MFE Programming Workshop

Interfacing R to Other Languages

Brett Dunn

Fall 2016

Why would you want to use another language?

- R is great, but it has some weaknesses
  - For example, loops can be slow
- It is sometimes desirable to call code written in other languages from R.
- Also, you may want to call R from a another language
- R interfaces have been developed for a number of other languages
  - We will focus on C/C++
- The main motivation in performance enhancement
  - C/C++ code may run much faster than R

Writing C/C++ Functions to be called from R

- Key points to remember:
  - All the argurments passedd from R to C/C++ are received by C/C++ as pointers
  - The C/C++ function itself must return `void`
    - Hence, we need to pass a pointer for the result
  - For R to work with C++ code (or even C code compiled with g++), you need to wrap your functions inside an an extern statement: extern "C" { yourC++_code_here.... } (or just declare with extern statement)
- We will learn to compile code using R (via gcc and g++) and Visual C++.
- The end product is a dynamic shared library file (.so) on Linux/OS X or a dynamic-link library
  - DLLs are a common way to incorporate number-crunching C++ code in a front-end like R or Excel.

Required software

- On windows, you need to install Rtools, available here
  - Just choose the version that matches your computer architecture (i.e. 64 bit or 32 bit)
  - You have the make sure Rtools is in you path (may need to restart)
- Please verify:
  - On Linux you need to have GNU gcc and g++ (probably already installed)
    - Do you need r-base-dev?
  - On OS X, you may need Xcode.

Our program: `timesTwo.cpp`

```
extern "C" void
  timesTwo(double *in, double *out)
{
  double value = in[0] * 2.0;
    out[0] = value;
}
```

What does `extern "C"` do?

- Remember R is written in C.
- `extern "C"` makes our C++ function available to a program written in C (i.e. R).
  - It declares the functions with C linkage
  - If we write a C program, we don't need it
- Note that the parameter and return types are constained.
  - For example, cannot write a function that passes a (nontrivial) C++ class to a C program
  - The C program would not know what to do about the contructors, destructors, and other class-specific operations.

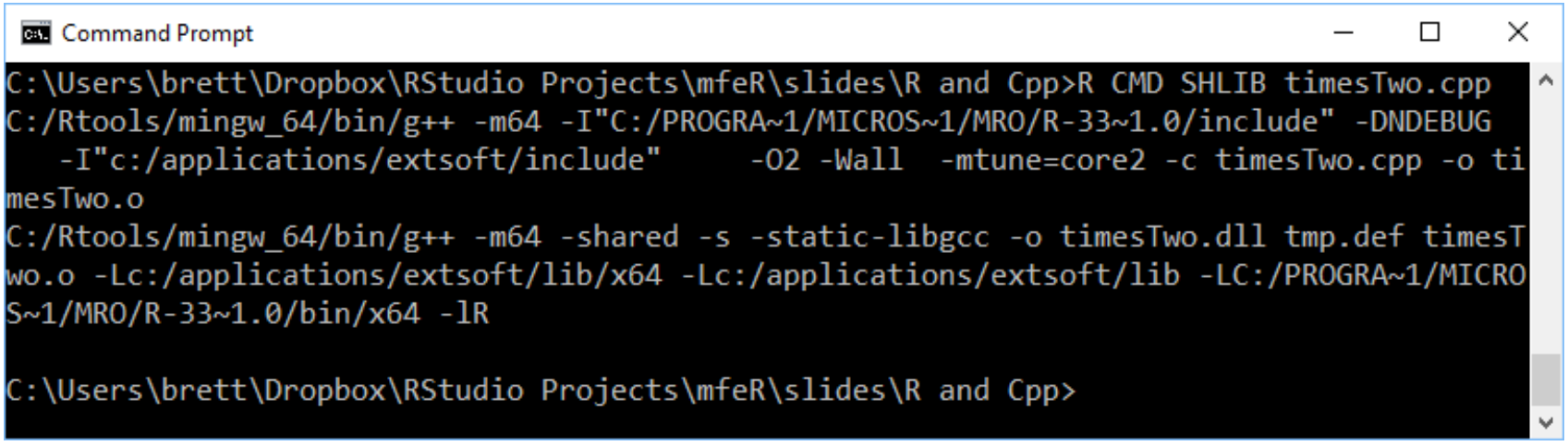
```
extern "C" void
  timesTwo(double *in, double *out)
{
  double value = in[0] * 2.0;
    out[0] = value;
}
```

Compile using R's command line tools

- In R, you can type:

```
system("R CMD SHLIB ./examples/timesTwo.cpp")
```

- Or, on the command line:



- Now we have timesTwo.dll (or timesTwo.so) ready to use in R

Now run the DLL in R

```
dyn.load("./examples/timesTwo.dll")
value_in <- 32; value_out <- 0
.C("timesTwo", as.double(value_in),
   res=as.double(value_out))$res
```

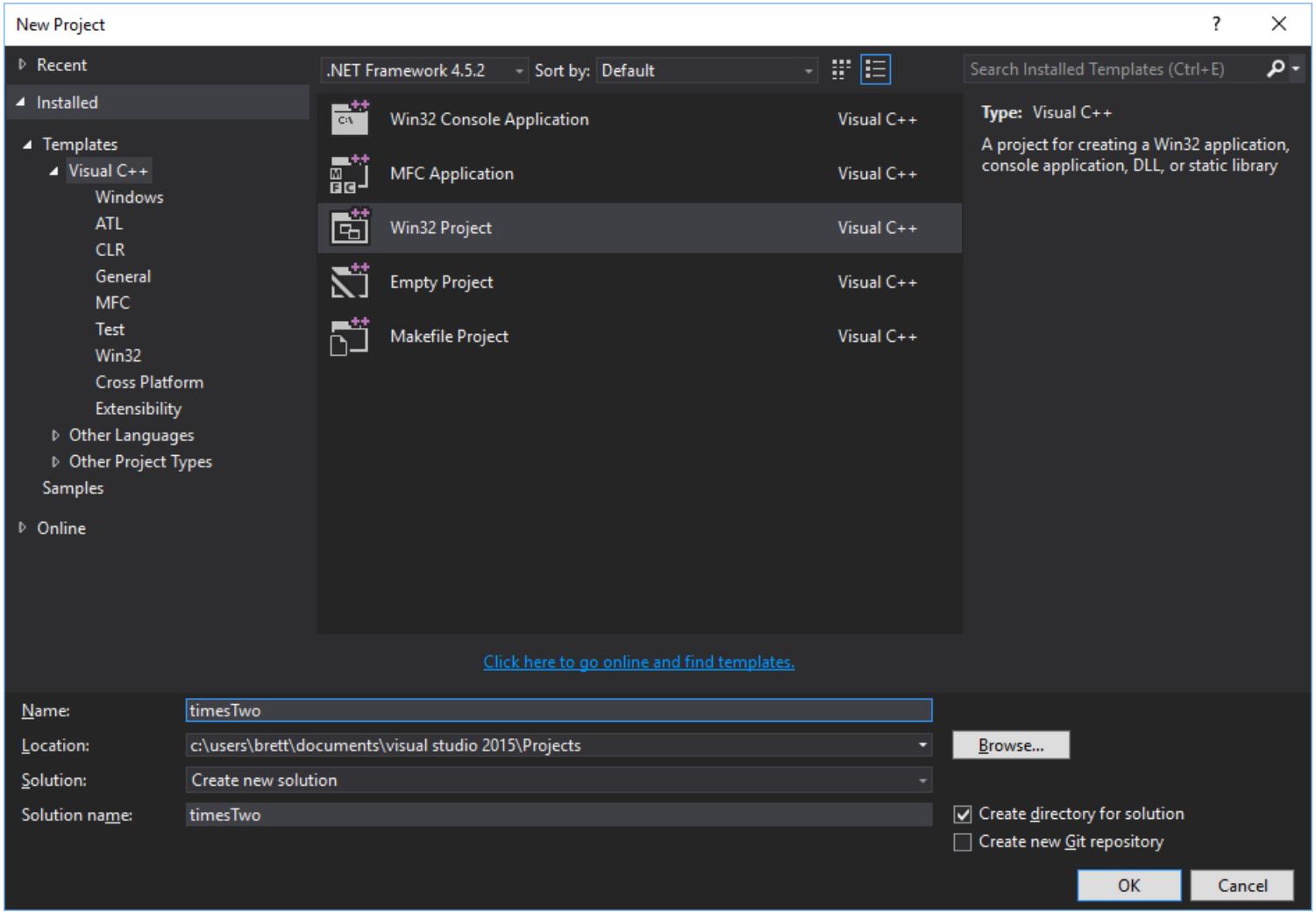
```
## [1] 64
```

```
dyn.unload("./examples/timesTwo.dll")
```

- `dyn.load` loads the .dll into R
- `.C` calls timesTwo, and passes value_in and value_out to the function
  - `.C` returns a list, so we define 'result' and extract 'result' from the list
- `dyn.unload` unloads the .dll from R (you need the unload the dll if you want to rebuild it).

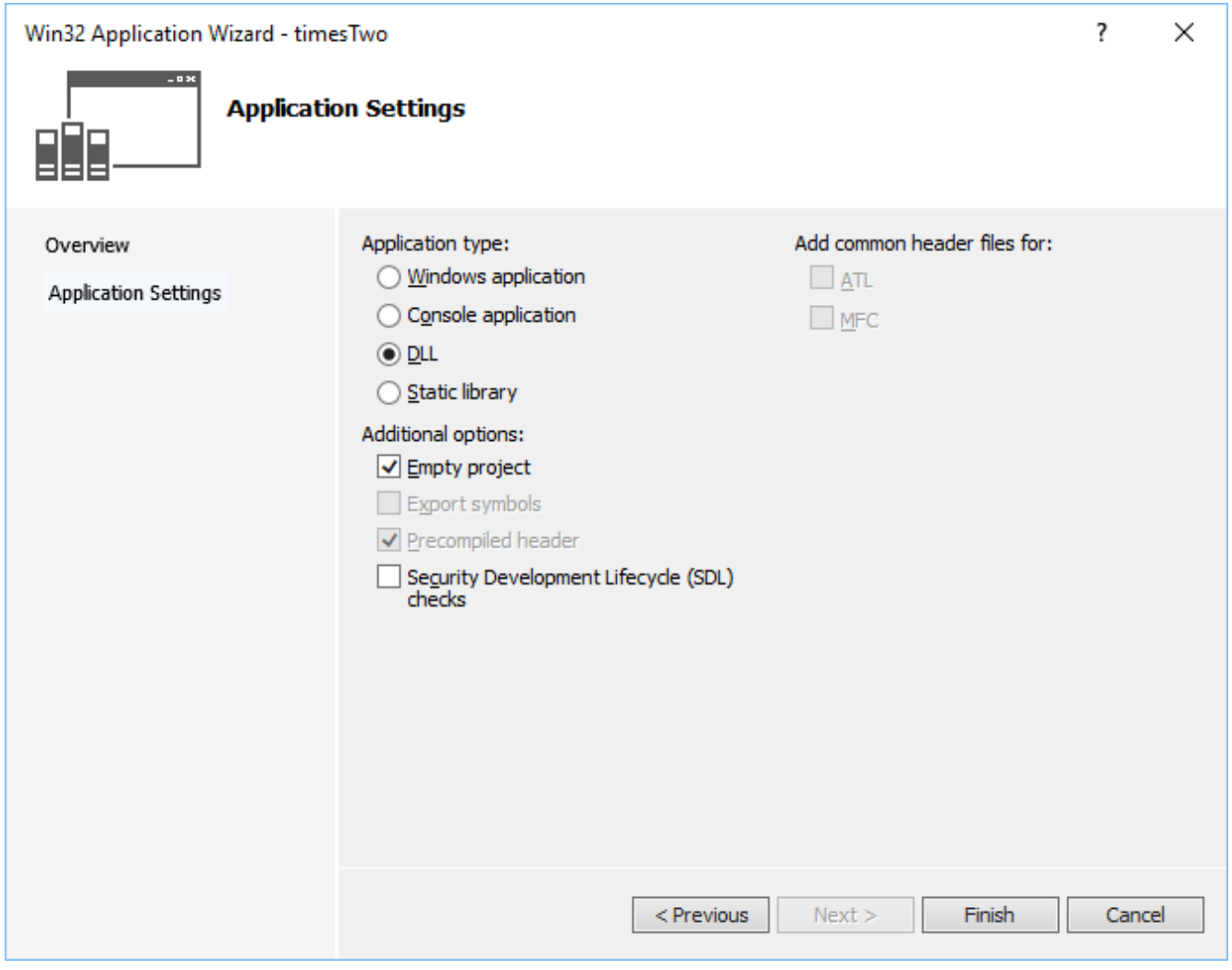
Creating a DLL project in Visual Studio 2015

- Choose File/New/Project../Win32 Project

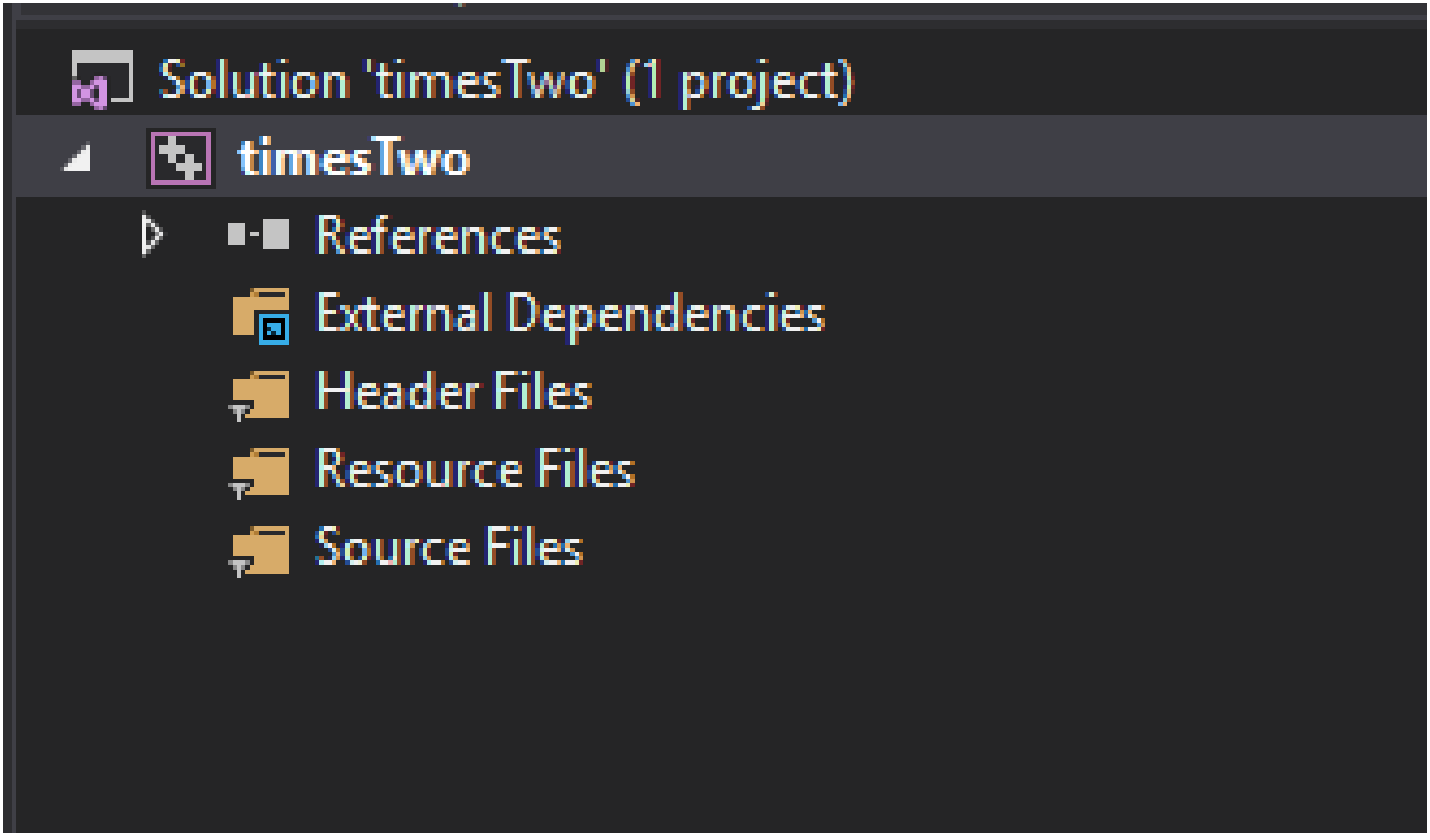


Creating a DLL project in Visual Studio 2015

- Specify a DLL and an Empty project

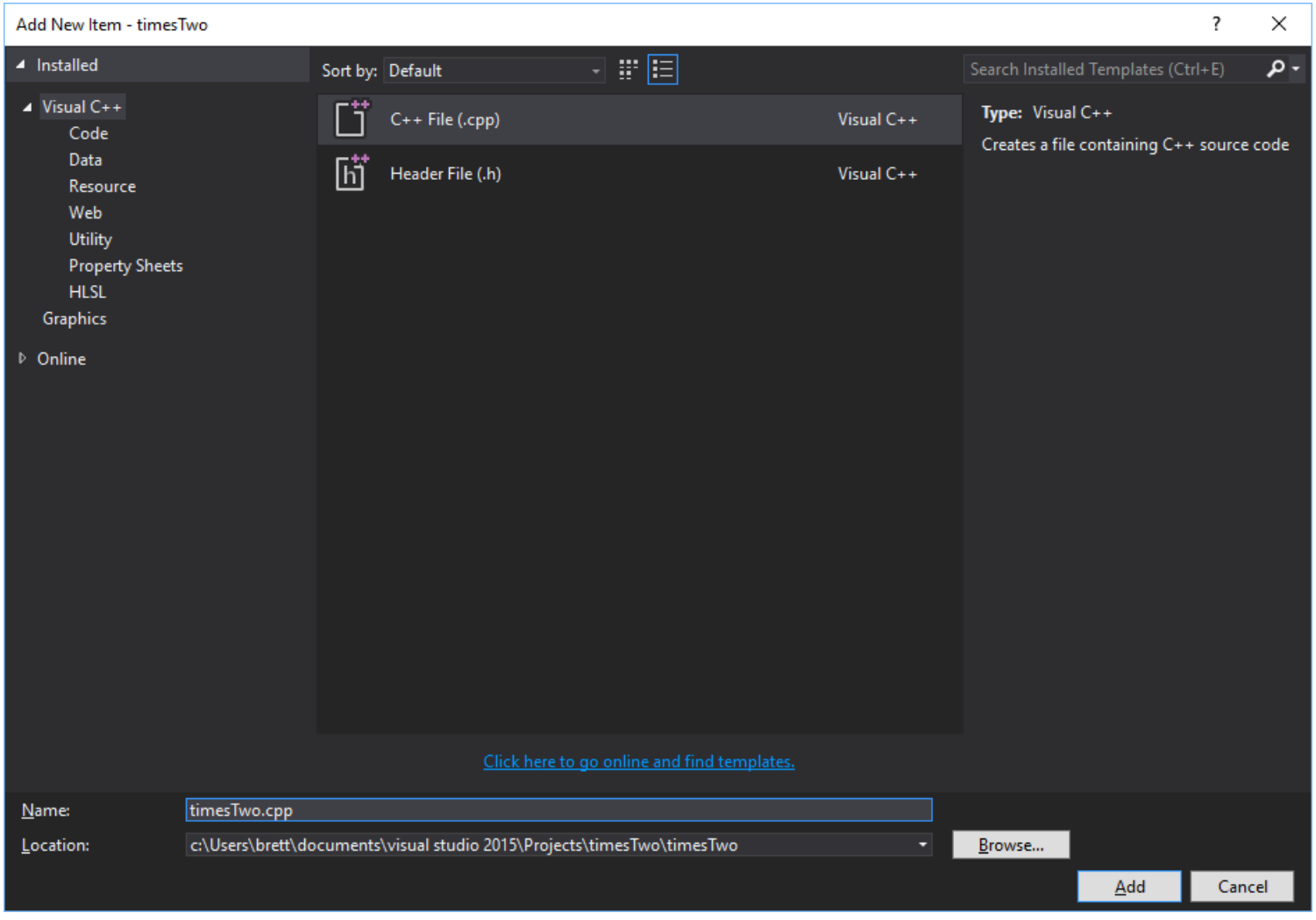


Project at This Point



Add a C++ Source File

-Right-click Source Files in the Solution Explorer, then select Add New Item, and then select C++ File (.cpp)



Add C++ Code to the Source File

```
extern "C" void __cdecl
  timesTwo(double *in, double *out)
{
  double value = in[0] * 2.0;
    out[0] = value;
}
```

What is `__cdecl` about?

- Applies only to Windows.
- The Visual C++ compilers allow you to specify conventions for passing arguments and return values between functions and callers.
- Two options we care about:
  - **`__cdecl`** is used by C/C++ programs, R, Matlab, SAS, others.
  - **`__stdcall`** is used by Excel, Win32 API fuctions, Pascal, others.
- This all essentially amounts to conventions for who (function caller or function) pops arguments off the stack.
- For more information, see this webpage.

```
extern "C" void __cdecl
  timesTwo(double *in, double *out)
{
  double value = in[0] * 2.0;
  out[0] = value;
}
```

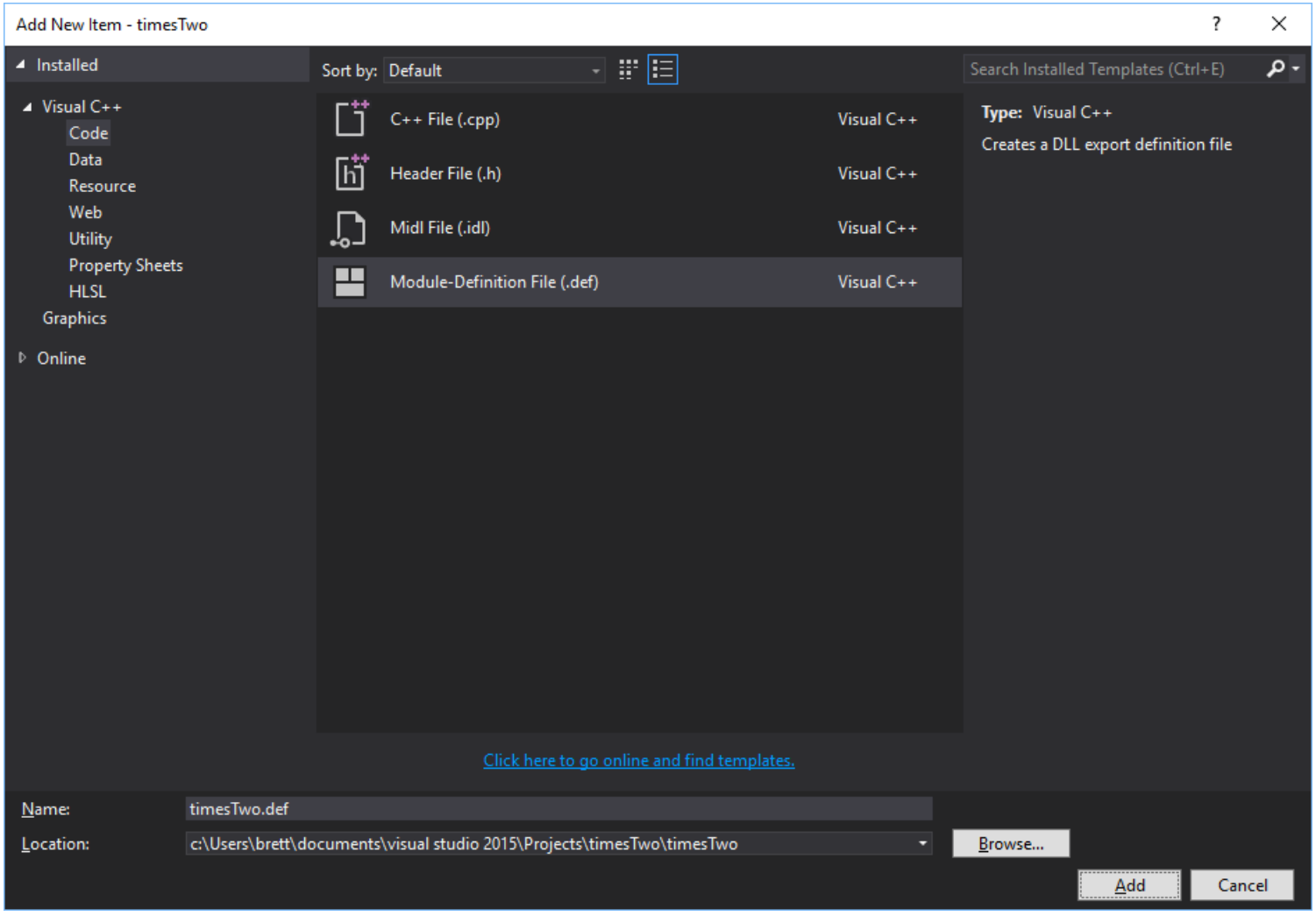
Why are we using pointers?

- In C++ `timesTwo(double& in, double& out)` works as well

```
extern "C" void __cdecl
  timesTwo(double *in, double *out)
{
  double value = in[0] * 2.0;
  out[0] = value;
}
```

Add a Module Definition File (.def)

- Add New Item. . . Under Visual C++ / Code you will find the .def file.



Module Definition File

- A .def file is a module definition file. This is a convenient way to tell the linker which parts of our C++ code we want to export.

```
// timesTwo.def
LIBRARY timesTwoDLL
EXPORTS
  timesTwo
```

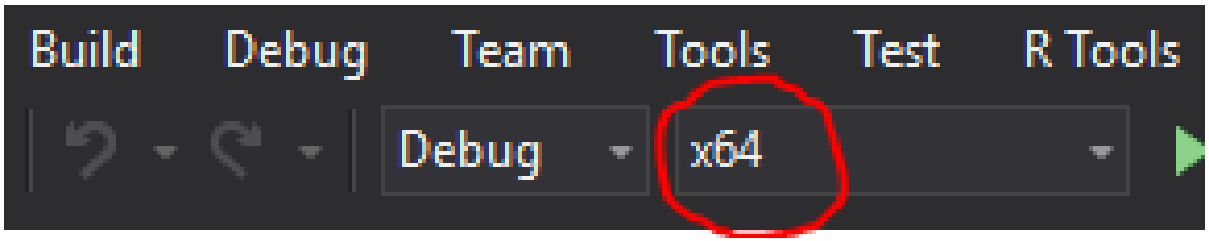
- `LIBRARY` is the name of the DLL
- `EXPORTS` lists the functions to be exported (each one on a seprate line)
  - If you want to use a different function name use `newName = oldName`

Another option: `__declspec(dllexport)`

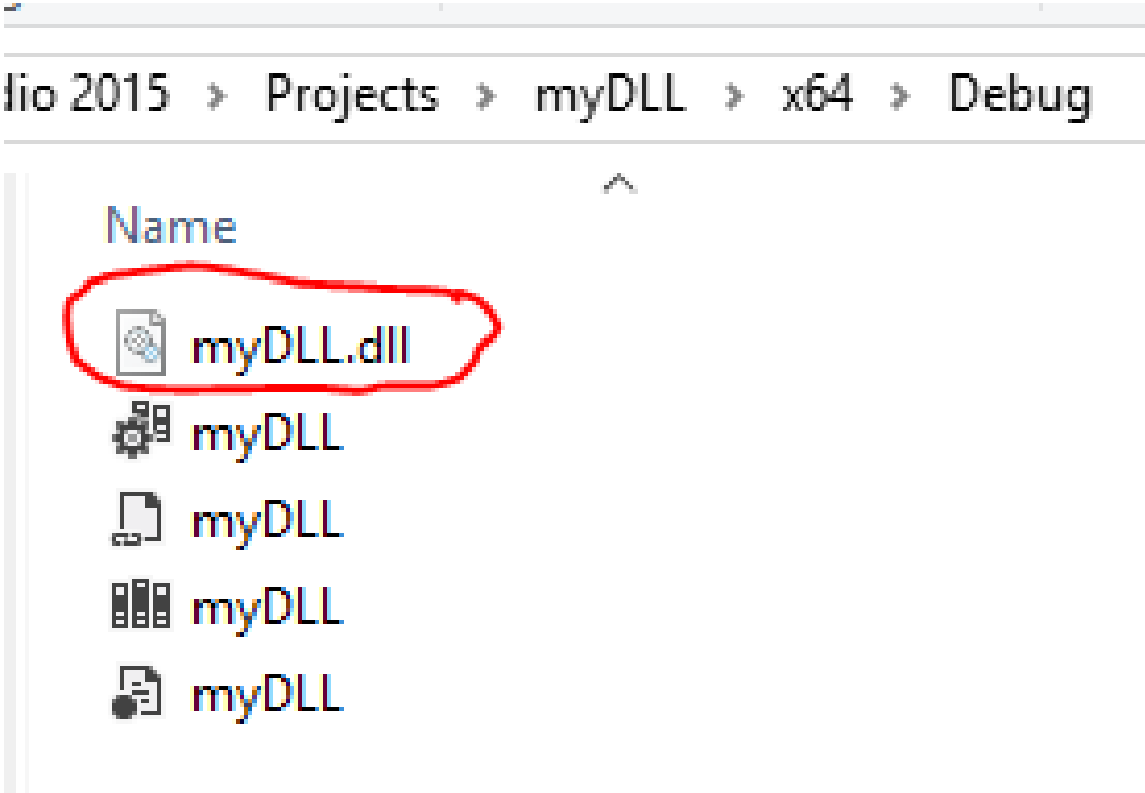
- Windows-specific.
- On Windows, we need to tell which functions are exported from the DLL.
  - That is, which funtions will be available in R.
- we
- When building your DLL, you typically create a header file that contains the functions you are exporting and add `__declspec(dllexport)` to the declarations in the header file.
- For more information, see this.
- Instead of `__declspec(dllexport)`, you can use a DEF file.

Build the Solution

- Make sure to change the architecture to x64 before building (unless you are using 32bit R)

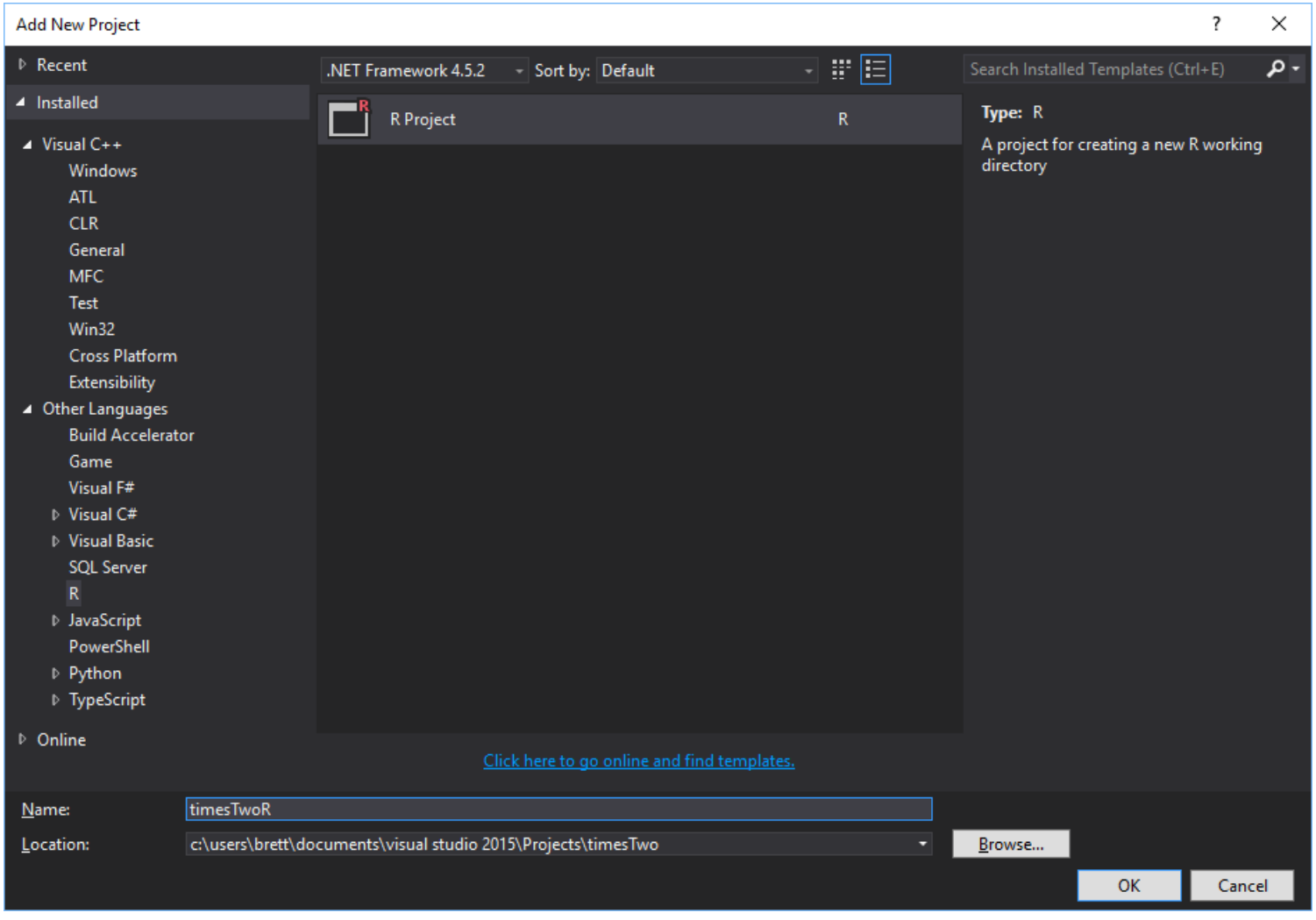


- Ctrl-Shift-B builds the solution.
- The DLL is found in the ./x64/Debug folder



Add an R project to Visual Studio

- Right click the solution. . . Add. . . New Project. . . Other Languages. . . R project



Now run the DLL in R

- `dyn.load` loads the .dll into R
- `.C` calls timesTwo, and passes value_in and value_out to the function
  - `.C` returns a list, so we define 'result' and extract 'result' from the list
- `dyn.unload` loads the .dll into R
  - you need the unload the dll if you want to rebuild it.

```
> dyn.load("../x64/Debug/timesTwo.dll")
> value_in <- 32
> value_out <- 0
> .C("timesTwo", as.double(value_in), result = as.double(value_out))$result
[1] 64
> dyn.unload("../x64/Debug/timesTwo.dll")
```

Let's change the code for Excel

- We don't need `extern "C"` anymore
- The function can return a double
- We need to use `__stdcall`
- Make sure the build matches the Excel version (x64 or x86)
- the .def file remains the same

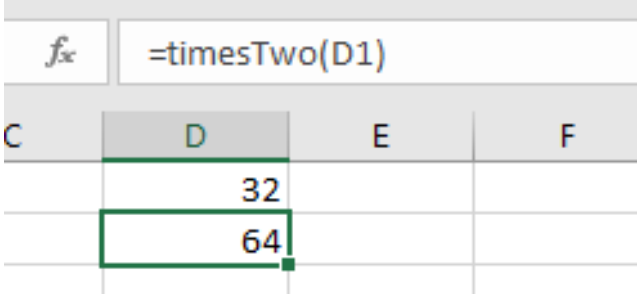
```
double __stdcall timesTwo(double *in)
{
    double value = in[0] * 2.0;
    return value;
}
```

In Excel

- Alt-F11 opens the VBA editor window. Right click on wookbook, Insert/Module
- We'll add a declaration for the function in the DLL.

```
Declare Function timesTwo _
    Lib "C:\PATH_TO_PROJECT\timesTwo\Debug\timesTwo.dll" _
    (ByRef valIn As Double) _
     As Double
```

- Now we can use the function in Excel



Rcpp

- I wanted to show you how build a DLL in visual studio, because it can be useful for more complicated projects
- Often it is easiest to use the Rcpp package instead.
- RCpp makes it easy to pass vectors, matricies, lists, ect, back to R.
    - However, there is overhead in doing this.
    - If you are concerned about speed, consider using the simplest structure

Rcpp Example

1. In RStudio, File / New File / C++ File.
2. Enter code in timesTwoRcpp.cpp

```
#include <Rcpp.h>
// [[Rcpp::export]]
Rcpp::NumericVector timesTwo(Rcpp::NumericVector x) {
  return x * 2;
}
```

3. In R,

```
library(Rcpp)
Rcpp::sourceCpp("./examples/timesTwoRcpp.cpp")
timesTwo(c(32,64))
```

```
## [1]  64 128
```

Using R's Library

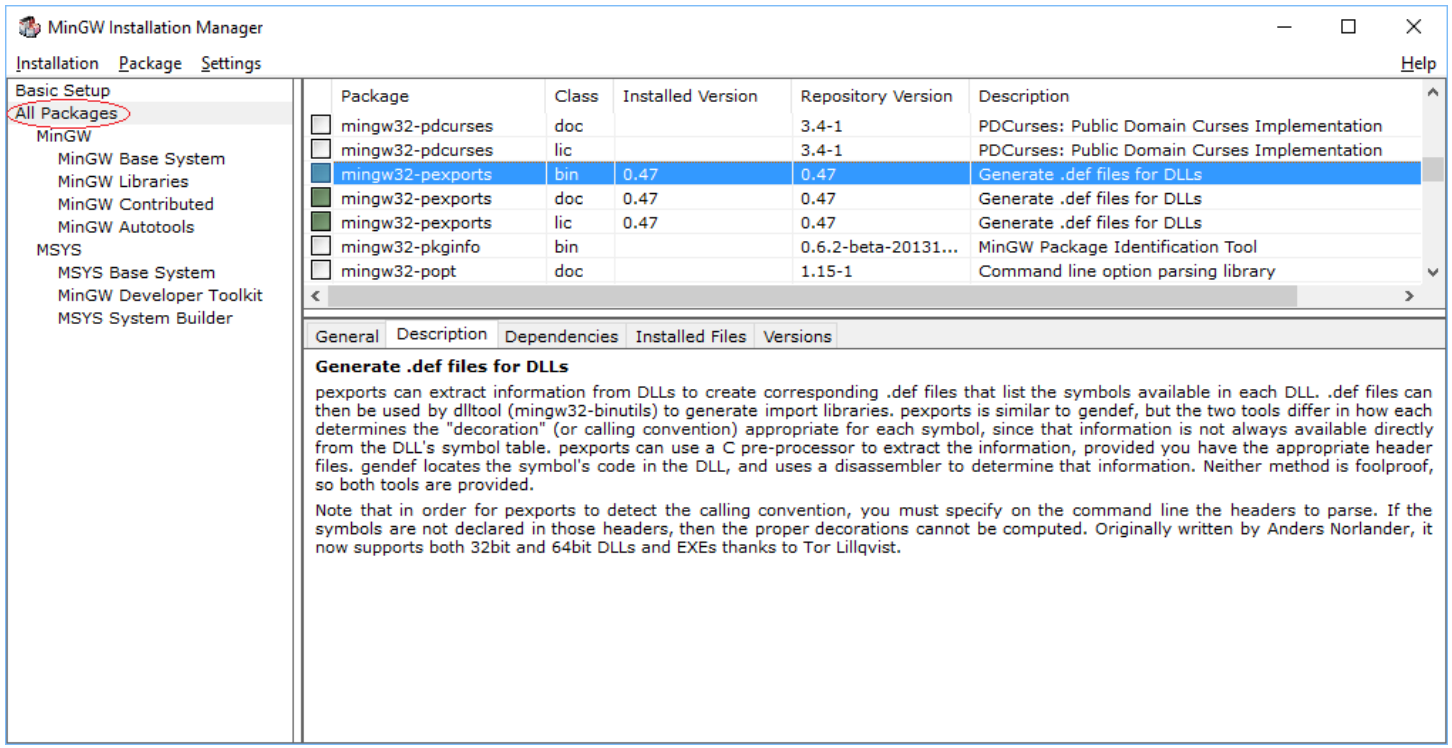
- Check out R-3.3.0\include
  - In that folder there are several header files with functions we can use in C/C++
  -

Using R insdie C/C++

- On linux this is easy and well-documents
- On Windows, it's another story...
- I will show you how to do it on Windows,
- Once you know what to do, it is really easy

Setting up the R API

- First you need `pexports` from MinGW.
  - We will use `pexports` to extract information from R.dll to create a list of symbols in the DLL
  - Then, we will use this file to generate an import library
- Go to MinGW.org to download the installer. Then, install `pexports`.



Setting up the R API

1. Create the exports definition file from R.dll

- From the command prompt type

```
$ cd "C:\Program Files\Microsoft\MRO\R-3.3.0\bin\x64\R.
$ pexports R.dll > R.exp
```

  - Note if pexports is not in your path, you will need to use the full path above

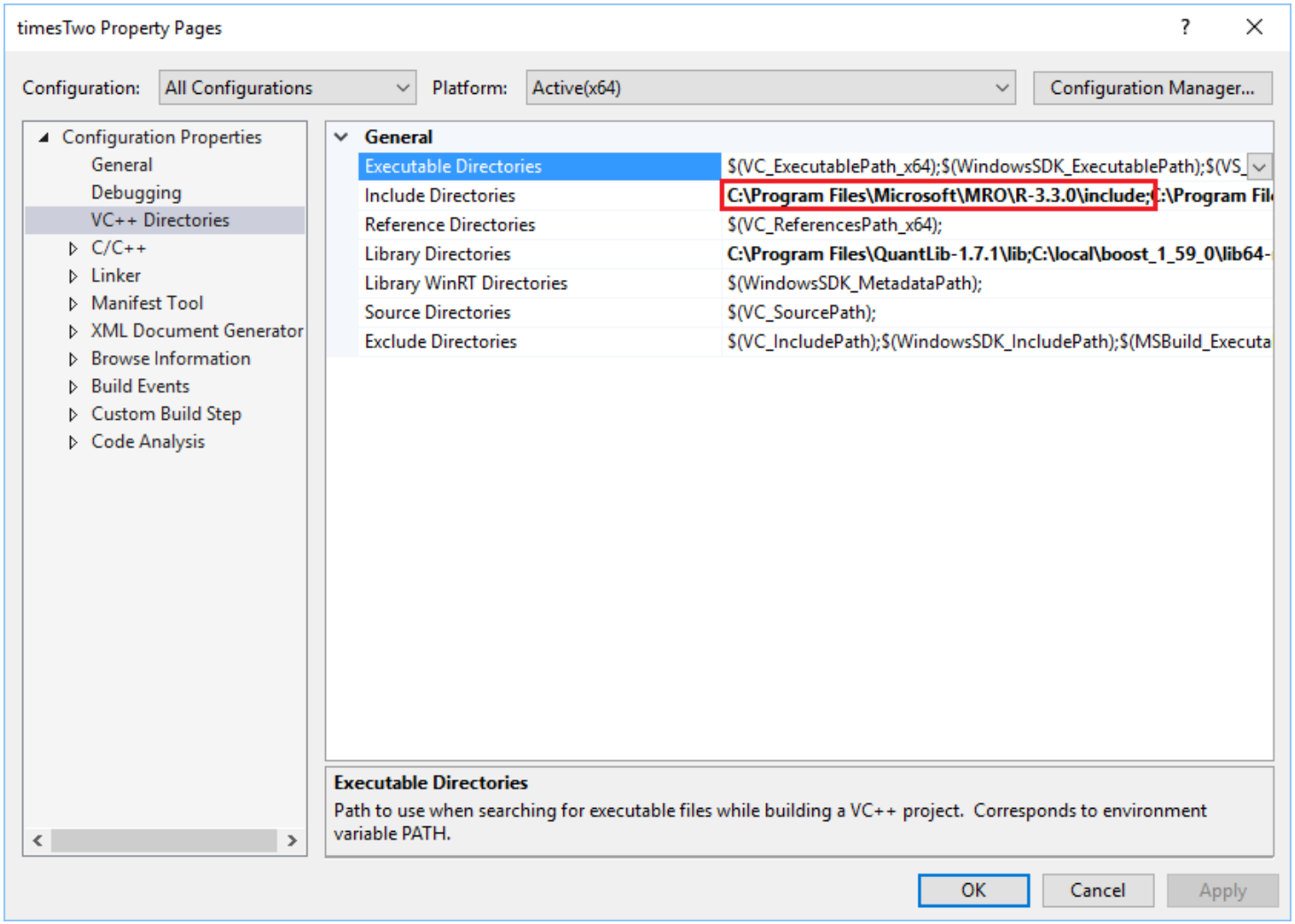
2. Then create the library file using VC++

```
$ lib /def:R.exp /out:Rdll.lib /MACHINE:X64
```

  - Now we can use this library in Visual Studio.

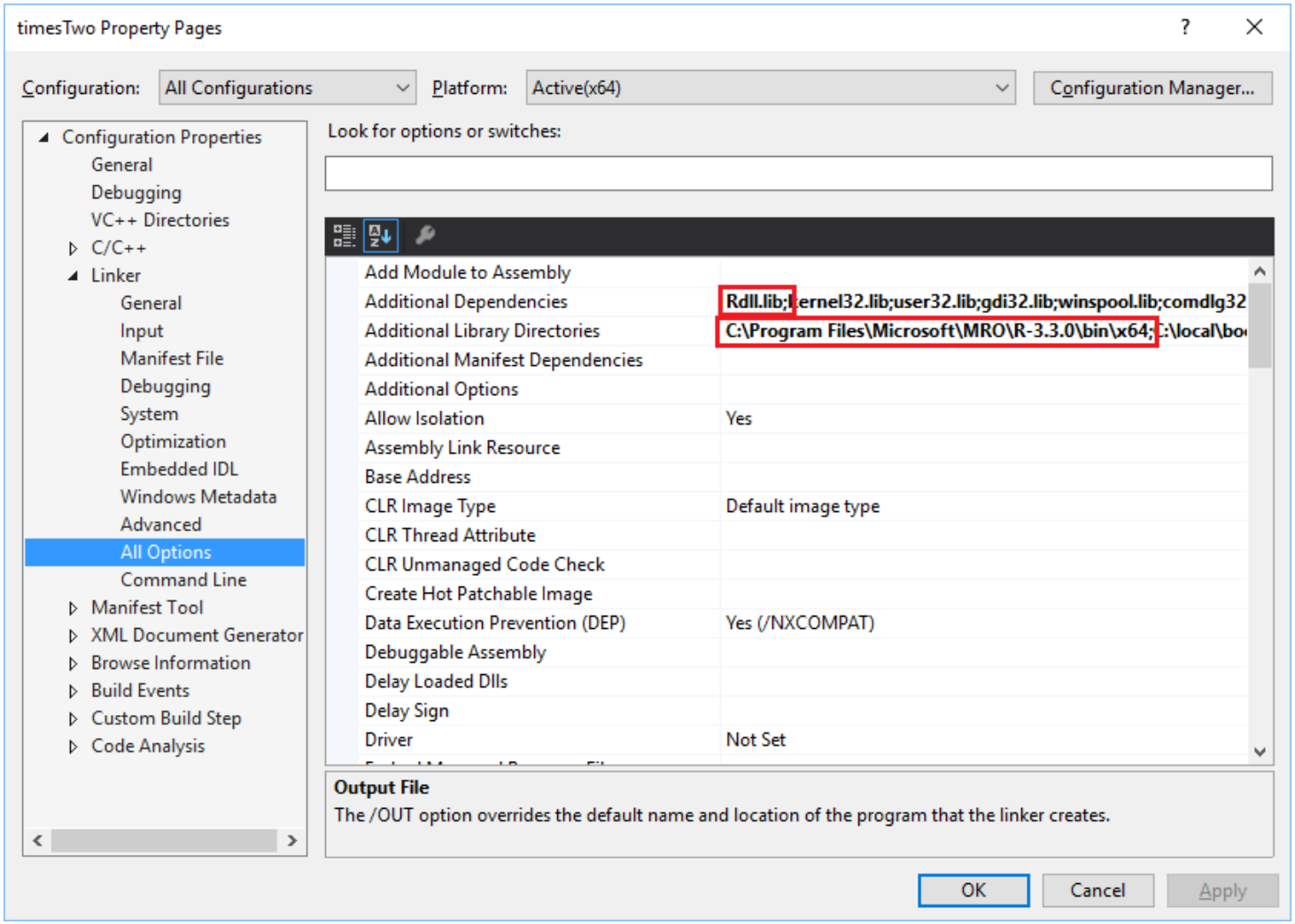
Add the path to the R-Version\include

- In Visual Studio, right-click the project to open up the property pages
- Add the path to the R header files



Add the Rdll.lib dependency

- Property pages/linker/all options
- Add Rdll.lib to the additional dependencies
- Add its path to the additional library Directories



R's random number generator in C++

```
extern "C" void __cdecl randNorm(double *out)
{
    GetRNGstate();
        out[0] = norm_rand();
        PutRNGstate();
}
```

Resources

http://adv-r.had.co.nz/C-interface.html