Subject: Re: Calling windows not in main.cpp
Posted by mr_ped on Mon, 23 Nov 2009 08:56:19 GMT
View Forum Message <> Reply to Message

You have to have already defined what is "splashScreen" here:
main.cpp: "splashScreen().Run();"

You can use Build / Preprocess (Alt+F7) to see what the compiler is trying to compile after he resolves the includes, you will see there's nothing about splashScreen definition in BrainBox.h.

Common C++ way of doing things is to have: (I'm writing it by hand, so sorry for any typos) (ignore all the "bar" things firstly before you get the idea how foo works, then re-read and try to resolve "bar")

foo.h:

```
class bar;   //a trick how to use class "bar" in other header without it's actual definition, see a bit down

class foo
{
protected:   //next definitions will be visible only inside the class and derived classes
  int x;     //variable definition
  bar *pointer_to_class_bar;   //you can use pointer even without actual definition of "bar" being known here

public:      //next definitions will be visible outside of the class
  foo() : pointer_to_class_bar(NULL)
  {
    //this is constructor - with (empty) code defined HERE */

    //pointer_to_class_bar->DoSomething();
    //^^ this would produce error, because compiler doesn't know here the bar has method "DoSomething"
  }

  int fwd_foo( int );  //this is forward definition of function, without actual code
};
```

foo.cpp:

```
#include "bar.h"   //in this file we will need to know bar class,
                   //so include it's full definition (similar file to foo.h)
#include "foo.h"   //include public definitions of foo class

//here comes the rest of foo implementation, like the missing body of int fwd_foo( int ) method
```

```
int foo::fwd_foo( int )  //notice how the name of method is preceded by class name
{
  if ( pointer_to_class_bar != NULL ) pointer_to_class_bar->SomeMethodOfBar();
  //^^ here it will work
  return 7;
}
```

other.cpp:

```
#include "foo.h"   //we want to use foo class

...  inside some function like main for example ...
{
  foo Foo;   //create instance of class foo at function stack memory
  int y = Foo.fwd_foo( 0 );
  //^^ compiler knows how to call this method, because it was forward-defined in the "foo.h", and
we did include that file
  //int my_x = Foo.x;
  //^^ error: while compiler knows the Foo has "x", you can't access it directly like this,
  //it's protected only for usage from withing Foo class family

  //bar Bar;  //this will create error, because compiler doesn't know full definition of class bar
  bar * another_bar_pointer;  //but this will work, because it was also forward-defined in foo.h,
  //but the usage is very limited, like only passing around the pointer value
  /* notice we don't need to include full bar.h to do that,
     so we will save compile time and when you edit bar.h,
     this file does not need to recompile, only files which do include full bar.h,
     like foo.cpp
  */
}
```

The basic idea is, that C++ compiler has to know EVERYTHING you are using in the file which is being compiled. It will not look at other files (unless you direct him by #include).
This is different from Java, where you only write the code parts (like .cpp files in C++), and you then do "import foo" and the compiler will take a look at foo and figure it out.
C++ is dumber in this aspect, the compiler has only single file during compilation (the one which you can see after Alt+F7), so what you use there, must be defined already there.

In your code you define the splashscreen in SplashScreen.cpp, but the main.cpp has no idea what you are talking about, it would have to include the .cpp file to see it (of course that's not the proper solution, the proper solution is to define the needed bits in some header file, and this header file include in both main.cpp and SplashScreen.cpp, and everywhere else where you want to work with splashscreen struct (as you are deriving it from TopWindow, I would rather suggest to make it "class", unless you have very good reasons to have it as struct (I can't think of any)).

Try to search some C++ language tutorial about definitions of classes (.h files) and

implementation (.cpp) to make sure you fully understand that concept. It's not difficult at all, once you get the idea, it's very natural. (natural in "computer-dumb" way, the Java is natural in "human-smart" way in this aspect)