# An Open Letter to the Kinetis KSDK Team

From: An embedded developer

## The Background

Before I get into the detail of my suggestions, I want to make several points very clear. I am what some would call an “occasional” developer. Since I specialize in a niche market with low volume demands, the need is not for teams of people working on the same project, where the usual focus of each team might be systems, hardware, software, integration, etc. In the majority of my applications, it is one person who is responsible for the development process from concept through to production. In this environment, the cost of the development tools and the time to learn them become predominant factors in the final product cost.

With this background, I have enjoyed the Kinetis offerings. The ARM architecture provides the computing power needed at a suitable hardware cost. On making the Kinetis Development Studio (KDS) available for the cost of downloading, the tools are appropriate for these kinds of projects. When I first encountered KDS, I was pleased with the concept of the Processor Expert (PE). As it was presented, I got the impression that the various software features, such as USB, I2C, etc., were as easy as drag-and-drop. However, that was not the case for me and I found that the basic operation of PE hid the details of the hardware too much for me. So, on hearing that PE was not going to be supported in the Kinetis Software Development Kit (KSDK), I was not really disappointed.

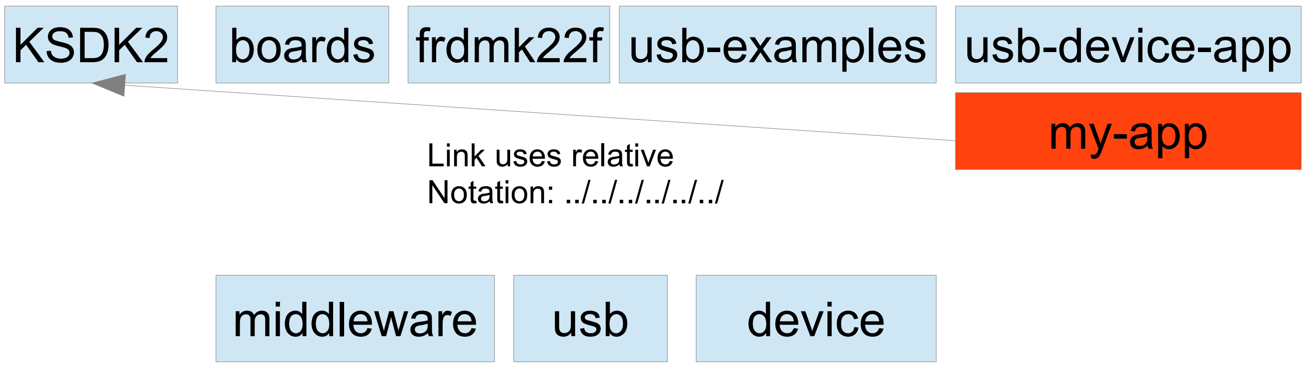
Moving on the the KSDK itself, once I learned how to use it with a basic board such as the FRDM-K22F, I became quite comfortable with it. There were a couple of projects that I did where the FRDM-K22F itself was the operational platform. I think that's the basic idea behind that form factor. The problem I had with this is the directory structure of the KSDK and how the demo projects are set up. There have been several write-ups on the support community about how to clone a project. I followed these in version 1.x of the KSDK and it was acceptable for prototype purposes. I was able to extend the process and contributed it to the community.

When version 2.x of the KSDK came out, I was intrigued by the idea of downloading just that portion of the KSDK that I needed, so I took that approach. However,I found that cloning of a project was no easier than before. So this letter is how I handled it and I hope that the team will review what I did and possibly adopt it. I'm sure other developers may find it of value.

## What's Wrong?

There are two basic problems with the current KSDK implementation. I'll handle the separately. The first is that although there are attempts to use and object oriented approach, they do not go far enough in the area of modularity and encapsulation. The simplest example is to look at any of the modules that contain the *main()* routine for the RTOS implementation. Previously, I was working with MQX, but currently I'm focusing on FreeRTOS and found the situation in both cases. Sure there is a *main()* routine, but it comprises only a small portion of the module. That's not my complaint. My complaint is that that once you have understood one case, i.e.: USB-MSD, and go to another, i.e.: USB-CDC, the base code is totally different with an almost identical *main()* routine. It becomes difficult to separate and identify just what code is appropriate for you case. The solution is fairly simple, but requires some thought.

The second problem has to do with how projects are set up and cloned. In this case,I think a couple of pictures will be worth more than the proverbial 1,000 words. In this first drawing, I have shown the various code sections in blue with the one of interest in red. That one was generated using the suggested cloning methods and is of interest as it will be my unique code. It is what I want to place under version control.



The figure is intended to show the approximate directory structure, but the key issue is that references to KSDK code and header files is via the relative notation. This means that I cannot easily move my unique code outside of the KSDK structure to work on it, save it to version control, etc. Sure, some VCS systems may be able to adapt, but this is forcing one practice on all developers.

**How can it be fixed?**

As I mentioned in the previous section, the modularity issue is easily fixed, and at the end, I'll show you why this is important. I take each of the modules containing the *main()* routine and separate them into two. For obvious reasons, I call the one **main.c**. It has no corresponding header file and all it has in it is the *main()* routine and any supporting code. It does what is necessary for setting up the primary processor and it's clocks. In itself, it does **not** set up auxiliary hardware that may be needed by the application code.

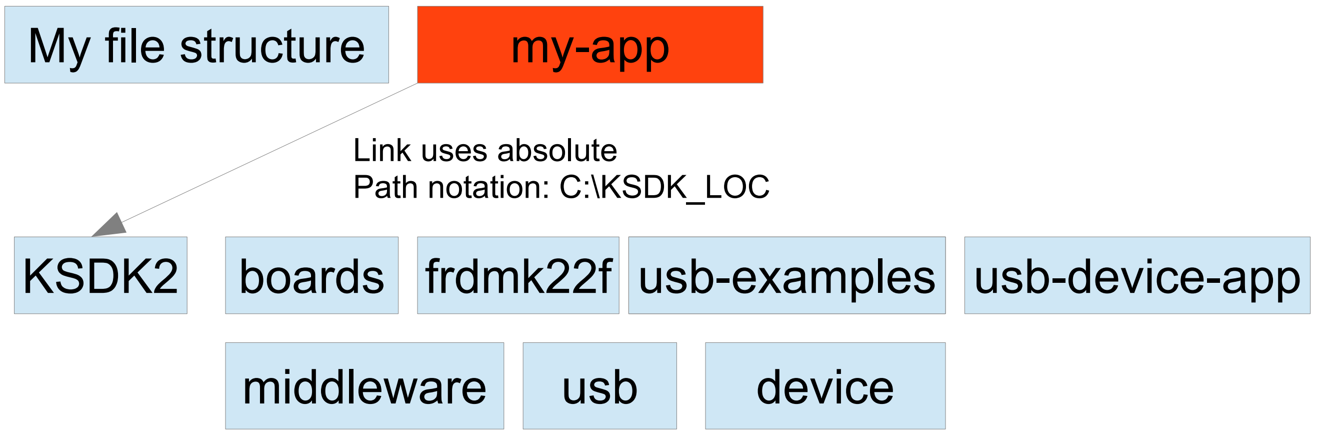
The other module, which I'll call by an appropriate name other than **app.c**, contains all the rest of the original code with the addition of a new routine called something like *I2C_InitPins()* where only those pins needed by the application are initialized.

Simple enough, now why was that change important? Once I have gone through the exercise of doing this for several cases, let's say I have an application that requires USB-CDC and I2C and that they are to be run under separate FreeRTOS tasks. This new implementation is as simple as including the USB-CDC and I2C tasks in the project and writing a new **main.c** that references the appropriate tasks. This is as close to drag-and-drop as it can get. No need to weed through the existing modules to see what is needed and try to deduce how this code might affect that other module. If the goal of modularity and encapsulation has been achieved, there should be no issues.

So, please design the base, or core modules, with the idea of reuse in mind.

The second issue is not quite as easy. This requires a little more thought, but once done it results in a much cleaner total package. In the previous drawing, note that due to the relative directory reference, I cannot move the red block outside of the KSDK tree structure, **unless** I make sure that it resides at the same depth in a separate tree. If, instead of using the relative addressing, we define a new path variable in the Project → Resource → Linked Resources as KSDK_LOC with the value of C:\KSDK2 which is where the KSDK is installed, then all of the ../../../../../../ references in the Project → C/C++ Build →

Settings → Cross ARM C Compiler → Includes can be replaced with the new path variable. Now the diagram looks like this and I am free to move my unique code to any subdirectory in the file system!



Note that all of my code is clearly located in a separate space and it is easy for me to identify it and handle it as necessary – copy, archive, etc. Also, note that at this point **all** of the code in the KSDK2 directory structure can be made read-only. This ensures that I will not inadvertently modify any of the KSDK code causing problems that may be blamed on the KSDK team.

There is another issue that I didn't address in my original complaints. That is one of unnecessary code duplication. The simplest example that I can pull up quickly is the **clock_config.h** module. Using a utility called **jdupes**, I found 112 identical copies! Since this all applies to the **frdmk22f** board, wouldn't it make more sense to have it reside once in that subdirectory? A quick glance shows that this would apply to the following list of files:

clock_config.c

MK22FN512xxx12_flash.ld

board.h

board.c

If the idea of breaking up the *BOARD_InitPins()* into multiple modules appropriate to each application, then this would probably apply to **pin_mux.c** and **pin_mux.h** as well. To be certain, I would need to look at this one closer as I continue this development.

Anyway, this approach would take advantage of the symbolic link capability of the KDS IDE. Since it is also possible to mix symbolic links and true modules within a section of the IDE, it would be possible to have the **boards** folder hold both unmodified KSDK code and code modified for my own purposes.

**The quick summary**

For me as an independent developer, these three changes would make the KDS/KSDK combination just that much more powerful. It is my feeling that there are others who would benefit from these changes as well. If done correctly, it would have any impact whatsoever on the casual user who wants to load up a demo and run it on the standard Kinetis hardware.