Kinetis SDK 2.0 Transition Guide

Rev. 2
02/2016

NXP Semiconductors, 6501 William Cannon Drive West, Austin, Texas 78735
www.nxp.com

Contents

1 Overview

This document describes the changes from Kinetis SDK 1.3 to 2.0. It also covers the transition from MQX to FreeRTOS, as MQX is not supported with Kinetis SDK 2.0.

2 New Features in Kinetis SDK 2.0

Kinetis SDK 2.0 is an evolution of SDK 1.x to provide a more optimized software solution. The largest change is the elimination of the concept of separate HAL and driver layers, which have now been combined into a single driver layer for each peripheral.

This new combined layer provides the low-level functionality of the HAL with the non-blocking interrupt-based functionality of the peripheral drivers, to allow users to select the right level of abstraction for their software needs.

One of the most notable changes in KSDK v2 is the focus on FreeRTOS. All RTOS examples now use the FreeRTOS kernel. Micrium µC/OS-II and µC/OS-III kernels are also pre-integrated into Kinetis SDK (under an evaluation license). RTOS peripheral driver wrappers for FreeRTOS and uC/OS-II & III are also provided that leverage native RTOS services (with no OS Abstraction layer). These are provided for use as-is or as examples for how to integrate KSDK drivers and middleware with other Operating Systems. The MQX kernel, stacks and middleware have been removed from the Kinetis SDK. However, MQX Software is still a supported software suite from NXP that is available in the MQX RTOS and MQX for KSDK v1.3 products. New ports of MQX following the "Classic" architecture (i.e. similar style to MQX v4.x) will soon be available for newer Kinetis devices. From here forward, the Kinetis SDK will focus on open-source RTOS solutions. This document will cover some of the differences when moving from MQX to FreeRTOS.

Other significant changes found in Kinetis SDK 2.0 include:

- The MQX RTCS Ethernet and MQX MFS File System stacks have been removed. lwIP and FatFs are still available for applications needing Ethernet and file system stacks.
- The USB stack has been re-written and is now a BSD licensed solution
- There is no longer a separate KSDK platform library. All examples and demos consist of a single project, which include all the necessary Kinetis SDK and application files.
- The Kinetis Expert tool is replacing Processor Expert, so there is no longer Processor Expert support with KSDK 2.0.
- Updates for Kinetis Design Studio to integrate it with Kinetis SDK are now provided via the online update tool, and no longer as part of a zip file included with Kinetis SDK. This update provides a wizard to create new SDK v2 projects inside of KDS.
- The Operating System Abstraction (OSA), Power Manager, and clock manager are no longer required by the drivers
- mbed TLS now included as part of the accelerated cryptography drivers

3 Kinetis SDK 2.0 Transition

While there is not a straight forward migration path to Kinetis SDK 2.0 from KSDK 1.3, there are important things to make note of that can make the process easier.

3.1 Supported Boards

The boards and devices supported by Kinetis SDK 2.x will be rolled out in waves. The latest roadmap can be found on the NXP Community.

3.2 File Structure

The file layout between KSDK 1.3 and KSDK 2.0 has changed significantly. One of the biggest changes is that all the examples and board related files are now in the **/boards** folder.

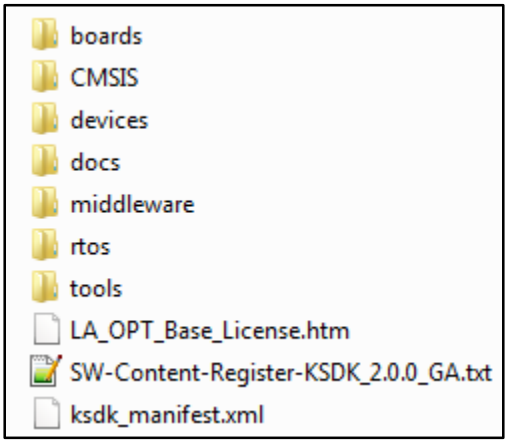

Figure 1: New SDK 2.0 File Layout

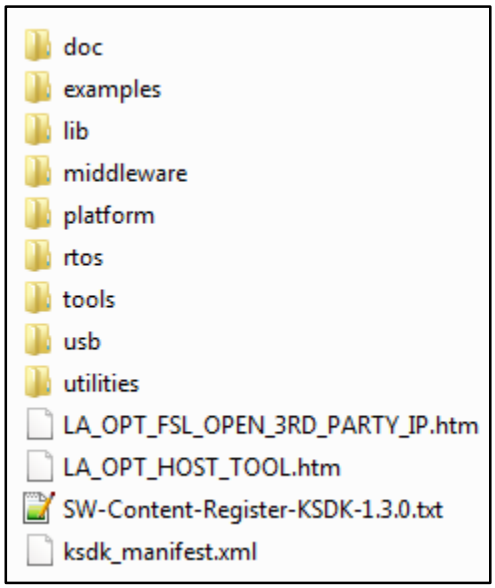

Figure 2: Old SDK 1.3 File Layout

Inside the **board** folder, you'll find a subdirectory for each board supported in this release of Kinetis SDK, and then below that, directories for each type of example.

Each example application will have its own unique copy of the board, pin_mux, and clock_config files. These files are no longer shared among all examples like in KSDK 1.3, but are instead specific to each particular example in KSDK 2.0.

The core of Kinetis SDK is now find in the **devices** folder, which is where the driver code is located, as well as the system startup files, linker files, and feature files.

The USB stack is now located in the **middleware** folder. The class files are also now located inside a particular USB example. As an example, the USB HID device files can be found in **\boards\<board_name>\usb_examples\usb_device_hid_mouse\bm**

3.3 Project Files

Kinetis SDK 2.0 does not have a separate platform library anymore, and so there is now only one project to compile. The project contains the necessary driver source code files under the **drivers** folder, as well as the board configuration files under the **board** folder. The **source** folder contains the application code for that particular example.

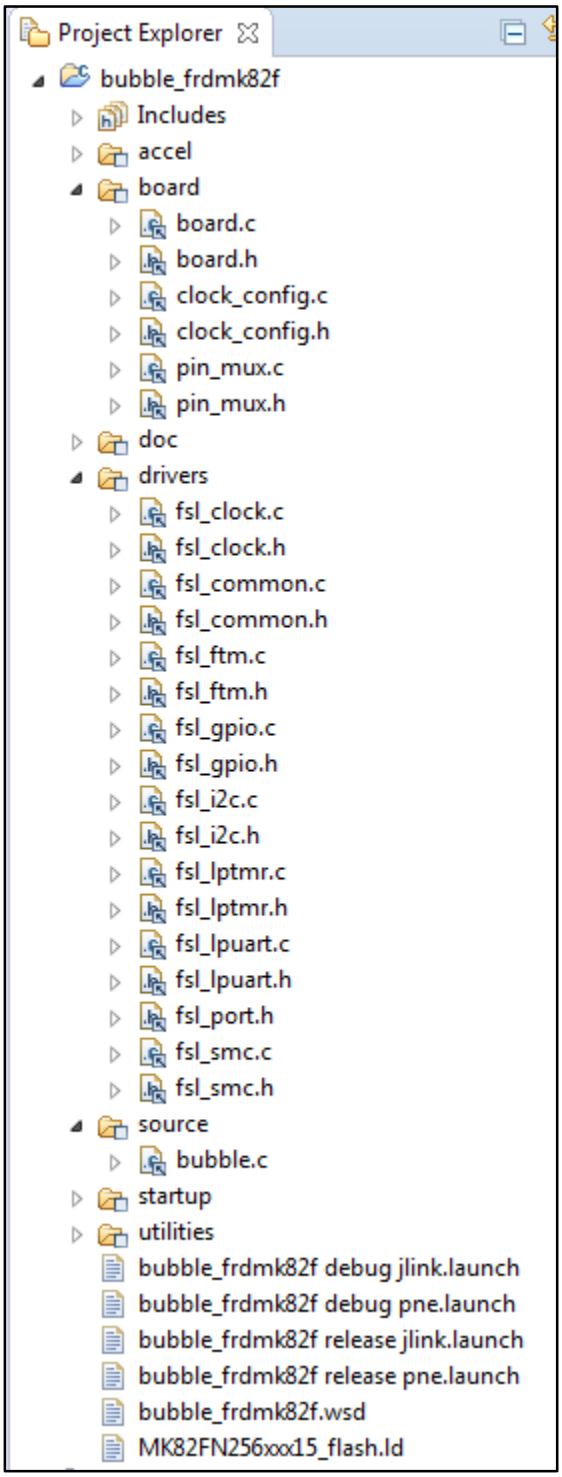

Figure 3: KSDK 2.0 Example Application Project

The project settings were also changed to reflect the new file structure layout and #defines used by Kinetis SDK 2.0. The include path is now:

$PROJ_DIR$/../../../../../CMSIS/Include
$PROJ_DIR$/../../../../../devices

$PROJ_DIR$/../../../../../devices/<device_name>/drivers
$PROJ_DIR$/..
$PROJ_DIR$/../../../../../devices/<device_name>/utilities
$PROJ_DIR$/../../..
$PROJ_DIR$/../../../../../devices/<device_name>

Also the path to the linker file is now in the same level as the example project, as each project has its own linker file now.

3.4 Kinetis Design Studio Updates

Updates for Kinetis Design Studio to add KSDK project generation functionality are now provided via the online update utility built inside of KDS. There is no longer a **KSDK_1.3.0_1.0.0_Eclipse_Update.zip** file to install from the installation directory first.

First go to **Help->Check for Updates** from inside KDS, and ***only*** install all the updates that are from Freescale and Somniumtech.

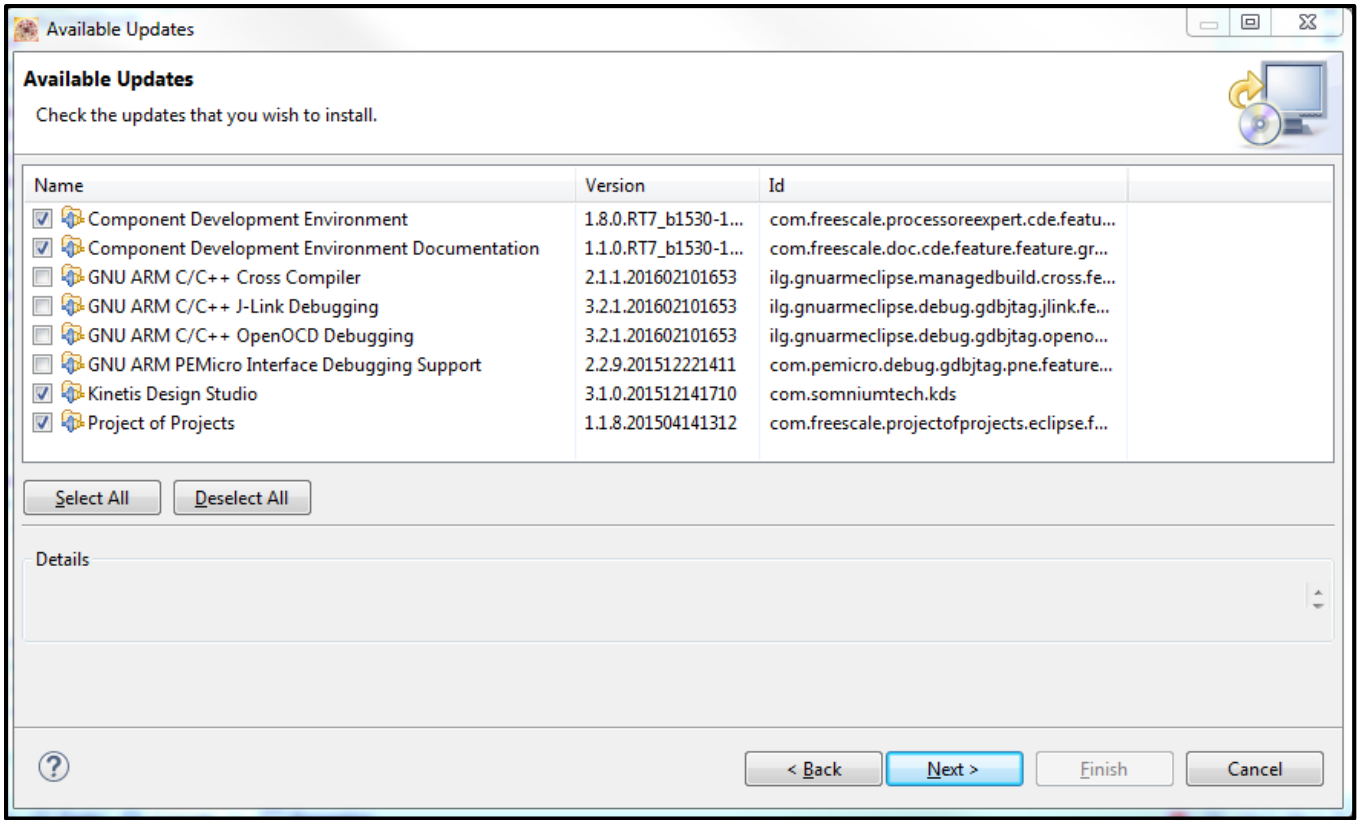

Figure 4: Kinetis Design Studio "Check for Updates" Dialog

After those updates are installed, then go to **Help->Install New Software** from the menu bar in KDS. In the **Work with** field, select the **Freescale KDS Update Site**. Then select the **New Kinetis SDK 2.x Project Wizard**, which provides a simple way to generate new KSDK 2.0 projects with KDS.

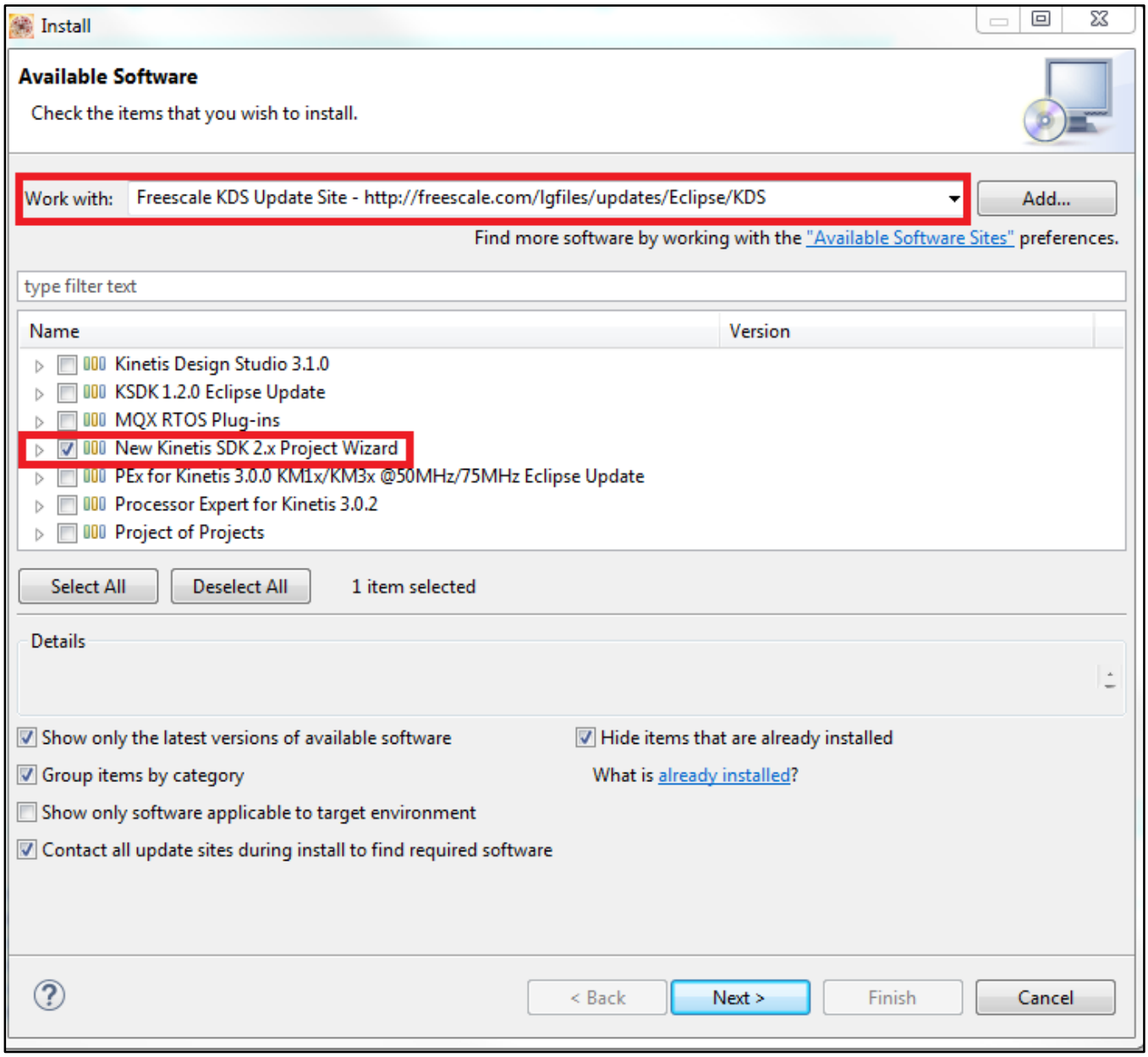

Figure 5: Kinetis Design Studio "Install New Software" Dialog

3.5 Distribution

Another change with Kinetis SDK 2.0 is how it is being distributed. Packages are generated using the new Kinetis Expert Tool, which is an online repository where the particular device/board, IDE/toolchain, RTOS, and features can be selected using a web interface, and then a unique package is generated with those specific options.

There will no longer be a monolithic installer that contains all of the KSDK supported boards and devices. This allows for much smaller download and installation sizes.

New features will be added over time to the Kinetis Expert Tool to allow for pin muxing selection and generation, clock configuration, and low power estimation.

3.6 Startup

Kinetis SDK 2.0 has a somewhat different startup and initialization sequence. Code initially begins executing in the Reset_Handler found in \devices\<device>\<compiler>\startup_<device>.s.

The SystemInit() function is now found at \devices\<device> and is slightly different since it no longer provides an option to configure the clock at this point. Its purpose is to enable cache (if available) and disable the watchdog timer.

The majority of the board initialization still occurs after reaching the main() function.

However the biggest change in how Kinetis SDK starts up is that there is no longer a hardware_init() function. Instead similar functions are called directly from main(). The following functions are typically called:

```
BOARD_InitPins();
BOARD_BootClockRUN();
BOARD_InitDebugConsole();
```

Inside of these initialization functions, the method and API for initializing pins and clocks has also been changed from 1.x.

3.7 KSDK API

Since the driver and HAL layers are now combined into a single layer, there have been significant API changes to Kinetis SDK from 1.x to 2.0. There is not a straight forward mapping of the changes, but the **Kinetis SDK v.2.0 API Reference Manual** contains the details of the new API.

3.8 Migrating a KSDK 1.3 project to Kinetis SDK 2.0

If you have an existing KSDK project, please keep in mind that it is not required to move to Kinetis SDK 2.0, and the work required to migrate an existing 1.x project to KSDK 2.0 is not trivial.

The most straight forward way to move an existing project to Kinetis SDK 2.0 is to create a new KSDK 2.0 project using the Kinetis SDK Project Generator tool, and then copy in the existing application code into this new project. There is also the option to use the New Project wizard in Kinetis Design Studio to create this new project too.

Creating a new project ensures that the project settings will be setup correctly. Then go through the hardware_init() function and setup the pins and clocks using the new KSDK 2.0 API. Finally, the driver code utilized by the application can be re-written to be compatible with the 2.0 API.

The examples provided under the **boards** folder will be very beneficial to learning about the new API used by KSDK 2.0 and how the code is structured.

4 FreeRTOS Transition

FreeRTOS is now the most popular RTOS to use with Kinetis SDK. MQX is no longer be available starting with KSDK 2.0. FreeRTOS provides many of the same RTOS features that were found with MQX. Similar features include:

- Multiple tasks with priority support
- Priority-based pre-emptive scheduler
- Semaphore and Mutexes (with priority inheritance)
- Message Queues and Message Passing
- Power saving in idle modes

FreeRTOS also provides similar usability features such as:

- Full source code available
- Small (<10KB) code size (configurable based on options)
- Kernel Aware Debugging in IDEs
- Example projects to showcase RTOS features

FreeRTOS is released under an open source license, so it is free of charge with no royalties. By using the modified GPL, you can distribute a combined work that includes FreeRTOS without being obligated to provide the source code for proprietary components. Only modifications to the kernel code must be published, and an OpenRTOS option is available if those kernel changes want to be kept proprietary.

4.1 New Features

FreeRTOS brings some new features that were not available on MQX. These include:

- Task notifications: An RTOS task notification is an event sent directly to a task that can unblock the receiving task, and optionally update the receiving task's notification value.
- Recursive mutex: A mutex used recursively can be 'taken' repeatedly by the owner
- Stack overflow hook/notification: There are two optional mechanisms that can be used to assist in the detection and correction of stack overflow events.
- Deferred interrupt handling: A mechanism is provided that allows the interrupt to return directly to the task that will subsequently execute the pended function.
- Blocking on multiple objects: Queue sets are a FreeRTOS feature that enables an RTOS task to block (pend) when receiving from multiple queues and/or semaphores at the same time.

4.2 Support Model

Support for FreeRTOS is provided in the NXP Kinetis SDK Community Forum and by the large development community that surrounds FreeRTOS. There are many textbooks, blogs, and forums focused on FreeRTOS. Professional Services from NXP is also available for custom code development.

4.3 High-Level Differences from MQX

There are some important conceptual changes that need to be made when moving from MQX to FreeRTOS. In FreeRTOS there is no longer the concept of a Task Template List. Instead tasks are created using **xTaskCreate()** API calls, and then the scheduler started by calling **vTaskStartScheduler().**

Also in FreeRTOS, the higher the priority number, the higher priority of that task. A task with priority number 10 is more critical than one with a priority number of 8. The idle task in FreeRTOS has a priority of 0, which is the lowest priority. In MQX this was the opposite, where a lower priority number meant that the task had a higher priority.

Several FreeRTOS example applications can be found at **boards\<board_name>\rtos_examples**

4.4 RTOS API

Migrating an application from MQX to FreeRTOS requires significant code re-write due to the differences in the API between the two RTOSes. Generally there is an equivalent function between the two, and at the high level they both have similar ways of doing things, but have different function names for them. The concepts of tasks, priorities, mutexes, and semaphores exist in both RTOSes though. There is not a direct mapping between the two RTOSes at the API level, but the API reference manual can be found online at the freertos.org website.

Some of the API changes are fairly straight forward, as you can see in the following examples:

- Create a Task:
  - MQX: **_task_create(…)**
  - FreeRTOS: **xTaskCreate(…)**
- Create a Semaphore:
  - MQX: **_lwsem_create(pointer, initialCount)**
  - FreeRTOS: **xSemaphoreCreateCounting(maxCount, initialCount)**
- Time Delay:
  - MQX: **_time_delay(time_to_delay)**
  - FreeRTOS: **vTaskDelay(ticks_to_delay)**

4.5 Migrating an MQX project to FreeRTOS

MQX for KSDK to FreeRTOS

Migrating from MQX for Kinetis SDK to FreeRTOS is a significant porting effort. The RTOS API will change, and the peripheral driver API, as discussed in the previous Kinetis SDK section, will change as well. The directory structure and project files are also significantly different. The most straight forward way to migrate a project would be is to create a new Kinetis SDK 2.0 project using the Kinetis SDK Project Generator tool, and then copy in the existing application code into this new project. This ensures the project settings will be setup correctly. However there will still be a significant amount of code re-write to use the new APIs.

MQX 4.x (Classic) to FreeRTOS

Migrating from MQX Classic (3.x or 4.x) will be a very significant effort. The RTOS API will change, and the peripheral driver API and model is very different from classic MQX to Kinetis SDK, and will need to be re-written from scratch. The directory structure and how things are organized are also very different. MQX Classic will continue to be supported on a paid basis with MQXv5, and that path is recommended going forward for classic users.

5 Middleware Stacks

Another major change is that the MQX RTCS (Ethernet) and MFS (Filesystem) stacks are no longer supported with Kinetis SDK 2.0 since they are dependent upon MQX. The USB stack was also rewritten from scratch for better efficiency and so it could be put under a BSD-style license.

5.1 Ethernet

The open source alternative lwIP is now the Ethernet stack available for use with Kinetis SDK. There are significant differences between MQX RTCS and lwIP, and so applications using this stack will need to be re-written. Example applications can be found at **boards\<board_name>\demo_apps\lwip** for boards supporting Ethernet

5.2 File System

The open source FatFs is now the file system stack available for use with Kinetis SDK. There are significant differences between the MQX MFS and FatFs, and so applications using this stack will need to be re-written. An example application showing how to use FatFs can be found at **\boards\<board_name>\driver_examples\sdcard_fatfs**

5.3 USB

The USB stack was rewritten for KSDK 2.x so that it could be released under a BSD license. There are numerous changes to the USB API and file structure. There is also no longer a separate USB library to compile like there was in KSDK 1.x. Instead all necessary source code is provided in a single USB example project for a particular class.

The USB examples are found in the **\boards\<board_name>\usb_examples** folder. For device examples, there are 'lite' versions available, which consist of a more tightly coupled example for smaller code size. The non-lite version is more generic for an easier to use example, with the tradeoff of larger code size. The **\docs\usb\USB Stack Device Reference Manual.pdf** document has more details on the differences.

6 References

Kinetis SDK 2.0 API Reference Manual

FreeRTOS API

Kinetis Expert Tool

NXP Communities:
Kinetis SDK
Kinetis Expert