("../layout-package.cmake" "${LIBSBML_SOURCE}")
      copy_file("layout-package.cmake" ${LIBSBML_SOURCE}/src)

      # copy language binding files
      foreach(bindingFile ${BINDING_FILES})
            copy_file_to_subdir( ${bindingFile} ${LIBSBML_SOURCE}/src)
      endforeach()

      # copy package files
      foreach(packageFile ${PACKAGE_FILES})
            copy_file_to_subdir( ${packageFile}
${LIBSBML_SOURCE}/src/packages/layout)
      endforeach()

      # copy header files to include directory just in case
      foreach(dir common extension sbml)

            # copy files
            copy_files( ${CMAKE_CURRENT_SOURCE_DIR}/${dir}/
                        ${LIBSBML_SOURCE}/include/sbml/layout *.h )

      endforeach()

      add_custom_target(integrate ALL)

      message(STATUS "Finished integrating the SBML layout package with
the libsbml source tree in:")
      message(STATUS "${LIBSBML_SOURCE}")

elseif(MODE STREQUAL "remove")
      # remove all package files from the specified libsbml source
directory

      remove_file(${LIBSBML_SOURCE}/layout-package.cmake)
      remove_file(${LIBSBML_SOURCE}/src/layout-package.cmake)

      # copy language binding files
      foreach(bindingFile ${BINDING_FILES})
            remove_file_in_subdir( ${bindingFile} ${LIBSBML_SOURCE}/src)
      endforeach()

      # copy package files
      foreach(packageFile ${PACKAGE_FILES})
            remove_file_in_subdir( ${packageFile}
${LIBSBML_SOURCE}/src/packages/layout)
      endforeach()

      # delete package directory
      file(REMOVE ${LIBSBML_SOURCE}/src/packages/layout)
      file(REMOVE_RECURSE ${LIBSBML_SOURCE}/include/sbml/layout)

      add_custom_target(remove ALL)

      message(STATUS "Finished removing the SBML layout package from the
libsbml source tree in:")
      message(STATUS "${LIBSBML_SOURCE}")


endif()
```

Package Build Files

Package build files are the files that, when integrated with the libSBML build system, provide an option to build the package and test the package. The most basic of them are the –package.cmake files. There are two of them, the first located in the package root directory, looks as follows:

```
option(ENABLE_LAYOUT     "Enable SBML Layout package"    OFF )


if(ENABLE_LAYOUT)
      add_definitions( -DUSE_LAYOUT )
      list(APPEND SWIG_EXTRA_ARGS -DUSE_LAYOUT)
endif()
```

It basically provides the option to build the package (and the default value for it, in the above the layout package is chosen to be OFF by default). When chosen a flag is set, which can be checked in code later on (as is done by the language bindings).

The second –package.cmake file, located in the package src directory, is more elaborate. The full listing below shows how the integration is done. Note first that this file will have no effect if the ENABLE_LAYOUT option defined in the previous file is not ON. Then all C++ files in the subfolders are discovered and stored in a variable LAYOUT_SOURCES. Also all header files are copied into the libsbml/include folder. Next the header files are marked for installation. And finally the LAYOUT_SOURCES are added to the LIBSBML_SOURCES. This will cause them to be compiled into the libsbml library. The file ends with an inclusion of the test cases that test the layout sbml classes. As these tests have not been ported to windows yet, they can only be run on UNIX operating systems like Linux and OSX.

```
if(ENABLE_LAYOUT)

#include common macro for copying files
include(${CMAKE_SOURCE_DIR}/common.cmake)

include(${CMAKE_SOURCE_DIR}/layout-package.cmake)

#build up sources
set(LAYOUT_SOURCES)

# go through all directtories: common, extension, sbml and util
foreach(dir common extension sbml util)

      # add to include directory
      include_directories(${CMAKE_CURRENT_SOURCE_DIR}/packages/layout/${
dir})

      # file sources
      file(GLOB current
${CMAKE_CURRENT_SOURCE_DIR}/packages/layout/${dir}/*.cpp)

      # add sources
      set(LAYOUT_SOURCES ${LAYOUT_SOURCES} ${current})

      # copy files
      copy_files( ${CMAKE_CURRENT_SOURCE_DIR}/packages/layout/${dir}/
                        ${CMAKE_SOURCE_DIR}/include/sbml/layout *.h )

endforeach()

# mark header files for installation
file(GLOB layout_headers ${CMAKE_SOURCE_DIR}/include/sbml/layout/*.h)
INSTALL(FILES ${layout_headers} DESTINATION include/sbml/layout)

# create source group for IDEs
source_group(req_package FILES ${LAYOUT_SOURCES})

# add layout sources to SBML sources
SET(LIBSBML_SOURCES ${LIBSBML_SOURCES} ${LAYOUT_SOURCES})

####################################################################
#
# add test scripts
#
if(WITH_CHECK)

      add_subdirectory(packages/layout/sbml/test)

endif()

endif()
```

Compile/Integrate/Remove the package

Once the package is in the above form (or you have obtained it from svn: http://sbml.svn.sourceforge.net/viewvc/sbml/branches/libsbml-packages/layout/) it can be compiled, integrated or removed from the libsbml 5 source tree. This can be done either via the command line, or the graphical cmake-gui application.

From the command line

Be sure to have cmake installed (and in your PATH). Next, change into the package directory. Next create the build directory:

```
mkdir build
cd build
```

Integrating the package

To integrate the package set the MODE variable to “integrate” and specify the libsbml source directory. For example (this has to be written in one line!):

```
cmake
     -DMODE:STRING=integrate
     -DLIBSBML_SOURCE:STRING=/Users/fbergmann/Development/libsbml-5/
     ..
```

Note the two dots at the end. They indicate that the main CMakeLists.txt file is located one folder down. Once this command is issued the package is integrated with the specified libsbml 5 source tree.

Compiling the package

To compile the package set the MODE variable to “compile” and specify the libsbml source directory, and a libsbml 5 library you want to link against. For example (this has to be written in one line!):

```
cmake
     -DMODE:STRING=compile
     -DLIBSBML_SOURCE:STRING=/Users/fbergmann/Development/libsbml-5/
     -DLIBSBML_LIBS:FILEPATH
      =/Users/fbergmann/Development/libsbml-5/build/lib/libsbml.dylib
..
```

Note the two dots at the end. They indicate that the main CMakeLists.txt file is located one folder down. Typing make after this will compile the package.

Removing the package

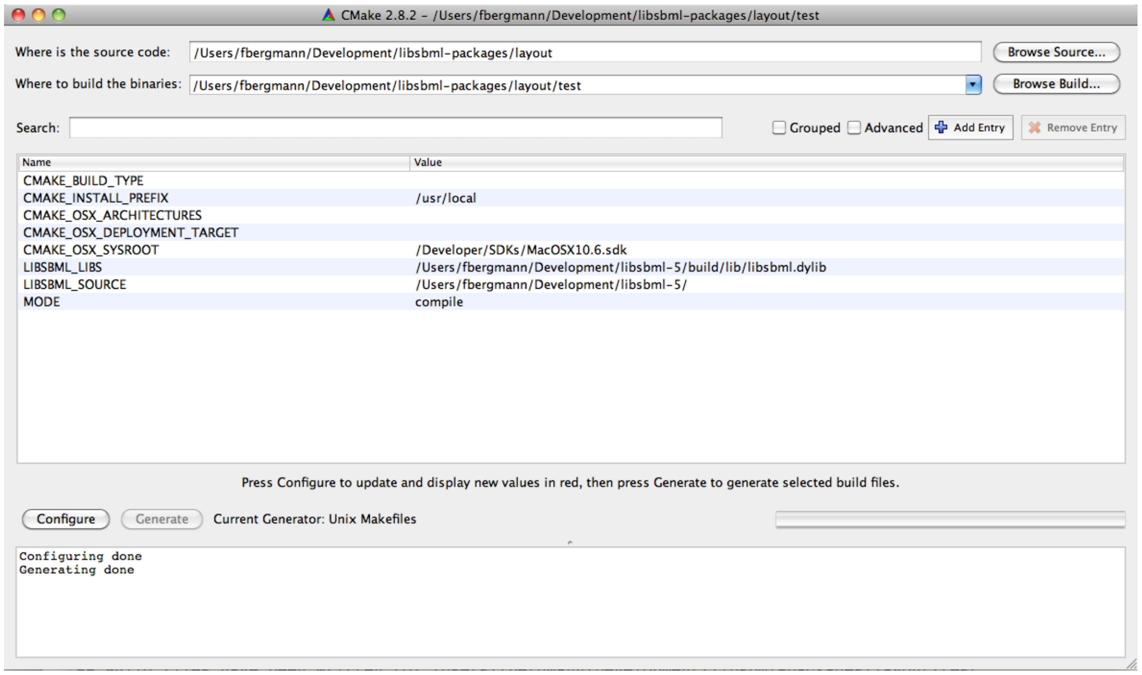
To remove the package set the MODE variable to “remove” and specify the libsbml source directory. For example (this has to be written in one line!):

```
cmake
     -DMODE:STRING=remove
     -DLIBSBML_SOURCE:STRING=/Users/fbergmann /Development/libsbml-5/
     ..
```

Note the two dots at the end. They indicate that the main CMakeLists.txt file is located one folder down. Once this command is issued the package is removed from the specified libsbml 5 source tree.

From CMake GUI

The configuration can also be made in a graphical interface. Here one simply locates the package directory. Again binaries are built in a separate folder here ‘test’. To integrate/configure/remove the package the MODE variable is set to its corresponding value and the user clicks on Configure and then Generate.



Where is the source code: /Users/fbergmann/Development/libsbml-packages/layout

Where to build the binaries: /Users/fbergmann/Development/libsbml-packages/layout/test

Name	Value
CMAKE_BUILD_TYPE	
CMAKE_INSTALL_PREFIX	/usr/local
CMAKE_OSX_ARCHITECTURES	
CMAKE_OSX_DEPLOYMENT_TARGET	
CMAKE_OSX_SYSROOT	/Developer/SDKs/MacOSX10.6.sdk
LIBSBML_LIBS	/Users/fbergmann/Development/libsbml-5/build/lib/libsbml.dylib
LIBSBML_SOURCE	/Users/fbergmann/Development/libsbml-5/
MODE	compile

Press Configure to update and display new values in red, then press Generate to generate selected build files.

Current Generator: Unix Makefiles

```
Configuring done
Generating done
```

If the package is to be compiled and debugged, it might be useful to select an IDE as generator. To change the generator: go to the File menu and delete the cache. The next time configure is hit the generator can be chosen from a menu.

References

SWIG: http://www.swig.org/

CMAKE: http://www.cmake.org/