Some lessons relevant to including external libraries in your R package

Kasper D. Hansen
Department of Biostatistics
Johns Hopkins University

kasperdanielhansen@gmail.com
@KasperDHansen
www.hansenlab.org

What

We stand on the shoulder of giants

R has excellent (bidirectional) interfaces to many languages.

It is often tempting, and sensible, to interface to an external library, to solve a problem.

But there are issues. This talk explores my experiences doing this.

Every library has a different story.

Two steps

1. Make it work on your platform of choice, for yourself.
2. Make it work on all platforms, and with your repository of choice (CRAN, Bioconductor, ?).

Step 2 can be very hard, and is easy to underestimate.

Check license.

Huge difference between CRAN and Bioconductor; I exclusively use the later.

Some lessons

Two approaches for using an external library

1. Link to an external library during R package building.
2. Include library code and build library as part of R package.

Different tradeofs between developer time and user frustration.

I favor (2), but there are issues with Windows.

Examples of (1) in the wild includes download library at package build time.

Crux of the problem

The external code is not under your control.

You may not understand it - apart from the API.

There are often portability issues; R runs on many platforms and compilers. It is hard to write portable code. Since you don't understand the core code (or control it), it may be very hard to address these issues.

Example: illuminaio

A package for parsing IDAT files, a series of binary formats for Illumina microarray data.

Initial version from 2012 (much of the code is older).

Mostly written using reverse engineering of the file format.

Early versions of IDAT files were encrypted (support due to M. Smith).

We use `des.[ch]` from FSF for decryption. No `Makevars`, works on all platforms.

Example: affxparser

An interface to Fusion SDK, reference code from a microarray manufacturer for parsing their binary file formats.

Initial version from 2005.

Fusion SDK is mostly written in C++; large complex codebase (roughly 90 cpp files plus headers).

Example: affxparser, approach

We include Fusion SDK as `src/fusion_sdk` and build the SDK as part of the R package building process.

Scrips for preparing Fusion SDK are in `inst/info` and subdirectories, including patches.

We only compile the (large) subset of Fusion SDK we need.

Getting it to work on Linux, OS X using GCC was easy. Fusion SDK had only been tested with Visual C on MS Windows (red flag). Luckily, it was not too hard to get to work on MS Windows (H. Bengstsson).

Example: affxparser, issues

Current

- (Embarrassingly) Throws exceptions which crashes R. These should be caught, but none of us know C++ exceptions.
- Non portable paths: `affxparser/src/fusion_sdk/calvin_files/writers/src/Calv`
- Calls to `std::cout`, `std:cerr`, `printf` and others.
- Hack in Makevars to decrease optimization level (assumes GCC style optimization flags). Caused by memory alignment issues related to memory mapping of files.

Historical

- Weird crashes with GCC 4.3. Perhaps compiler bug.
- `#pragma` and unclear `#ifdef` usage.

Example: Rgraphviz, introduction

Interfaces to Graphviz, a graph layout library written by a group formerly at AT&T.

From 2002 (very early Bioconductor) by the old core team (mostly Jeff Gentry).

Has always worked on UNIX systems; historically major pain on Windows (now solved)

Graphviz has a huge, old, complicated codebase (source tarball is 25 MB), mostly written in C. Conceptually, code factors into two parts: layout and rendering. Only use the former in Rgraphviz.

Example: Rgraphviz, history

- 2002: Initial version links to external Graphviz; this does not work on Windows (At the time, Graphviz is was built on Windows using Visual C).
- 2005: S. Falcon manages to hand build Graphviz using MinGW (Windows); the pre-built Graphviz is included in Rgraphviz. Still unknown how this was accomplished.
- 2007: The Graphviz team supplies pre-built binaries which works for linking on Windows. The 2005 solution is abandoned; users now have to install Graphviz themselves.

This proved to be a major headache.

- Graphviz is updated frequently, in a non-backwards compatible manner.
- Windows users had to locate the exact version used to build the binary version of Rgraphviz. But older Graphviz versions hidden on website.

Example: Rgraphviz, history, continued

- 2009: Updated Rgraphviz to support Graphviz 2.21 needed for 64 bit Windows (K. Hansen). Still had crashes on some versions of Windows.
- 2009: Using .onLoad to catch errors when loading (M. Morgan). This allows us to print useful messages regarding version mismatch between Rgraphviz and Graphviz. Installation still major headache though.
- 2012: Complete refactoring of the build process (K. Hansen). Now installs easily on Windows.

Example: Rgraphviz, 2012 refactoring

1. Needed to separate rendering from layout in codebase. I did this to remove many unnecessary dependencies from the build process; needed to modify Automake files.
2. Rtools does not support running autoconf; many tools are missing. I made a mixture of MinGW and Rtools to get missing tools.
3. Several bugs, including memory leaks, in the Graphviz code were identified; I had to debug without really understanding what was going on.
4. Finally built it on Windows, developed using a dualboot machine, then a VM.

End result: we now include `src/graphviz` and build Graphviz as part of Rgraphviz. This allows us total control. We include pre-built libraries for Windows in `src/libwin` and link to these libraries during Rgraphviz building.

My checklist

Has the library been built

- using GCC (including GCC 4.2), clang
- using the Intel compilers
- on OS X?, Solaris?
- on Windows, using MinGW

Bonus for using autoconf as a build system.

What are the dependencies?

Issues to be solved by the community

1. Very painful to develop on platforms you have little access to (for me, Windows).
   Possible solution: make VM with everything installed, available to developers.
2. Rtools does not support autoconf.
3. Develop system for delivering pre-built libraries, especially for Windows but also OS X.