Hello. Welcome to CalculatorLab++!

CalculatorLab++ illustrates that you can integrate C++, Objective C and Interface Builder code together into one program. You may want to do this if you wish to leverage C++ which already exists at your site. This example is remotely based on another one named "CalculatorLab" by Randy Nelson. You should be familiar with CalculatorLab before tackling this one.

CalculatorLab++ also serves as an example for error handling in NeXTStep.

We created the user interface using Interface Builder. In that interface specification we created an object called "SimpleCalcInstance". This object is the "dispatch center" for the program. It receives notification when the user presses any of the keys in the calculator window and hands that message off to the C++ object "CalcEngine". We chose to implement in in this way because it's most efficient -- we can take advantage of Interface Builder's powerful capabilities, and leverage our C++ code. The "SimpleCalcInstance" is necessary because Interface Builder only supports Objective C classes.

One other way to achieve the same effect and eliminate the need for "SimpleCalcInstance" is to create the interface directly using Objective C code instead of the nib file, and integrate the C++ objects into that. With an example this trivial, this is a feasible solution as the interface is neither large nor complicated. However, in a real-world situation, the time saved by using Interface Builder is significant.

Steps to integrating C++, Objective C and Interface Builder. For details concerning syntax etc. check the source code.

- Create the interface using interface builder. Make sure that you have a ªdispatch objectº

- Fill in the code for the ªdispatch objectº
- Add the new .m and .h files to the project in Project Builder.
- Create a Makefile.preamble that uses the ObjC++ compiler option to compile the source files as Objective-C++ files.
- Now that you are using a C++ compiler you must tell the compiler when you are using Objective C. Add the extern declarations to the _main.m and any other .m files. Check the source on how to do that. NOTE: Project Builder generates _main.m, so be careful not to let it overwrite the file.
- Remove the "void" declaration in front of the main routine in the _main.m file.
- Compile.

Yep. It's as easy as that!

Many thanks to Steve Naroff for advice and code!

modified for 3.0 (Warp10o) - mai
Note: A current bug in Project Builder prevents us from using other extensions such as .cc .C or .cxx to recognize C++ files. Hence, all files are named .m files for simplicity even though they could be C++ files.