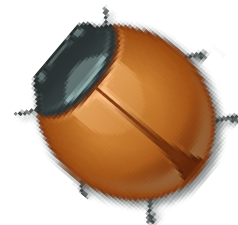


F2A Shell

Tutorial





How to run F2A shell in your PC	3
How to debug a own code using F2A shell	4
FirstTest.h	4
FirstTest.cpp	4
Trying to debug the FirstTest code.....	4
DLL creation	4
Modifying Template.h	5
Modifying Template.cpp	6
Making the DLL file	7
MinGW Stand-Alone	7
Eclipse	7
Loading the DLL file into the Shell	7



How to run F2A shell in your PC

The first step is to download the last version of F2A shell. Usually, the application is in a RAR file.

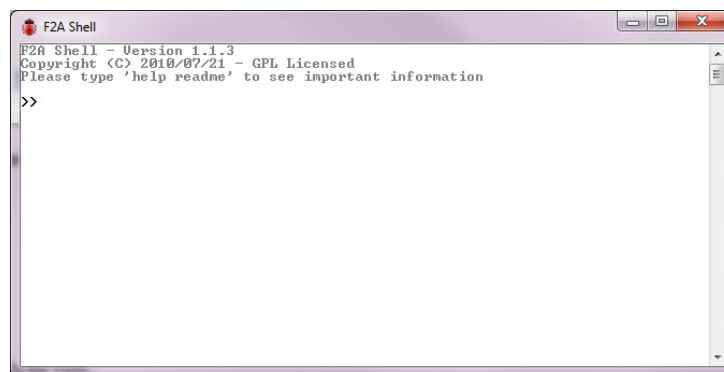
Extract the entire F2A shell's folder form RAR file into the folder assigned by you.

Name	Date modified	Type	Size
Dat	24/07/2010 09:50 ...	File folder	
Libs	24/07/2010 09:50 ...	File folder	
F2Ashell.exe	24/07/2010 09:53 ...	Application	299 KB
license.txt	21/07/2010 10:42 a...	Text Document	35 KB

F2A shell folder installed.

Well, F2A is ready to run.

Do double-click in F2Ashell.exe. The next image shows you how the windows must look.



F2A shell is running.

Now F2A shell is ready.



How to debug a own code using F2A shell

In order to explain how use **F2A shell** to debug a particular code, I'm going to develop the explanation through a small and easy code write in C++.

The first thing you have to be sure is that F2A shell is running in your PC.

My project to debug is named 'FirstTest' (you can download this project from the web).

FirstTest has two files to debug. They are: 'FirstTest.h' and 'FirstTest.cpp'

FirstTest.h

```
#ifndef FIRSTTEST_H_
#define FIRSTTEST_H_

#include <stdio.h>

void vTest();
int iAdd(int a, int b);

#endif
```

FirstTest.cpp

```
#include "FirstTest.h"

void vTest(){
    printf("This is working!\n");
}

int iAdd(int a, int b){
    return (a + b);
}
```

Trying to debug the FirstTest code

If you want to debug this code you must do a console code to insert the two parameters of **iAdd** function, to show the result of the **iAdd** function and to execute the **vTest** function.

You can choose F2A shell to execute the different function of your code. The only requirement is to do an interface between your code and F2A shell. The way to do that is creating a DLL and mounting it into the shell. This interface allows you relate a specific text command with each one of your functions.

You can download the **DLL template** to get an easy and fast way to develop the interface.

DLL creation

If it's your first time to create a DLL as interface between your code and F2A shell, follow the next steps, if not, you can add new text command into an old DLL created by you which execute the FirstTest's functions.

When you extract the file into the RAR file that contains DLL template's folders and files, you must to have this:



Name	Date modified	Type	Size
Debug	21/07/2010 12:18 ...	File folder	
headers	21/07/2010 12:17 ...	File folder	
lib	21/07/2010 12:17 ...	File folder	
sources	21/07/2010 12:17 ...	File folder	
license.txt	21/07/2010 10:42 a...	Text Document	35 KB
readme.txt	21/07/2010 09:40 a...	Text Document	2 KB

DLL template folder

In **Debug** there is the folder of the DLL created.

The folder **headers** contains all the headers (*.h) that you need.

The folder **lib** contains the file **libF2Ashell.a**, this file is a static library with the compiled code necessary.

The folder **sources** contains the file template.cpp. You can use this one to fill with the code that you need.

If you don't want to re-use the template code you must to create a new file, for example:

FirstTestInterface.cpp into sources folder and **FirstTestInterface.h** into headers folder.

Copy (merge if it's necessary) the **sources**, **headers** and **lib** folder in your project folder. (If you want to re-use the template code, copy **sources** folders as well).

Modifying Template.h

```
#ifndef TEMPLATE_H_
#define TEMPLATE_H_

#include <windows.h>
#include "ShellLink.h"
#include "../FirstTest.h"

void vShellPort_Test(ShellParamToPorts* params);
void vShellPort_Add(ShellParamToPorts* params);

#endif /* TEMPLATE_H_ */
```

The changes are:

1. Include the reference to the header file of your code.
2. Modify/Add the new 'Port Functions'. The Port Functions process the parameters that have been introduced by the shell to reach your functions. Here, we create two Port Functions, one for **vTest** function and other for **iAdd** function.



Modifying Template.cpp

```
#include    "../headers/template.h"
```

```

BOOL WINAPI __declspec(dllexport) FirstTestInterface (HINSTANCE hInst, DWORD
Reason, LPVOID Reserved){
    if(Reason==DLL_PROCESS_ATTACH){        return    TRUE; }
    if(Reason==DLL_PROCESS_DETACH){        return    TRUE; }
    return    FALSE;
}

extern "C"{
unsigned int __declspec (dllexport) LibRun(ShellParamToPorts* params){
    unsigned int        RunReturn;

    RunReturn = PortRunReturn_OK;           //The Command has been processed

    if (!strcmp(params->pcGetCmd(), "test"))    vShellPort_Test(params);
    else if (!strcmp(params->pcGetCmd(), "add")) vShellPort_Add(params);
    else RunReturn = PortRunReturn_UC;         //Unknown Command
    return RunReturn;
}

void        __declspec (dllexport) LibIni(ShellParamToPorts* params){ }

} //end extern "C"

void vShellPort_Test(ShellParamToPorts* params){
    vTest();
}

void vShellPort_Add(ShellParamToPorts* params){
    unsigned int        result;
    char                line[64];

    if(params->uiGetNbrArg() == 2){            //check the amount of arguments
        result = iAdd( atoi(params->pcGetArg(0)), atoi(params->pcGetArg(1))
);
        sprintf(line,"%d + %d = %d", atoi(params->pcGetArg(0)), atoi(params-
>pcGetArg(1)), result);
        params->vMessage(line, "message");
    }
    else{
        params->vMessage("Format Error:", "error");
        params->vMessage("Add param1 param2", "info");
        params->vMessage("Result = param1 + param2", "info");
    }
}
}

```

The changes are:

1. The name of the first function was changed to **FirstTestInterface**.
2. In the function **LibRun** I've introduced the new commands: **test** and **add**. Then, I associate each command to a specific Port Function, **vShellPort_Test** and **vShellPort_Add** respectively.
3. In the Port Function **vShellPort_Test** I've introduced the first function to test: **vTest**. This function doesn't need parameters, then it's doesn't matter to check them.



4. In the Port Function `vShellPort_Add` I've introduced the second function to test: **iAdd**. The result of this function is put into **result** variable. The different messages to show have been modified according to the **iAdd** function.

Making the DLL file

There is one more step to reach our DLL file; this is linking the static library.

MinGW Stand-Alone

If you are compiling and linking with MinGW you have to do the next thing at the end of your process of compiling (at this point you should to have the **main.o**, **FirstTest.o** and **template.o** files):

```
g++ -shared -olibFirstTest.dll sources\template.o main.o FirstTest.o
..\lib\libF2Ashell.a
```

Eclipse




If you are using the tool Eclipse to do all the programming, compiling and linking process, this explanation will be useful for you.

1. First, do right-button click on **FirstTest** project folder.
2. Go to **Properties**.
3. Now, expand **C/C++ Build** and select **Settings**.
4. In the right side of the windows look for **Tool Settings** label.
5. Expand **MinGW C++ Linker** and select **Libraries**.
6. Click on **Add** button in the sub windows **Library search path (-L)** and select the lib folder in your workspace (or you can copy and paste this: `"${workspace_loc:/FirstTest/lib}"`, check it before).
7. Click on **Add** button in the sub windows **Libraries (-l)** and write **F2Ashell**.
8. Press OK button and then build your entire project.

Loading the DLL file into the Shell

Now, you have your DLL interface file: **libFirstTest.dll** (you can change the name if you want). The next step is loading it into the shell workspace.

To do this, you must to copy the DLL file into **Libs** folder in **F2Ashell** application folder (all this is a recommendation, because you can load a library from a different folder). You should to see this:

Name	Date modified	Type	Size
 libFirstTest.dll	25/07/2010 01:06 a...	Application extens...	1.466 KB
 ShellPort_system.dll	18/07/2010 06:41 ...	Application extens...	1.465 KB
 ShellPort_test.dll	18/07/2010 04:51 ...	Application extens...	1.465 KB

Libs folder after you have copied libFirstTest.dll

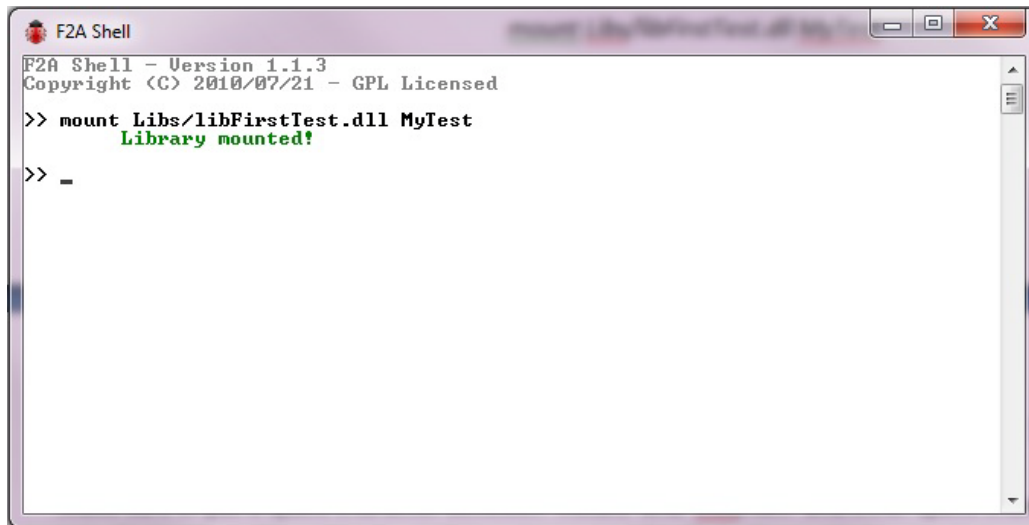
Then, run **F2A shell**.

Execute the next command:

```
mount Libs/libFirstTest.dll MyTest
```



Make sure of put a space character between 'mount' and 'Libs/lib...' and other space character between 'Libs/libFirstTest.dll' and 'MyTest'. This action mount (load) the library that you've just created into **MyTest** domain.



```
F2A Shell - Version 1.1.3
Copyright (C) 2010/07/21 - GPL Licensed

>> mount Libs/libFirstTest.dll MyTest
      Library mounted!

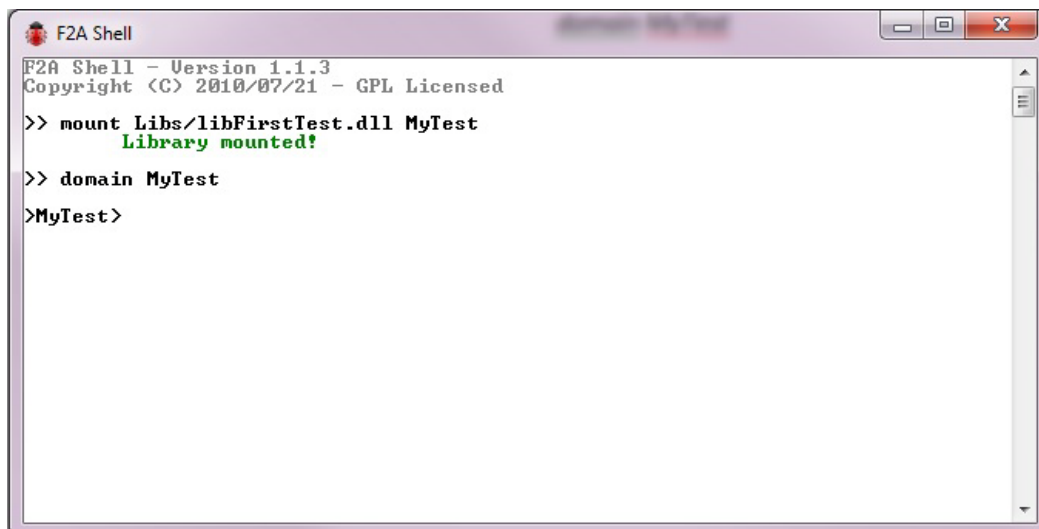
>> _
```

DLL file loaded into the shell.

Go to MyTest domain executing the next command:

domain MyTest

Make sure of put a space character between 'domain' and 'MyTest'.



```
F2A Shell - Version 1.1.3
Copyright (C) 2010/07/21 - GPL Licensed

>> mount Libs/libFirstTest.dll MyTest
      Library mounted!

>> domain MyTest
>MyTest>
```

MyTest domain selected.

Now, you are ready to run your function. Remember that the command **test** executes the **vTest** function of your library and the command **add [param 1] [param 2]** executes the **iAdd** function of your library. Run these commands in the shell.



```
F2A Shell
F2A Shell - Version 1.1.3
Copyright (C) 2010/07/21 - GPL Licensed

>> mount Libs/libFirstTest.dll MyTest
      Library mounted!

>> domain MyTest

>MyTest> test
This is working!

>MyTest> add 2 3
      2 + 3 = 5

>MyTest> _
```

The commands **test** and **add** executed.

You can see the feedbacks of your functions in a grey text below each command executed.

Your functions have been tested by now, but you can finish testing the DLL interface. On the one hand, the first command executed (test) is independent of the number of parameters, so you can put anything else after the command and it has to work in the same way. On the other hand, the command add only allows two parameters, so you can try to execute it with a different among of parameter or different type of them.

```
F2A Shell

>MyTest> test 6 :> ... #^@#~?~_
This is working!

>MyTest> add 3
      Format Error:
      Add param1 param2
      Result = param1 + param2

>MyTest> add 6 8 7
      Format Error:
      Add param1 param2
      Result = param1 + param2

>MyTest> add 5 hj
      5 + 0 = 5

>MyTest> add
      Format Error:
      Add param1 param2
      Result = param1 + param2

>MyTest>
```

Commands executed with different among and types of parameters.

At least, execute **exit** command to finish.