How To Make an R Package Based on C++ And Manage It With R-Forge: A Tutorial

José M. Maisog
Medical Numerics, Inc.,
and Georgetown University Medical Center
joe.maisog@medicalnumerics.com
jmm97@georgetown.edu

http://howtomakeanrpackage.pbworks.com

Version 1.14

September 24, 2011

Table of Contents

List of Figures

Chapter 1

Introduction

On January 13, 2009, my thesis advisor, George Luta, forwarded me some C++ code provided by Jianfei Liu, a computer scientist/engineer based in Peking University. Dr. Liu's code performs a *point containment test*: given a closed three-dimensional polyhedron defined by a triangular mesh, and given a point in three-dimensional space, the code tests whether the point is inside, outside, or exactly on the polyhedron. Dr. Luta thought that a possible thesis project for me might be to wrap up Dr. Liu's C++ code into an **R** package, and then write a thesis examining the use of the code on some real data; the idea would be to perform *gating* on three-variable flow cytometry data. (See, e.g., Matt Wand's curvHDR package.) As it turned out, I did my thesis topic on a completely different different topic, non-negative matrix factorization. Still, with some encouragement from Dr. Luta, I continued work on building an **R** package from Dr. Liu's C++ code.

It took a while for us to figure out how to do this. Chapter 4 from Chambers' book [2] was helpful, as was Rob Hyndman's tutorial [3]. But the single most important document for my purposes was Alan Lenarcic's tutorial [5]; although it doesn't quite get through the entire process, this document got me about 95% of the way there. It took me a while to figure out the remaining 5%, but with lots of help from my classmate, Yuan Wang, I finally created the C++-based **R** package ptinpoly.

There was still one problem: on my Windows system, I was able to make builds only for Windows and Linux, but we also wanted a MacOS build. And it seemed that I needed access to a Macintosh computer in order to create a build for MacOS, but nobody I knew had a Macintosh computer that I could use. So, for a while only Linux and Windows builds of the ptinpoly package were available. (In retrospect, I now think that I could have submitted the Linux build to **CRAN**, and **CRAN** would have generated the MacOS build for me. But I didn't think of this at the time.)

Then, during my poster presentation at JSM 2009, Tom Lumley suggested that we try **R-Forge**. The way it was described to us, you upload the raw source material for your **R** package to **R-Forge**, and **R-Forge** automatically makes Linux, Windows, and MacOS builds for you. Learning how to use **R-Forge** took me a little longer than it really should have. Part of the problem was that there seemed to be *two* other pieces of software that I had to learn about: **Subversion**, which is the version control system that **R-Forge** uses, and **TortoiseSVN**, which provides a nice graphical user interface (GUI) to **Subversion**. In essence, I had an extra degree of freedom that I was needlessly worrying about. I finally figured out that I needed to "cone down" only on learning **TortoiseSVN**, and only on two basic operations: *check out* and *commit*. Once I got **TortoiseSVN** under my belt, I was then able to use **R-Forge** with ease. And once I was able to use **R-Forge**, I was able to generate builds of the ptinpoly package for all three platforms, Linux, Windows, *and* MacOS. (If you're interested, the method on which the ptinpoly package is based has been published as a paper [6].)

Since then, I have found two other tutorials on building **R** packages that have helped improve my understanding of the process: a tutorial by Schafer [7], in which he describes how to build an **R** package based on FORTRAN 95, and a tutorial by Leisch [4], in which he describes how to build an **R** package based solely on **R** scripts (i.e., not based on compiled languages like FORTRAN or C++).

This document is my attempt to put together in one place the things I've learned about building a C/C++-based **R** package, and then managing it using **R-Forge**. In the first chapter of the tutorial, I step you through the installation of required software. Then in the second chapter, I introduce you to a small C++ class that we'll use for the tutorial; in that chapter, we'll build a small test program that exercises the class without involving **R**. In the third chapter, I first step you through the process of building a *dynamically linked library* (DLL) from the C++ code, and then accessing that DLL from within **R**; then I step you through the process of building an **R** package based on the C++ code. In the final chapter, I show how to use **R-Forge** to make Linux, Windows, and MacOS builds of your **R** package, and how to transmit your **R** package from **R-Forge** to **CRAN**.

Now, this tutorial may already have been rendered obsolete by the recent development of an interesting **R** package, **Rcpp** (URL: `http://dirk.eddelbuettel.com/code/rcpp.html`), but for what it's worth, here's my take on the matter. I'm running a Windows XP system, so this tutorial is Windows-centric. Still, I think that many of the procedures should be very similar under either Linux or MacOS. If you notice any errors in this tutorial, please let me know.

Chapter 2

Software Setup

In this chapter, I discuss the software needed for this tutorial;. It is assumed that you have root privileges on your computer, so that you can install the software. If not, you'll need the system administrator to install the software for you, if you don't already have the necessary software installed. In the first section, I go over software for making **R** packages based on underlying C++ code, while in the second section I cover the two pieces of software you'll need to use **R-Forge**.

2.1 Software For Making R Packages Based on C++

2.1.1 R (required)

This may be obvious, but you'll need a copy of **R** (URL: `http://www.r-project.org`) itself to be installed. In the **bin** subfolder of the **R** installation, there is (in addition to **R.exe** itself) a utility program named **Rcmd.exe**, which we'll use to check (section 5.2 and section 5.5) and build (section 5.6) **R** packages.

2.1.2 RTools (required)

RTools (URL: `http://www.murdoch-sutherland.com/Rtools`) is a suite of utilities for R. It provides a version of the Gnu C/C++ **gcc** compiler, called **MinGW**, which we'll use in the next chapter.

Recent versions of **RTools** also include **perl**, which I believe is required for running the **rcmd** utility program that comes packaged with **R** itself. At least, if I try to run **rcmd** on a machine that has **R** installed but not **RTools** or **perl**, I get an error message about **perl** not being found.

2.1.3 MiKTeX (optional)

This is optional; it is needed to create .pdf help files. However, it is possible to build a working **R** package without .pdf help files [3]. A. Lenarcic's tutorial seemed to imply that **MiKTeX** should be installed before **RTools**, so I have listed it here first. Go to **the MiKTeX website** (URL: http://www.miktex.org/) and download the latest version of **MiKTeX**. After you've downloaded the **MiKTeX** installer, run it, perhaps by double-clicking on its icon or perhaps by selecting "Open" in your downloader.

2.1.4 Microsoft HTML Help Workshop (optional)

As with **MikTeX**, HTML Help Workshop is optional. It is needed to create .chm help files. However, it is possible to build a working **R** package without .chm help files [3]. Download it from this URL:
http://www.microsoft.com/downloads/details.aspx?FamilyID=00535334-c8a6-452f-9aa0-d597d16580ccdisplaylang=en; or do a web search for HTMLHelp.exe.

2.1.5 perl (optional)

As mentioned above (subsection 2.1.2), the latest version of **RTools** should come with **perl**. However, if you try to run **rcmd check** (section 5.2 and section 5.5) and get an error message that the command **perl** is not found, it means that you're missing **perl** and that you need to download and install it. If so, you can obtain it from here:
http://www.activestate.com/Products/activeperl

2.1.6 Set Up Your Windows Path Variable

The **Path** variable is a list of folders that are searched when a program is invoked at the DOS prompt or from within a script. This like the **Path** environment variable under Linux/UNIX, or to the **MATLABPATH** variable for **MATLAB**. If the program isn't found in any of those folders, and if it isn't a native DOS command such as **CHDIR**, **MKDIR**, **RMDIR**, and so on, then an error is generated, even though it may be a program that you have in fact installed somewhere – you still need to tell DOS where it is.

You need to set up the **Path** variable so that it contains the folders listed in Table 2.1 [5]. These are the executables that you have installed in the previous subsections. The steps to modify the **Path** variable are listed below; this is a somewhat meticulous and tedious step (A. Lenarcic called it "awkward"), so be careful. If you get any of the folder paths wrong, the rest of this tutorial will probably not work.

1. On your computer desktop, double-click on the icon labeled **My Computer**. This should pop up a window as shown in Figure 2.1.
2. Select the tab labeled **Advanced**.
3. Click on the button labeled **Environment Variables**. This should pop up a window as shown in Figure 2.2.
4. Select **System Variables** This should pop up a window as shown in Figure 2.3.
5. Click in the entry box labeled **Variable value** and select the entire contents.

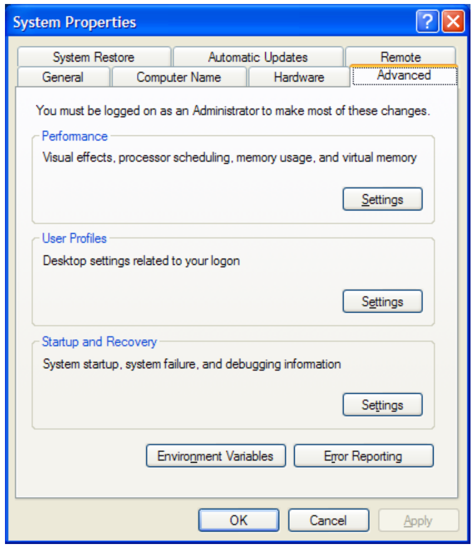

Figure 2.1: The **System Properties** dialog box.

6. Copy and paste the contents into a text editor like **WordPad** or **NotePad++** (see subsection 2.1.7 below).

7. In the text editor, add the following folders to the *beginning* of the list of folders, so that they are searched first. Delimit folders with semicolons, as per the **Path** variable convention; see Figure 2.4.

   - **R bin** – this is the folder in which your **R** executable, **R.exe**, is kept. On my system, this is the folder **C:\Program Files\R\R-2.10.1\bin**; of course, the exact path name may differ on your own system.
   - **RTools bin** – this is a subfolder of your **RTools** installation. On my system, this is the folder **C:\Rtools\bin**.
   - **MinGW bin** – this is another **bin** subfolder of your **RTools** distribution. On my system, this is the folder **C:\Rtools\MinGW\bin**
   - **perl bin** – Recent versions of **RTools** come with **perl**, kept in a separate bin folder in the **RTools** distribution, e.g **C:\Rtools\perl\bin**.
   - **MiKTeX** – If you have indeed installed it. E.g., **C:\Program Files\MiKTeX 2.7\miktex\bin**
   - **HTML Help Workshop** – If you have indeed installed it. E.g., **C:\Program Files\HTML Help Workshop**

Again as per Alan Lenarcic, we need to run a few commands at a DOS prompt to confirm that your **Path** variable has been properly set up [5]. Bring up a new DOS shell by clicking

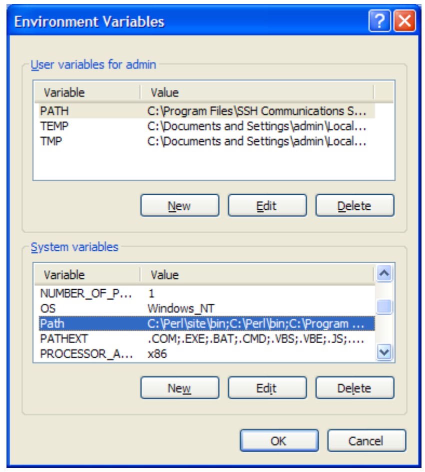

Figure 2.2: The **Environment Variables** dialog box.

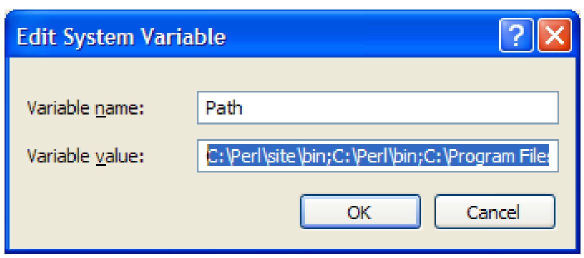

Figure 2.3: The **Edit System Variables** dialog box.

Software	folder	Test Command
R	R/bin	rcmd –help
Gnu Compiler	Rtools/bin	gcc –help
perl	perl/bin	perl –help
MiKTeX	miktex/bin	tex –help
MinGW utilities	Rtools/bin	grep –help
HTML Help Workshop	HTML Help Workshop	hhc –help

Table 2.1: Table of folders to list in the **Path** variable, and commands to confirm.

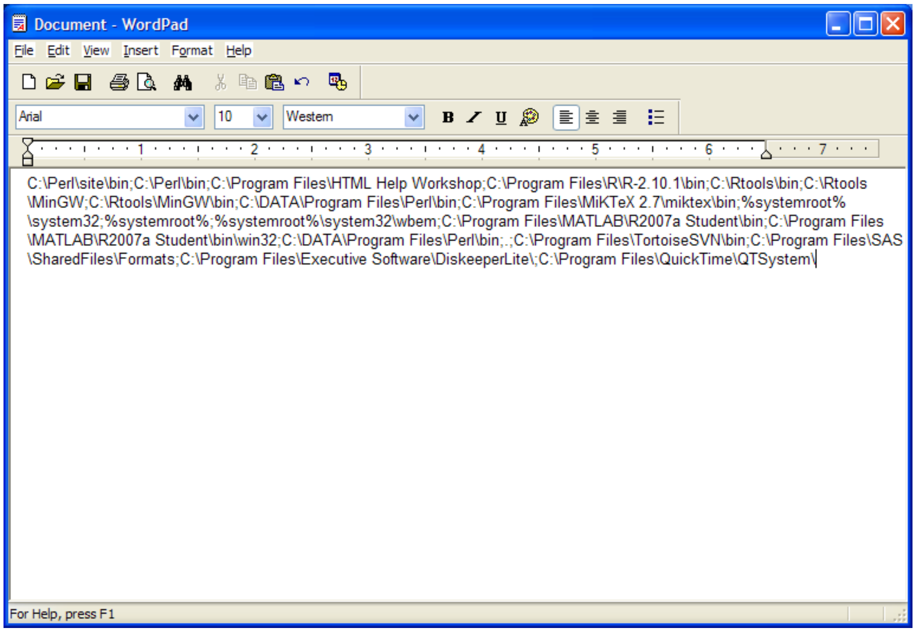

Figure 2.4: Example of editing the **Path** variable in WordPad.

on the Start button in the lower left corner of your Windows desktop, and select **Run...**; this should bring up the **Run** dialog box, as show in Figure 2.5. Type **cmd** in the entry box, and then click on the OK button. This should then pop up a window containing an old-fashioned DOS prompt; see Figure 2.6. (Alternatively, you can select Start → Programs → Accessories → Command Prompt.)

Then, type the following seven commands at the DOS prompt. The first command, 'path', simply lists prints out the value of your **Path** variable. The remaining six commands invoke various programs that are needed to create an **R** package.

```
path
rcmd --help
gcc --help
perl --help
tex --help
grep --help
hhc --help
```

The latter six commands are listed in Table 2.1. If you run any of these commands and get an error message like "command not found", it means either that you haven't set up the Windows **Path** variable properly, or that you haven't installed the software itself properly.

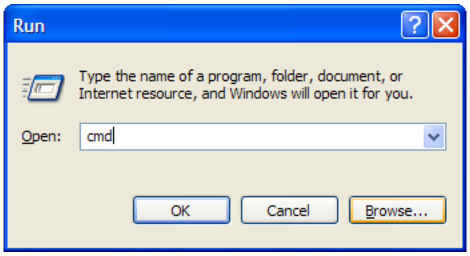

Figure 2.5: The **Run** dialog box.

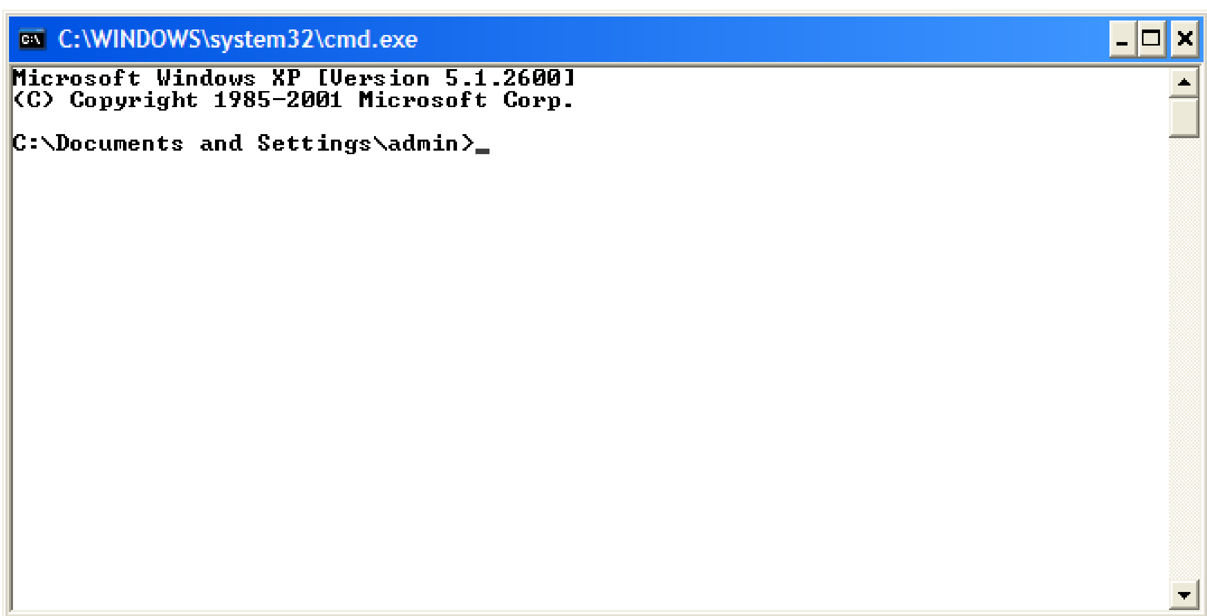

Figure 2.6: DOS prompt.

2.1.7 Dev-C++ or NotePad++ (optional)

In his tutorial, A. Lenarcic suggested obtaining a general coding text editing program for C++ code. He suggested the **Crimson Editor** (URL: http://crimsoneditor.com/english/download.html), but qualified his suggestion with the observation that this editor (at least that particular version) didn't eliminate CRLF returns to his satisfaction. I believe that the "Convert to UNIX Format" feature of **Notepad++** (URL: http://notepad-plus-plus.org/) may do the trick. Also, **Notepad++** recognizes multiple standard programming languages, including C++, and is able to show computer code in context-sensitive colors – e.g., comments in green, reserved symbols in purple, and preprocessor macros in red. For this tutorial, we'll use **gcc**, the Gnu compiler that comes with **RTools**; this will be fine for the example we'll be dealing with.

For larger-scale programming efforts, you might consider using a full-fledged integrated development environment (IDE) such as Microsoft Visual Studio. For a freely available option, consider **Dev-C++** (URL: http://www.bloodshed.net/dev). Note that **Dev-C++** uses the **MinGW** port of the Gnu compiler; I think that this means that if you install **Dev-C++**, then you'll have *two* copies of the **MinGW** Gnu compiler, one from **RTools** and one from **Dev-C++**. It may be possible to set up **Dev-C++** so that it uses the copy of the **MinGW** Gnu compiler that came with **RTools**, but I haven't looked into that. (Maybe somebody out there can confirm or refute this idea.)

You can find a tutorial that I wrote on getting started with **Dev-C++** at this URL: http://scs.gmu.edu/~wss/wss0804.shtml#student

2.2 Software for Using R-Forge

If you'd like to use **R-Forge** to store and maintain your **R** project (see chapter 6), you'll need to install **TortoiseSVN** and the **PuTTY** distribution.

2.2.1 TortoiseSVN

Subversion is an open source software for version control. **TortoiseSVN** provides a nice graphical user interface (GUI) to **Subversion** functionality. Download the **TortoiseSVN** distribution from this URL: http://tortoisesvn.tigris.org

As an aside, I thought at first that one needed to install *both* **Subversion** *and* **TortoiseSVN**, and that **TortoiseSVN** would run "on top" of **Subversion**; i.e., one would have both a **Subversion** process and a **TortoiseSVN** process running simultaneously on the computer. I mistakenly thought it would be exactly analogous to the relationship between **MiKTeX** (which provides LaTeX functionality) and **TeXnicCenter** (which runs "on top" of **MiKTeX** and provides a nice GUI). But it turns out that **TortoiseSVN** itself provides **Subversion** functionality, so you *don't* need to install **Subversion** as well.

2.2.2 PuTTY

PuTTY is a free SSH, Telnet and Rlogin client for 32-bit Windows systems. We need the **PuTTY** distribution for its public key encryption software, since **R-Forge** uses this particular encryption system for security. Download the **PuTTY** distribution from this URL: `http://www.chiark.greenend.org.uk/~sgtatham/putty`. After downloading and unpacking it, you should see a few files, including two named **puttygen.exe** and **pageant.exe**; we will be using these two programs in section 6.3 and section 6.4. For further information on public key encryption in the context of **PuTTY**, see
`http://tartarus.org/~simon/putty-snapshots/htmldoc/Chapter8.html#pubkey`

Chapter 3

Example C++ Code: A Simple Matrix Class

In the three sections of this chapter, I list the code listing for example C++ code that we'll use in this tutorial:

- **RMat.h** – header file declaring the interface to a simple C++ matrix class named **RMat**, which we'll interface with an **R** matrix
- **RMat.cpp** – C++ code defining the member functions of the **RMat** class. (This is *not* the most efficient implementation for a C++ linear algebra library, but it will suffice for this tutorial. For a discussion of efficient scientific programming in C++, see, e.g., [10] and [11]; see also Chapter 5 of R. Davies' documentation for his **NewMat** C++ linear algebra library, URL: `http://www.robertnz.net/nm11.htm#design`.)
- **testRMat.cc** – C++ code for a driver program defining a **main** function, which exercises the member functions of the **RMat** class to demonstrate that it works.

Cut and paste the source code given in section 3.1 into a plain text file named **RMat.h**. Similarly, place the source code given in section 3.2 into a file named **RMat.cc**, and the source code given in section 3.3 into a file named **testRMat.cc**. For your convenience, a ZIP file file containing this source code is downloadable from this URL:
`http:\\howtomakeanrpackage.pbworks.com\f\Code.zip`.

At a DOS prompt (see subsection 2.1.6), navigate to the folder containing the three source code files; use the DOS commands **chdir** and **dir** to change and list folders, in exactly the same way you'd use the **cd** and **ls** commands under UNIX. Then use **gcc** from **MinGW** to compile the test program:

```
gcc -o testRMat testRMat.cpp RMat.cpp -lstdc++
```

Then invoke the test program:

```
testRMat
```

If all went well, you should see this output:

```
vMat1 =
1  3  5
2  4  6

vMat1(2,2) = 4

vMat2 =
3  5  7
4  6  8

vMat3 =
1  3  5
2  4  6

vMat4 =
1  3  5
2  4  6

vMat5 =
1  2
3  4
5  6

vMat6 =
35  44
44  56

Now, vMat6 =
0  0
0  0
Press any key to continue . . .
```

3.1 RMat.h

```
#ifndef _RMAT_H
#define _RMAT_H

#define COMPILE_WITH_R 0

class RMat {
public:
    //- Constructors and destructor.
    //
    RMat();                                // Constructor #1
    RMat(int x, int y);                    // Constructor #2 for x-by-y matrix.
    RMat(double *vElements, int x, int y); // Constructor #3 for R interface
    RMat(const RMat &vRhs);                // Constructor #4 (copy constructor)
    ~RMat();                               // Destructor

    //- Overloaded operators.
    //
    RMat& operator= (const RMat &vRhs);     // Assignment #1
    RMat& operator= (double vValue);        // Assignment #2
    RMat operator* (const RMat &vRhs);      // Multiplication
    double& operator()(int x, int y) const; // Element access

    //- Some utility functions.
    //
    RMat Transpose();                 // Transpose
    int NumRows() const;              // Returns number of rows.
    int NumCols() const;              // Returns number of columns.
    void Print();                     // Print matrix to stdout
    void Print(const char *vString);  // Print matrix to stdout with string.
#if COMPILE_WITH_R
```

```
    void RPrint();                    // Print matrix to R console.
    void RPrint(const char *vString); // Print matrix to R console with string.
#endif

    //- A function for deallocating memory is made publicly available because
    //  I want to not only run test this code in a pure C++ test
    //  program outside of R, testRMat.cc, but I also want to run this code
    //  from within R.  In the latter case, I will let R handle memory
    //  management.  This means that I can't automatically deallocate memory
    //  in the destructor, which is the usual thing one might do.
    //  For symmetry, the function for memory allocation is also made publicly
    //  available, although it didn't have to be.  (Maybe it should be private!)
    //
    bool AllocateMemory(int x, int y); // Allocate memory for x-by-y matrix.
    void DeallocateMemory();           // Deallocate memory

private:
    double  *mValues;           // Matrix values
    int     mNumRows, mNumCols; // Matrix dimensions
    double  mNaN;               // Return NaN on bad input
};

#endif // _RMAT_H defined
```

3.2 RMat.cc

```
#include "RMat.h"   // For RMat class
#include <limits>   // For NaN
#include <iostream> // For cout, cerr, endl, and flush
#include <assert.h> // For assert

#if COMPILE_WITH_R
#include "R.h"      // For Rprintf
#endif

using std::cout;
using std::cerr;
using std::endl;
using std::flush;

//- Constructor #1
//
RMat::RMat() : mValues(0), mNumRows(0),
mNumCols(0), mNaN(std::numeric_limits<double>::quiet_NaN())
{
}

//- Constructor #2
//
RMat::RMat(int x, int y) : mValues(0), mNumRows(0),
mNumCols(0), mNaN(std::numeric_limits<double>::quiet_NaN())
{
    AllocateMemory(x,y);
}

//- Constructor #3
//  This constructor will be used to point an RMat object to the memory
//  location provided by R, via the '.C()' function.
//
RMat::RMat(double *vElements, int x, int y) :
mNaN(std::numeric_limits<double>::quiet_NaN())
{
    mValues  = vElements;
    mNumRows = x;
    mNumCols = y;
```

```
}

//- Constructor #4 (copy constructor)
//
RMat::RMat(const RMat &vRhs) : mValues(0), mNumRows(0),
mNumCols(0), mNaN(std::numeric_limits<double>::quiet_NaN())
{
    AllocateMemory(vRhs.NumRows(),vRhs.NumCols());
    for (int i=1; i<=mNumRows; i++)
    {
        for (int j=1; j<=mNumCols; j++)
        {
            (*this)(i,j) = vRhs(i,j);
        }
    }
}

//- Destructor
//
RMat::~RMat()
{
    DeallocateMemory();
}

//- Assignment operator #1
//
RMat&
RMat::operator= (const RMat &vRhs)
{
    DeallocateMemory();
    AllocateMemory(vRhs.NumRows(),vRhs.NumCols());
    for (int i=1; i<=mNumRows; i++)
    {
        for (int j=1; j<=mNumCols; j++)
        {
            (*this)(i,j) = vRhs(i,j);
        }
    }
    return (*this);
}

//- Assignment operator #2
//
RMat&
RMat::operator= (double vValue)
{
    for (int i=0; i<mNumRows*mNumCols; i++)
    {
        mValues[i] = vValue;
    }
    return (*this);
}

//- Multiplication operator
//
RMat
RMat::operator* (const RMat &vRhs)
{
    assert(mNumCols==vRhs.mNumRows);
    RMat vProduct(mNumRows,vRhs.NumCols());
    for (int i=1; i<=mNumRows; i++)
    {
        for (int j=1; j<=vRhs.NumCols(); j++)
        {
            for (int k=1;k<=mNumCols;k++)
            {
                vProduct(i,j)+=(*this)(i,k)*vRhs(k,j);
            }
```

```
        }
    }
    return vProduct;
}

//- Element access
//
double&
RMat::operator()(int x, int y) const
{
    //- Basic range checks.
    //
    if ( ( x < 1 ) || ( x > mNumRows )
      || ( y < 1 ) || ( y > mNumCols ) )
    {
         cerr << "Rmat::operator(): range error!" << endl;
         return (double &) mNaN;
    }

    //- Offset is ((y-1)*mNumRows)+(x-1)
    //  rather than ((x-1)*mNumCols)+(y-1)
    //  because we want to organize memory the same
    //  way as it is in an R matrix.  R is column-major.
    //  Reference:
    //  http://en.wikipedia.org/wiki/Row-major_order#Column-major_order
    //
    return mValues[((y-1)*mNumRows)+(x-1)];
}


//- Transpose
//
RMat
RMat::Transpose()
{
    RMat vTranspose(mNumCols,mNumRows);
    for (int i=1; i<=mNumRows; i++)
    {
        for (int j=1; j<=mNumCols; j++)
        {
            vTranspose(j,i) = (*this)(i,j);
        }
    }
    return vTranspose;
}

//- Print #1
//
void
RMat::Print()
{
    for (int i=1; i<=mNumRows; i++)
    {
        for (int j=1; j<=mNumCols; j++)
        {
            cout << (*this)(i,j) << "  ";
        }
        cout << endl;
    }
}

//- Print #2
//
void
RMat::Print(const char *vString)
{
    cout << vString;
    Print();
```

```
}

#if COMPILE_WITH_R
//- RPrint #1
//
void
RMat::RPrint()
{
    for (int i=1; i<=mNumRows; i++)
    {
        for (int j=1; j<=mNumCols; j++)
        {
            Rprintf("%g  ",(*this)(i,j));
        }
        Rprintf("\n");
        R_FlushConsole();
        R_ProcessEvents();
    }
}

//- RPrint #2
//
void
RMat::RPrint(const char *vString)
{
    Rprintf("%s",vString);
    RPrint();
}
#endif

//- Get number of rows
//
int
RMat::NumRows() const
{
    return mNumRows;
}

//- Get number of columns
//
int
RMat::NumCols() const
{
    return mNumCols;
}

//- Allocate memory
//  Initialize all entries to 0.
//
bool
RMat::AllocateMemory(int x, int y)
{
    DeallocateMemory();
    try {
        mValues  = new double [x*y];
    }
    catch (...) {
        cerr << "Rmat::AllocateMemory(): unable to allocate memory for "
             << x << " by " << y << " matrix!" << endl;
        return false;
    }

    mNumRows = x;
    mNumCols = y;
    for (int i=1; i<=mNumRows; i++)
    {
        for (int j=1; j<=mNumCols; j++)
        {
```

```
            (*this)(i,j) = 0.0;
        }
    }
    return true;
}

//- Deallocate memory
//
void
RMat::DeallocateMemory()
{
    if ( mValues != 0 )
    {
        delete [] mValues;
        mValues = 0;
    }
}
```

3.3 testRMat.cc

```
#include <cstdlib>
#include <iostream>

#include "RMat.h"

#if COMPILE_WITH_R
#include <Rmath.h>
#endif

using namespace std;

int main(int argc, char *argv[])
{
    //- Test element access (set value / get value).
    //
    RMat  vMat1(2,3);
    int   vCount = 0;
    for (int i=1; i<=2; i++)
    {
        for (int j=1; j<=3; j++)
        {
            vMat1(i,j) = ++vCount;
        }
    }
    vMat1.Print("vMat1 = \n");
    cout << endl << "vMat1(2,2) = " << vMat1(2,2) << endl;

    //- Test special constructor for pre-allocated memory.
    //  We'll use this when passing data from R to C++.
    //  vMat2 should be a 2 by 3 matrix containing values between 3 and 8.
    //
    double *vValues;
    vValues = new double [6];
    for (int i=0; i<6; i++)
        vValues[i] = i+3;
    RMat vMat2(vValues,2,3);
    vMat2.Print("\nvMat2 = \n");

    //- Test assignment.
    //
    RMat  vMat3;
    vMat3 = vMat1;
    vMat3.Print("\nvMat3 = \n");

    //- Test copy constructor.
```

```
    //
    RMat  vMat4(vMat3);
    vMat4.Print("\nvMat4 = \n");

    //- Test transpose.
    //
    RMat vMat5 = vMat4.Transpose();
    vMat5.Print("\nvMat5 = \n");

    //- Test matrix multiplication.
    //
    RMat  vMat6 = vMat1 * vMat5;
    vMat6.Print("\nvMat6 = \n");

    //- Test setting all matrix elements to a scalar value.
    vMat6 = 0.0;
    vMat6.Print("\nNow, vMat6 = \n");

    //- Note that we must manually deallocate memory
    //  with the RMat class, since its destructor doesn't
    //  do this.  This is in anticipation of using the RMat
    //  class with R.
    //
    vMat1.DeallocateMemory();
    vMat2.DeallocateMemory();
    vMat3.DeallocateMemory();
    vMat4.DeallocateMemory();
    vMat5.DeallocateMemory();
    vMat6.DeallocateMemory();

#if COMPILE_WITH_R
    printf("\nUsing R math library to compute R_pow(2,3) = %g\n",R_pow(2,3));
#endif

    system("PAUSE");
    return EXIT_SUCCESS;

    return 0;
}
```

Chapter 4

Make a Shared (Dynamically Linked) Library

As an intermediate step between building a stand-alone program that exercises the C++ code (chapter 3) and making an **R** package (chapter 5), I think it's very useful to *first* create a dynamically linked library and debug it, because the Edit-Build-Run debugging cycle is faster. Once it's debugged, you can then proceed to make the full-blown **R** package.

For this chapter and the next, I have drawn heavily on Alan Lenarcic's tutorial [5].

4.1 RInterface.cc

Below is the code for C/C++ functions that will interface the **RMat** class with **R**. Cut and paste this source code into a plain text file named **RInterface.cc**. For your convenience, this source code is included in the ZIP file given previously, `http:\\howtomakeanrpackage.pbworks.com\f\Code.zip`.

In the C/C++ interface code below, note the following.

- the C "wrapper"function **CWrapper** does not contain any instantiations of C++ objects. However, it *does* make a call to a function **CppMultiply** that itself instantiates and uses C++ **RMat** objects. The C++ code is thus "hidden" from **R** by the intervening C "wrapper"function, **CWrapper**.
- The C "wrapper"function **CWrapper** is enclosed in an `extern` statement, indicating that it is a C rather than C++ function.

For some reason, you need to "hide" C++ code from **R**, presenting **R** with only a C interface; my guess is that the part of **R** that interfaces with C/C++ code was compiled in C, not C++. Within the C code, however, you can then make calls to functions which themselves use C++. The C code thus functions as a sort of "wrapper" for the C++ code.

```
#include "RMat.h"
#include "R.h"     // R functions
#include "Rmath.h" // Rmath

//-----------------------------------------------------------------------------
//- This file defines TWO functions.
//
//  The first one is a C++ function named 'CppMultiply' that directly accesses
//  the C++ RMat class.
//
//  The second one is a C function named 'CWrapper', that INdirectly accesses
//  the C++ RMat class, via a call to 'CppMultiply'.

//-----------------------------------------------------------------------------
//- CppMultiply
//
//  C++ function which uses the RMat class defined in RMat.h and RMat.cpp.
//  Since it is a C++ function, it cannot be called directly from R,
//  but instead will be called through an intermediary C function,
//  CppMultiply, defined below.
//
//  'result' is a raw pointer to the result of matrix multiplication.
//  'numRows' and 'numCols' is the number of rows and columns of the first
//  matrix; 'numCols2' is the number of columns of the second matrix.
//  The number of rows of the second matrix must be the number of
//  columns of the first matrix; this will be checked at the level
//  of the R code.
//  'mat1' and 'mat2' are raw pointers to the matrix elements of the two
//   matrices.
//
void CppMultiply(double *result,
                   int *numRows, int *numCols, int *numCols2,
                   double *mat1, double *mat2)
{
    RMat  vMat1(mat1,*numRows,*numCols);
    RMat  vMat2(mat2,*numCols,*numCols2);

    //- Compute matrix product using overloaded operator,
    //  just to demonstrate that we get the same answer.
    //
    RMat  vMat3 = vMat1 * vMat2;
    vMat3.RPrint("CppMultiply(): way down in the C++ code,\nvMat3 =\n");

    //- Use Constructor #3 to point vMat4's mValues pointer to the memory
    //  location provided by R.  Note that this constructor assumes that
    //  memory allocation/deallocation will be managed up in the R level,
    //  not down here in the C++ code.
    //
    //  Then, "manually" compute the matrix product.  We couldn't use the
    //  overloaded multiplication operator as we did with vMat3 above,
    //  because the first thing that the overloaded assignment operator '='
    //  does is deallocate memory.  This means that the memory location that
    //  R provided would probably become corrupted or inaccessible.
    //  (In any case, you're not supposed to use the regular C++ 'delete'
    //  statement with memory that was allocated by R.)  This is still using
    //  C++ stuff -- we're using the overloaded '()' operator.
    //
    RMat  vMat4(result,*numRows,*numCols2);

    vMat4 = 0.0; // Initialize to zero.
    for (int i=1; i<=*numRows; i++)
    {
        for (int j=1; j<=*numCols2; j++)
        {
            for (int k=1;k<=*numCols;k++)
            {
                vMat4(i,j) += vMat1(i,k) * vMat2(k,j);
            }
```

```
        }
    }
    vMat4.RPrint("CppMultiply(): way down in the C++ code,\nvMat4 =\n");

    //- When program execution finishes the above three nested FOR loops,
    //  vMat4 will contain the result of multiplying the two matrices.
    //  When program execution exits this function vMat4 will go out of scope,
    //  but the result of multiplication will remain in memory.
    //  Note that the RMat destructor does NOT deallocate memory; if it did,
    //  result of multiplication would be lost when vMat4 goes out of scope!
    //  Not to mention the problem of deallocating memory in vMat1 and vMat2.
}

//------------------------------------------------------------------------------
//- CWrapper
//
//  C function that in turn invokes the above C++ function 'CppMultiply'.
//  R can access C code but can't access C++ code directly.
//  This C function provides a C interface to the C++ code that R can access.
//  See: http://www.parashift.com/c++-faq-lite/mixing-c-and-cpp.html
//  In this C function, you must NOT include class declarations,
//  instantiate any C++ objects, or do any oher obvious C++
//  things.
//  The EXTERN statement tells the C++ compiler that the
//  enclosed function 'CWrapper' is a C function.
//  Although apparently we can insert C++ style comments and
//  we can even declare variables in the middle of the function,
//  which I thought you can't do in regular C.
//
//  'result' is a raw pointer to the result of matrix multiplication.
//  'numRows' and 'numCols' is the number of rows and columns of the first
//  matrix; 'numCols2' is the number of columns of the second matrix.
//  The number of rows of the second matrix must be the number of
//  columns of the first matrix; this will be checked at the level
//  of the R code.
//  'mat1' and 'mat2' are raw pointers to the matrix elements of the two
//   matrices.
//
extern "C" {
    void CWrapper(double *result,
                 int *numRows, int *numCols, int *numCols2,
                 double *mat1, double *mat2)
    {
        //- Invoke second function which internally can do C++ things.
        //
        CppMultiply(result,numRows,numCols,numCols2,mat1,mat2);
    }
}
```

4.2 Run RCMD SHLIB

Edit the file **RMat.h**, and change the line

```
#define COMPILE_WITH_R 0
```

to

```
#define COMPILE_WITH_R 1
```

This enables some lines of code that use **R** functions such as **Rprint**.

Examine the newly enabled lines in **RMat.cc**. There, you'll see the following function.

```
void
RMat::RPrint()
{
    for (int i=1; i<=mNumRows; i++)
    {
        for (int j=1; j<=mNumCols; j++)
        {
            Rprintf("%g  ",(*this)(i,j));
        }
        Rprintf("\n");
        R_FlushConsole();
        R_ProcessEvents();
    }
}
```

The **RMat** member function **RPrint** calls the **R** function **Rprint**, which prints character strings to the **R** console. But the invocation of **Rprint** must be followed by calls to the two functions, **R_FlushConsole** and **R_ProcessEvents**, otherwise the character output won't be printed immediately. Apparently, character strings get stored in some buffer that needs to be manually flushed. For more on the **R** API, see Chapter 6 of [8] (URL: http://cran.r-project.org/doc/manuals/R-exts.html#The-R-API). That chapter lists many of the **R** functions that you might like to try calling from C/C++ code.

Then, to create a dynamically linked library, run the following command:

```
rcmd SHLIB RMat.cc RInterface.cc -o multmat.dll
```

This should create a new DLL file named **multmat.dll**.

4.3 Load the DLL into R

To load the **multmat** DLL into **R**, run the following command at the **R** prompt:

```
dyn.load("C:/YourFolder/multmat.dll")
```

As usual, you'll need to change the exact path to the **multmat.dll** file to reflect the actual folder structure on your own system.

4.4 Check that the DLL has been properly loaded

To confirm that the DLL has been properly loaded, run the following command at the **R** prompt:

```
is.loaded("CWrapper")
```

It should return TRUE rather than FALSE.

4.5 MatrixMult.r

The **R** function **MatrixMult** defined below doesn't do much beyond calling the C wrapper function **CWrapper** (which in turn calls the underlying C++ code). Cut and paste this code into a plain text file named **MatrixMult.r** For your convenience, the code is downloadable from this URL: http:\\howtomakeanrpackage.pbworks.com\f\MatrixMult.r.

There are two things to note about **MatrixMult**:

1. First, note that this **R** function makes a call to the **R** function **.C**, which accepts the name of the C wrapper function, **CWrapper**, as one of its input arguments. It will be able to access the C wrapper function if the DLL has been properly loaded, which we checked in the previous subsection.

2. Also note that this **R** function makes some basic checks on its input arguments such as making sure that the input is numeric and of the proper dimensions. Sometimes such checks can be placed down at the level of the C wrapper function, or perhaps even at the C++ level. Schafer offers some suggestions for where to place checks [7].

This **R** function might itself be considered a "wrapper function". Therefore, it had been tempting to name it **RWrapper**, but I thought that it would be better to give it a more descriptive name, **MatrixMult**, since this is the function that the user will interact with in practice.

```
MatrixMult <-
function(A,B) {
    # Make sure A and B are numeric.
    if ( ( ! is.numeric(A) ) || ( ! is.numeric(B) ) ) {
        print("MatrixMult(): input must be numeric!")
        return(NaN);
    }

    # Make sure A and B are matrices.
    if ( ( ! is.matrix(A) ) || ( ! is.matrix(B) ) ) {
        print("MatrixMult(): input must be matrices!")
        return(NaN);
    }

    # Check inner dimensions of matrices.
    ncolA = ncol(A)
    nrowB = nrow(B)
    if ( ncolA != nrowB ) {
        print("MatrixMult(): improperly dimensioned matrices!")
        return(NaN);
    }

    # Create a matrix of zeros which will hold the result
    # of multiplication.  This step allocates memory that
    # the C++ code will use.
    nrowA      = nrow(A)
    ncolB      = ncol(B)
    multResult = matrix(rep(0,nrowA*ncolB),nrow=nrowA)

    # Invoke C function 'CWrapper'.
    # The first argument of .C() is the name of the C wrapper function.
    # The rest of the arguments provide pointers to six R variables, in order:
    # (1) The matrix 'multResult', which will contain the result of multiplication
```

```
    # (2) The number of rows in the first matrix.
    # (3) The number of columns in the first matrix.
    # (4) The number of columns in the second matrix.
    # (5) The actual values of the first matrix.
    # (6) The actual values of the second matrix.
    output =.C("CWrapper",
        product = as.double(multResult),
        nRows   = as.integer(nrow(A)),
        nCols   = as.integer(ncolA),
        nCols2  = as.integer(ncol(B)),
        matrix1 = as.double(A),
        matrix2 = as.double(B)
    )

    # Reshape 'product' back into an nrowA by ncolB matrix.
    # Then return the matrix.
    multResult = matrix(output$product,nrow=nrowA)
    return(multResult)
}
```

4.6 Test the R Wrapper Function

In an **R** session, source the file **MatrixMult.r**, so that the **R** wrapper function **MatrixMult** is defined.

```
source("C:/Your/Folder/MatrixMult.r")
```

Then define two matrices A and B, and multiply them in **R**.

```
A = matrix(1:6,nrow=2)
B = t(A)
Product1 = A %*% B
print(Product1)
```

The **print** function should produce the following output.

```
> print(Product1)
     [,1] [,2]
[1,]   35   44
[2,]   44   56
```

Then, use the **MatrixMult** function to perform the matrix multiplication.

```
Product2 = MatrixMult(A,B)
print(Product2)
```

You should see the following in the **R** console:

```
> Product2 = MatrixMult(A,B)
CppMultiply(): way down in the C++ code,
vMat3 =
35  44
44  56
CppMultiply(): way down in the C++ code,
vMat4 =
35  44
44  56
> print(Product2)
     [,1] [,2]
[1,]   35   44
[2,]   44   56
>
```

This confirms that the **R** function **MatrixMult** (which invokes the C wrapper function **CWrapper**, which in turn invokes the C++ function **CppMultiply**, which computes the matrix product) gets the same answer that **R** does.

4.7 Unload the DLL

When debugging your C++ code and testing it in **R**, you may need to fix your C++ code and then re-build the DLL. But you won't be able to re-build the DLL if you currently have it loaded in an **R** session. You must first unload the DLL from **R** before attempting to rebuild it.

To unload the DLL, run the **dyn.unload** command at the **R** prompt:

```
dyn.unload("C:/YourFolder/multmat.dll")
```

Now that you have unloaded the DLL, you can proceed to re-build it using **rcmd SHLIB** as in section 4.2, and then re-load it into **R** as in section 4.3.

Chapter 5

Make An R Package

Once you've got the DLL and **R** wrapper function working properly (chapter 4), you can then proceed to make an **R** package.

5.1 Create the R Package Folder Structure

Get into an **R** session if you aren't already in one. In this **R** session, define matrices A and B as well as the **R** function **MatrixMult**, as described previously in section 4.6. Then run the following **R** commands:

```
rObjects = c('A','B','MatrixMult')

package.skeleton(name = "multmat",
    path = "C:/Your/Example/Folder",
    list = rObjects
)
```

This should create a folder named **multmat**, which in turn contains three subfolders named **R**, **data**, and **man**, as well as two plain text files named **DESCRIPTION** and **Read-and-delete-me**. This new **multmat** folder will be placed in the folder specified by the **path** argument in the call to **package.skeleton**; you should modify this to point to some appropriate folder on your own system.

The subfolder **R** should contain an **R** script named **MatrixMult.r**, which contains the definition of the function **MatrixMult**. The subfolder **data** should contain two **R** data files, **A.rda** and **B.rda**, which contain the two example matrices. Finally, the subfolder **man** should contain four **R** documentation files, **multmat-package.Rd**, **multmat-package.Rd**, **A.Rd**, and **B.Rd**; these files will be further discussed below in section 5.5. You can open these files in a plain text editor such as **WordPad** or **NotePad++**; if you do so, you will see that the content of these files looks very similar to LaTeX.

The **path** of the **package.skeleton** command indicates where you'd like to create the **R** package folder skeleton. At the end of his tutorial, A. Lenarcic notes that your choice of where to place this folder can cause problems. I have noticed this myself intermittently, and am not sure where the problem arises. I considered the possibilities that the problem arises because the folder path is too long (e.g., perhaps not enough memory was allocated for the folder path), or perhaps the path name should not contain spaces or other "funny" characters, but these don't seem to be the problem. If the **rcmd check** and **rcmd build** commands give you error messages about an "unspecified package", consider moving your project high up in the folder tree, perhaps to **C:\tmp**, such that the folder path is very short and doesn't contain any "funny" characters.

For details on the structure of R packages, see Section 1.1 of [8]
(URL: `http://cran.r-project.org/doc/manuals/R-exts.html#Package-structure`).

5.2 Run RCMD CHECK

At this point, it might be instructive to try running the **rcmd check** command. To do this, perform the following steps:

1. Bring up a DOS prompt as before (see subsection 2.1.6). Click on the **Start** button in the lower left corner of your desktop, and select Start → Programs → Accessories → Command Prompt. Alternatively, you can select Start → Run..., and in the dialog box that pops up type **cmd**, and then click on the OK button. Either way, a DOS prompt running in a window should pop up (Figure 2.6).

2. At this DOS prompt, navigate to the parent folder containing the **multmat** folder you created in section 5.1. (Again, use the DOS commands **chdir** and **dir** to change and list folders, exactly the same way you'd use the **cd** and **ls** commands under UNIX.)

3. At the DOS prompt, invoke the **dir** command. You should see the **multmat** folder listed.

4. At the DOS prompt, type the following command:

```
rcmd check multmat
```

The last step will perform a check on the **multmat** folder, to see whether it is ready to be built into an **R** package. At this point of the tutorial, the **multmat** folder is *not* ready to be built into an **R** package, so you should see an error message like informing you that it cannot be installed. The very last line should indicate the folder path to a plain text log file with a name like **00install.out** that contains details regarding the check failure. Open this file in a plain text file and examine the contents; it should contain an error message indicating that no packages were specified. We will remedy this shortly, in section 5.5.

5.3 The DESCRIPTION and NAMESPACE files

Examine the contents of the **DESCRIPTION** file, using a plain text editor. You'll note that it contains basic information on the package, such as the name, a very short title, and the author. Leave this file unchanged for now. (Although we won't get into it here, note that the **DESCRIPTION** file has a line for the type of license that you want to make your package available under. See http://www.r-project.org/Licenses/ and Section 1.1.1 in [8] for more information about licenses.)

Create a plain text file named **NAMESPACE** in the **multmat** folder, i.e., in the same folder as the **DESCRIPTION** file. Make the contents of the **NAMESPACE** file the following two lines:

```
useDynLib(multmat)
export(MatrixMult)
```

This tells **R** that we need to load a shared library named **multmat.dll**, as well as the **R** function **MatrixMult**, defined in the file **MatrixMult.r**.

5.4 Add src folder for C++ Code

Create a subfolder named **src** in the **multmat** folder. The **multmat** folder should now have *four* subfolders: **man**, **data**, **R** and **src**. In the new **src** subfolder, place the files **RMat.h**, **RMat.cc**, and **RInterface.cc**. Be sure that you have edited the file **RMat.h** so that the line

```
#define COMPILE_WITH_R 0
```

now reads

```
#define COMPILE_WITH_R 1
```

as described in section 4.2.

(You don't need to include the file **testRMat.cc**, since this was only a stand-alone program that tested the C++ class **RMat**. But I have found that it seems to be okay if you do include this file.)

5.5 Fix the Documentation Files

Examine the file **00install.out**, created by the **rcmd check** command in section 5.2. You will see an error message stating:

```
Warning: ./man/multmat-package.Rd:34: All text must be in a section
Warning: ./man/multmat-package.Rd:35: All text must be in a section
```

This indicates that the **section** definition in the file **multmat-package.Rd** is empty, and that you should look at lines 34 and 35. Edit the file **multmat-package.Rd** with a text editor; you'll note that the formatting in this file is similar to LaTeX. Note that lines 34 and 35 are:

```
~~ Optionally other standard keywords, one per line, from file KEYWORDS in ~~
~~ the R documentation directory ~~
```

Remove these lines (or insert percent % signs at the beginning of each line, to comment them out), and re-run the **rcmd check multmat** command. It will still generate an error message. Examine the file **00install.out** again. This time, you will see an error message stating:

```
Error in Rd_info(db[[i]]) : Rd files must have a non-empty \title.
```

This indicates that at least one of the **.Rd** documentation files has an empty **title** section. Edit the file **MatrixMult.Rd**. You'll see that line 5 contains this text:

```
%%  ~~function to do ... ~~
```

Recall that in LaTeX, a line beginning with a percent sign % is a comment. So, the **title** section (lines 4-6 in the file **MatrixMult.Rd**) is considered empty. Change the **title** section in the **MatrixMult.Rd** so that it now look like this:

```
\title{
Example function to multiply matrices
}
```

The **title** section in the files **A.Rd** and **B.Rd** are also empty. Edit both of these files so that their **title** sections look like this:

```
\title{
Sample data matrix.
}
```

Re-run the **rcmd check multmat** command. This time, you'll see this error message:

```
Error: unexpected symbol in ''`~~simple examples''
```

So there's a problem in the **examples** section of one of the **.Rd** files. Edit the file **multmat-package.Rd** so that the **examples** section looks like this:

```
\examples{
data(A)
data(B)
C = MatrixMult(A,B)
}
```

Re-run the **rcmd check multmat** command. This time, it should go to completion! Examine the file **00install.out**, just to see what it looks like when **rcmd check** goes to completion without error. Admittedly, a few warnings are generated, and ideally you'd modify the package so that no warnings were generated. But for the purposes of this tutorial, we'll ignore the warnings and proceed.

A ZIP file containing a copy of the **multmat** folder that passes the **rcmd check** command can be downloaded from this URL:
http:\\howtomakeanrpackage.pbwiki.com\f\multmat.zip

5.6 Run RCMD BUILD

To build a gzipped tarfile package (for Linux):

```
rcmd build multmat
```

This should generate a file named **multmat.tar.gz**. You can use this file to install the **multmat R** package on a Linux machine.

To build a ZIP file package (for Windows), run this command instead:

```
rcmd build -binary multmat
```

This should generate a file named **multmat.zip**. You can use this file to install the **multmat R** package on a Windows machine.

The ZIP files that these commands generated can be downloaded from the following URLs:
http:\\howtomakeanrpackage.pbwiki.com\f\multmat_1.0.zip
http:\\howtomakeanrpackage.pbwiki.com\f\multmat_1.0.tar.gz

5.7 Install/Uninstall Your R Package

To install the **multmat** package, run this command at an **R** prompt:

```
install.packages(repos = NULL, pkgs = ''C:/My Documents/MultMat.zip'')
```

Of course, you'll need to modify the value of the **pkgs** argument to point to wherever the ZIP file for the **multmat** package is kept. Alternatively, you can select Packages → Install package(s) from local zip files... from the **R** menu.

Load the **multmat** library into **R** with the usual command:

```
library(multmat)
```

Check the documentation for the project by trying these four commands at the **R** prompt:

```
?multmat
?MatrixMult
?A
?B
```

The documentation is extremely minimal, because we made the minimum number of changes necessary in the **.Rd** files to get the **R** package to build. When you get around to making a "real" package, you'll need to flesh out the documentation much better than we have here. For details on the various sections in the **.Rd** files, see chapter 2 of [8] (URL: http:\\cran.r-project.org\doc\manuals\R-exts.html\#Writing-R-documentation-files). As mentioned earlier, the documentation markup language is very similar to LaTeX.

Try running the example from the documentation page for the **multmat** package by typing the following commands at the **R** prompt:

```
data(A)
data(B)
C = MatrixMult(A,B)
```

You should see the following output:

```
CppMultiply(): way down in the C++ code,
vMat3 =
35  44
44  56
CppMultiply(): way down in the C++ code,
vMat4 =
35  44
44  56
```

Also, the matrix C should now be the product of matrices A and B:

```
> C
     [,1] [,2]
[1,]   35   44
[2,]   44   56
```

You can confirm that we got the right answer by using **R** itself to compute the matrix product:

```
> A %*% B
     [,1] [,2]
[1,]   35   44
[2,]   44   56
```

During the course of developing an **R** package, you might find it necessary to uninstall an old version of the package with a new improved version. To uninstall the **multmat** **R** package, run this command at the **R** prompt:

```
remove.packages("multmat")
```

Unfortunately, I have found it necessary to completely exit **R** before attempting to re-install a library. Perhaps this isn't necessary, but I haven't figured out a way around it. If you know how to do this, please let me know.

Chapter 6

R-Forge: Cross-Platform Builds and Submitting to CRAN

R-Forge (http://r-forge.project.org; [9] and [1]) provides three pieces of functionality which we'll use in this tutorial:

- **R-Forge** provides a central version-controlled repository for **R** packages (section 6.4, section 6.5, and section 6.6). This can be important for large-scale projects which involve more than one developer. Also, if you discover that you have inadvertently introduced a bug while you're developing a package, you can revert your package to a previous version.
- **R-Forge** performs nightly builds for **R** packages on three platforms: Linux, Windows, and MacOS (see section 6.6). This is convenient, because you might not have access to all three types of machines; in my case, I didn't have access to a Macintosh computer, and so I relied on **R-Forge** to build the MacOS version of the ptinpoly package.
- Finally, **R-Forge** provides a facility for you to submit your package from directly **R-Forge** to **CRAN** (section 6.7).

At this point in the tutorial, we have built a example **R** package named **multmat** based on underlying C++ code. Don't submit this example package to **R-Forge**!! I don't think that the **R-Forge** team wants you to submit "test" **R** packages that don't really do anything. I think that if you submit a "test" **R** package to **R-Forge**, the project won't be approved, and you won't get past section 6.2. Submit *only* "real" **R** packages to **R-Forge**.

(Note that there is a similar website named **RForge**, located at `http://www.rforge.net`. This tutorial doesn't use that website. In retrospect, I am wondering whether Tom Lumley had been referring to this **RForge** rather than the **R-Forge** that I ultimately used!)

The following sections step you through the basic use of **R-Forge**.

6.1 Create Your R-Forge Account

Go to **R-Forge**, `http://r-forge.r-project.org`. You should see the **R-Forge**, as shown in Figure 6.1. Create an **R-Forge** account, and then log in. After you log in, the web page should look something like Figure 6.2.

6.2 Register Your Project

After logging in to **R-Forge**, click on the link **Register Project**. The resulting web page should look similar to Figure 6.3. Fill in the information to register your project. (Again, don't submit the example **multmat** package to **R-Forge**!! Submit *only* "real" **R** packages to **R-Forge**.) After your project is successfully registered, a link for the project will appear under the **Projects** tab; see Figure 6.4. Clicking on that project link will bring you to a page for the project; see Figure 6.5.

6.3 Configure Your Security

Configure your security with the following steps.

1. Invoke **puttygen.exe** by double-clicking on its icon; this should pop up the PuTTY key generator, as shown in Figure 6.6.

2. Leave **SSH-2 RSA** selected, and click on **Generate**. See Figure 6.7 and Figure 6.8. The newly generated public key is the selected text highlighted in blue in Figure 6.8.

3. Go to the **Account Maintenance** page in **R-Forge**; see Figure 6.9.

4. At the bottom of the **Account Maintenance** page, you should see a link labeled **Edit Keys**. Click on this link.

5. You should now see a web page that looks like Figure 6.10. Note the large entry box towards the bottom, labeled **Authorized keys**. Copy and paste your *public* key (Figure 6.8) into the entry field.

6. Then wait for at least an hour.

For more information on using **puttygen**, see this page. This corresponds to copying and pasting your public key into **R-Forge**.

6.4 Check Out Files from R-Forge

To check out files from the **R-Forge** repository, perform the following steps.

1. Invoke **pageant.exe** by double-clicking on its icon. **pageant.exe** will appear as a new icon in your system tray, usually found in the bottom right corner of your computer desk top; see Figure 6.11.

Figure 6.1: **R-Forge** home page.

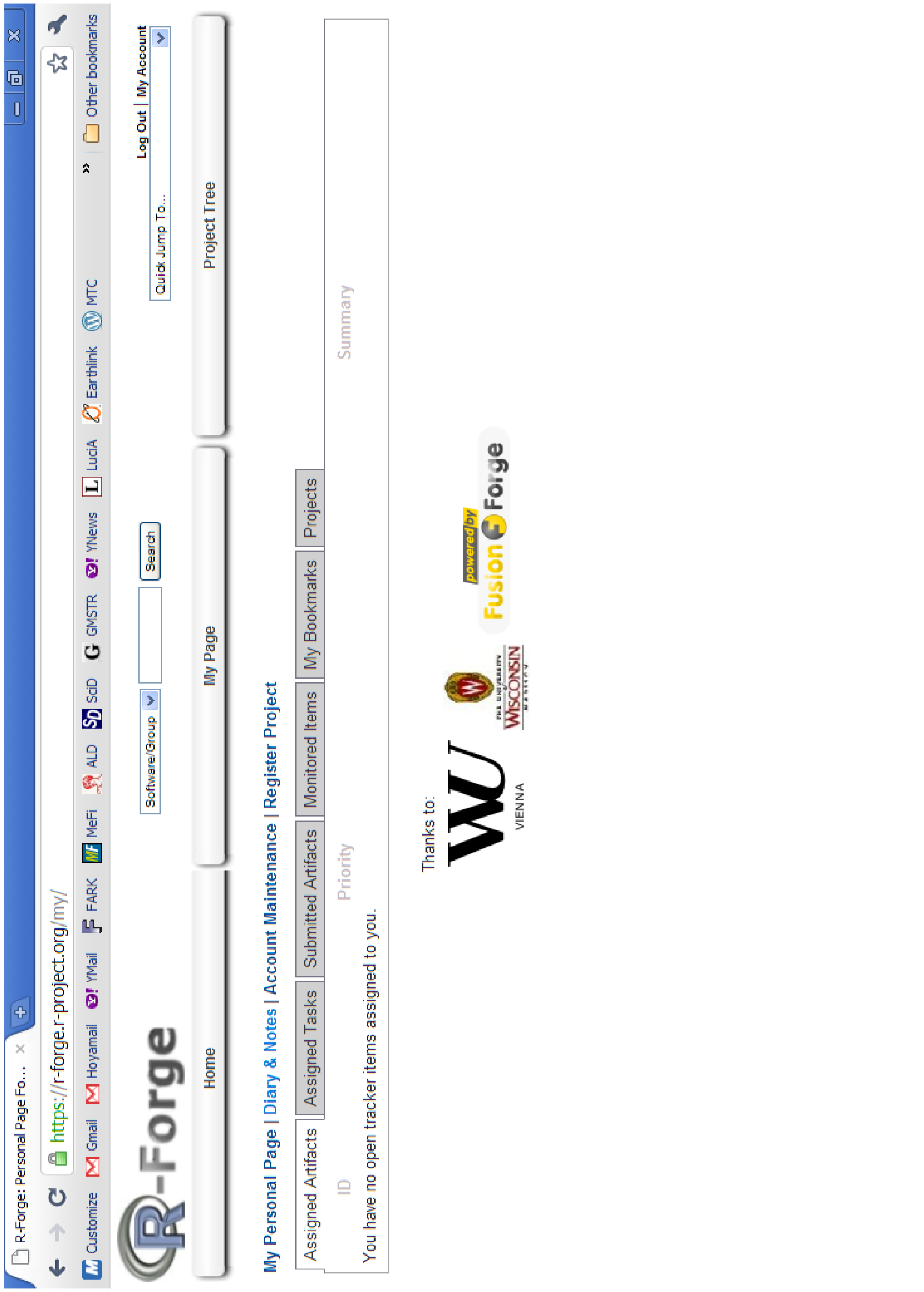

Figure 6.2: **R-Forge**, just after logging in. Note links **Account Maintenance** and **Register Project**, button **My Page**, and tab **Projects**.

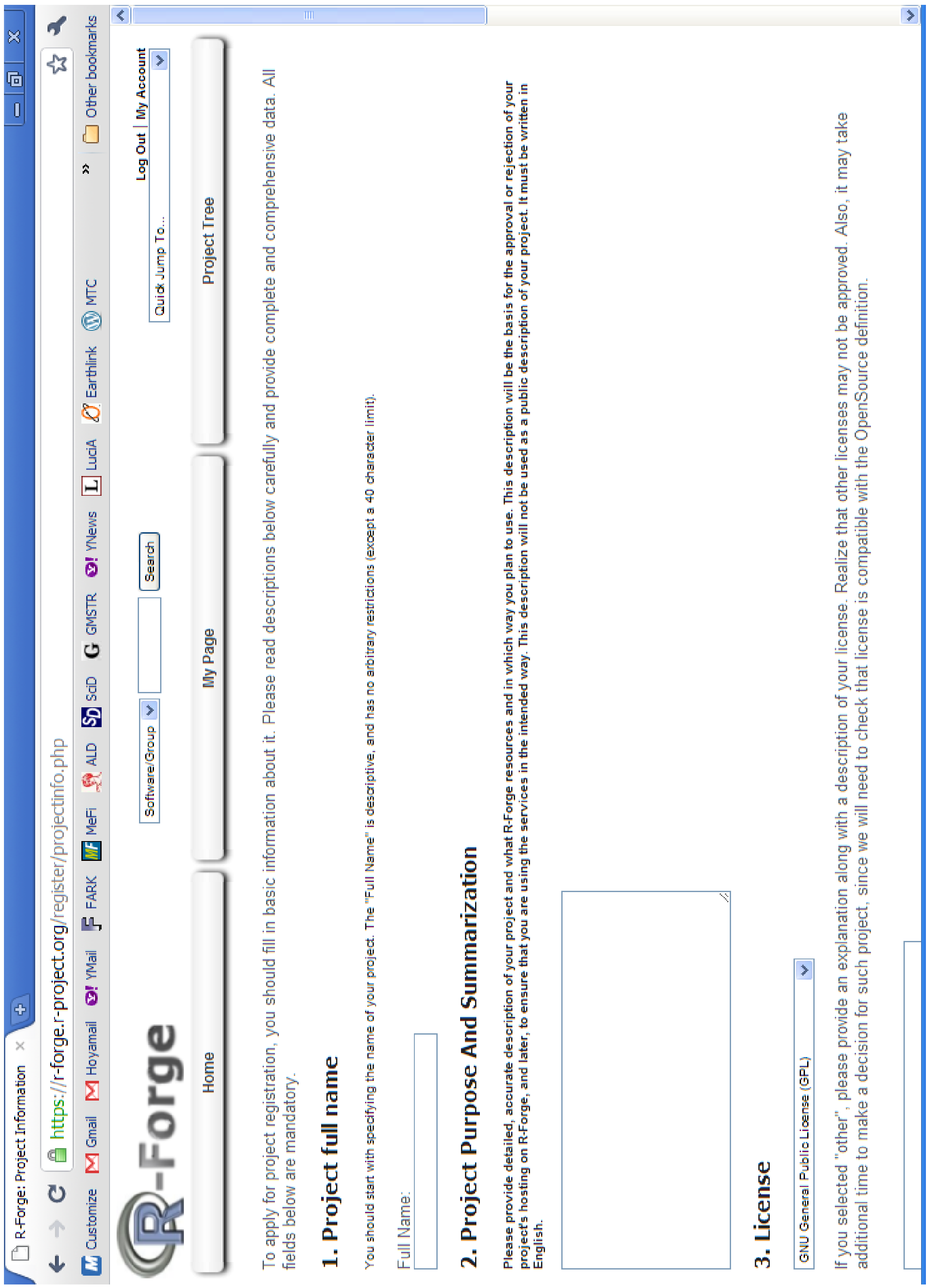

Figure 6.3: **R-Forge** Register Project Page.

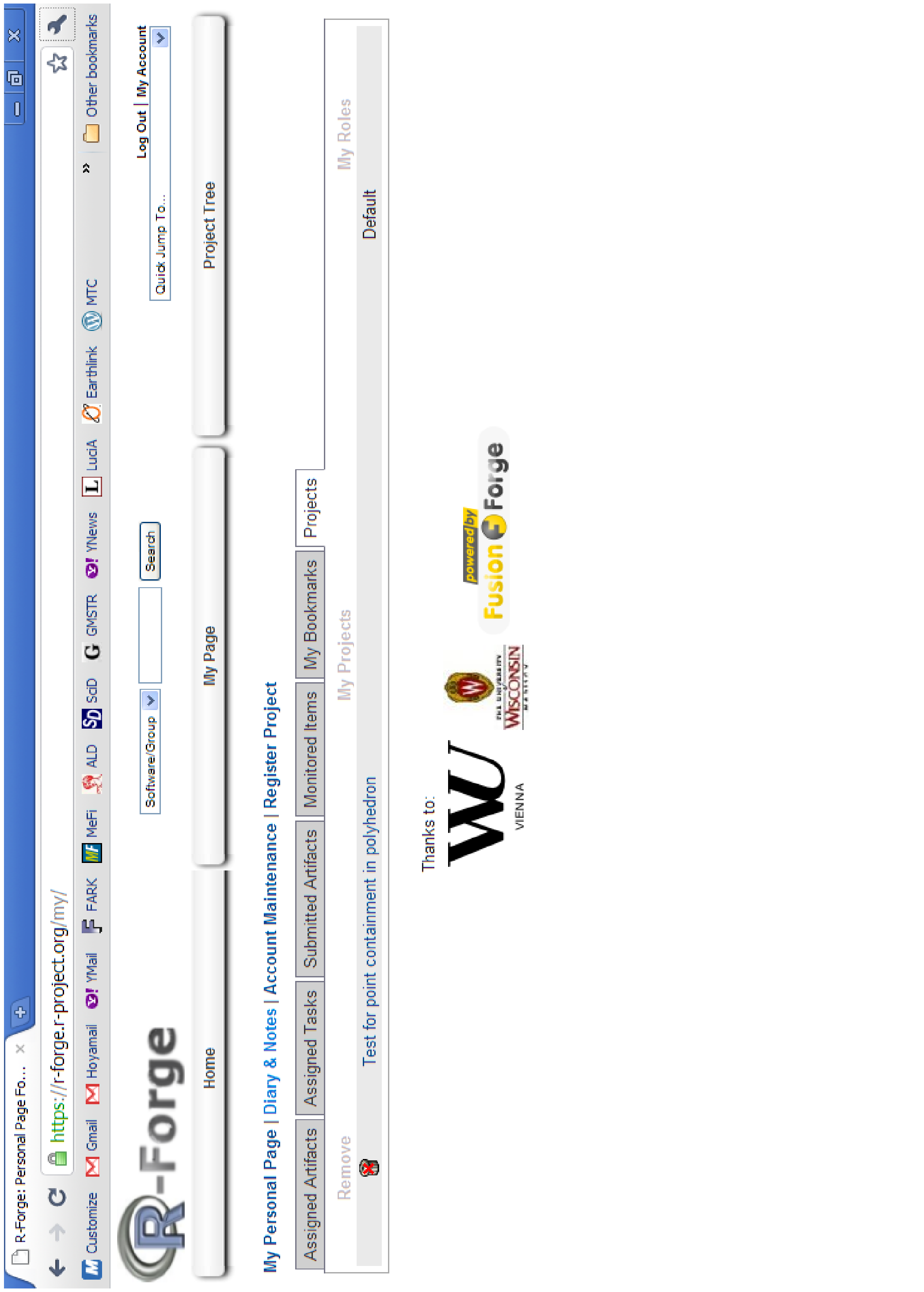

Figure 6.4: **R-Forge**, My Projects. After your project is successfully registered, a link for the project will appear under the **Projects** tab.

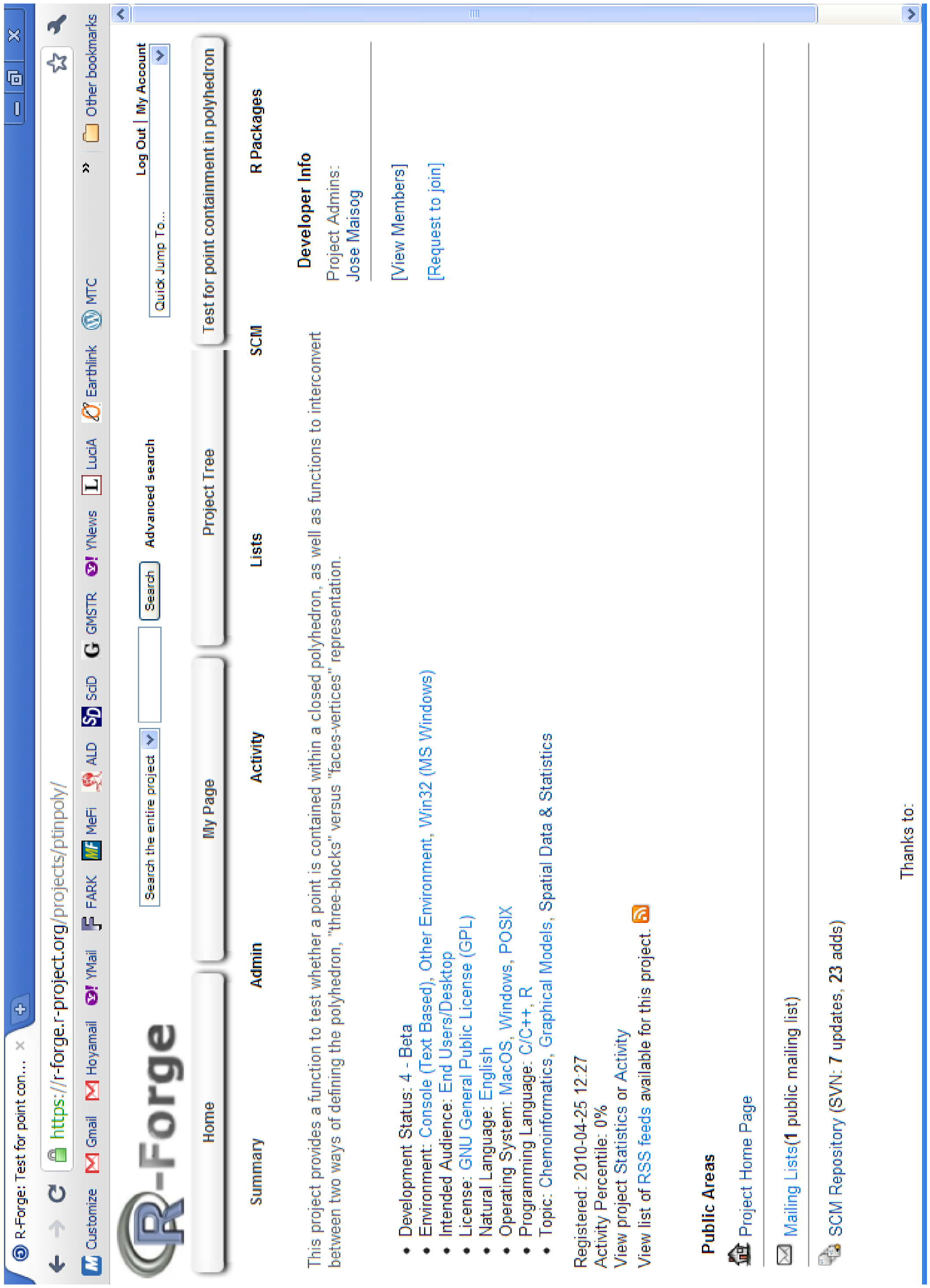

Figure 6.5: **R-Forge** PtInPoly Project Page. Note **SCM Repository** link in bottom left corner.

Figure 6.6: **PuTTY** Key Generator Start Up.

2. Right-mouse-button-click on this icon to pop up its context menu, and select **Add Key**. This pops up a browser window; see Figure 6.12.

3. Browse to the folder where you saved the .ppk containing your private key from **puttygen.exe**, select the .ppk file, and then click on the **Open** button. Because the private key is now loaded into **pageant.exe**, **TortoiseSVN** will be allowed to check out your project from the **R-Forge** repository.

4. Browse to some folder where you'd like to check out your project from the **R-Forge** repository. I'd suggest making a new folder in which to keep ongoing "snap shots" of the repository, because you may be periodically updating your project over the weeks and months to come.

5. Right-click in the empty space of the folder to pop up the context menu (see Figure 6.13) Since you have installed **TortoiseSVN**, some extra items specific to **TortoiseSVN** appear in this context menu.

6. Select **SVN Checkout...** from the context menu (alternatively, you could go to your

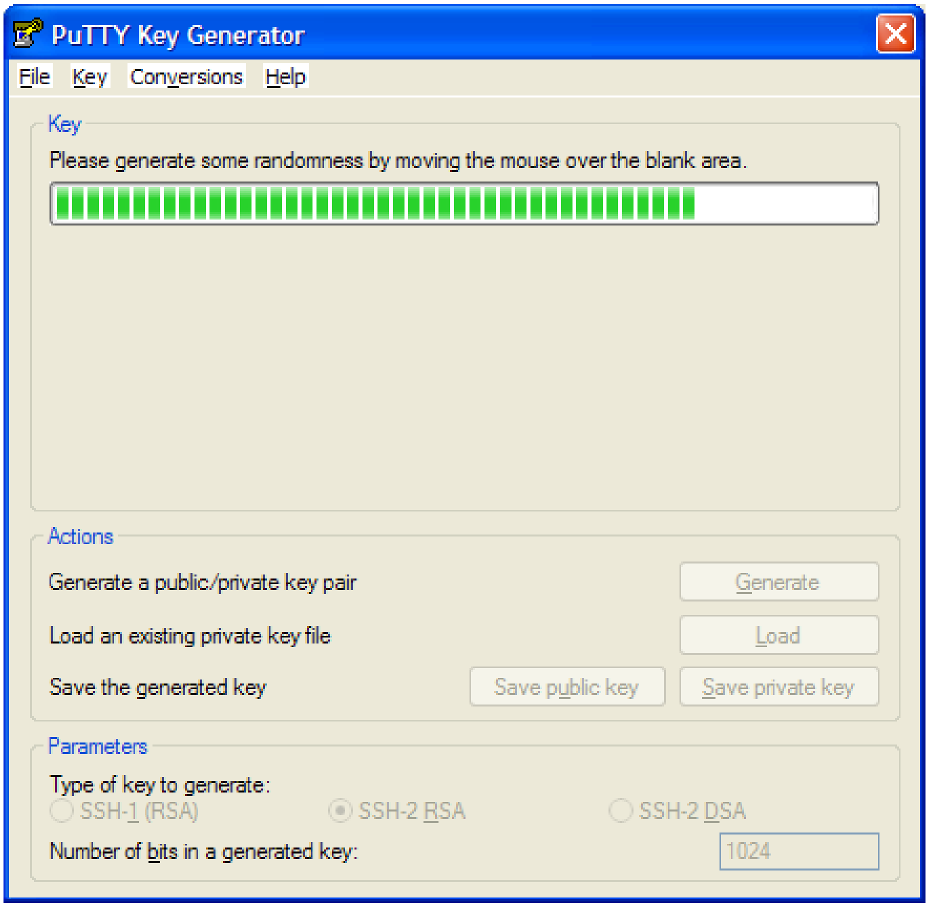

Figure 6.7: **PuTTY** Key Generator in the process of generating public and private keys.

Windows browser and select File → TortoiseSVN Checkout...). This will pop up the **TortoiseSVN Checkout** dialog box, as seen in Figure 6.14. This dialog box will create a new folder which will contain the checked out files. You must now type in a name to be given to this "check out" folder.

7. I'd suggest giving the check out folder a name that indicates the project and the current date and/or some sort of version number, to distinguish it from check outs that you may do in the future. E.g., if you were checking out a project named **multmat** for the second time on the date December 17, 2010, you might name the folder something like **multmat_B_121710**.

This should create a "check out" folder with the name you indicated. The check out folder's icon will look slightly different from the usual folder icon. It will have a small red exclamation point overlaid on top, which is **TortoiseSVN**'s way of indicating a "check out" folder that hasn't been committed ("checked in") yet.

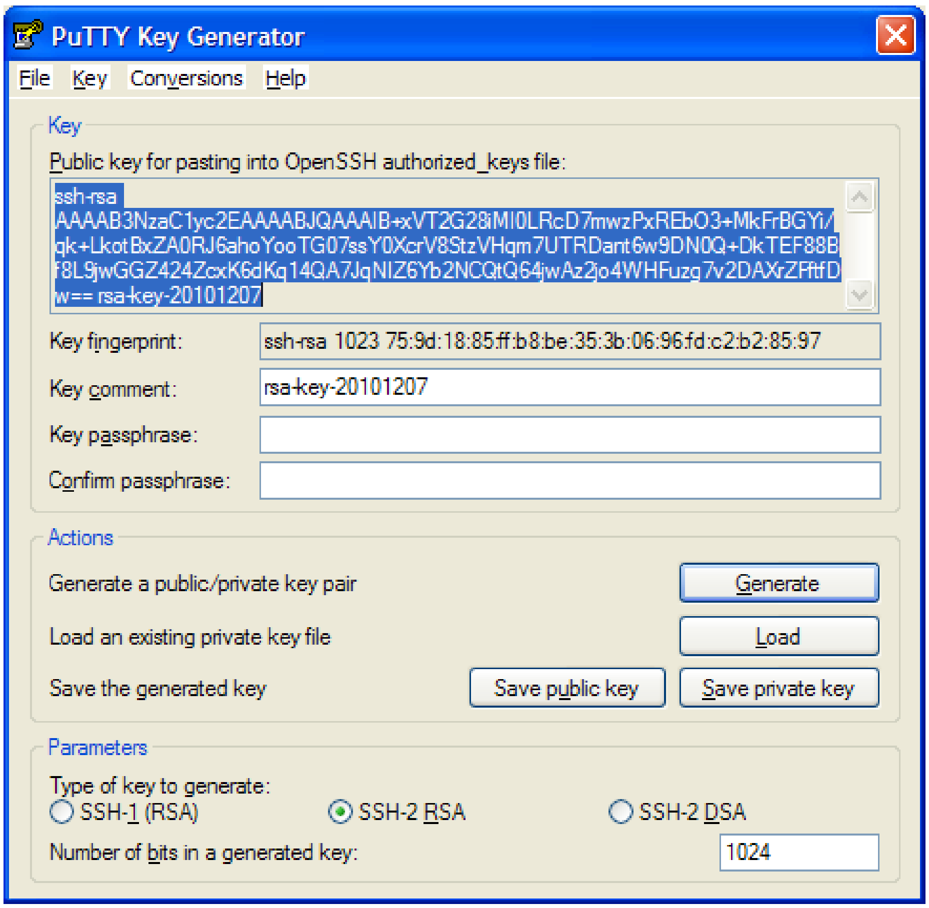

Figure 6.8: **PuTTY** Key Generator, showing keys that were generated. The newly generated public key is the selected text highlighted in blue.

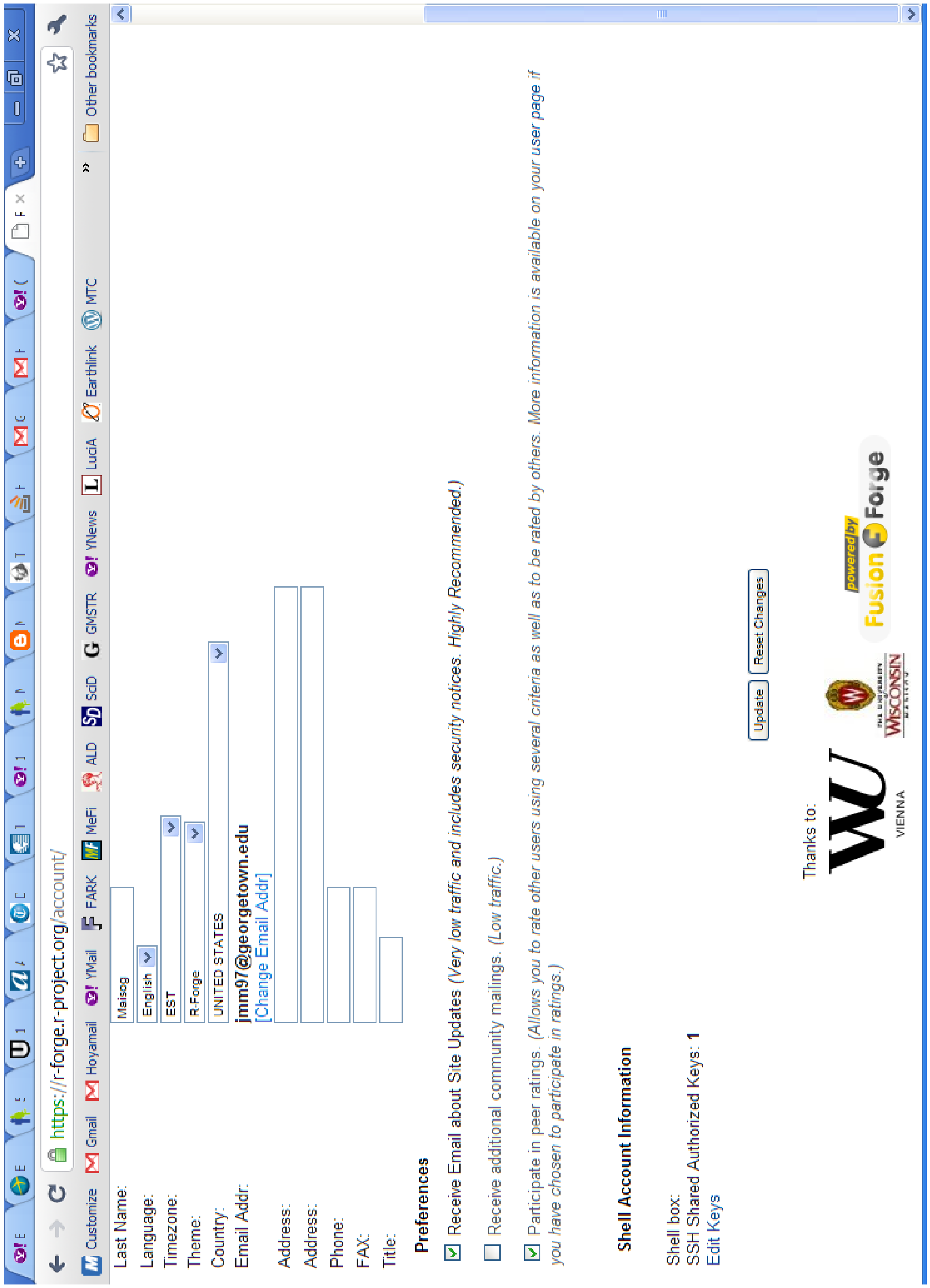

Figure 6.9: Bottom of the **Account Maintenance** Page. Note **Edit Keys** link at the bottom, under **Shell Account Information**.

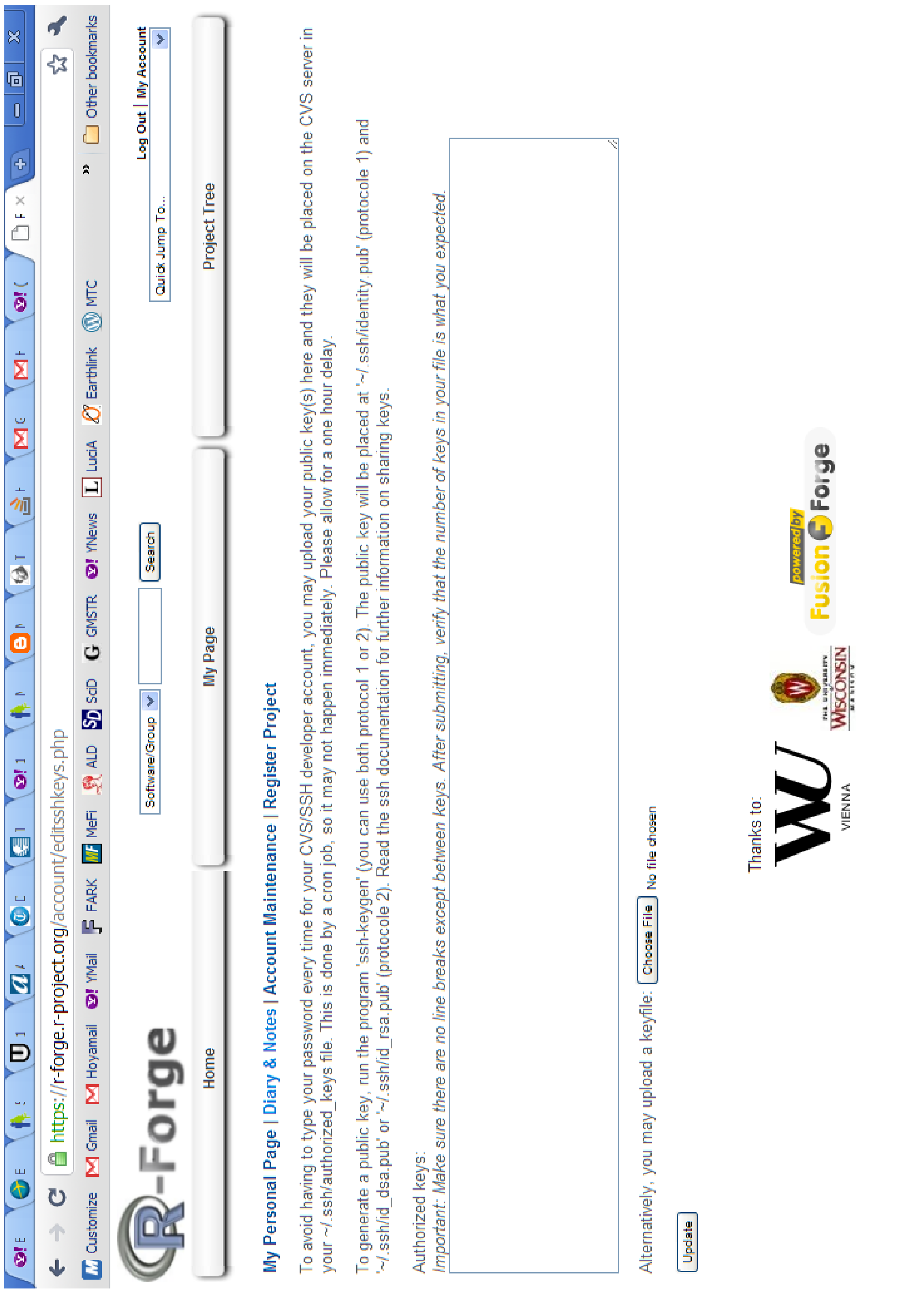

Figure 6.10: **R-Forge** Edit Keys Interface. Paste your *public* key into the entry field.

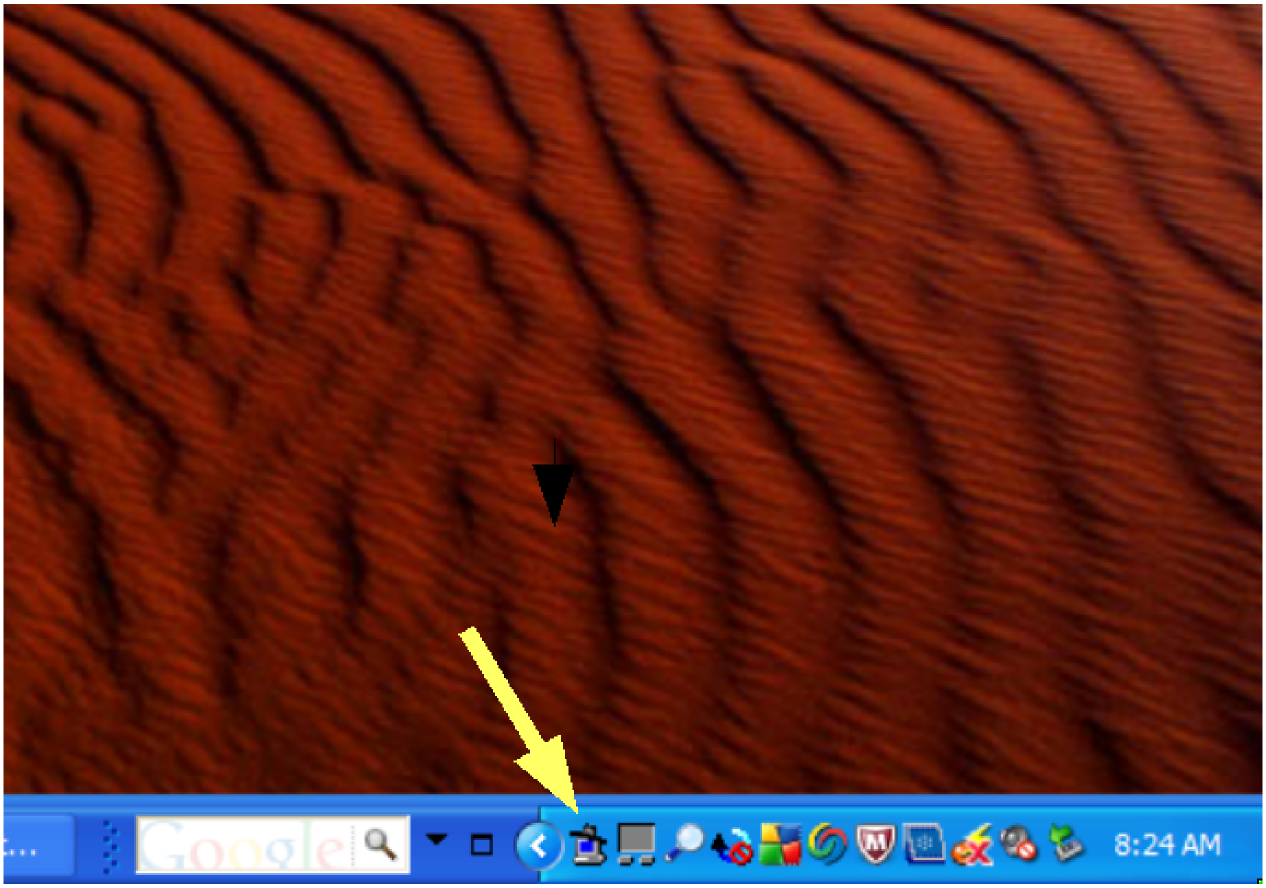

Figure 6.11: **Pageant** System Tray Icon. It looks like a computer monitor wearing a hat.

6.5 Modify Check Out Folder Contents

The check out folder may contain several folders with names like **pkg** and **web**. If you see this set of folders, place the contents of your **R** package into the **pkg** folder; i.e., place your **data**, **man**, **R**, **src** folders and **DESCRIPTION** and **NAMESPACE** files into the **pkg** folder. If the check out folder is empty, then place the contents of your **R** package directly into it.

6.6 Commit Modification To R-Forge

To check the folder back in to **R-Forge**, go into the parent folder of the "check out" folder created in section 6.4. Right-mouse-button-click on the icon for the check out folder, which should pop up a context menu that include **TortoiseSVN** options; see Figure 6.15. From the **TortoiseSVN** context menu, select **SVN Commit...**. This should pop up the **TortoiseSVN Commit** dialog box, as seen in Figure 6.16. Click on the **OK** button to upload the modified files back to the repository on **R-Forge**. After checking the files back into the **R-Forge** repository, the red exlamation point on the icon of the "check out" folder will be

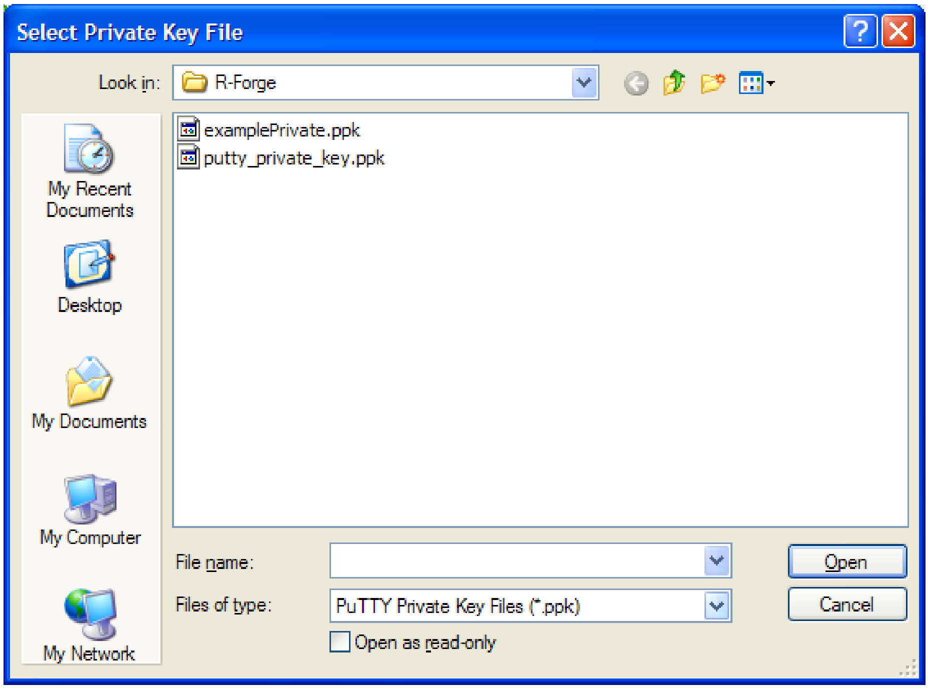

Figure 6.12: **Pageant** Add Key Dialog Box.

changed to a green check sign; this is **TortoiseSVN**'s way of indicating that the "check out" folder has been committed to the respository.

Now you must wait. **R-Forge** performs nightly builds, and if an error is found a log should be generated. You can see the schedule for nightly builds by logging in to your **R-Forge** account and going to the **Contributed R Packages** page for your **R** project; see Figure 6.17. Note the blurb that states "All packages are built/checked according to this schedule"; this sentence provides a link to the schedule for the nightly builds.

After the nightly builds are completed, you can check the build logs for problems by going to the **Contributed R Packages** page for your **R** project. In Figure 6.17, note the table with four columns, two for Linux (32- and 64-bit), and one each for Windows and MacOS. If you want to see whether there was a problem with, e.g., the Windows build, click on one of the **patched** or **devel** links in the Windows column.

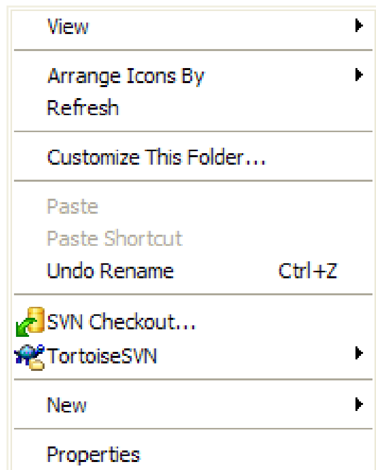

Figure 6.13: **TortoiseSVN** Context Menu.

To install the package directly from **R-Forge**, type, e.g. (within the **R** environment):

```
install.packages("your_R_package", repos="http://R-Forge.R-project.org")
```

6.7 Submit R Package to CRAN

Submitting your **R** package to **CRAN** from R-Forge is trivial. Just go to the **Contributed R Packages** page and click on the link "Submit this package to CRAN", found just below the table for the Linux, Windows, and MacOS builds; see Figure 6.17.

Now you must wait again. It may take a few days for your **R** package to show up on **CRAN**. Also, don't be surprised if one build shows up quickly on **CRAN** whereas another build takes a few more days to show up; the different builds are apparently managed by different teams, and at any one time one team may be busier than another. I don't think that the builds are automated to the extent that they are on **R-Forge**.

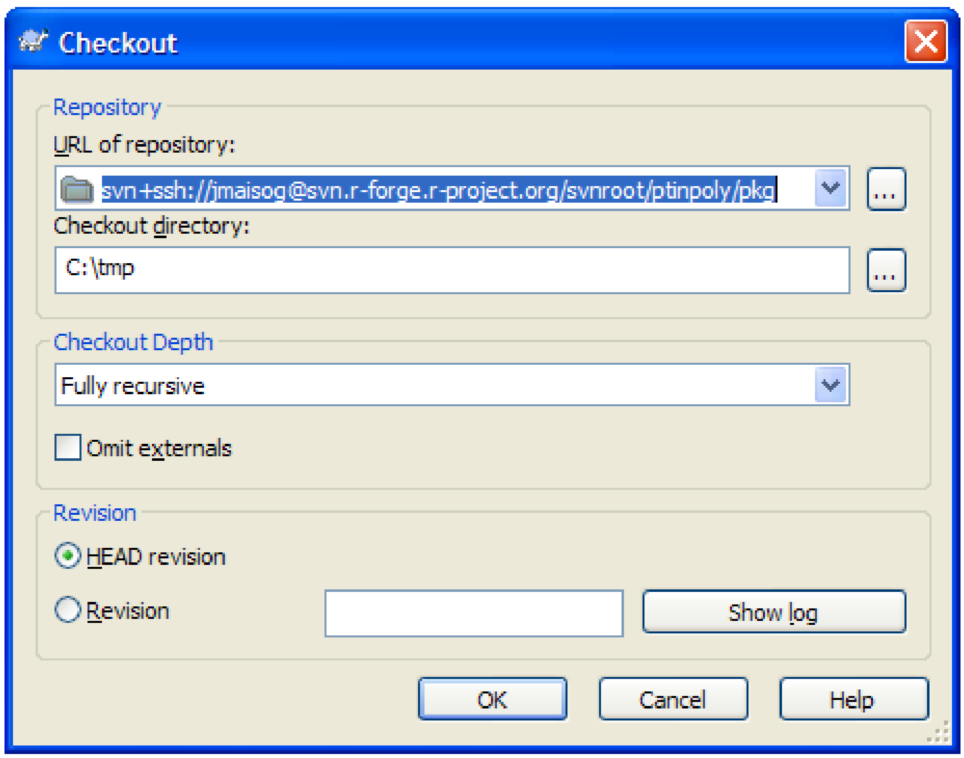

Figure 6.14: **TortoiseSVN** Checkout Dialog Box.

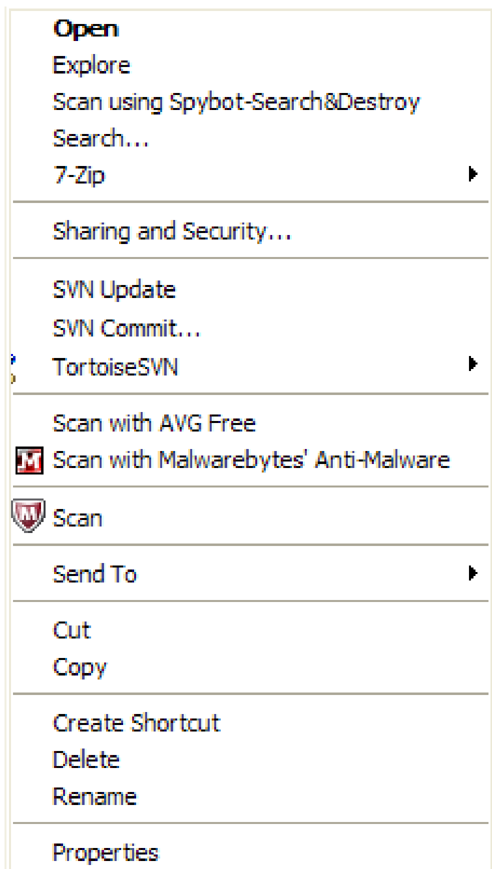

Figure 6.15: **TortoiseSVN** Context Menu With Commit Option.

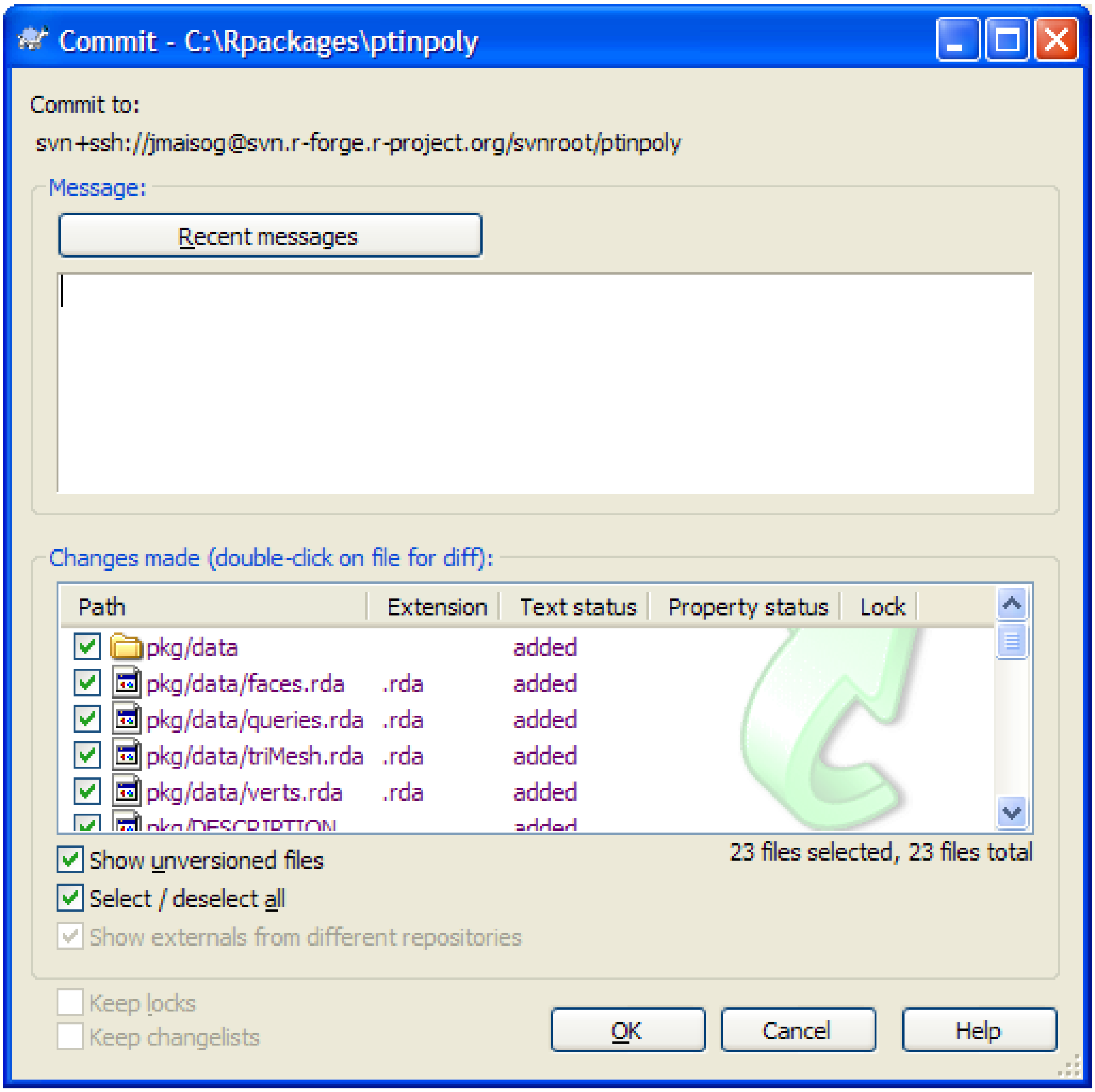

Figure 6.16: **TortoiseSVN** Commit Dialog Box.

To install directly from **CRAN**, type, e.g.:

```
install.packages("your_R_package", repos="http://cran.r-project.org")
```

Later, if you discover that your **R** package has a bug, check out the files from **R-Forge** as described in section 6.4. This time, the check out folder will contain local copies of the **R** package files stored in **R-Forge** Modify the files in the check out folder to fix the bug; after making your edits, you might want to update the date and version number in the **DESCRIPTION** file. When you're all finished, commit your modifications to **R-Forge** as described in section 6.6. Don't forget to re-submit the package to **CRAN**.

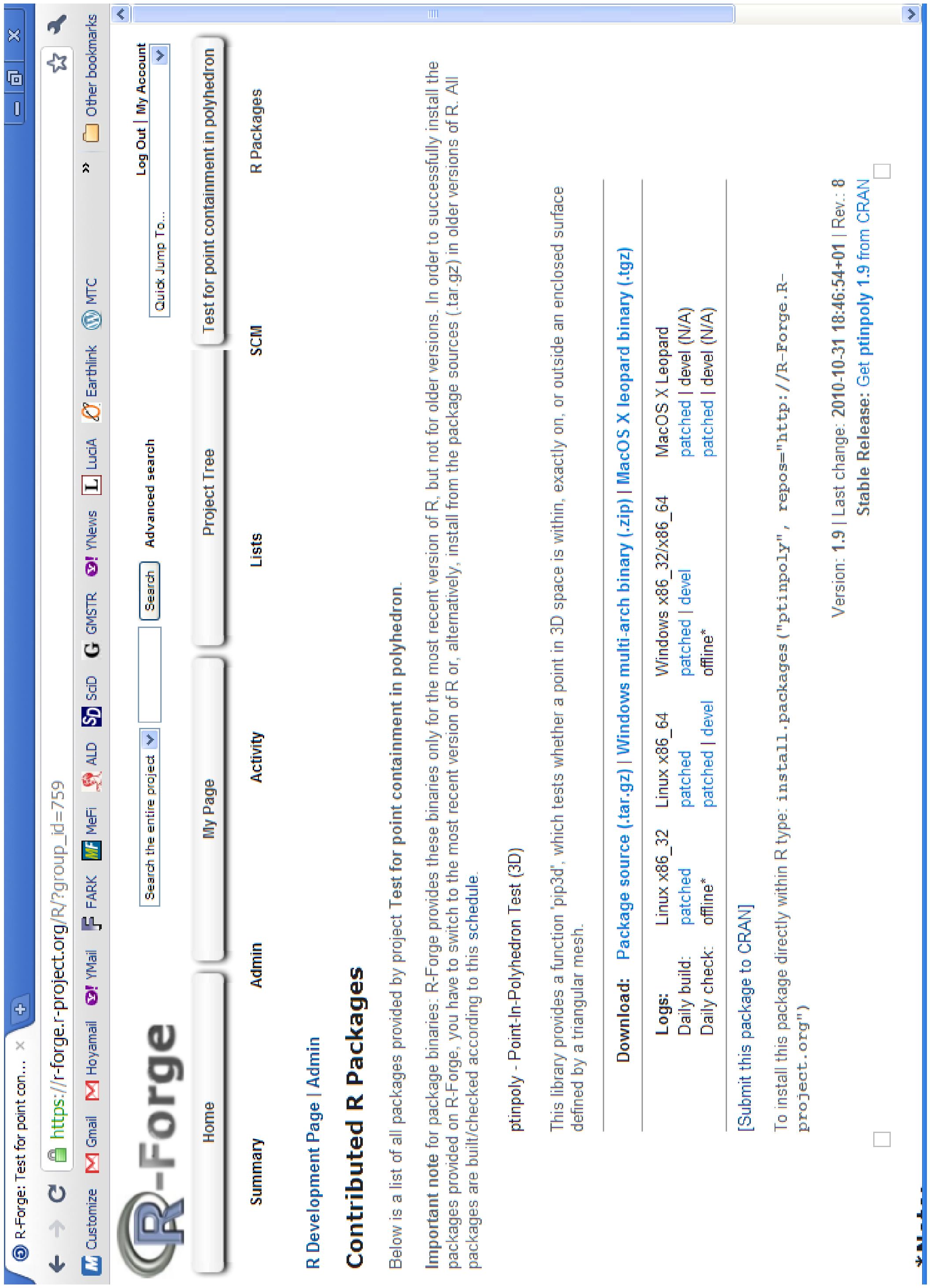

Figure 6.17: R Packages Page. Note table with four columns, two for Linux (32- and 64-bit), and one each for Windows and MacOS. Also note the **Submit this package to CRAN** link just under the table.

Bibliography

[1] R-Forge Administration and Development Team. R-Forge User's Manual (BETA). Technical report, The R Project, 2009.

[2] J Chambers. Software for Data Analysis: Programming with R. Springer, New York, 1st edition, 2008.

[3] RJ Hyndman. Building R packages for Windows. Technical report, Monash University, 2008.

[4] F Leisch. Creating R Packages: A Tutorial. Technical report, Ludwig-Maximilians-Universität München, and R Development Core Team, 2009.

[5] A Lenarcic. R Package Writing Tutorial. Technical report, Harvard University, 2008.

[6] J Liu, YQ Chen, JM Maisog, and G Luta. A new point containment test algorithm based on preprocessing and determining triangles. Computer-Aided Design, 42(12):1143–50, 2010.

[7] JL Schafer. How to Make an R Package for Windows With Native Routines in Fortran 95. Technical report, Pennsylvania State University, 2008.

[8] The R Development Core Team. Writing R Extensions, v. 2.8.0. Technical report, The R Project, 2008.

[9] S Theußl and A Zeileis. Collaborative Software Development Using R-Forge. The R Journal, 1(1):9–14, 2009.

[10] T Veldhuizen. Linear Algebra with C++ Template Metaprograms. Dr. Dobb's Journal, 21(8):38, 40–42, 44, 1996.

[11] T Veldhuizen. Techniques for Scientific C++. Technical report, Indiana University, 2000.

Appendix A

MinGW gcc: Link Stand-Alone Example C++ Code To R Libraries

Suppose that for some reason you want to compile the *stand-alone* C++ code given in chapter 3, but with the **R** libraries linked in as they are in the **R** package. In this appendix, I show how this might be done using **gcc**. This demonstrates the possibility of invoking **R** functions (e.g., math functions) outside of R, in a stand-alone C++ program.

Edit the file **RMat.h** so that the macro

```
COMPILE_WITH_R
```

is set to 1 rather than 0 (see section 4.2).

In addition to enabling lines to access the **R** libraries in the files **RMat.h** and **RMat.cc**, this will also enable a line in the driver program **testRMat.cc** that calls an **R** math function.

Then, build the two **.o** object files **RMat.o** and **testRMat.o** with commands that look like this:

```
gcc -c RMat.cc -o RMat.o -I"C:/RTools/MinGW/include" -I"C:/Program Files/R/R-2.10.1/include"
gcc -c testRMat.cc -o testRMat.o -I"C:/RTools/MinGW/include" -I"C:/Program Files/R/R-2.10.1/include"
```

You'll note that each invocation of **gcc** has two **-I** arguments. These two arguments tell the compiler where to find the *include files* for standard C++ functions and for the **R** functions. Of course, your installation of **RTools** and for **R** may differ from mine, so you'll need to modify the folder paths accordingly. (The code in **RInterface.cc** is not involved, since we're not accessing the C++ code through **R** here.)

Then, link the two object files with the **R** library as follows:

```
gcc testRMat.o RMat.o -o testRMat.exe -L"C:/RTools/MinGW/lib" -L"C:/Program Files/R/R-2.10.1/bin" -lR -lstdc++
```

Here, the two **-L** arguments tell the compiler where to find the *binary libraries* for standard C++ functions and for the **R** functions. Again, you'll need to modify the folder paths according to your own computer system.

Then, run the the newly built executable by typing

```
testRMat.exe
```

at the DOS prompt. You should see the same output that you saw in chapter 3. But in addition, you should see this line at the end:

```
Using R math library to compute R_pow(2,3) = 8
```

This demonstrates that the C++ code successfully accessed the **R** math function **R_pow**. As mentioned earlier, see Chapter 6 of [8] for more on the **R** API (URL: `http://cran.r-project.org/doc/manuals/R-exts.html#The-R-API`); that chapter lists many **R** functions that you might find useful.

Appendix B

Dev-C++: Link Stand-Alone Example C++ Code To R Libraries

Here I show how to link the stand-alone example C++ code to the **R** libraries, but using **Dev-C++** rather than **gcc**. The process is pretty much the same as for **gcc**, except that specifying library and include directories is done through a graphical user interface rather than on the command line. Again, be sure that the

```
COMPILE_WITH_R
```

macro is set to 1 rather than 0 (see section 4.2).

1. In **Dev-C++**, create a new Console Application project.
2. Add the three files **testRMat.cc**, **RMat.cc**, and **RMat.h** to the project.
3. In **Dev-C++**, select Project → Project Options. This should bring up the **Project Options** dialog box, as shown in Figure B.1.
4. In the **Project Options** dialog box, click on the **Parameters** tab. In the white pane under **Linker**, type in the text

   ```
   -lR
   ```

   (The second character is a small L, not a numeral 1.) This is to tell the linker to link in the **R** library at build time. This is illustrated in Figure B.2.
5. In the **Project Options** dialog box, click on the **Directories** tab. This should bring up the **Directories** sub-dialog box as shown in Figure B.3. The **Library Directories** sub-tab should be selected by default. Add the **R bin** folder by clicking on the little square button with an icon of a folder tree structure, browsing to the **R bin** folder, then clicking on the **Add** button (you *must* click on the **Add** button).

6. Still in the **Directories** sub-dialog box, click on the **Include Directories** sub-tab. Add the **R include** folder by clicking on the little square button with an icon of a folder tree structure, browsing to the **R include** folder, then clicking on the **Add** button (you *must* click on the **Add** button). This is illustrated in Figure B.4.

7. You can now dismiss the **Project Options** dialog box.

8. Then select Execute → Compile. This should build the stand-alone executable.

9. Then select Execute → Run. This should run the program. You should see the same output that you saw in chapter 3. But in addition, you should see this line at the end:

   ```
   Using R math library to compute R_pow(2,3) = 8
   ```

   This demonstrates that the C++ code successfully accessed the **R** math function **R_pow**.

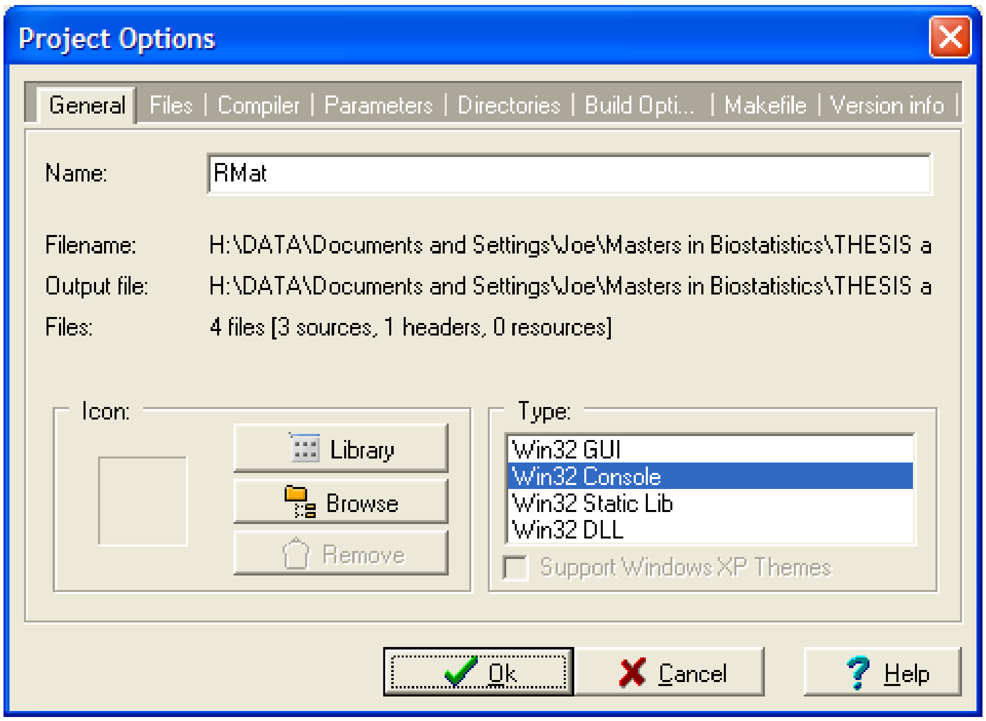

Figure B.1: The **Dev-C++ Project Options** dialog box.

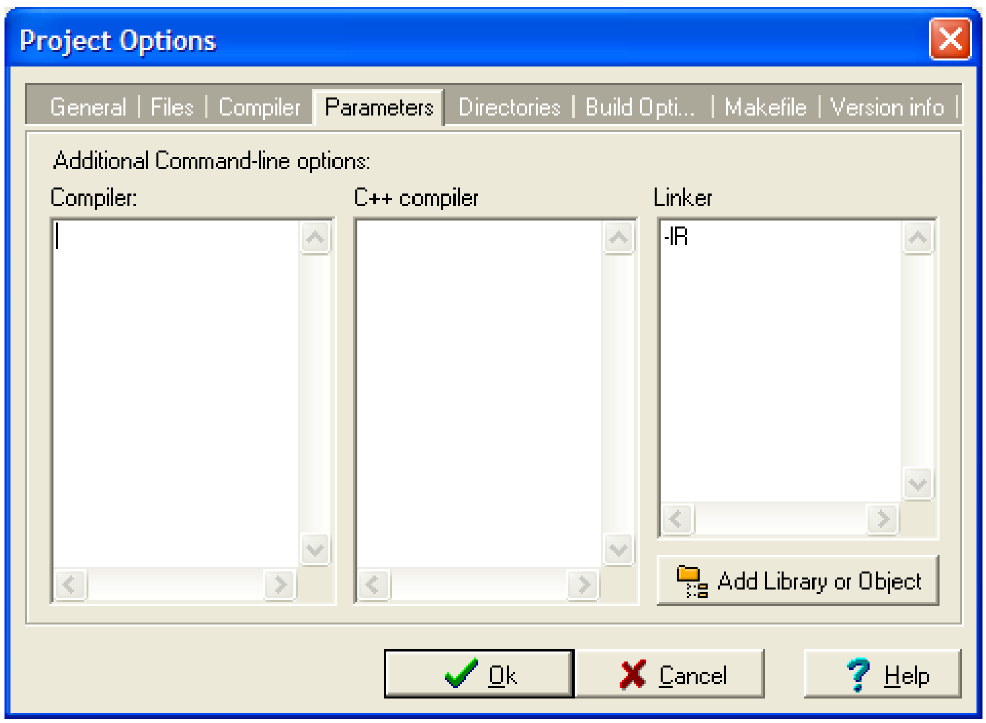

Figure B.2: The **Dev-C++ Parameters** dialog box.

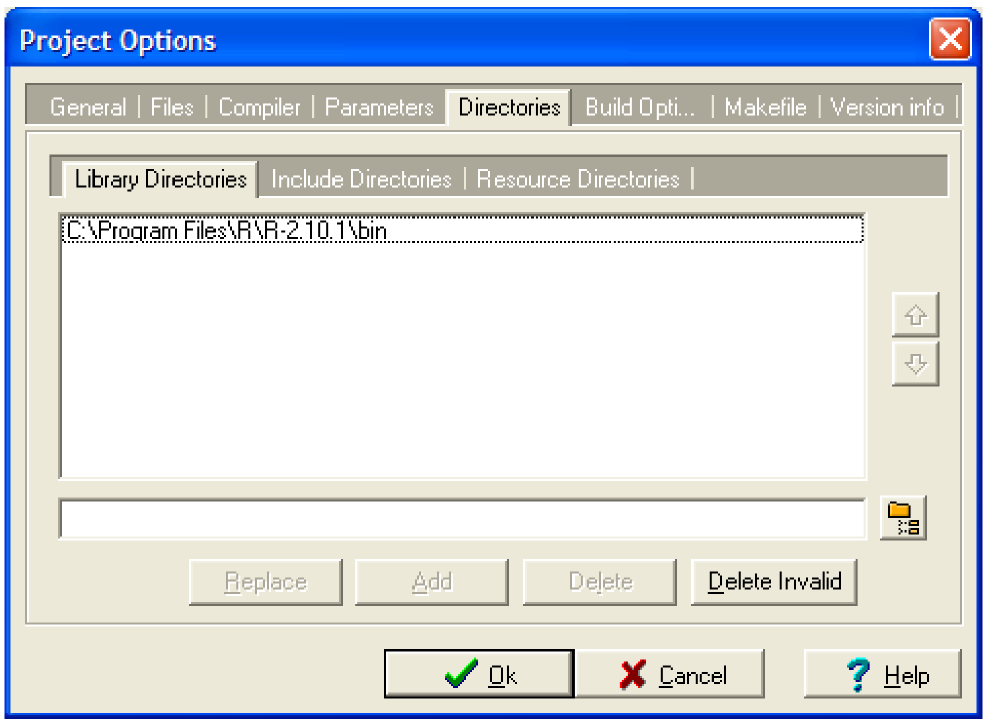

Figure B.3: The **Dev-C++ Library Directories** dialog box.

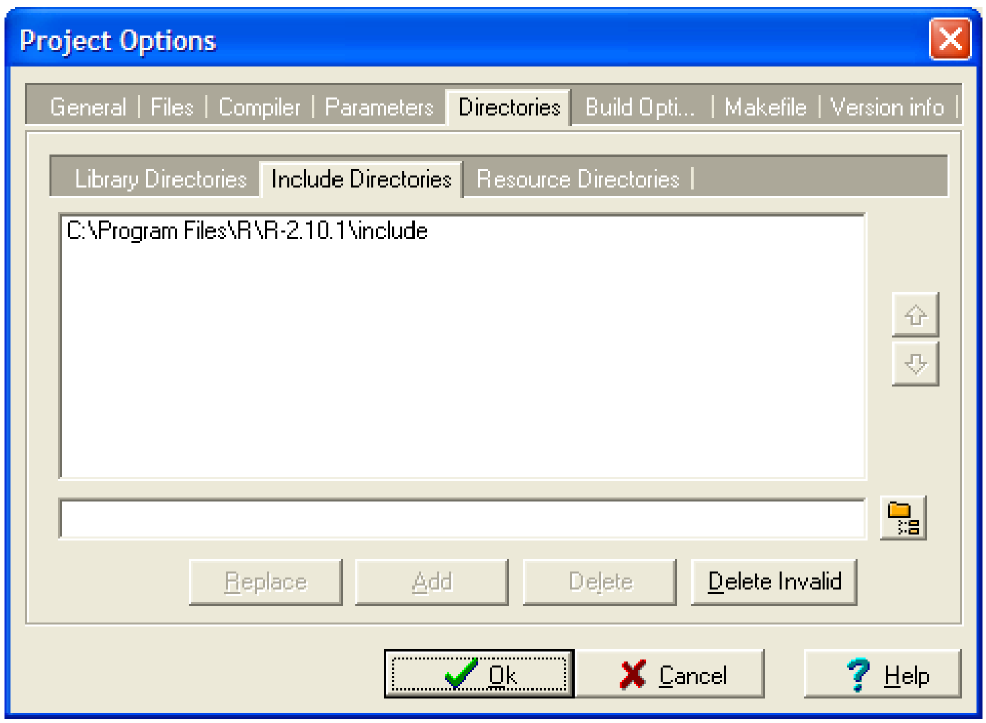

Figure B.4: The **Dev-C++ Include Options** dialog box.