

# USING A HEADER FILE FOR USER DEFINED SPECIFIC TASK FUNCTIONS\*

Kenneth Leroy Busbee

This work is produced by OpenStax-CNXX and licensed under the  
Creative Commons Attribution License 2.0<sup>†</sup>

## Abstract

Concepts and an example of how to create a user library within the C++ programming language.

## 1 Concept: User Defined Specific Task Functions

Most companies have certain tasks that are unique to their company. Collectively the programming staff may decide to build several functions and organize them into one or more user libraries. Specific task functions are often built using a testing shell program. The sole purpose of the testing shell program is to create the specific task functions and to test them to insure that they are working properly. Think of a clam, its shell surrounds the important part, the pearl. A testing shell program surrounds the specific task function (the important part). Usually the testing shell program will be used to create several functions that will be placed into a user defined library. The process flows as follows:

1. The **testing shell** program with the specific task functions is built and thoroughly tested.
2. A copy of the **testing shell** source code is saved as the **header file** that once modified will be placed in the user library. You delete the main part of the program leaving a comments area, any needed include file references and the specific task functions.

---

\*Version 1.7: Mar 17, 2010 4:11 pm -0500

<sup>†</sup><http://creativecommons.org/licenses/by/2.0/>

## Header File Creation

Monitor_Testing_Shell.cpp	udst_monitor.h
Program Comments	Program Comments <b>Edit as appropriate</b>
Headers & Other	<del>Headers &amp; Other</del> <b>Delete all but needed libraries. Keep: #include &lt;ctime&gt;</b>
Prototypes	Prototypes
Variables	Variables
main	<del>main</del>
clear_m	clear_m
delay_m	delay_m <b>the clock function needs: ctime</b>
pause_m	pause_m

**Figure 1:** Creating a header file from a copy of the testing shell.

- Another copy of the **testing shell** source code is saved as the **prototypes file**. This is a text file that retains only the prototypes for the functions that were placed into the **header file**. The functions should be using meaningful identifier names, thus the prototypes should provide adequate information to others on how to call the function with appropriate parameter passing.

## Prototypes File Creation



**Figure 2:** Creating a prototypes file from a copy of the testing shell.

- Another copy of the **testing shell** source code is saved as the **verify header program**. You delete the functions prototypes and definitions then provide an include that points to the header file. This program is compiled and run to make sure the **header file** is working properly.

## Verify Header File Creation

<code>Monitor_Testing_Shell.cpp</code>	<code>Monitor_Verify_Header.cpp</code>
<b>Program Comments</b>	<b>Program Comments</b> <b>Edit as appropriate</b>
<b>Headers &amp; Other</b>	<b>Headers &amp; Other</b> <b>Add user libraries as needed:</b> <b>#include "c:\\Dev-Cpp\\user_library\\monitor.h"</b>
<b>Prototypes</b>	<b>Prototypes</b>
<b>Variables</b>	<b>Variables</b>
<code>main</code>	<code>main</code>
<code>clear_m</code>	<code>clear_m</code>
<code>delay_m</code>	<code>delay_m</code>
<code>pause_m</code>	<code>pause_m</code>

**Figure 3:** Creating a verify header file from a copy of the testing shell.

A good way to understand the concept is to review the four files described above that have been created by a programmer. We will be using the C++ programming language, however the code is easy to understand and will serve our needs well at explaining the concepts; even if you are not familiar with C++.

## 2 Demonstration Using C++

### 2.1 Creating a Folder or Sub-Folder for the Four Files

Depending on your compiler/IDE, you should decide where to download and store source code files for processing. Prudence dictates that you create these folders as needed prior to downloading source code files. A suggested sub-folder for the **Bloodshed Dev-C++ 5 compiler/IDE** might be named:

- `Monitor_Header`

If you have not done so, please create the folder(s) and/or sub-folder(s) as appropriate.

### 2.2 Download the Four Files

Download and store the following files to your storage device in the appropriate folder. You may need to right click on some of the links and select "Save Target As" in order to download some of the files.

- Download from Connexions: Monitor\_Testing\_Shell.cpp<sup>1</sup>
- Download from Connexions: udst\_monitor.h<sup>2</sup>
- Download from Connexions: udst\_monitor\_prototypes.txt<sup>3</sup>
- Download from Connexions: Monitor\_Verify\_Header.cpp<sup>4</sup>

### 2.3 Study the Files Collectively to Understand the Concepts

Take a few moments to review the files in conjunction with the concept discussion above. You should compile and run the **Monitor\_Testing\_Shell.cpp** program.

### 2.4 Creating a Folder or Sub-Folder for your User Library

Depending on your compiler/IDE, you should decide where to create a folder that will hold the header files you create. We suggest that you create the folder in conjunction with the compiler/IDE software. If you were using the **Bloodshed Dev-C++ 5 compiler/IDE** you most likely installed the compiler/IDE software at: **C:\Dev-Cpp\** if you installed it on your machine or at: **DriveLetter:\Dev-Cpp\** (where the **DriveLetter** is the drive that represents your flash drive) if you installed it on a flash drive. We suggest that you create a sub-folder at that location named:

- user\_library

The path of: **C:\Dev-Cpp\user\_library** would be created as the location for your user library if using your machine installation. You can literally place it anywhere and name the library any name, but once you decide on a place and name; you do not want to move or rename the folders.

### 2.5 Placing the Header File into the User Library

You need to copy the **udst\_monitor.h** file placing it into the **user\_library** folder just created. As you can guess the **udst** stands for user defined specific task. The functions within this header file would be used to control the interaction a user has with the monitor. The **.h** is a convention of the C++ programming language and indicates a header file. Thus the identifier name for the header file is very meaningful and descriptive.

### 2.6 Verify that the Header File Works Properly

Review the **Monitor\_Verify\_Header.cpp** source code file and note the two include commands are different.

1. The Standard Library uses a less than and a greater than to bracket the Standard Library name of: `iostream`
2. The user library uses quote marks to bracket the location of the header file. This identifies to the compiler that we are specifying the exact file we want. We provide a complete file specification (drive, path information, filename and extension).
3. Because this item is technically a string within C++, we must use two back slashes between the drive, path(s) and filename. This is because the first back slash assumes that the next character is an escape code and if we really don't want an escape code but a back slash, the second back slash says no I wanted a back slash. This string: "C:\\Dev-Cpp\\user\_library\\udst\_monitor.h" will be interpreted to mean: **C:\Dev-Cpp\user\_library\udst\_monitor.h**

<sup>1</sup>See the file at <[http://cnx.org/content/m19346/latest/Monitor\\_Testing\\_Shell.cpp](http://cnx.org/content/m19346/latest/Monitor_Testing_Shell.cpp)>

<sup>2</sup>See the file at <[http://cnx.org/content/m19346/latest/udst\\_monitor.h](http://cnx.org/content/m19346/latest/udst_monitor.h)>

<sup>3</sup>See the file at <[http://cnx.org/content/m19346/latest/udst\\_monitor\\_prototypes.txt](http://cnx.org/content/m19346/latest/udst_monitor_prototypes.txt)>

<sup>4</sup>See the file at <[http://cnx.org/content/m19346/latest/Monitor\\_Verify\\_Header.cpp](http://cnx.org/content/m19346/latest/Monitor_Verify_Header.cpp)>

Depending on what drive you are using, what path folder structure you are using and what you called your folder; you may need to correct the include reference within the source code so that it properly references the header file.

Compile and run the Monitor\_Verify\_Header.cpp program. Note: It should work exactly as the Monitor\_Testing\_Shell.cpp program.

### 3 Definitions

**Definition 1: udst**

User Defined Specific Task

**Definition 2: testing shell**

A program used to create specific task functions.

**Definition 3: header file**

A file that contains items we want to have included toward the top of our source code.