Document Number: MQXKSDKRN
Rev 1.0.0, 07/2014

Freescale MQX™ RTOS for Kinetis SDK Release Notes

version 1.0.0 beta

Contents

1 Read Me

This is the release notes for Freescale MQX™ RTOS for Kinetis SDK. The product is based on the MQX kernel, MFS and RTCS stack version 4.1.0. The product is provided in two packages:

- MQX Kernel only – should be extracted to previously downloaded SDK
- MQX pre-integrated package – already contains SDK, TCPIP stack, File system

Note that this release note describes the complete version containing TCP/IP stack and MQX file system. It is not relevant for the MQX Kernel only package.

1.1 Development Tools Requirements

Freescale MQX RTOS was compiled and tested with these development tools:

- Kinetis Development Studio version 1.1.0
  - See build projects in `kds`subdirectories
- IAR Embedded Workbench® for ARM® Version 7.20.1
  - See build projects in `iar` subdirectories
- ARM-MDK™ - Keil µVision® version 5.11.0
  - See build projects in the `uv4` subdirectories

1.2 System Requirements

System requirements are based on the requirements for the development tools. There are no special host system requirements for hosting the Freescale MQX RTOS distribution itself.

The minimum PC configuration is determined by the development tools. The recommended PC configuration is 2 GHz processor, 2 GB RAM and 2 GB free disk space.

1.3 Target Requirements

Freescale MQX RTOS supports the evaluation boards mentioned below. There are no special requirements for the target hardware other than what each board requires for its operation (power supply, cabling, jumper settings etc). For more details about the board-specific setup for MQX applications, see Kinetis SDK board-related documentation.

Evaluation boards supported:

- Kinetis
  - TWR-K64F120M
  - FRDM-K64F
  - TWR-K22F120M
  - FRDM-K22F
  - TWR-KV31F120M

1.4 Set up installation instructions and technical support

Product is provided in two distribution options:

- MQX Kernel only as part of the standard Kinetis SDK installer
- MQX pre-integrated package with the RTCS TCP/IP stack and the MFS file system

MQX Kernel only package

This package is distributed as part of the Kinetis SDK package. To install MQX RTOS, select the corresponding option during the Kinetis SDK installation. Note that the package contains the MQX kernel only, not the TCP/IP stack, File system and other components.

MQX pre-integrated package

This package and contains the Kinetis SDK driver set, MQX kernel, RTCS - MQX TCP/IP stack and MFS - MQX File System. Unzip/install the package to your target system.

It is recommended that you install MQX RTOS to the path without spaces to avoid build problems with certain tool chains.

For a description of available support including commercial support options, visit the MQX RTOS support site on freescale.com.

2 What is new

This section describes the major changes and new features implemented in this release.

- This is a beta release of the MQX RTOS for Kinetis SDK. The final product will be based on the Kinetis SDK 1.1.0. The feature set and the directory layout will be changed in the final version of the product.

- The MQX kernel, RTCS stack and MFS stack is based on version 4.1.0.
- Peripheral I/O drivers are fully based on the Kinetis SDK driver set. MQX RTOS provides POSIX wrappers for the I/O console and filesystem only.
- The file API was changed to comply with POSIX standard (FILE* is used instead of legacy MQX_FILE)
- MQX RTOS provides POSIX-based I/O drivers for serial console (tty, uart), file system, Ramdisk and other MQX native I/O drivers (TFS, NULL, PIPE ...)
- MQX RTOS provides its own re-entrant stdlib.h implementation.

3 Release Content

This table describes the release contents:

Table 1. Release Contents

Deliverable	Location
Configuration Files	`<install_dir>/rtos/mqx/config/...`
Mass-build project for all supported boards	`<MQX_DIR>/build/<tool>/<board>`
MQX PSP, BSP Source Code, and Examples	`<install_dir>/rtos/mqx/...`
MQX PSP source code for Kinetis ARM Cortex-M core	`.../mqx/source/psp/cortex_m`
MQX BSP source code	`.../mqx/source/bsp/...`
RTCS source code and examples	`<install_dir>/tcpip/rtcs/...`
MFS source code and examples	`<install_dir>/filesystem/mfs/...`
NShell Library Source Code	`<install_dir>/rtos/mqx/nshell/...`
Keil Task Aware Debugging plugin (TAD)	`<install_dir>/tools/mqx_plugins/keil_extensions/...`
IAR Task Aware Debugging plugin (TAD)	`<install_dir>/tools/mqx_plugins/iar_extensions/...`
KDS Task Aware Debugging plugin (TAD)	`<install_dir>/tools/mqx_plugins/kds_extensions/...`
TFS Make Utility	`<install_dir>/tools/mktfs.exe`
AWK interpreter (GNU General Public License)	`<install_dir>/tools/gawk.exe`
SNMP code generation scripts	`<install_dir>/tools/snmp/*.awk`
TAD string and configuration files	`<install_dir>/tools/mqx_plugins/tad/...`
Documentation	`<install_dir>/rtos/mqx/doc`

This figure shows the Freescale MQX RTOS directories installed to the user host computer (subdirectories reduced for clarity):

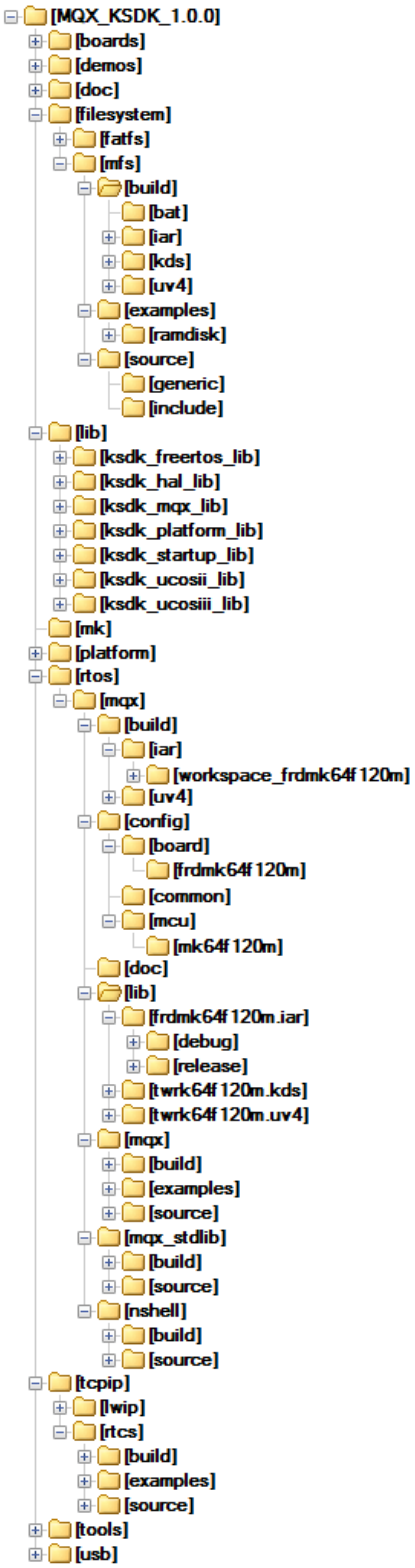

Figure 1. Freescale MQX RTOS Directories

4 MQX Release Overview

The Freescale MQX RTOS for Kinetis SDK consist of:

- MQX real time kernel and system components
- TCP/IP networking stack (RTCS)
- FAT file system (MFS)
- Platform and Board support packages

4.1 MQX RTOS PSP

Freescale MQX RTOS release contains ARM Cortex-M Platform Support Packages. Contact MQX RTOS support on freescale.com for other Freescale platforms.

The platform-specific code from `/mqx/source/psp/<platform>` is built together with the generic MQX core files. These two parts form a static library generally referred to as "PSP" which enables the target application to access RTOS features.

4.2 MQX RTOS BSPs

Freescale MQX RTOS release includes Board Support Packages for the boards mentioned in Target Requirements .

The board-specific code from `/mqx/source/bsp/<board>` is built together with I/O driver files from `/mqx/source/io`. These two parts form a static library generally referred as "BSP". The functions included in this library enable the board and operating system to boot up and use the I/O driver functions.

The following section describes drivers supported by the MQX BSPs.

4.3 I/O drivers supported

I/O drivers in MQX for Kinetis SDK are based on peripheral drivers and HAL layers provided as a part of the SDK framework. For some of these drivers, MQX brings POSIX compliant API wrappers. MQX also provides a set of platform independent drivers which simplify apps coding.

The driver code is located in `<install_dir>/rtos/mqx/mqx/source/nio/drivers`.

The following list describes POSIX based I/O drivers available in the latest MQX release. The full set of peripheral I/O drivers (non POSIX based) can be found in Kinetis SDK documentation

nio_dummy

The dummy driver does very little. This driver internally keeps some information on how many bytes have been read or written per file and per device.

nio_null

The null driver does nothing. This is the simplest possible driver, and is usually used as a template for new drivers.

nio_mem

This driver provides I/O operations on memory block. The memory block is specified by address and size.

nio_pipe

This driver provides a FIFO buffer where I/O operations are used to access the buffer.

nio_serial

This driver is a NIO subsystem wrapper built on top of the SDK UART driver. This driver is mainly used by nio_tty driver.

nio_tfs

Trivial Filesystem is used as a simple read-only file repository instead of the fully featured MFS. TFS is not installed in the BSP startup code. Applications must initialize the TFS and pass a pointer to the filesystem data image. The mktfs tool is available (both as executable and Perl script) to generate the image from the existing directory structure. The RTCS HTTP example demonstrates the use of TFS.

nio_tty

Tty driver is installed on top of driver used for standard input/output. This driver provides basic formatting for terminal such as echo and end of line handling. When MQX starts, nio_tty driver is installed on the top of nio_serial driver. It is then opened for stdin, stdout, and stderr.

4.4 Default I/O Channel

An I/O communication device installed by MQX BSP can be used as the standard I/O channel.

4.5 MQX PSP and BSP Directory Structure

RTOS files are located in the /rtos/mqx subdirectory of the Freescale MQX RTOS installation.

4.6 MQX MFS

MFS files from the `/filesystem/mfs/source` directory are built into a static library. When linked to the user application, the MFS library enables the application to access FAT12, FAT16, or FAT32-formatted drives.

4.7 MQX RTCS

RTCS files from the `/tcpip/rtcs/source` directory are built into a static library. When linked to the user application, the RTCS library enables the application to provide and consume network services of the TCP/IP protocol family.

The MQX 4.1.0 RTCS stack is IPv6 ready with respect to IPv6 Ready Logo certification and has passed all required tests. IPv6 support is available as a separate update package available from Freescale.

4.8 MQX USB Host and Device

USB Host and Device stack is provided as a part of the Kinetis SDK release. These stacks use an OS abstraction layer (OSA) to adapt to MQX RTOS. See USB documentation for details.

4.9 MQX nShell

The shell and command-line handling code is implemented as a separate library called nShell.

4.10 Building the MQX libraries

When using MQX RTOS for the first time and making changes to the compile-time user configuration file or MQX kernel source files, rebuild MQX libraries to ensure that the changes are propagated to the user applications.

4.11 Example applications

Demo applications are in this directory:

`<install_dir>/demo`

The examples are written to demonstrate the most frequently used features of the Freescale MQX RTOS.

In addition to these demo applications, there are simpler example applications available in MQX, RTCS, MFS, and USB directories.

The tables summarize all demo and example applications provided in this release.

MQX Example Applications

Table 2. `<install_dir>/mqx/mqx/examples`

Name	Description
demo	Shows MQX multitasking and inter-process communication using standard objects like semaphores, events, or messages. See lwdemo for the same example using the lightweight objects.
event	Simple demonstration of MQX events.
hello	A trivial Hello World application spread across two tasks.
isr	Shows how to install an interrupt service routine and how to chain it with the previous handler.
klog	Shows kernel events being logged and later the log entries dumped on the console.
log	Shows the application-specific logging feature.
lwdemo	Same as the "demo" application, but implemented using lightweight components only.
lwevent	Simple demonstration of MQX lightweight events.
lwlog	Simple demonstration of MQX lightweight log feature.
lwmsgq	Simple demonstration of MQX lightweight inter-process messaging.
lwsem	Simple demonstration of MQX task synchronization using the lightweight semaphore object.
msg	Simple demonstration of MQX inter-process message passing.
mutex	Simple demonstration of MQX task synchronization using the mutex object.
nill	Even simpler than Hello World. A void application which may be used for copy/paste to start custom application.
sem	Simple demonstration of MQX task synchronization using the semaphore object.
taskat	Shows how task can be created within statically allocated memory buffer (avoid heap allocation for task stack and context).
taskq	Shows custom task queue and how the queue can be suspended and resumed.
test	Shows the self-testing feature of each MQX component.
timer	Simple demonstration of MQX timer component.

Table continues on the next page...

Table 2. `<install_dir>/mqx/mqx/examples` (continued)

Name	Description
watchdog	Simple demonstration of the MQX task timeout detection using the kernel (not to be confused with watchdog) component.

RTCS Example Applications

Table 3. `<install_dir>/mqx/rtcs/examples/...`

Name	Description
eth_to_serial	Simple character passing between the UART console and the telnet session. Shows custom "lightweight" telnet.
httpsrv	Simple web server with CGI-like scripts and web pages stored in internal flash.
shell	Shell command line providing commands for network management.
snmp	SNMP protocol example providing microprocessor state information.
benchmark	Throughput benchmark application. Fnet benchmark is required on the PC side.

MFS Example Applications

Table 4. `<install_dir>/filesystem/mfs/examples/...`

Name	Description
ramdisk	Shows use of MFS accessing the external RAM (or MRAM).

5 Known issues and limitations

Idle Task Required on Kinetis Platforms

The Kinetis kernel, by design, cannot operate without an idle task. The MQX_USE_IDLE_TASK configuration option must be set to 1.

MFS Does Not Check Validity of Directory Rename

MFS_Rename_file() function does not check the necessary precondition when renaming a directory. If the directory is renamed to its own subdirectory, the directory becomes inaccessible and lost cluster chains are created.

UTF8 support in MFS

The UTF8 support in MFS is limited to read-only access and for long file names. The UTF8 support for write access will be implemented in a future release.

Interrupt handling and priorities

The KSDK package allows the application developer to use the NVIC_* CMSIS functions. A set of functions handle the NVIC interrupt priorities. The MQX scheduler, however, limits the usage of interrupt priorities. Follow these criteria when using NVIC_SetPriority with MQX:

- priority level should be an even number
- priority level should be equal or higher than 2 times the value of MQX_HARDWARE_INTERRUPT_LEVEL_MAX value input in mqx_init structure if you want to use the MQX services in the interrupt service handler

These limitations give the application developer a maximum of seven levels of interrupt priorities.

KDS project setup

KDS project can be built only with a selected C99 standard. Set the standard in Properties->C/C++ Build->Settings->Tool Settings->Cross ARM C Compiler -> Optimization->Language standard->GNU ISO C99(-std=gnu99).

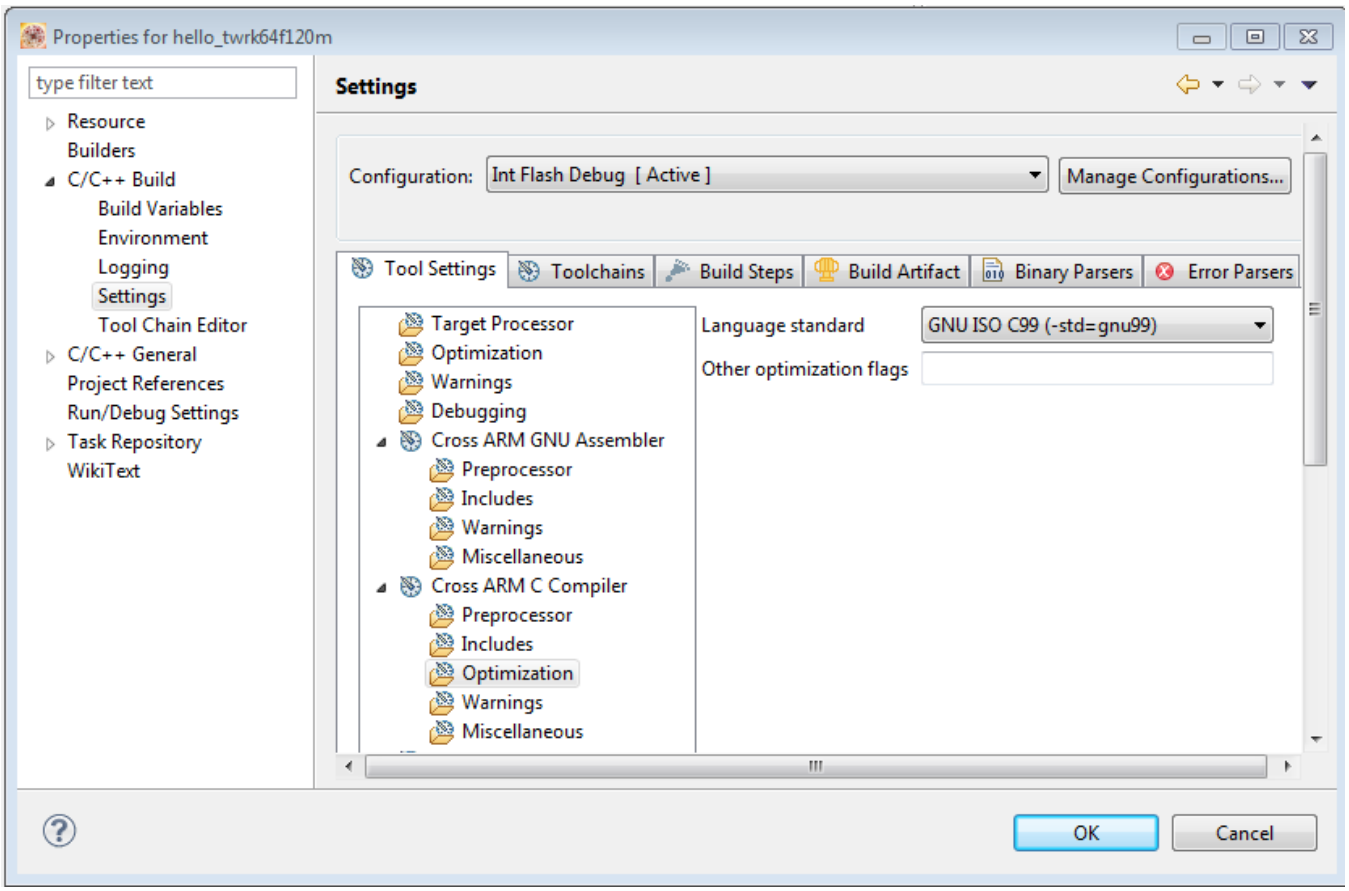

Figure 2. KDS C99 language standard setup

Concurrent access to core resources issue (MQX-1486)

If multiple tasks run on the same priority and are waiting for an event or if time slicing scheduling is used, an application might crash. This is caused by unsafe concurrent access to the core resources in the MQX memory handlers.

6 Revision History

This table summarizes revisions to this document.

Revision History		
Revision number	Date	Substantial changes
1.0.0	7/2014	Initial release

How to Reach Us:

Home Page:
freescale.com

Web Support:
freescale.com/support

Information in this document is provided solely to enable system and software implementers to use Freescale products. There are no express or implied copyright licenses granted hereunder to design or fabricate any integrated circuits based on the information in this document.

Freescale reserves the right to make changes without further notice to any products herein. Freescale makes no warranty, representation, or guarantee regarding the suitability of its products for any particular purpose, nor does Freescale assume any liability arising out of the application or use of any product or circuit, and specifically disclaims any and all liability, including without limitation consequential or incidental damages. “Typical” parameters that may be provided in Freescale data sheets and/or specifications can and do vary in different applications, and actual performance may vary over time. All operating parameters, including “typicals,” must be validated for each customer application by customer's technical experts. Freescale does not convey any license under its patent rights nor the rights of others. Freescale sells products pursuant to standard terms and conditions of sale, which can be found at the following address: freescale.com/SalesTermsandConditions.

Document Number MQXKSDKRN
Revision 1.0.0, 07/2014