Borst Automation Embedded Solutions

Walter Borst
Kapitaen-Alexander-Strasse 39
27472 Cuxhaven, GERMANY

Fon: +49 (0) 4721 6985100
E-Mail: walter.borst@borst-automation.de
Home: https://www.borst-automation.de/

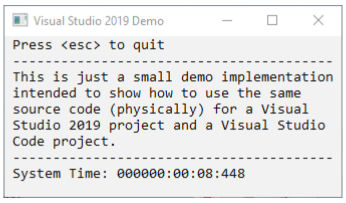
Technical Data Sheet

One for Two SCM

Overview

Visual Studio 2019 and Visual Studio Code are probably the two most important development environments at Microsoft at the moment. This also connects to the topic of reusable code.

The present source code module therefore first answers the question of what an environment can look like in which the same source code is accessed in both Visual Studio Code and Visual Studio 2019.



```
Press <esc> to quit
-------------------------------------
This is just a small demo implementation
intended to show how to use the same
source code (physically) for a Visual
Studio 2019 project and a Visual Studio
Code project.
-------------------------------------
System Time: 000000:00:08:448
```

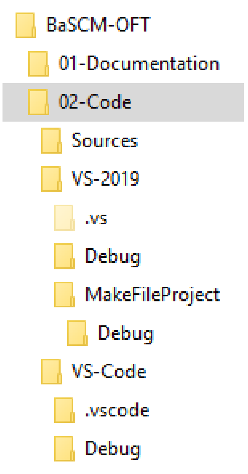
As an application, I chose a simple console implementation, which is coded in C++ like everything else in this SCM. The timer (in ms) that is displayed runs in a separate thread.

Actually, the task is very simple. But as is so often the case, it turns out that it is not quite as easy as initially thought.

However, the principles on which the IDE Visual Studio 2019 is based and the principles on which the source code editor Visual Studio Code was constructed are very different.

The compilers used for the build are cl.exe from Visual Studio 2019 environment and g++.exe from GNU package MinGw.

Directory Structure



The source code for the application is only in the 'Sources' directory. This directory does not contain any files related to either development environment.

Besides that, there are two sections: 'VS-2019' and 'VS Code'.

The solution file for VS 2019 is located directly in the path 'VS-2019'. In addition, I have added a make project, which is located in the 'MakeFileProject' folder.

The make file project in the VS 2019 environment uses the MinGw compiler g++.exe.

As usual, the JSON files for the VS Code project are housed in the .vscode area. The workspace is accessed via a file (OneForTwoSCM.code-workspace) also located there.

Prerequisites

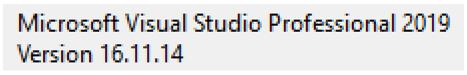
The 'normal' project under VS 2019 can be used immediately. However, a GNU compiler is required for the make projects. It's called MinGw and can be installed and configured via the following link:

https://code.visualstudio.com/docs/cpp/config-mingw

It is necessary to follow the instructions in said document scrupulously to have a complete development environment.

Visual Studio 2019

I used the following version of Visual Studio 2019:

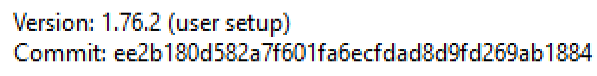


Microsoft Visual Studio Professional 2019
Version 16.11.14

Further installations for VS 2019 are not required.

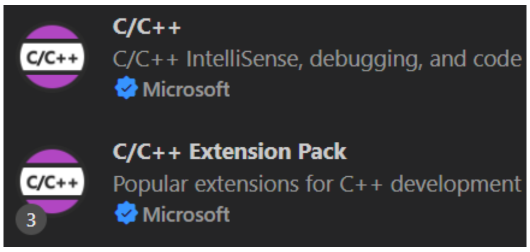
Visual Studio Code

I used the following version of Visual Studio 2019:



Version: 1.76.2 (user setup)
Commit: ee2b180d582a7f601fa6ecfdad8d9fd269ab1884

Only two extensions are required:



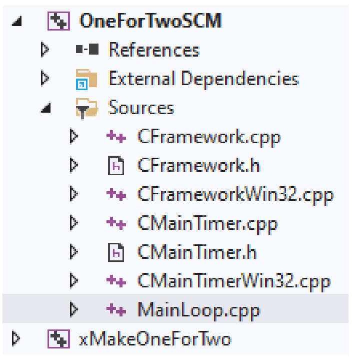
C/C++
C/C++ IntelliSense, debugging, and code
Microsoft

C/C++ Extension Pack
Popular extensions for C++ development
Microsoft

Getting Started

Visual Studio 2019

Open the solution:
.\BaSCM-OFT\02-Code\VS-2019\OneForTwoSCM.sln



OneForTwoSCM
References
External Dependencies
Sources
CFramework.cpp
CFramework.h
CFrameworkWin32.cpp
CMainTimer.cpp
CMainTimer.h
CMainTimerWin32.cpp
MainLoop.cpp
xMakeOneForTwo

The source code for the application is only in the 'Sources' directory. This directory does not contain any files related to either development environment.

Besides that, there are two sections: 'VS-2019' and 'VS Code'.

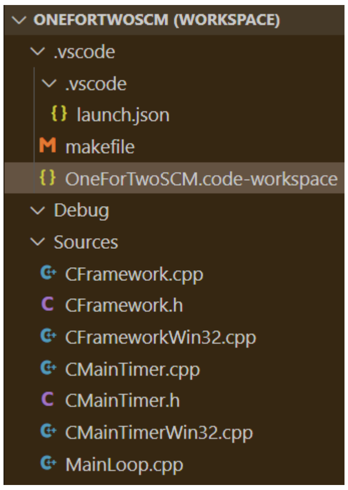
The solution file for VS 2019 is located directly in the path 'VS-2019'. In addition, I have added a make project, which is located in the 'MakeFileProject' folder.

The make file project in the VS 2019 environment uses the MinGw compiler g++.exe.

Visual Studio Code

As usual, the JSON files for the VS Code project are housed in the .vscode area. The workspace is accessed via a file (OneForTwoSCM.code-workspace) also located there.

Open the file OneForTwoSCM.code-workspace.



```
ONEFORTWOSCM (WORKSPACE)
  .vscode
    .vscode
      launch.json
    makefile
    OneForTwoSCM.code-workspace
  Debug
  Sources
    CFramework.cpp
    CFramework.h
    CFrameworkWin32.cpp
    CMainTimer.cpp
    CMainTimer.h
    CMainTimerWin32.cpp
    MainLoop.cpp
```

Only two JSON files are needed to accommodate the definitions. Launch.json is there to configure the debugger, while the OneForTwoSCM.code-workspace file is used to specify the possible paths and to describe one task for the build and one for the clean command.

Since the file ‘OneForTwoSCM.code-workspace’ is on the root of the workspace, a second directory '.vscode' was required because the debugger looks for the settings in the ‘.vscode’ folder within the root directory of the workspace to retrieve the full path of the program that is to be started for debugging.

As with the corresponding project in Visual Studio 2019, a makefile is required here. It is a second makefile because the relative paths to the sources and the debug area are different.

Details

Two of the source code files are marked with the 'Win32' qualifier. These are files that specifically serve the Windows environment. Actually, this also applies to the MainLoop.cpp. But since the internal functions are abstracted enough, it wouldn't make that much of a difference.

Initialization and Startup

```
// Entry point of the application
int main()
{
    char            in_key = 0x00;
    uint32_t       sys_time = 0;
    uint8_t           count = 50;

    // Start the timer task
    if (CFramework::CTask::Start((TY_char*)"TimerTask",
      (void (*)(void*))executeTimerTask,
      NULL, &m_task_ctr_timer) == CError::NONE)
    {
        m_timer_task_started = TRUE;
    }
```

Since the application is 'only' a Windows program and not a real embedded system, explicit initialization is not necessary. The start in main() and all that really matters is starting the timer task.

Operation

The program runs in an endless loop until the <esc> key is pressed or the application is otherwise closed.

The console application reads the time from the timer task approximately twice a second and displays it.

Walter Borst
Kapitaen-Alexander-Strasse 39
27472 Cuxhaven, GERMANY

Fon: +49 (0) 4721 6985100
E-Mail: walter.borst@borst-automation.de
Home: https://www.borst-automation.de/

Termination

```
    // Stopp the timer task
    if (m_timer_task_started == TRUE)
    {
        CFramework::CTask::Terminate(&m_task_ctr_timer);
    }
```

When the program is terminated, it is important to also terminate all threads in a controlled manner beforehand.

Download the Source Code

You can download the source code of the presented demo at any time via the following link.

https://borst-automation.com/downloads/OneForTwoSCM-V1.0.zip.

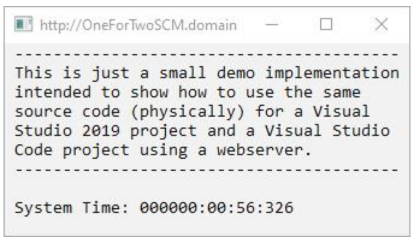
What's Next

Migration to Linux

I think, porting to a Raspberry Pi 4 would be a good idea for this purpose. Of course, the MainLoop module must then be further abstracted to the Linux console, and two more files are added to the special files marked ‘Win32’.

These new files will be marked as ‘Linux’.

Adding a Web Server



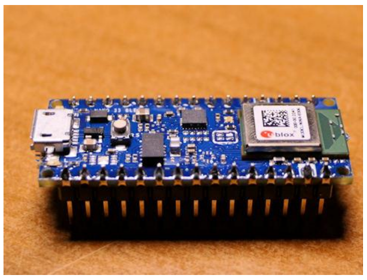
```
----------------------------------------
This is just a small demo implementation
intended to show how to use the same
source code (physically) for a Visual
Studio 2019 project and a Visual Studio
Code project using a webserver.
----------------------------------------

System Time: 000000:00:56:326
```

There are enough examples on the Internet of how such a web server is created. These also include those that visualize a console in the browser.

The web server would have the advantage that the display no longer needs to be specific to the system on which it is running.

Implementation on nRF52840



Finally, porting to an nRF52840 would show whether the concept delivers what it promises.

A serial interface would be used as the connection, which could be replaced by a Bluetooth connection in a further modification.