Visual C++ 6.0 Quick Guide

Preliminaries

- Install Visual Studio (which contains Visual C++) from the CD. Ignore the prompt for MSDN, since we do not have the MSDN CDs (Use the on-line MSDN instead).
- After installing the program, be sure to download and install Service Pack 6.
- Select `Tools.Customize.(Toolbars)`. Check `Build` and uncheck `Build MiniBar`. The `Build` bar gives you a nice dropdown box that displays whether you are in Debug or Release mode.
- The model:
    - A *project* is a set of source code that compiles to an executable or library. The .dsp file is essentially a makefile.
    - A *workspace* (.dsw file) contains one or more projects. There is always at least one workspace, but the simplest configuration of a single workspace and a single project works pretty well.
    - Source code is contained in .h, .c, and .cpp files. The C compiler is automatically invoked for .c files, while the C++ compiler is invoked for .cpp files.
    - Object code and executables are contained in the Debug and/or Release directories, which are created automatically.
    - The resource file .rc contains resources, and is primarily useful for GUI-based programs.
    - Ignore (but leave) the StdAfx.[h,cpp] files, which are used for precompiled headers.
    - Files such as .clw, .ncb, .aps, and .opt can be ignored and/or deleted. They are automatically generated by the VC++ program.

Create a New Program

There are two main types of executables: and Windows-based applications.

- To create a console-based application,
    - `File.New.(Projects).Win32 Console Application.` Type a name for the project, and specify a location (directory) on the hard disk. A folder with the specified name will be created in the specified directory, and files will be created in the folder.
    - Select "`An Application that supports MFC`". This choice is recommended, because it gives you access to MFC if you ever decide later that you need it. MFC stands for Microsoft Foundation Classes, which are C++ wrappers around the Win32 API that make it much easier to program on Windows.
- To create a windows-based application,
    - `File.New.(Projects).MFC AppWizard (exe)`. Type a name for the project, and specify a location (directory) on the hard disk.
    - Select `Dialog based`. (The other options are useful for creating end-user applications, but they are significantly more complicated.)

    - Uncheck `ActiveX Controls`.
    - As before, a folder with the specified name is created in the specified directory, and files are created in the folder.
- In either case, precompiled headers are an unnecessary headache. To turn them off, select `Project.Settings.(C/C++)`. Choose "`Precompiled Headers`" in the drop-down box. Choose "`All Configurations`" in the left drop-down box. Select "`Not using precompiled headers`."

<u>Editing Program</u>

- For console-based application, ignore the MFC stuff. Put all of your code into the else clause.
- For windows-based application, drag widgets onto dialog. To create a callback, click on the widget and select `View.Class Wizard`. Click the desired Message (e.g., `BN_CLICKED`), `Add Function`, `Edit Code`. Type code in callback function in `ProjectDlg.cpp` file, where `Project` is the name of the project.
- To add an existing file to the project, right-click on the project name in the `FileView` and select `Add Files to Project...` and then select the filename. To create a new file, select `File.New.(Files).C++ Source File` (or `C/C++ Header File`), select the project, and enter the filename and location. Alternatively, click the `New Text File` icon, save the file, then add the file to the project.

<u>Running Program</u>

- To compile, select `Build.Compile`.
- To run, `Build.Execute` (which is the same as double-clicking the .exe file in the Debug or Release folder). To run with the debugger attached, `Build.Start Debug.Go`.
- While debugging, you can view the variables, call stack, memory, registers, etc. using `View.Debug Windows`. In the call stack, double-click on line to jump to the code. There are two windows for viewing variables: The `Variables` window gives you automatic access to commonly-used variables using `Auto`, `Locals`, and `this` tabs; The `Watch` window allows you to select and modify the variables to view.
- In the `Watch` window, names can be dragged from the `Variables` window or typed by hand. Append `,n` to the name (where n is an integer) to view several items in an array. Append `,x` to the name to convert the number to hexadecimal. The Watch window also accepts dereferencing (`*`), indexing (`[]`), and arithmetic operations (`+`, `-`, `*`, `/`). For example, if `p` is an array of integers, then `p,5` lists the first five values, `p[3]` lists the fourth value, and `p[1]+p[3]` adds two of the values.

<u>Keyboard Shortcuts</u>

- ESC Puts cursor in editor

- Alt+0 Puts cursor in Workspace window (containing FileView, etc.)
- Ctrl+PageUp/Down Cycles between ClassView, ResourceView, and FileView when cursor is in Workspace window
- Ctrl+F Find (in current file)
- F3 Find next
- Ctrl+I Incremental search. Type string, then Ctrl+I to find next.
- Ctrl+Shift+F Find in files.
  (This keyboard shortcut is not standard. To activate: `Tools.Customize.(Keyboard).(Category:Edit).(Commands:FindInFiles)`, type `Ctrl+Shift+F`, `Assign`.)
- F7 Compile
- Ctrl+F7 Execute
- Ctrl+F2 Toggle bookmark
- F2 Go to next bookmark
- F4 Jump to next warning or error (Shift+F4 jumps back)
- F9 Toggle breakpoint (for debugger)
- F5 Execute with debugger attached. (Only allowed in Debug mode.)
- F10 (Debugger) Perform next instruction
- F11 (Debugger) Perform next instruction, stepping into it
- Ctrl+F10 (Debugger) Run to cursor
- Shift+F5 (Debugger) Exit debugger
- Ctrl+Shift+8 Toggle view whitespace (spaces are dots, tabs are >>)
- Ctrl+K Next `#endif`
- Ctrl+J Previous `#ifdef`
- `Alt+F12` Go to definition of constant under cursor (e.g., `NULL`)
  (only works if you have built project with Browse information)