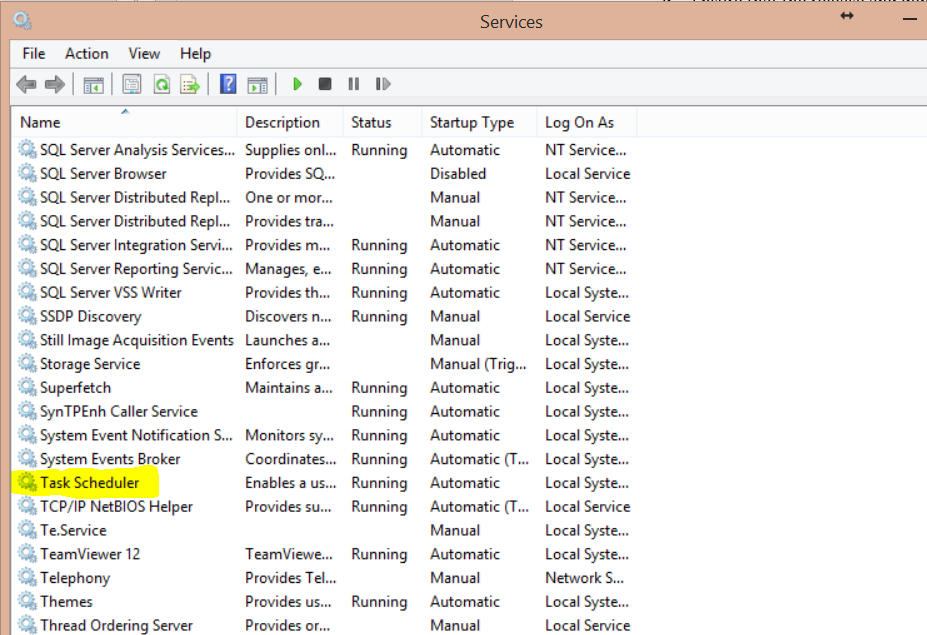
# Solution Summary

## Implementation Facts/Decisions/Assumptions

* I wanted to illustrate other aspects of coding that are important to me – debug logging and unit testing.
  + Good debug logging is important when supporting code running in the field; it can save the day. I always attempt to log errors/etc. And I attempt to log detailed debug-level info to aid in field diagnosis.
  + In the past, for unit testing C/C++, I’ve used CppUnit and CppTest. I like CppTest better than CppUnit for things like memory leak checking, etc. But I used CppUnit for this coding exercise.
  + Since the initial requirements didn’t specify debug logging, I’m assuming that debug logging was not necessary for this coding exercise.
    - However, I implemented a debug logging hack just so I could feel comfortable with some level of debug logging.
    - In the code, where I’d normally log a message, I originally outputted the message to stdout and added a TODO for the debug logging. This made the code noisy. So, I wrote a utility log function to wrap that all up to make the code a bit cleaner.
* As directed, I used a git repo to manage code changes. The GitHub repo is <https://github.com/dougbeyer/ComScheduledTask.git> .
* For a very long time now, I’ve used a separate code editor for my editing. It has very powerful template facilities. Every time I start using a new language, I build up extensive templates for that language. I attempt to put “best practices”, etc directly into the templates. C/C++ is no exception. So, when you see things like source code file layout, class layout, template typdefs, etc, that’s all coming from my templates. I don’t have to think about syntax and I certainly don’t type out every line of code. Enter a few characters, hit the space bar, and voila, there’s my file, statement, etc.
* I chose to use TCHAR even though current MSDN reading says to always use widechar. But my templates all use TCHAR. So rather than worry about it, I just continued to use TCHAR. Of course, for production code, I’d use whatever standards Cofense has in place.
* I didn’t spend any time worrying about extensive use of namespaces. I just used the same namespace for all classes.
* Note that, in general, I tend to prefer that my functions return error codes versus exceptions. It’s easy to extend the range of info they return simply by defining more error codes. But, at the same time, I’m inclined to write my functions first using a bool return value to indicate success/failure; it’s easier initially compared to defining return codes. I then refactor as needed if I later determine that a function needs to return more than two possible values.
* I am an advocate of using goto’s within a single function for the sole purpose of avoiding deeply-nested IFs. My goto’s *only* drop down; no weird cases of jumping all over the place.
* I originally started with a scheduled task that launched Notepad.exe. Then, for my unit tests, I wrote code to enum processes looking for instances with a specific name (e.g. Notepad.exe). But then I realized that the CppUnit solution I had didn’t define an x64 build configuration. Therefore, I couldn’t build an x64 version of my unit test app. That was necessary to check for instances of Notepad.exe that might exist prior to my unit test runs. So, I did some minor refactoring to the code to create a scheduled task that would copy a file. This made it easier to unit test.

## Running the code

1. Prepare CppUnit:
   1. Download a recent version.
   2. Define the CPP\_UNIT\_HOME environment variable as described in the CppUnit documentation.
   3. Build x86 release and debug versions of the CppUnit dll.
   4. Ensure that the release and debug versions of the CppUnit DLL (i.e. cppunit\_dll.dll and cppunitd\_dll.dll) and the corresponding import libraries (i.e. cppunit\_dll.lib and cppunitd\_dll.lib) are in the $(CPP\_UNIT\_HOME)\lib directory. If not, copy them from the respective output directories.
2. Clone my repo.
3. Build my coding exercise solution using VStudio 2015. I have Update 3 at home.
4. Verify that the Task Scheduler service is running.



1. If you run ComScheduledTask.exe, it will run a scheduled task to launch Notepad.exe as soon as the task is registered. The app will pause after launching Notepad to give you the chance to view the scheduled task in Windows, etc. Hit enter, the app will continue by deleting the scheduled task, and then pausing again before existing. Hit enter again to exit.
   1. You can view the scheduled task using the Windows Task Scheduler GUI. Hit Start Menu, type “task s”, and select the option entitled “Schedule tasks”. My task is registered in the root folder.
2. UnitTests.exe uses a scheduled task that copies a file. The file and the associated Windows .bat file get copied to C:\ProgramData\TaskSchedulerUtilTests by the UnitTests VStudio project’s post-build steps. The unit tests do not delete that directory; so there is some garbage left on the machine.