

# NXP MQX™ RTOS 5.2 Release Notes

(May 2022)

## 1 Introduction

This document is the Release Notes for the MQX™ RTOS version 5.2. The software is built based on the MQX RTOS version 5.1. It includes the full set of RTOS services and a standard set of peripheral drivers.

NXP MQX™ RTOS is released for specific i.MX RT, i.MX, Kinetis, Vybrid, ColdFire, and Power Architecture processors. Support for other NXP processors is available upon request.

### 1.1 Development Tools Requirements

NXP MQX RTOS was compiled and tested with these development tools, though not with all tools for every release:

- MCUXpresso IDE from NXP
- IAR Embedded Workbench for ARM®
- DS-5 Development Studio from ARM
- CodeWarrior Development Studio from NXP

#### Contents

<b>1</b>	<b><i>Introduction .....</i></b>	<b><i>1</i></b>
<b>2</b>	<b><i>What Is New.....</i></b>	<b><i>3</i></b>
<b>3</b>	<b><i>Release Content .....</i></b>	<b><i>5</i></b>
<b>4</b>	<b><i>MQX RTOS Release Overview .....</i></b>	<b><i>7</i></b>



## 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 NXP MQX RTOS distribution itself.

Minimum PC configuration:

- As required by Development and Build Tools

Recommended PC configuration:

- 2 GHz processor – 2 GB RAM - 2 GB free disk space

Software requirements:

- OS: Windows® 7 or later

## 1.3 Supported Processors

Below is a summary of the processors supported by MQXv5 and the evaluation board that is supported for each release. MQX can be easily ported to your custom hardware. Details are available by contacting [mqxsales@nxp.com](mailto:mqxsales@nxp.com). The evaluation boards in the table below are all available from NXP or one of their distributors, except where otherwise noted. There are no special requirements for the target hardware other than what each board requires for its operation (power supply, cabling, jumper settings, etc.).

Support Processor	Evaluation Board
Kinetis K26	TWR-K65F180M
Kinetis K60 100 MHz	TWR-K60N512
Kinetis K60 120 MHz	TWR-K60F120M
Kinetis K64	FRDM-K64F
Kinetis K65	TWR-K65F180M
Kinetis K66	FRDM-K66F
Kinetis K70	TWR-K70F120M
Kinetis K81	TWR-K80F150M
Kinetis KV58	TWR-KV58F220M
i.MX6SX M4 core	MCIMX6SX-SDB
i.MX6ULL A7 core	Phytec i.MX6ULL SOM
i.MX7S / i.MX7D M4 core	MCIMX7SABRE
i.MX7S / i.MX7D A7 core	MCIMX7SABRE
i.MX8M M4 core	MCIMX8M-EVK
i.MX RT 1020	MIMXRT1020-EVK
i.MX RT 1024	MIMXRT1024-EVK
i.MX RT 1050	IMXRT1050-EVKB
i.MX RT 1060	MIMXRT1060-EVK
i.MX RT 1064	MIMXRT1064-EVK
i.MX RT 1160 (coming soon)	MIMXRT1160-EVK
i.MX RT 1170 M7 (+ support for the M4 core coming soon)	MIMXRT1170-EVK

## 1.4 Set up installation instructions and technical support

Unzip the provided package to your hard drive. There is no prescribed folder to install that package to, but it is recommended to install MQX RTOS to a path without spaces to avoid build problems with certain tools, and it is recommended to not install MQX to the C:\ root directory.

### NOTE

Since version 4.0, the pre-built libraries are not distributed in the MQX RTOS release package, which makes it necessary to compile MQX RTOS libraries for a particular board before the first use. For detailed build instructions, reference the release specific documentation providing installation instructions that was shipped with your release.

For a description of available support including commercial support options, click [here](#) or contact [mqxsales@nxp.com](mailto:mqxsales@nxp.com)

## 2 What Is New

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

### 2.1 Added in version 5.2

#### New Features:

- Task Aware Debugging for IAR Embedded Work bench ver 8.x and 9.x
- Performance Monitoring Code to monitor cpu loading by task
- Mbedtls 3.0
- Integration of mBedTLS and WolfSSL/SSH (optional components – extra fees may apply)

#### New Ports:

- i.MX RT 1170 M7 core (M4 core support coming soon)
- i.MX RT 1060
- i.MX RT 1064
- i.MX RT 1020
- i.MX RT 1024
- i.MX7 Colibri BSP
- KV58
- K81

#### General Clean up and Enhancements:

- Re-organization of some i.MX RT related files to accommodate the growing list of supported i.MX RT processors
- Updates to the Task Aware Debugging plug-in for MCUXpresso
- Added support for mbedTLS 3.0
- Updates to address reported vulnerabilities:
  - Updates to address 3 cases in MQX where allocating memory with a size that exceeds the size of the signed integer will cause a wrap-around of the integer. This results in the allocation of a smaller buffer and potentially a heap overflow.
  - Updates to address a vulnerability that affects the Link-Local Multicast Name Resolution (LLMNR) Server and in the TFTP Server. The vulnerability exists in the code that processes the host name for an LLMNR request, and the filename for a TFTP request. A host name or filename that is not null terminated may result in data corruption that could lead to a hard fault of the application.

- Added new processor specific SDKs
- Ported in new USB stack
- MQX
  - Added Timed Task Queues
  - Added a schedule rotate function
  - Added time related functions to get elapsed time and busy wait (in microseconds)
  - Added interrupt enter and exit functions
  - Added functions: io\_strtok\_r, io\_strdup()
  - Updates to SD Card driver initialization
  - Update to dispatch.s for M7 and M4 cores
  
- RTCS:
  - Added functions to load and execute S Record images over TFTP
  - Added a function to load and execute a boot image from a server
  - Added TFTP functions (open, read, eof, timeout\_restart, timeout\_update, and close).
  - Updates to socket management
  - Updates to DHCP pad size handling
  - Added IP interface socket release function
  
- Web Server:
  - Updated handling of https scripts
  
- MFS:
  - added ioctl for device identify
  
- Shell:
  - added new shell commands:
    - Performance monitoring data
    - Load and execute an executable file
    - Load an executable file
    - Monitor ADC inputs
    - Display GIC interrupts
    - Print memory blocks from a NOR flash
    - TFTP client commands
  - Updates to existing shell commands such as the Task Aware Debugging (TAD) command

#### Updates to Address Identified Vulnerabilities

- Implemented updates to address potential vulnerabilities with memory allocation functions as well as a host name length issue. Copies of the relevant bulletin are available by contacting us at [mqxsales@nxp.com](mailto:mqxsales@nxp.com)

#### Bug Fixes

- Various updates for general consistency and modernization of the code.

#### Deprecated Features Removed:

- None

### 3 Release Content

Table 1 lists the contents of this release. Note that not every sub-folder is included with each release since only supported and relevant libraries are included with each release of MQXv5.

**Table 1. Release Contents**

Deliverable	Location
Configuration Files	<install_dir>/config/...
MQX PSP, BSP Source Code, Project Files, and Examples	<install_dir>/mqx/...
MQX PSP source code for i.MX RT, Kinetis, Vybrid ARM Cortex-M core	.../mqx/source/psp/cortex_m
MQX PSP source code for i.MX, Vybrid ARM Cortex-A core	.../mqx/source/psp/cortex_a
MQX PSP build projects	.../mqx/build/<compiler>/psp_<board>
MQX BSP source code	.../mqx/source/bsp/<board>
MQX BSP build projects	.../mqx/build/<compiler>/bsp_<board>
RTCS source code and examples	<install_dir>/rtcs/...
RTCS source code	.../rtcs/source
RTCS build projects	.../rtcs/build/<compiler>/rtcs_<board>
RTCS example applications	.../rtcs/examples
MFS source code and examples	<install_dir>/mfs/...
MFS source code	.../mfs/source
MFS build projects	.../mfs/build/<compiler>/mfs_<board>
MFS example applications	.../mfs/examples
USB Host driver source code and examples	<install_dir>/usb/host/...
USB Host source code and class drivers	.../usb/host/source
USB Host build projects	.../usb/host/build/<compiler>/usbh_<board>
USB Host example applications (HID, MSD, HUB)	.../usb/host/examples
USB Device drivers source code and examples	<install_dir>/usb/device/...
USB Device source code	.../usb/device/source
USB Device build projects	.../usb/device/build/<compiler>/usbh_<board>
USB Device example applications (HID, MSD, CDC, PHDC)	.../usb/device/examples
Shell Library Source Code	<install_dir>/shell/...
Shell source code	.../shell/source
Shell build projects	.../shell/build/<compiler>/shell_<board>
Lua Library Source Code and examples	<install_dir>/lua/...
Lua source code	.../lua/src
Lua build projects	.../lua/build/<compiler>/lua_<board>

*Table continues on the next page...*

**Table 1. Release Contents (continued)**

Deliverable	Location
Lua example projects	.../lua/examples
TFS Make Utility	.../tools/mktfs.exe

<compiler> represents the supported compiler











In addition to the above components, the following optional components may be included. Additional license fee may apply to access these components, please contact [mqxsales@nxp.com](mailto:mqxsales@nxp.com) for further details.

- Amazon AWS Client
- Boot Loader
- FATFS File System
- LFS File System
- LVGL Graphics library
- Storyboard graphics integration
- Wolf SSL/SSH integration

Note that only the Debug build configurations for the provided libraries and examples are verified for each package. The provided project files may contain a Release build configuration for reference, but this is typically not verified. Should you prefer to release code to the field that is compiled with optimizations turned on, it is recommended to either clone the Debug build configuration to create your own build configuration with the desired optimization settings, or to adjust the optimization settings in the Debug build configuration. However, since code behaviour can vary with different compiler optimization settings, it is recommended that prior to releasing code to the field that full regression testing be done with the final optimization settings that you intend to use.

### 3.1 Directory Structure

Figure 1 shows a typical directory structure for a release of NXP MQX RTOS directories installed to the user's host computer (subdirectories not shown for clarity). Note that the layout of your release may differ since more than one USB version is not normally available and there may be other components added that are required to support your specific processor.

 mx_5_2	
 config	Configuration Files
 lua	Lua Scripting Language
 mfs	MS-DOS File System
 middleware	Middleware files such as inter-core communication software
 mx	MQX RTOS
 rtcs	RTCS TCP/IP Stack
 shell	Shell console interface
 tools	Tools – task aware debugging and boot loader code (if applicable)
 usb	USB –USB Host and Device (some releases use a different USB Stack)

**Figure 1. NXP MQX RTOS Directories**

## 4 MQX RTOS Release Overview

The release consists of the following libraries:

- MQX RTOS real time kernel and system components
- TCP/IP networking stack (RTCS)
- FAT file system (MFS)
- Shell
- Lua
- USB Host and Device stacks
- Platform and Board support packages
- I/O drivers

This release contains the following components and I/O drivers; however drivers will only be supported for processors that have the corresponding circuitry.

- Audio driver I2S or SAI
- Compact Flash Card driver
- DCU driver
- ESDHC driver
- Ethernet Driver
- FlashX Flash driver
- FlexCAN, msCAN
- I2C driver (polled and interrupt driven version)
- I2C driver (polled and interrupt driven version)
- LWADC – light weight ADC
- LWGPIO – light weight GPIO
- NAND Flash driver
- QuadSPI Driver
- RTC / IRTC Real Time Clock driver
- SD Card driver (SPI or SDHC based)
- SPI driver
- TSS Touch Sensing driver
- UART Serial driver (polled and interrupt driven version)

## 4.1 MQX RTOS PSP and BSP Directory Structure

RTOS files are located in the `mqx` subdirectory of the NXP MQX RTOS installation. The directory structure is shown in Figure 2.

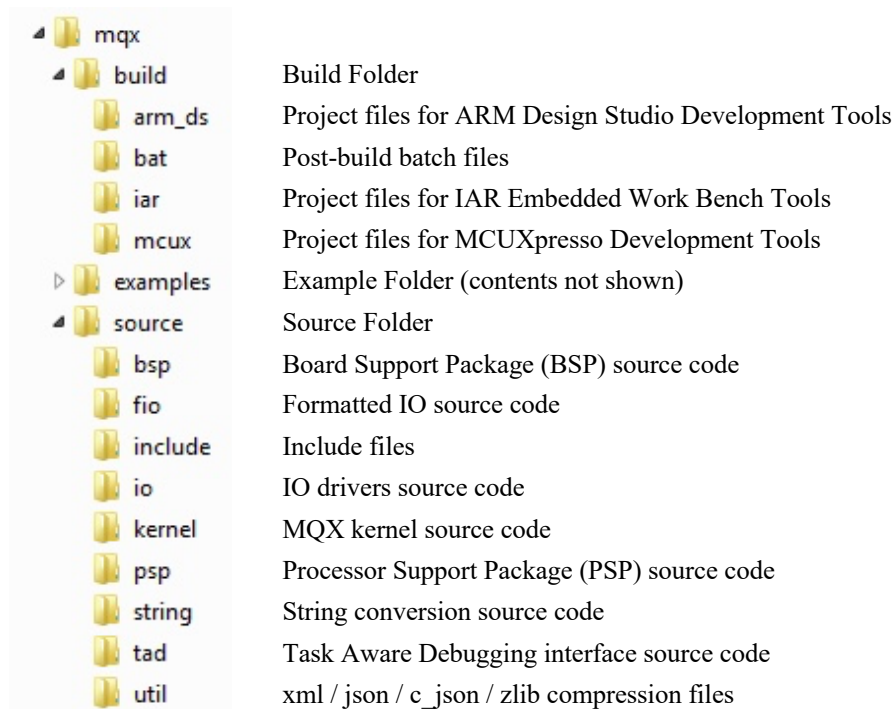


Figure 2. MQX PSP and BSP Directory Structure

## 4.2 MQX RTOS PSP

This release of NXP MQX RTOS contains support for specific ARM Cortex-A, Cortex-M, ColdFire V4 Platform Support Packages. Contact [mqxsales@nxp.com](mailto:mqxsales@nxp.com) for ports to other NXP 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 a Processor Support Packages (PSPs) which enables the target application to access RTOS features.

## 4.3 MQX RTOS BSPs

NXP MQX RTOS release includes Board Support Packages (BSPs) for the boards mentioned in section 1.3.

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 a BSP. The functions included in this library enable the board and operating system to boot up and use the I/O driver functions.

Subsequent sections describe drivers supported by the MQX BSPs.



## 4.4 Changing the MQX RTOS source files

The NXP MQX RTOS is distributed in source code form. It is recommended to not modify any of the source files other than the compile-time configuration files. This recommendation applies to all files under “source” and “build” sub-directories in all MQX RTOS, RTCS, MFS, USB, and other core components folders.

If you are creating custom board support packages or adding additional I/O drivers, add the new files and subdirectories to the following directories:

```
<install_dir>/mqx/source/bsp  
<install_dir>/mqx/source/io
```

## 4.5 Building the MQX RTOS libraries

For more details about building MQX RTOS libraries and applications, reference the release specific documentation that was provided with your package. 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 RTOS libraries to ensure that the changes are propagated to the user applications.

## 4.6 I/O drivers supported

The following list describes I/O drivers available in the latest MQX RTOS release. The drivers are an optional part of the MQX RTOS and their installation can be enabled or disabled in the BSP startup code. To provide the optimal code and RAM application size, most of the drivers are disabled by default in the `/config/<board>/user_config.h` file.

Note that not all drivers are supported by all releases. In some cases the required hardware, either on-chip or on the evaluation board, is not available. In other cases a driver has been ported but not tested so some testing may be required. Table 2 at the end of this section defines the drivers that are available for each release.

### NOTE

When `BSPCFG_driver-enabling` macros are missing in the `/config/<board>/user_config.h` file, the default setting is taken from the BSP-specific header file located in the `/mqx/source/bsp/<board>/<board>.h`. The user decides whether to enable the automatic installation of the driver in the BSP startup code (by enabling the appropriate `BSPCFG_ENABLE_XXX` macro in the `user_config.h`), or manually in the application code.

### TFS – Trivial Filesystem

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 `mkfts` 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.

### I2C I/O Driver

This driver supports the I2C interface in both master and slave mode. If enabled in user configuration, the I2C driver is installed during the BSP startup code as the “i2cx” in polled mode and as the “ii2cx” in interrupt mode where “x” stands for a specified I2C channel number. Example applications are provided in the MQX RTOS source tree for both master and slave mode.

**I2S and SAI I/O Driver**

This driver supports an I2S interface in a master mode. If enabled in user configuration, the I2S device driver is installed during the BSP startup code as "i2s0:". An example application is provided in the MQX RTOS source tree.

**SPI I/O Driver**

This driver supports the operation master mode. If enabled in user configuration, the SPI device drivers are installed during the BSP startup code as "spi0:" (or "spiX:" where X is index of the SPI module used).

**QuadSPI I/O Driver**

This driver provides a C language API to the QuadSPI peripheral module. If enabled in user configuration, the QuadSPI device drivers are installed during BSP startup code as "qspi0:" (or "qspiX:" where X is index of QSPI module used).

**FlexCAN Driver**

This driver provides a C language API to the FlexCAN peripheral module. An example application is provided in the MQX RTOS source tree.

**RTC Driver**

This driver provides a C language API to the Real Time Clock peripheral module and functions, and synchronizes the clock time between RTC and MQX RTOS systems. If enabled in user configuration, the RTC module is initialized and MQX RTOS time is renewed automatically during BSP startup.

**Serial I/O Driver**

The standard SCI (UART) driver supports both polled and interrupt-driven modes. If enabled in user configuration, the serial devices are installed as "ttya:", "ttyb:" and "ttyc:" (polled mode) and "ittya:", "ittyb:" and "ittyc:" (interrupt mode) automatically during BSP startup.

**LWGPIO I/O Driver**

This the light weight GPIO driver which provides a C language API to all GPIO ports available on a particular device.

**LWADC I/O Driver**

This driver provides a C language API to ensure a uniform access to ADC peripheral basic features.

**Flash I/O Driver**

This I/O driver provides a standard interface to either internal or external Flash memory. If enabled in user configuration, the Flash driver (called FlashX) is installed as "flashx:" device automatically by the BSP startup code. Note that "flash0", "flash1" etc. device names are used for FlashX devices installed for external Flash memory. For devices with internal Flash memory, the FlashX driver depends on several parameters passed in a form of global symbols from an application or from a Linker Command File. For more information, see driver installation code in the BSP and an example application provided in the MQX RTOS source tree.

**ENET Driver**

The low-level Ethernet driver is used by the RTCS TCP/IP software stack. The driver is initialized directly by the application before RTCS is used for the first time. The RTCS Shell and HTTP examples demonstrate the use of this driver.

**SD Card I/O Driver**

This I/O driver implements a subset of the SD protocol v2.0 (SDHC). The driver can use either the MQX RTOS SPI driver or the MQX RTOS (e)SDHC driver to communicate with the SD Card device. Install the driver at the application level, and pass a lower-layer driver handle to it. The MFS file system can be installed on top of this device.

**(E)SDHC I/O Driver**

This I/O driver covers the (e)SDHC peripheral module and provides low-level communication interface for various types of cards including SD, SDHC, SDIO, SDCOMBO, SDHCCOMBO, MMC, and CE-ATA.

**Resistive Touch-Screen Driver**

This I/O driver accesses the ADC and GPIO modules to detect touch events and acquire touch coordinates on a resistive touch-screen unit.

**HWTimer Driver**

This driver provides a C language API for uniform access to the features of various HW timer modules such as PIT and SysTick.

**DMA Driver**

This driver provides the C language API and essential functionality to control the DMA peripheral module.

**I/O Expander Driver**

This driver controls an off-chip I/O expander device and provides a convenient interface for individual pin handling. Currently, it only supports the MAX7310 device.

Table 2. Driver Availability

	MQX PSP+BSP Libraries	MFS Library (FAT File System)	RTCS Library (IPv4/IPv6 TCP/IP Stack)	Shell Library	USB Host Library	USB Device Library	UART (polled and interrupt driven)	I2C (polled and interrupt driven)	SPI	LWGPIO	HW Timer (PIT< Systick, GPT)	LWADC	FlashX	u/eSDHC	Ethernet Driver	SD Card driver (SPI or SDHC based)	FlexCAN / mCAN	Audio driver I2S or SA1	QuadSPI	NAND flash driver	Compact Flash Card Driver	RTC, RTC (Real Time Clock)	RPmsg	TSS_touch Sensing	DCU
<b>Kinetis</b>																									
FRDM-K64F	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	1	2	n/a	n/a	n/a	1	n/a	2	n/a
FRDM-K66F	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	1	3	n/a	n/a	n/a	2	n/a	3	n/a
TWR-K60D100M	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	1	2	n/a	n/a	2	1	n/a	2	n/a
TWR-K60F120M	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	1	2	n/a	2	2	1	n/a	2	n/a
TWR-K65F180M	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	3	3	3	n/a	2	2	1	n/a	2	n/a
TWR-K70F120M	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	1	2	n/a	2	2	1	n/a	2	n/a
TWR-K80F150M	●	●	●	●	3	3	●	●	3	3	3	3	3	●	3	3	3	●	n/a	3	3	3	n/a	3	n/a
TWR-KV58F220M	●	●	●	●	3	3	●	●	●	●	3	●	3	3	3	3	3	3	n/a	3	3	3	n/a	3	n/a
<b>i.MX</b>																									
MIMXRT1170-EVK	●	●	●	●	●	●	●	●	●	●	●	●	3	●	●	3	3	3	3	3	3	3	●	n/a	1
MIMXRT1064-EVK	●	●	●	●	●	3	●	●	3	●	●	●	3	●	●	3	3	3	3	3	3	3	n/a	n/a	1
MIMXRT1060-EVK	●	●	●	●	●	●	●	●	●	●	●	●	3	●	●	3	3	3	3	3	3	3	n/a	n/a	1
MIMXRT1050-EVK	●	●	●	●	●	3	●	3	3	●	1	3	3	1	●	●	3	3	3	3	n/a	1	n/a	n/a	n/a
MIMXRT1024-EVK	●	●	●	●	3	3	●	●	●	●	1	3	3	3	●	3	●	3	3	3	3	3	n/a	n/a	n/a
MIMXRT1020-EVK	●	●	●	●	3	3	●	●	●	●	1	●	3	3	●	3	3	3	●	3	3	3	n/a	n/a	n/a
RD-IMX6SX-SABRE	●	●	●	●	1	1	●	●	●	●	●	●	3	3	●	3	●	3	3	●	n/a	●	●	n/a	n/a
i.MX6ULL	●	●	●	●	●	●	●	●	●	●	●	3	3	3	●	●	3	3	3	3	3	3	n/a	n/a	n/a
MCIMX7DSABRE	●	●	●	●	1	1	●	●	1	1	1	3	3	1	1	1	1	3	3	1	n/a	1	1	n/a	n/a
MCIMX7S-Toradex	●	●	●	●	3	3	●	1	1	1	1	3	3	1	1	1	1	3	3	3	n/a	1	1	n/a	n/a
imx8M	●	●	●	●	3	3	●	1	1	1	1	3	3	1	1	1	1	3	3	3	n/a	1	1	n/a	n/a
<b>Power Architecture</b>																									
TWR-PXS30	●	●	●	●	3	3	●	●	●	●	3	●	3	3	●	●	3	3	3	3	3	3	3	3	3
<b>ColdFire</b>																									
MCF54455 EVB	●	●	●	●	3	3	●	●	●	●	●	3	3	3	●	3	3	3	3	3	3	3	n/a	3	3

**Legend:**

- Verified on this development board
- 1 Driver/Library has been ported to this board but is not supported or verified and a little work may be needed to get it going.
- 2 4.2 Driver is available for this board, work may be needed to get it going with 5.x.
- 3 Driver/Library has not been ported to this board. Some porting effort will be required.
- n/a Driver is not applicable to this board since the required circuitry / components are not provided.

**Notes**

A) The Low Power, User Mode, and Flash File System found in MQX Classic have been deprecated. However, they can be ported and provided as separately licenseable items.

## 4.7 Default I/O Channel

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

## 4.8 MFS for MQX RTOS

MFS files from the `/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.

## 4.9 RTCS for MQX RTOS with IPv4 and IPv6 support

RTCS files from the `/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 RTOS RTCS stack is IPv6 ready with respect to IPv6 Ready Logo certification and has passed all required tests. The IPv6 protocols for RTCS were previously separately licensed for a fee. However, starting with MQXv5 they are included with the MQX package.

## 4.10 USB Host for MQX RTOS

NXP MQX RTOS release includes a USB Host stack and a set of class drivers. The Mass Storage Device (MSD) class driver is typically supported with an example for releases that require USB Host support. This example utilizes the MFS File System to mount and to access a USB Mass Storage Device such as a memory stick or flash storage device. The following is a complete list of the class drivers provided that you can utilize:

- Audio
- Communication Devices Class (CDC)
- Communication Devices RNDIS
- Human Interfaces Device (HID)
- Hub
- Mass Storage Device (MSD)
- Personal Healthcare Devices Class (PHDC)
- Printer
- video

## 4.11 USB Device for MQX RTOS

NXP MQX RTOS release includes a USB Device stack and a set of class drivers. The RNDIS (virtual NIC) class driver is typically provided with ports that require USB Device support. The following is a complete list of the class drivers provided that you can utilize:

- Audio
- Composite
- Human Interface Device (HID)
- Mass Storage Device (MSD)
- Personal Healthcare Devices Class (PHDC)
- Printer
- Virtual Comm
- Virtual NIC

## 4.12 MQX RTOS Shell

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

## 4.13 Lua

The Lua Scripting Language is a powerful mechanism to interface to your application remotely to automate testing or to run a series of commands conveniently. It can also be called locally by an application task to run a number of commands sequentially and consistently. Lua is built as a separate library.

## 4.14 Example applications

Example applications are available in MQX RTOS, RTCS, MFS, and USB directories. Note that only a subset of these examples will have project files created for your specific set of tools and processor. Typically, the following examples are provided with project files for your environment, plus possibly others that are required to verify peripheral interfaces that are supported with your specific release:

- `mqx\examples\hello`
- `mfs\examples\ramdisk`
- `rtcs\examples\shell`

Tables 3 through 7 summarize the example applications that are provided in source code that are more commonly leveraged as examples on how to interface to MQX kernel objects and in some cases external peripherals. Some of these may be provided with project files for your compiler / debugger tools, but that should not be expected. However, you can create project files for any of these which can be done by cloning one of the provided example application project files (such as the hello example) and pulling in the associated source code.

Table 8 summarizes other examples that are provided in source code but project files will not be provided for. However, similar to the other examples, this source code can be a good reference for accessing MQX features and external peripherals.

**Table 3. MQX Examples**

Name	Description
Accel	Shows accessing the accelerometer via the I2C bus
can/flexcan	Shows usage of FlexCAN API functions to transmit and receive CAN frames.
demo	Shows MQX RTOS multitasking and inter-process communication using standard objects like semaphores, events, or messages. See <code>lwdemo</code> for the same example using the lightweight objects.
event	Simple demonstration of MQX RTOS events.
flashx	Demonstration of FlashX driver functionality.
flashx_swap	A demonstration of FlashX driver's swap and reset functionality.
hello	A trivial Hello World application using a single task.
hwtimer	Shows usage of HW timer driver abstraction. Demonstrates how to initialize HW timer for various modules, set frequency, callback, start, and stop the timer.
i2c	Shows how to read/write data from/to external EEPROM. Additional HW setup is needed.
lwadc	Shows usage of the ADC driver, sampling analog values from the two ADC channels.

qspi	Demonstrates basic operation of QuadSPI driver, interfacing to QSPI flash.
rtc	Shows the Real Time Clock module API. Demonstrates how to synchronize RTC and MQX RTOS time and how to use RTC alarm interrupts.
spi	Demonstrates the use of the spi channel to access a memory device

**Table 4. RTCS Examples**

Name	Description
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.

**Table 5. MFS Examples**

Name	Description
mfs_usb	Console shell-based example showing how to access MFS filesystem mounted on the USB mass storage.
ramdisk	Shows use of MFS accessing the external RAM (or MRAM).
sdcard	Shows use of MFS accessing the SDHC or SPI-connect SD Card.

**Table 6. USB Examples**

Name	Description
USB Host with Mass Storage Class Driver	Executes the standard "mass storage device" commands to a USB connected mass storage device.
USB Device with VNIC	Acts as a USB Device connected to a computer for implement USB over ethernet (virtual NIC)

**Table 7. Lua Examples**

Name	Description
shell	show the general use of Lua

**Table 8. Additional Examples**

The following examples are also provided for reference in source code. Project files are not included, but the examples are helpful demonstrations of how to access MQX features and peripherals

Name	Description
<b>MQX Examples</b>	
benchmrk	Contains benchmarks codes for timing and code size for different components.
cplus	Shows simple C++ application.
fp	Shows creation of a floating point task
ftm	Demonstrates how to use the FTM Quaddec driver on the Vybrid A5 processor
ftm_pwm	Demonstrates the use of the FTM in PWM mode for controlling an LCD backlight
hello2	A trivial Hello World application spread across two tasks.
i2c_scan	Scans for active peripherals on the i2c bus across the entire address range
imx7	Contains some lwgpio examples for the i.MX7 processor
ipc	Demonstrates the use of the inter-processor communications link on a multi-core processor
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.
lcdifv2	Demonstrates the use of the LCDIFv2 interface on some i.MX RT processors
log	Shows the application-specific logging feature.
lwbrsem	Shows the use of the light weight binary recursive semaphore
lwdemo	Same as the "demo" application, but implemented using lightweight components only.
lwevent	Simple demonstration of MQX RTOS lightweight events.
lwgpio	Demonstrates use of the light weight GPIO driver
lwlog	Simple demonstration of MQX RTOS lightweight log feature.
lwmsgq	Simple demonstration of MQX RTOS lightweight inter-process messaging.
lwsem	Simple demonstration of MQX RTOS task synchronization using the lightweight semaphore object.
lwtimer	Demonstrates the use of the light weight timer
msg	Simple demonstration of MQX RTOS inter-process message passing.
multicore	Demonstrates the use of core mutexes and shared memory for a multi-core processor
mutex	Simple demonstration of MQX RTOS task synchronization using the mutex object.
nandflash	Shows the use of the NAND flash driver included with MQX for some Kinetis processors
perf_mon	Performance monitoring code that illustrates the CPU loading by task
qspi	Illustrates the interface of quad SPI serial flash memory modules found on Vybrid boards
rpsmsg_pingpong	Demonstration of the RP message interface between MQX and Linux on a i.MX multi-core processor
rtc	Illustrates the use of the real time clock and interrupts for creating alarms
sem	Simple demonstration of MQX RTOS task synchronization using the semaphore object.
serial_test	Illustrates the use of accessing the serial port and changing the communication parameters
shell	Demonstrates the use of the shell to support a set of commands
spi_master	Demonstrates the use of a SPI channel between two boards and operating as the master
spi_slave	Demonstrates the use of a SPI channel between two boards and operating as the slave
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 RTOS component.
tfs	Shows the usage of ROM-based Trivial File System in an MQX RTOS application.
time	Prints out the current time in seconds and milliseconds
timedelay	Illustrates the use of the <code>_time_delay()</code> function
timer	Simple demonstration of MQX RTOS timer component.
watchdog	Simple demonstration of the MQX RTOS task timeout detection using the kernel (not to be confused with watchdog) component.
wifi_module	Illustrates a connection to a wifi module over a serial connection
zlib	Demonstrates the use of the zlib compression/decompression function
<b>RTCS Example</b>	
eth_to_serial	Simple character passing between the UART console and the telnet session. Shows custom "lightweight" telnet.