C++ / VBA interface

1 Solution

The purpose of this document is to describe the steps to obtain three projets in a Visual Studio 2010 solution, and to describe the C++ / VBA interface. These three projects will be:

- a project to create a static library, named **rational_stat**, which will contain the definition of class(es) and its (their) methods,
- a project to create an application, named **rational_app**, which is actually a test program which verifies if our class is working correctly,
- a project to create a dynamic library (DLL), named **rational_dll**, which will be used for the C++ / VBA interface.

The application and the DLL will use the code of the static library, which will contain the definition of the methods of our class(es). Hence, the code of the **rational_stat** project will not be *copied* in the projects **rational_app** and **rational_dll**, but instead it will be *used*.

On the hard disk, the solution will be in the folder "Mes Documents/Visual Studio/Projects". This folder will be called "project_folder" below. Here is the structure of the solution **Rational** (relevant suubfolders and files) that will be created on the hard disk in "project_folder". The italic files are those added in the following sections, the bold ones are those generated by the compilation of the solution.

- Rational/
  - Debug/
    - **rational_stat.lib**
    - **rational_app.exe**
    - **rational_dll.lib**
    - **rational_dll.dll**
  - rational_stat/
    - *rational.h*
    - *rational.cpp*
  - rational_app/
    - *rational_app.cpp*
  - rational_dlll/
    - *rational_dll.h*
    - *rational_dll.cpp*

1.1 Static library

To create the project of the static library :

- Fichier → Nouveau → Projet
- Choose “Win32” (on the left) and “Application Console Win32” (on the right)
- Set the name of the project in the bottom part : **rational_stat**
- Set the name of the solution (just below) : **Rational**
- Click on “OK”. In the new window, click on “Suivant” at the bottom, then, on the right, click on “Bibliothèque statique”. Uncheck “En-tête précompilé”.
- Click on “Terminer”

Now that the project is created, the files must be added.

1.1.1 Header file

To add a header file :

- In **rational_stat** (on the left), right click on “Fichiers d’en-tête”.
- Then click on “Ajouter” → “Nouvel Elément”.
- Choose on the right “Code” and “Fichier d’en-tête (.h)”
- Set the file name at the bottom : **rational.h**

We now fill the file **rational.h** :

```
#ifndef RATIONAL_H
#define RATIONAL_H

class Rat
{
private:
  int num_;
  int den_;

public:
  Rat(int num, int den = 1);
  double double_get();
};

#endif // RATIONAL_H
```

1.1.2 Source file

To add a source file :

- In **rational_stat** (on the left), right click on “Fichiers sources”.
- Then click on “Ajouter” → “Nouvel Elément”.
- Choose on the right “Code” and “Fichier source (.cpp)”
- Set the file name at the bottom : **rational.cpp**

We now fill the file **rational.cpp** :

```
#include "rational.h"

Rat::Rat(int num, int den)
{
  num_ = num;
  den_ = den;
}

double Rat::double_get()
{
  return (double)num_ / den_;
}
```

To compile and create the static library, press the F7 key. The file **rational_stat.lib** is located in the directory *project_folder/Rational/Debug*.

1.2 Application

To create the project of the application :

- Right click on the solution **Rational** on the left
- “Ajouter” → “Nouveau Projet”
- Choose “Win32” (on the left) and “Application Console Win32” (on the right)
- Set the name of the project in the bottom part : **rational_app**
- Click on “OK”. In the new window, click on “Suivant” at the bottom, then, on the right, select “Projet vide”
- Click on “Terminer”

1.2.1 Source file

Now that the project is created, the source file must be added:

- In **rational_app** (on the left), right click on “Fichiers source”.
- Then click on “Ajouter” → “Nouvel Elément”.
- Choose on the right “Code” and “Fichier source (.cpp)”
- Set the file name at the bottom : **rational_app.cpp**

We now fill the file **rational_app.cpp** :

```
#include <iostream>

#include  "rational.h"

int main()
{
  Rat r(1,5);

  std::cout << r.double_get() << std::endl;

  return 0;
}
```

1.2.2 Configuration

Now, in order to compile the project, we must configure it so that it uses the static library.

- Right click on **rational_app** (on the left) and click on “Propriétés”. A new window appears with, on the left, a set of options.
  - C/C++
    - Général
      - On the right, click on “Autres Répertoires Include”
      - Click on the black arrrow, then “Modifier...” and a new window appears
      - Click on the yellow folder icon then on “...”. Select the folder where the file **rational.h** is located (it should be *project_folder/Rational/rational_stat.*
      - Click on “OK”
  - Editeur de liens
    - Général
      - On the right, click on “Répertoires de bibliothèques supplémentaires”

        - Click on the black arrrow, then “Modifier...” and a new window appears
        - Enter *project_folder/Rational/Debug*.
      - Entrée
        - Click on “Dépendances supplémentaires”
        - Click on the black arrrow, then “Modifier...” and a new window appears
        - Enter **rational_stat.lib**
  - Click on “OK”
- Click on “OK”

To compile both projects, press the F7 key. pressing Ctrl-F5 will not execute this program, as the default project that will be executed is the static library (which is even not a binary...). To allow **rational_app** to be executed, right click on **rational_app** and select “Définir comme projet de démarrage”. Now press Ctrl-F5.

1.3 Dynamic library

To create the project of the DLL :

- Right click on the solution **Rational** on the left
- “Ajouter” $\rightarrow$ “Nouveau Projet”
- Choose “Win32” (on the left) and “Application Console Win32” (on the right)
- Set the name of the project in the bottom part : **rational_dll**
- Click on “OK”. In the new window, click on “Suivant” at the bottom, then, on the right, select “DLL” and select “Projet vide”
- Click on “Terminer”

The DLL will be used into Excel, via VBA. A header file and a source file need to be written.

1.3.1 Header file

First, the header file :

- In **rational_dll** (on the left), right click on “Fichiers d’en-tête”.
- Then click on “Ajouter” $\rightarrow$ “Nouvel Elément”.
- Choose on the right “Code” and “Fichier d’en-tête (.h)”
- Set the file name at the bottom : **rational_dll.h**

In order for VBA to call the functions, these ones must have the keyword **__stdcall** added just before their name and **__declspec(dllexport)** in the header file, that is, their signature must be of the form :

```
__declspec(dllexport) type_ret __stdcall fct_name(params)
```

and they must be written in a C style (no class, etc...).

Here is the header file :

```
#ifndef RATIONAL_DLL_H
#define RATIONAL_DLL_H

#define DLL_EXPORT __declspec(dllexport)

extern "C"
{
// add here all the functions that will be called by VBA

DLL_EXPORT double __stdcall rat_double_get(int num,
                                           int den);

}

#endif /* RATIONAL_DLL_H */
```

1.3.2 Source file

Second, the source file :

- In **rational_dll** (on the left), right click on “Fichiers source”.
- Then click on “Ajouter” → “Nouvel Elément”.
- Choose on the right “Code” and “Fichier source (.cpp)”
- Set the file name at the bottom : **rational_dll.cpp**

In order for VBA to call the functions, these ones must have the keyword **__stdcall** added just before their name in the source file, that is, their signature must be of the form :

```
type_ret __stdcall fct_name(params)
```

Usually, one has to use a class in the body of the function, hence it is quite common that the function takes as arguments the arguments of the constructor of the class. It is not mandatory, though.

Here is the source file that will be used :

```
#include <rational.h>
#include <rational_dll.h>

double __stdcall rat_double_get(int num, int den)
{
  Rat r(num, den);

  return r.double_get();
}
```

Now, the project must be configured exactly like in section 1.2.2, except that in *Editeur de liens→Entrées→Dépendances supplémentaires*, **rational_dll.lib** must also be added.

2 C++-VBA interface

To use C++ functions in Excel, VBA must be used. Actually, VBA can't call directly C++ functions, it can only call C functions. So one has to write C functions which call the C++ functions we want. That is the purpose of the DLL build in section 1.3.
Here are the steps to use that DLL in VBA:

- In Excel, open a new sheet and save it with the name **rational_dll.xls** for example.
- Press Ctrl-F11 to open VBA Editor.
- Add a module (right click on **VBA Project (rationall_dll.xls)**, then Insert → Module.
- At the beginning of the VBA program, one must write the functions to use in VBA, and the C function that the VBA one calls. The syntax is the following :

  ```
  Declare Function name_fct_vba "path_to_DLL" Alias _
   "_name_fct_c@n" (VBA parameters) type_ret_vba
  ```

  where:

  – *name_fct_vba* is the name of the function used in the VBA program
  – *path_to_DLL* is the name of the DLL with its full path
  – *_name_fct_c@n* is the name of the C function in the DLL, prepended with an undersore, and to which we append @n, where n is a number which value will be described later.
  – *VBA parameters* are the parameters of the C function, but with the VBA syntax.

- *type_ret_vba* is the returned type of the C function but with the VBA syntax
- the underscore '_' that is at the end of a line allows the command to be splitted into several lines.

Here is an example of VBA program using the **rational_dll.dll** DLL :

```
Declare Function rat_double_get_vba Lib _
"c:\path\to\rational_dll.dll" Alias "_rat_double_get@8" _
(ByVal n As Long, ByVal d As Long) As Double

Sub test()
  Feuil1.Cells(1, 1).Value = rat_double_get_vba(2,7)
End Sub
```

It is sufficient to launch the VBA macro in the Excel sheet.
As a final note, the computation of n is the following : it is the sum of the length of the type of each argument of the C function in byte. An **int** in C is 4 bytes long, so $n = 4 + 4 = 8$.