Renesas Synergy™ Platform

FileX Port Block Media Framework Module Guide

Introduction

This module guide will enable you to effectively use a module in your own design. Upon completion of this guide, you will be able to add this module to your own design, configure it correctly for the target application and write code, using the included application project code as a reference and efficient starting point. References to more detailed API descriptions and suggestions of other application projects that illustrate more advanced uses of the module are available in the Renesas Synergy™ Knowledge Base (as described in the References section at the end of this document), and should be valuable resources for creating more complex designs.

The FileX Port Block Media Framework module, implemented on sf_el_fx, supports the Express Logic FileX® system, a complete FAT format media and file management system for deeply embedded applications. The FileX Port Block Media Framework module provides I/O calls for FileX to access Synergy Media drivers through the Block Media Interface and adaptation layers. The SD/MMC HAL module and the SDIO HAL module are implemented on r_sdmmc and are used to read/write and control SD/MMC media devices as well as SDIO cards. The sf_el_fx module uses the SD/MMC peripherals on the Synergy MCU.

Contents

1. FileX Port Block Media Framework Module Features

- The FileX Port Block Media Framework module features include:
  - Support for FAT32, FAT16, and FAT12
  - Support for mounting the first FAT partition on a given media
  - Support for unlimited FileX objects (media, directories, and files)
  - Support for dynamic FileX object creation/deletion
  - Flexible memory usage
  - Size scales automatically
  - Small footprint
  - Complete integration with ThreadX.
- SD card interface
  - Compatible with SD, SDHC, and SDXC formats
  - Supports 1-bit and 4-bit bus width
  - Card detect function when supported by the hardware
  - Write protect support.
- eMMC interface
  - Supports 1-bit, 4-bit, and 8-bit bus width.
- SD bus interface
  - Compatible with SD memory card and SDIO card
  - Transfer bus mode selectable from 4-bit wide bus mode or 1-bit default bus mode
  - Compatible with SD, SDHC, and SDXC formats.
- SD and MMC shared
  - DMAC and DTC triggerable by the SBFAI interrupt SD buffer is read and write accessible using the DMAC.

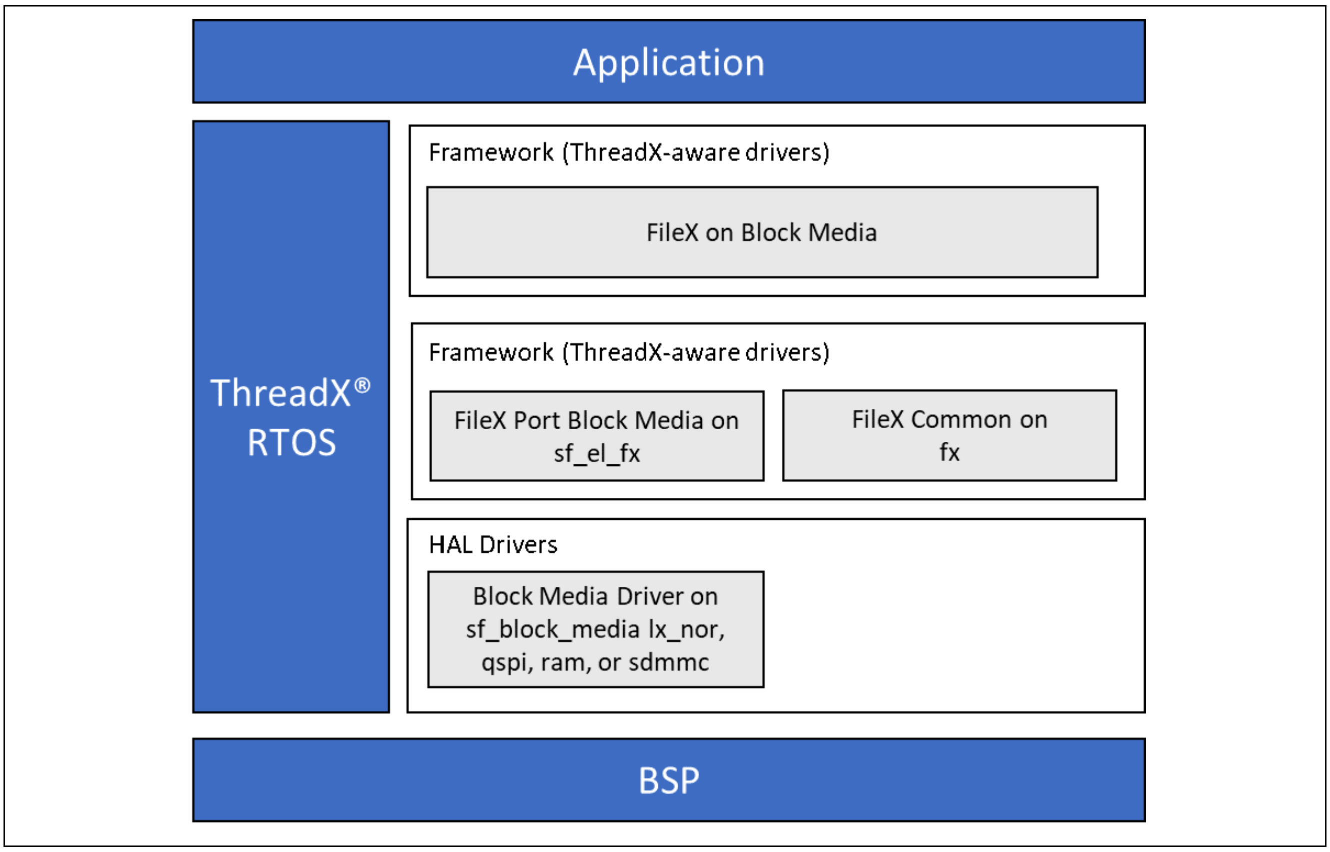

Application
ThreadX® RTOS
Framework (ThreadX-aware drivers)
FileX on Block Media
Framework (ThreadX-aware drivers)
FileX Port Block Media on sf_el_fx
FileX Common on fx
HAL Drivers
Block Media Driver on sf_block_media lx_nor, qspi, ram, or sdmmc
BSP

Figure 1. FileX Port Block Media Framework Module Organization, Options, and Stack Implementations

2. FileX Port Block Media Framework Module API Overview

The FileX Port Block Media Framework implements the Express Logic FileX operations for file organization and access. The complete list of FileX related APIs is too long to provide here, so only some of the most important ones are provided in the following table. Refer to the *FileX User's Manual*, as listed in the References section at the end of this document.

Table 1. FileX Port Block Media Framework API Summary (Selected examples only)

Function Name	**Example API Call and Description**
fx_directory_attributes_read	`fx_directory_attributes_read(&my_media, "mydir", &attributes);` Reads the directory's attributes from the specified media.
fx_file_open	`fx_file_open(&my_media, &my_file, "myfile.txt", FX_OPEN_FOR_READ);` Opens "myfile.txt" file for read.
fx_file_create	`fx_file_create(&my_media, "myfile.txt");` Creates file with name "myfile.txt."
fx_media_read	`fx_media_read(&my_media, 22, my_buffer);` Reads the logic sector (in this example from sector 22) from media specified by &my_media, and places it into the buffer my_buffer.

Note: For more complete descriptions of operation, status return values, and definitions for the function data structures, typedefs, defines, API data, API structures and function variables, review the *SSP Reference Manual* as described in the References section at the end of this document. For FileX API documentation, refer to X-Ware and NetX Component Documents for Renesas Synergy, as described in the References section.

3. FileX Port Block Media Framework Module Operational Overview

FileX is a complete File Allocation Table (FAT) format media and file management system for deeply embedded applications. FileX supports an unlimited number of media devices at the same time:

- RAM disks
- FLASH managers
- Multiple other physical devices.

FileX supports 12-bit, 16-bit, and 32-bit FAT formats and contiguous file allocation. FileX is highly optimized for both size and performance. You can find the API reference for the FileX module in the *FileX User's Manual* as described in the References section at the end of this document.

3.1 FileX Port Block Media Framework Operational Notes

The media must be formatted to either a FAT12, FAT16, or FAT32 filesystem before it can be opened. This can be done prior to inserting the media, or at run time.

3.2 FileX Block Media Framework Limitations

- SF_EL_FX does not currently support exFAT
- When using SF_EL_FX with SD/eMMC, the FileX block size must be 512 bytes
- Refer to the most recent *SSP Release Notes* for any additional operational limitations for this module.

4. SDMMC HAL Module Operational Overview

The SD/MMC Driver and the SDIO Driver are implemented on r_sdmmc and are used to read/write and control SD/MMC media devices as well as SDIO cards. The SD/MMC module can be used as a standalone driver, or it can be used with FileX or any other compatible file system.

- The SDMMC HAL Module supports the following memory devices: SDSC (SD Standard Capacity), SDHC (SD High Capacity), SDXC (SD Extended Capacity), and eMMC (embedded Multi Media Card).
  - Supports reading, writing, and erasing SD and eMMC memory device
  - Supports 1, 4, or 8-bit data bus (8-bit bus is supported for eMMC only)
  - Supports detection of hardware write protection (SD cards only)
  - Automatically selects between backwards compatible mode and high speed SRD mode (eMMC).
- Supports SDIO
  - Supports SDIO single register access (CMD52)
  - Supports SDIO multiple register access (CMD53)
  - Supports SDIO interrupts.
- Automatically configures the clock to the maximum clock rate supported by both host (MCU) and device.

Interrupt Configurations

When using FileX, sf_el_fx, and sf_block_media_sdmmc, the callback is implemented at the sf_block_media_sdmmc layer and does not have to be set up at the SDMMC module level. However, the interrupts and transfer drivers still must be configured.

The interrupts required are listed as follows:

- Using SD/MMC with DTC:
  - Access Interrupt
  - DTC Interrupt.
- Using SD/MMC with DMAC:
  - Access Interrupt
  - DMAC Interrupt (in DMAC instance).
- Using SDIO with DTC:
  - Access Interrupt
  - SDIO Interrupt
  - DTC Interrupt.
- Using SDIO with DMAC:
  - Access Interrupt
  - SDIO Interrupt
  - DMAC Interrupt (in DMAC instance).

The Card interrupt is optional, and is only available on MCU packages that have the SDHIn CD pin (n = channel number) available on the **Pins** tab of the Synergy Configuration Tool.

Block Size Configuration

Block size configuration is provided for use with SDIO cards only. For SDIO cards, block size may be configured to 1 - 512 bytes. Block size must remain at the default 512 bytes for SD cards and e/MMC memory devices.

Card Detection Configuration

See Card Detection Limitations before using card detection in the `r_sdmmc` driver. If card detection is not available or is not desired for the application, card detection must be set to Not Used in the Properties for the r_sdmmc instance in the Synergy Configuration tool. To enable card detection, set **Card Detection** to **CD Pin** and **Media Type** to **Card** in the Properties for the `r_sdmmc` instance in the Synergy Configuration tool.

Access Interrupt Priority

When data transfers are not 4-byte aligned or not a multiple of 4 bytes, a software copy of the block size (up to 512 bytes) is done in the access interrupt. This blocks other interrupts that are a lower or equal priority to the access interrupt until the software copy is complete.

General Timing Notes

Several functions in this driver block. `sdmmc_api_t::open()` and `sdmmc_api_t::erase()` block until the entire operation is complete. `sdmmc_api_t::read()`, `sdmmc_api_t::write()`, `sdmmc_api_t::readIo()`, `sdmmc_api_t::writeIo()`, `sdmmc_api_t::readIoExt()`, and `sdmmc_api_t::writeIoExt()` block for a single command response cycle. In a multi-threaded system, care should be taken to use this driver in a lower priority thread if other threads require a response time faster than the time this driver could block for, during one of these function calls.

Timing Notes for Open

The `sdmmc_api_t::open()` API completes the entire device identification and configuration process. This involves several command-response cycles at a bus width of 1-bit and a bus speed of 400 kHz or less.

Timing Notes for Read, Write, ReadIoExt and WriteIoExt

The read and write media (`sdmmc_api_t::read()` and `sdmmc_api_t::write()`) and `extended read` and `write` SDIO APIs (`sdmmc_api_t::readIoExt()` and `sdmmc_api_t::writeIoExt()`) block until the response is received from the device. The data transfer operation is non-blocking and requires interrupts and a transfer instance, either DMAC or DTC. These APIs return SSP_SUCCESS to indicate that the initial operations have started successfully. The application must wait for a callback with the event SDMMC_EVENT_TRANSFER_COMPLETE or SDMMC_EVENT_TRANSFER_ERROR to indicate completion of the read or write.

Timing Notes for ReadIo and WriteIo

The SDIO `sdmmc_api_t::readIo()` and `sdmmc_api_t::writeIo()` APIs block until the response is received from the device. The read or write operation is complete when these APIs return.

Timing Notes for Erase

The `sdmmc_api_t::erase()` API blocks until the erase operation is complete. This may be several seconds or more depending on how many sectors are being erased.

SDIO Interrupts

Many SDIO cards use level interrupts, meaning that the interrupt is not deasserted until the interrupt is cleared on the device. In order to ensure that SDIO interrupts are cleared appropriately, one of the following methods are recommended:

- Ensure that the SDIO interrupt is cleared on the device before exiting the callback function with the event SDMMC_EVENT_SDIO. If this method is chosen and any SDIO API must be used in the interrupt, the access interrupt must be a higher priority than the SDIO interrupt.
- Disable the SDIO interrupt in the callback function with the event SDMMC_EVENT_SDIO using `sdmmc_api_t::IoIntEnable()`. Clear the SDIO interrupt elsewhere in the application, then re-enable SDIO interrupts if desired using `sdmmc_api_t::IoIntEnable()`.

4.1 SDMMC HAL Module Operational Notes

SD HALA Compliance

When developing host devices that are compliant with the SD specifications, the host must comply with the SD Host/Ancillary Product License Agreement (SD HALA).

4.2 SDMMC HAL Module Limitations

Data Alignment and Size

Data transfers should be 4-byte aligned and a multiple of 4 bytes in size whenever possible. This recommendation applies to the `read()`, `write()`, `readIoExt()`, and `writeIoExt()` APIs. When data transfers are 4-byte aligned and a multiple of 4-bytes, the r_sdmmc driver is zero copy and takes full advantage of hardware acceleration by the DMAC or DTC. When data transfers are not 4-byte aligned or not a multiple of 4 bytes, an extra CPU interrupt is required for each block transferred (see Access Interrupt Priority).

Card Detection Limitations

Card detection support in the r_sdmmc driver is limited. Card detection is only available after `sdmmc_api_t::open()` is complete, and `open()` cannot be completed unless a card is inserted. Card detection in the r_sdmmc driver is therefore only useful to detect card removal and reinsertion. After reinsertion is detected, the driver must be closed and reopened, and card detection will not be available until the reopen is complete. Card detection is available only on MCU packages that have the SDHIn CD pin (n =

channel number) available on the **Pins** tab of the Synergy Configuration tool. If the MCU does not have the SDHIn CD pin, or card detection is not desired for the application, card detection must be disabled in the Properties for the r_sdmmc instance in the Synergy Configuration tool. An alternative to using the card detection feature of the r_sdmmc driver is to use an External IRQ instance and handle card detection at the application layer. If card detection is handled at the application layer, the `sdmmc_api_t::open()` should be called after card insertion is detected, and the `sdmmc_api_t::close()` should be called after card removal is detected.

Other Limitations

Refer to the most recent *SSP Release Notes* for the most up to date limitations for this module.

5. Including the FileX Port Block Media Framework Module in an Application

This section describes how to include the FileX Port Block Media Framework module in an application using the SSP configurator.

Note: This section assumes that you are familiar with creating a project, adding threads, adding a stack to a thread and configuring a block within the stack. If you are unfamiliar with any of these items, refer to the first few chapters of the *SSP User's Manual* to learn how to manage each of these important steps in creating SSP-based applications.

To add the FileX Port Block Media Framework module to an application, simply add it to a thread using the stacks selection sequence given in the following table. (The default name for the FileX Port Block Media Framework is g_sf_el_fx. This name can be changed in the associated Properties window.)

Table 2. FileX Port Block Media Framework Port Selection Sequence

Resource	ISDE Tab	Stacks Selection Sequence
g_sf_el_fx0 FileX Port Block Media Framework on sf_el_fx	Threads	New Stack> Framework > File System> FileX Port Block Media Framework on sf_el_fx

When the FileX Port Block Media Framework module on sf_el_fx is added to the thread stack as shown in the following figure, the configurator automatically adds any needed lower-level modules. Any drivers that need additional configuration information have text highlighted in Red. Modules with a Gray band are individual modules that stand alone. Modules with a Blue band are shared or common and need only be added once and can be used by multiple stacks. Modules with a Pink band can require the selection of lower-level drivers; these are either optional or recommended (this is indicated in the block with the inclusion of this text.) If the addition of lower-level drivers is required, the module description includes **Add** in the text. Clicking on any Pink banded modules brings up the **New** icon and then displays the possible choices.

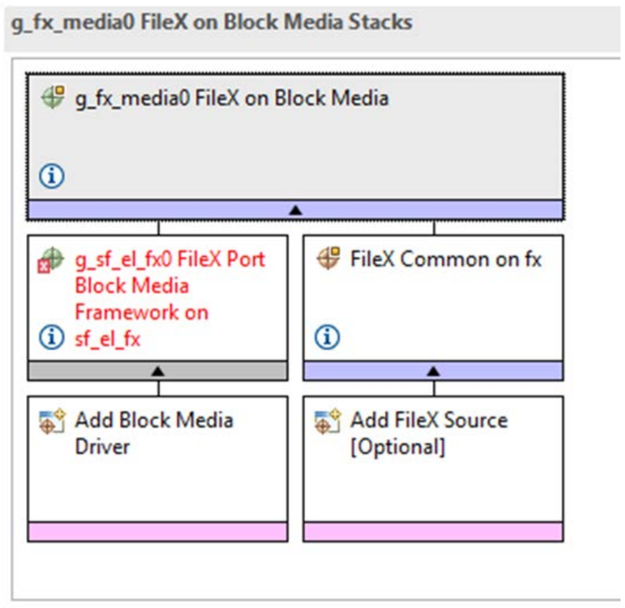

g_fx_media0 FileX on Block Media Stacks
g_fx_media0 FileX on Block Media
g_sf_el_fx0 FileX Port Block Media Framework on sf_el_fx
FileX Common on fx
Add Block Media Driver
Add FileX Source [Optional]

Figure 2. FileX Port Block Media Framework Module Stack

6. Configuring the FileX Port Block Media Framework Module

You must configure the FileX Port Block Media Framework module for the desired operation. The SSP configuration window automatically identifies (by highlighting the block in red) any required configuration selections, such as interrupts or operating modes, which must be configured for lower-level modules for successful operation. Furthermore, only those properties that can be changed without causing conflicts are available for modification. Other properties are 'locked' and not available for changes and are identified with a lock icon for the 'locked' property in the **Properties** window in the ISDE. This approach simplifies the configuration process and makes it much less error-prone than previous 'manual' approaches to configuration. The available configuration settings and defaults for all the user-accessible properties are given in the **Properties** tab within the SSP configurator, and are shown in the following tables for easy reference.

One of the properties most often identified as requiring a change is the interrupt priority. This configuration setting is available within the Properties window of the associated module. Simply select the indicated module and then view the **Properties** window. The interrupt settings are often toward the bottom of the properties list, so scroll down until they become available. Also note that the interrupt priorities listed in the **Properties** window in the ISDE include an indication as to the validity of the setting based on the MCU targeted (CM4 or CM0+). This level of detail is not included in the following configuration properties tables, but is easily visible with the ISDE when configuring interrupt-priority levels.

Note: You may want to open your ISDE, create the module, and explore the property settings in parallel with looking over the following configuration table settings. This will help orient you and can be a useful 'hands-on' approach to learning the ins and outs of developing with SSP.

Table 3. Configuration Settings for the FileX Port Block Media Framework Module

ISDE Property	Value	Description
Parameter Checking	Enabled, Disabled, BSP Default: BSP	Selects if code for parameter checking is to be included in the build
Name	Default: g_sf_el_fx0	FileX Port Block Media framework Module name

Note: The example values and defaults are for a project using the Synergy S7G2 MCU Group. Other MCUs may have different default values and available configuration settings.

Table 4. Configuration Settings for the Block Media Framework Module on sf_block_media_sdmmc

ISDE Property	Value	Description
Parameter Checking	Enabled, Disabled, BSP Default: BSP	Selects if code for parameter checking is to be included in the build
Name	Default: g_sf_block_media_sdmmc0	Block Media framework Module name
Block size of media in bytes	512	Media Block size

Note: The example values and defaults are for a project using the Synergy S7G2 MCU Group. Other MCUs may have different default values and available configuration settings.

In some cases, settings other than the defaults for lower level modules can be desirable. For example, the bus width can be different for different media types. The configurable properties for the lower-level stack modules are given in the below sections for completeness and as a reference.

Note: Most of the property settings for lower-level modules are intuitive and usually can be determined by inspection of the associated Properties window from the SSP configurator.

6.1 Configuration Settings for the SD/MMC Driver and SDIO Driver

Typically, only a small number of settings must be modified from the default for lower level drivers and these are indicated with red text in the Thread Stack block. Notice that some of the configuration properties must be set to a certain value for proper framework operation and is locked to prevent user modification. The table below identifies all the settings within the properties section for the module.

Table 5. Configuration for the SD/MMC and SDIO Driver

ISDE Property	Setting	Description
Parameter Checking	Enabled, Disabled, BSP Default: BSP	Selects if code for parameter checking is to be included in the build
Name	Default: g_sdmmc0	Module name
Channel	0,1 Default: 1	Channel of SDMMC peripheral, channel 0 or 1
Media type	Embedded, Card Default: Embedded	Media is a card or an embedded device. This allows the firmware to know whether to look for card insertion/removal and write protect pins.
Bus Width	1 bit, 4 bits, 8 bits	Data bus as defined by hardware interface (8 bits for eMMC only)
Block Size	512	Size of block
Callback	NULL (default) Name of user callback function.	(Not required if using FileX) Set to name of user callback function. Provides event that caused interrupt: SDMMC_EVENT_CARD_REMOVED, SDMMC_EVENT_CARD_INSERTED, SDMMC_EVENT_ACCESS, SDMMC_EVENT_SDIO, SDMMC_EVENT_TRANSFER_COMPLETE, SDMMC_EVENT_TRANSFER_ERROR
Access Interrupt Priority	Priority level 0-15 Default: Disabled	Interrupt level priority required for reading, writing, and errors for SD, eMMC, and SDIO. Calls the interrupt callback if enabled.
Card Interrupt Priority	Priority level 0-15 Default: Disabled	(Optional) Interrupt for card removal. Automatically closes device when card is removed. Calls the interrupt callback if enabled.
DMA Request Interrupt Priority	Priority level 0-15 Default: Disabled	Interrupt priority level required for SDIO reading, writing, and errors. Calls the interrupt callback if enabled. Not used for memory cards.

Note: The example values and defaults are for a project using the Synergy S7 Family MCUs. Other MCUs may have different default values and available configuration settings.

6.2 FileX Block Media Framework Module Clock Configuration

The SDHI uses PCLKA for its clock source. There is no need to configure the clock specifically for the SDMMC peripheral, unless you need to optimize the data rate. The SDMMC driver selects the appropriate built-in divider based on the PCLKA frequency and the maximum clock rate allowed by the SD, SDIO, or eMMC device, obtained at media device initialization.

6.3 FileX Block Media Framework Module Pin Configuration

Use the e^2 studio pin configurator to configure the I/O pins for SDMMC peripheral. The drive capacity for each pin should be set to "Medium" or "High" for most boards and high-speed memory and SDIO devices. The following tables illustrate the method for selecting the pins within the SSP configuration window and provide an example selection for the module pins.

Note: The operation mode selection mode determines what peripheral signals are available and thus what MCU pins are required.

Table 6. Pin Selection Sequence for SDHI Peripheral

Resource	ISDE Tab	Pin selection Sequence
SDHI	Pins	Select **Peripherals > Storage:SHDI > SDHI0**

Note: The above selection sequence assumes that SDHI0 is the desired hardware target for the driver.

Table 7 Pin Configuration Settings for SDHI Peripheral

Pin Configuration Property	Settings	Description
Operation Mode	Disabled, Custom, SD_MMC 1 bit, SD_MMC 4 bit, MMC 8 bit Default: Custom	Select mode according to the application
CLK	None, P413 Default: P413	Clock pin
CMD	None, P412 Default: P412)	Command pin
DAT0-7	None, PXXX Default: PXXX	Data pin
CD	None, P903 Default: P903	Card detection pin
WP	None, P414 Default: P414	Card write protection pin

Note: The above example values are for a project using the Synergy S7G2 MCU and the DK-S7G2 Kit. Other Synergy Kits and other Synergy MCUs may have different available pin configuration settings.

6.4 FileX Block Media Framework Module Other Settings

The read and write media (R_SDMMC_Read and R_SDMMC_Write) and extended read and write SDIO functions (R_SDMMC_ReadIoExt and R_SDMMC_WriteIoExt) have been made non-blocking in version 1.1.0 and now require interrupts and a transfer function such as DMAC or DTC. The read and write functions return SSP_SUCCESS to indicate that the initial operations have started successfully. However, the user application must wait for the user callback and check for event SDMMC_EVENT_TRANSFER_COMPLETE or SDMMC_EVENT_TRANSFER_ERROR to indicate completion of the read or write.

7. Using the FileX Port Block Media Framework Module in an Application

The typical steps in using the FileX Port Block Media Framework module on sf_el_fx in an application are:

1. Initialize the media using the FileX API `fx_system_initialize` (sf_el_fx calls it automatically if “Auto Initialization” property is set to **Enabled** in **FileX Common on fx**).
2. Optionally, format the media using the FileX API `fx_media_format.`
   - sf_el_fx automatically formats the media during the generated initialization function if **Format media during initialization** property is set to **Enabled**).
   - If **Filesystem on SDMMC** is enabled, the entire media after the hidden sectors is formatted and the **Total Sectors** configuration in **FileX on Block Media** is ignored.
3. Open the media using the FileX API `fx_media_open` (sf_el_fx opens the media automatically if **Auto Initialization** is set to **Enabled** in the FileX on Block Media instance).
4. Create and delete files and directories as required using one of the FileX APIs (for example, `fx_file_create`, `fx_file_delete`, `fx_directory_create`, `fx_directory_delete`).
5. Read from and write to files on the media as required using one of the FileX API (for example, `fx_file_read` or `fx_file_write`).
6. Read from and write to the media directly as required using one of the FileX API (for example, `fx_media_read` or fx_`media_write`).
7. Close the physical media using the FileX API, `fx_media_close.`

Note: After a successful `fx_media_open` call, all FileX file- and directory-related APIs can be used. Refer to the *FileX User Guide* for documentation on all available functions.

These common steps are illustrated in a typical operational flow diagram in the following figure.

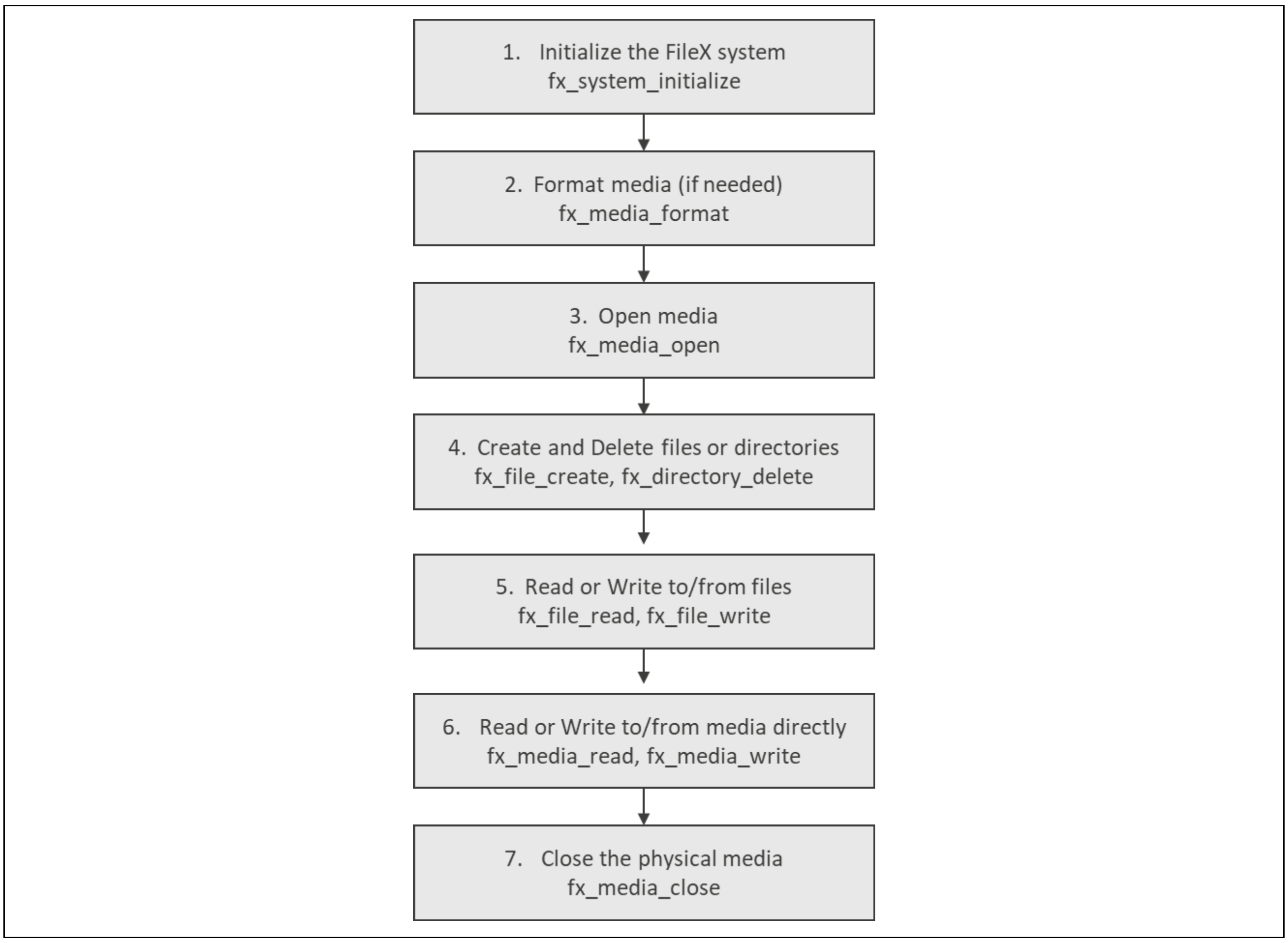

1. Initialize the FileX system
fx_system_initialize
2. Format media (if needed)
fx_media_format
3. Open media
fx_media_open
4. Create and Delete files or directories
fx_file_create, fx_directory_delete
5. Read or Write to/from files
fx_file_read, fx_file_write
6. Read or Write to/from media directly
fx_media_read, fx_media_write
7. Close the physical media
fx_media_close

Figure 3. Flow Diagram of a Typical FileX Port Block Media Application

The typical steps for using the r_sdmmc driver with an SD or eMMC memory device are:

1. Open the driver using `sdmmc_api_t::open()`.
2. Read data from the card using `sdmmc_api_t::read()` or write data to the card using `sdmmc_api_t::write()`.
3. Wait for a callback with the event SDMMC_EVENT_TRANSFER_COMPLETE (which indicates that the operation completed successfully) or SDMMC_EVENT_TRANSFER_ERROR (which indicates that the operation did not complete successfully) after each time `sdmmc_api_t::read()` or `sdmmc_api_t::write()` is called.

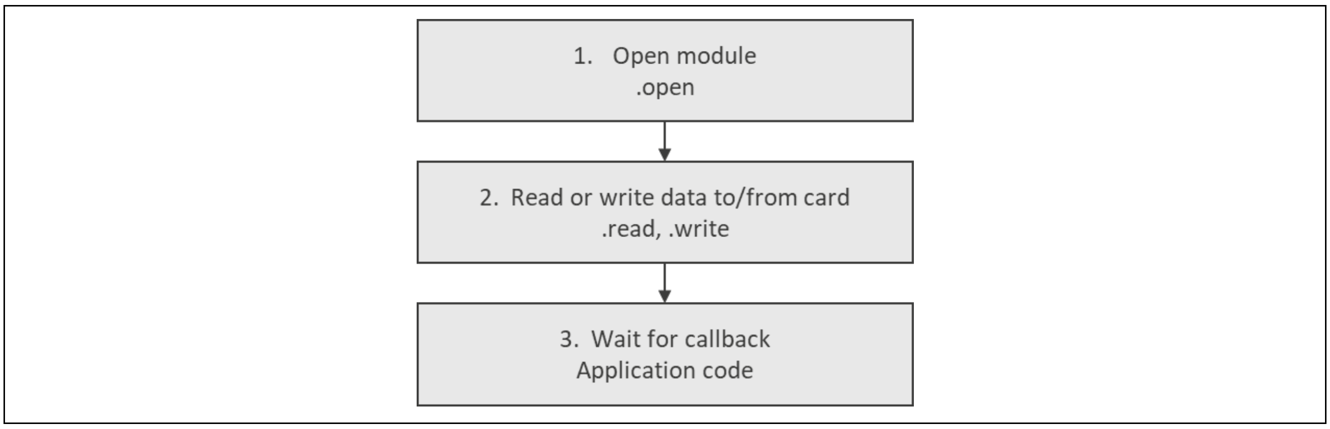


Figure 4. Flow Diagram of a Typical SDMMC SD or eMMC Application

The typical steps for using the r_sdmmc driver with an SDIO card are:

1. Open the driver using `sdmmc_api_t::open()`.
2. Set or read back single registers using `sdmmc_api_t::readIo()` or `sdmmc_api_t::writeIo()`. The operation is complete immediately after these calls and there is no need to wait for a callback.
3. Set or read back multiple registers using `sdmmc_api_t::readIoExt()` or `sdmmc_api_t::writeIoExt()`. The block size can be changed using `sdmmc_api_t::control()` between calls if necessary.
4. Wait for a callback with the event SDMMC_EVENT_TRANSFER_COMPLETE (which indicates the operation completed successfully) or SDMMC_EVENT_TRANSFER_ERROR (which indicates that the operation did not complete successfully) after each time `sdmmc_api_t::readIoExt()` or `sdmmc_api_t::writeIoExt()` is called.
5. If the card requests an interrupt, the callback is called with the event SDMMC_EVENT_SDIO. Handle the SDIO interrupt as described in the documentation for the card (see the SDIO Interrupts section). SDIO interrupts from the card can be enabled or disabled at any time using `sdmmc_api_t::IoIntEnable()`.

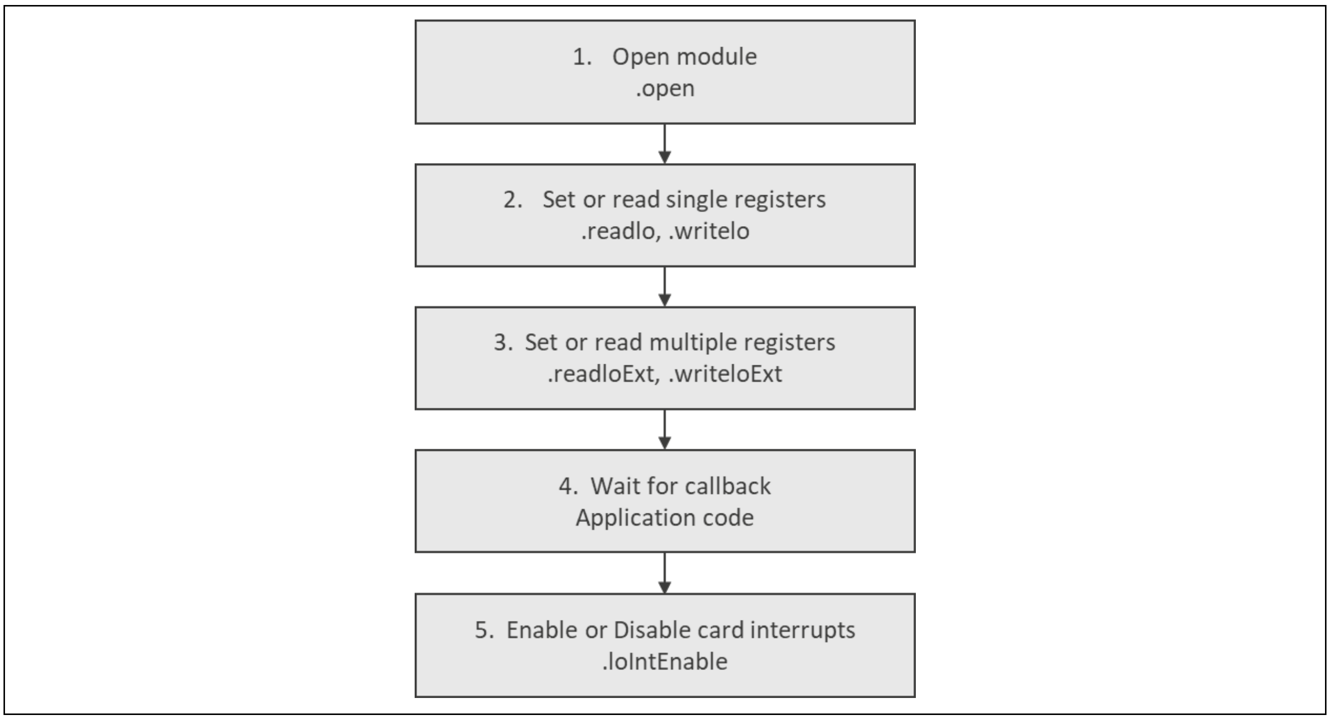


Figure 5. Flow Diagram of a Typical SDMMC SDIO Application

8. FileX Port Block Media Framework Module Application Project

The application project associated with this module guide demonstrates the aforementioned steps in a full design. The project can be found using the link provided in the References section at the end of this document. You may want to import and open the application project within the ISDE and view the configuration settings for the FileX Port Block Media Framework Module. You can also read over the code (in `sdmmc_thread_entry.c`), which is used to illustrate the FileX on Block Media APIs in a complete design.

The application project demonstrates the typical use of the FileX on Block Media Framework APIs. The application project auto-generated code initializes FileX framework using the FileX Port Block Media Framework. It is a driver which aims to implement FileX-specific accesses using the Block Media Framework. The underlying implementation of the Block Media Framework uses the SD/MMC Driver. The application project is designed to work with onboard eMMC memory, however, it is easy to change configuration settings to access the SD card. The following table identifies the target versions for the associated software and hardware used by the application project.

Table 8. Software and Hardware Resources Used by the Application Project

Resource	Revision	Description
e² studio	v7.3.0 or later	Integrated Solution Development Environment
SSP	v1.6.0 or later	Synergy Software Platform
IAR EW for Synergy	v8.23.3 or later	IAR Embedded Workbench® for Renesas Synergy™
SSC	v7.3.0 or later	Synergy Standalone Configurator
DK-S7G2	v3.0 to v4.1	Development Kit

A simple flow diagram of the application project is given in the following figure.

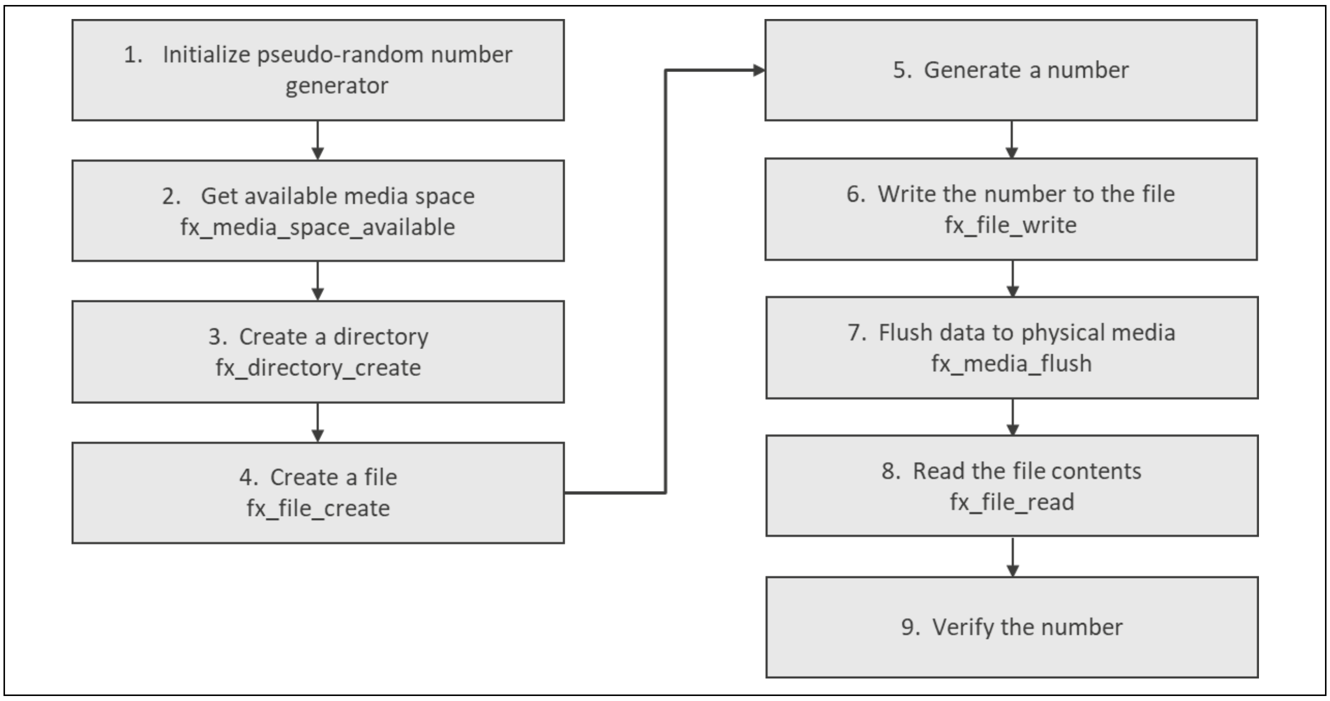

1. Initialize pseudo-random number generator
2. Get available media space fx_media_space_available
3. Create a directory fx_directory_create
4. Create a file fx_file_create
5. Generate a number
6. Write the number to the file fx_file_write
7. Flush data to physical media fx_media_flush
8. Read the file contents fx_file_read
9. Verify the number

Figure 6. FileX Port Block Media Framework Module Application Project Flow Diagram

The complete application project can be found using the link provided in the References section at the end of this document. The `sdmmc_thread_entry.c` file is located in the project once it has been imported into the ISDE. You can open this file within the ISDE and follow along with the description provided to help identify key uses of APIs.

The first section of `sdmmc_thread_entry.c` has the header files which reference the FileX Media structure and a code section which allows semi-hosting to display results using `printf()`. The next section contains macro constants and variable definitions. Then, there are function prototypes. The first function generates a pseudo-random number. Then, two functions for two-way conversion between integers and strings are defined. There is also simple code section for error handling. If the semi-hosting is enabled, it displays the error code. The next function is used for writing a number to a file. At first, the number is converted into a string. Then it accesses `FileX` APIs in order to open, clear, write, close the file, and flush data to physical media. The last step is necessary because the FileX operations are buffered, and not all recent changes might be reflected after reset. Alternatively, FileX flushes data when `fx_media_close` is called. The following function is very similar to the previous one, except that it is used for reading a number from given file instead of performing a write.

The last section is the thread entry function section. At the beginning, pseudo-random number generator is initialized using a predefined number. If the semi-hosting is enabled, the available space is displayed. Then, the application needs to create a file, but a directory is created first. Then the software enters the 'forever' while loop, and it starts with generating a number. This number is then written to the previously created file. Afterwards, the number is read and verified with the generated one. In case of an error, the message is provided. Then, a thread sleep function pauses execution for several ThreadX timer ticks and the while loop functions are repeated.

Note: The above description assumes that you are familiar with using `printf()` with the Debug Console in the Synergy Software Package. If you are unfamiliar with this, refer to the **How do I Use Printf() with the Debug Console in the Synergy Software Package** given in the Reference Section at the end of this document. Alternatively, the user can see results using the watch variables in the debug mode.

A few key properties are configured in this Application Project to support the required operations and the physical properties of the target board and MCU. Following are the properties with the values set for this specific project. You can also open the application project and view these settings in the property window as a hands-on exercise.Threads can be found under **X-Ware > FileX > FileX** on block media.

Table 9. FileX Port Block Media Framework Configuration Settings for the Application Project

ISDE Property	Value Set
Name	g_fx_media
Format media during initialization	Enabled
File System is on block media	True
Volume Name	Volume 1
Number of FATs	1
Directory Entries	256
Hidden Sectors	0
Total Sectors	500,000
Bytes per Sector	512
Sectors per Cluster	1
Working media memory size	512

Table 10. SD/MMC Driver on r_sdmmc Configuration Settings for the Application Project

ISDE Property	Value Set
Parameter Checking Enable	Default (BSP)
Name	g_sdmmc
Channel	0
Media Type	Embedded
Bus Width	8 Bits
Block Size	512
Card Detection	Not Used
Callback	NULL
Access Interrupt Priority	Priority 8 (CM4: valid, CM0+: invalid)
Card Interrupt Priority	Disabled
DMA Request Interrupt Priority	Disabled

Table 11. Transfer Driver on r_dmac Configuration Settings for the Application Project

ISDE Property	Value Set
Parameter Checking Enable	Default (BSP)
Name	g_transfer0
Channel	0
Mode	Normal
Transfer Size	1 Byte
Destination Address Mode	Fixed
Source Address Mode	Incremented
Repeat Area (Unused in Normal Mode)	Source
Destination Pointer	NULL
Source Pointer	NULL
Number of Transfers	0
Number of Blocks (Valid only in Block Mode)	0

ISDE Property	Value Set
Activation Source	Software Activation
Auto Enable	False
Callback	NULL
Interrupt Priority	Priority 8 (CM4: valid, CM0+: invalid)

9. Customizing the FileX Port Block Media Framework Module for a Target Application

Some configuration settings are normally changed from those shown in the application project, by the developer. For example, the user can easily change the configuration settings for the FileX instance, especially the formatting options. The user can also change the Media Type setting to Card.

10. Running the FileX Port Block Media Framework Module Application Project

To run the FileX Port Block Media Framework application project and to see it executed on a target kit, you can simply import it into your ISDE, compile, and run debug.

To implement the FileX Port Block Media Framework application in a new project, follow the steps for defining, configuring, auto-generating files, adding code, compiling, and debugging on the target kit. Following these steps is a hands-on approach that can help make the development process with SSP more practical, while just reading over this guide tends to be more theoretical.

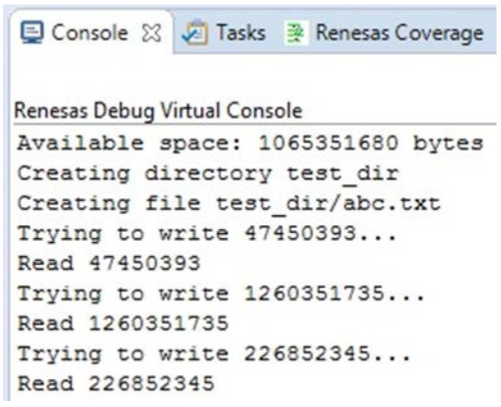
Note: The following steps are described in sufficient detail for someone experienced with the basic flow through the Synergy development process. If these steps are not familiar, refer to the first few Chapters in the *SSP User's Manual*, as described in the Reference Section at the end of this document.

To create and run the FileX Port Block Media Framework application project, simply follow these steps:

1. Create a new Renesas Synergy project for the DK-S7G2 board called SDMMC_HAL_FRWK_FX_MG_AP.
2. Select the **Threads** tab.
3. Add a new thread as follows:

— Symbol	SDMMC Thread
— Name	sdmmc_thread

4. Add the FileX on Block Media to the SDMMC Thread.
5. Click on the **Generate Project Content** button.
6. Add the code from the supplied project files "`sdmmc_thread_entry.c`" and "`semihosting_cfg.h`" or copy over the generated "`sdmmc_thread_entry.c`" and "`semihosting_cfg.h`" files.
7. Connect to the host PC using a micro USB cable to J17 on DK-S7G2. You may need to set the MMC switch to "On" and the SD switch to "Off" on S101 for the DK-S7G2 v3.1/v3.0 and "SD Socket" switch on S8 for DK-S7G2 v4.1.
8. Start to debug the application.
9. The output can be viewed, after a wait (The FileX media format needs 4-5 mins for formatting the specified sectors in the configuration), on the debug console (Renesas Debug Virtual Console).



```
Available space: 1065351680 bytes
Creating directory test_dir
Creating file test_dir/abc.txt
Trying to write 47450393...
Read 47450393
Trying to write 1260351735...
Read 1260351735
Trying to write 226852345...
Read 226852345
```

Figure 7. Example Output from FileX Port Block Media Framework Application Project

11. FileX Port Block Media Framework Module Conclusion

This module guide has provided all the background information needed to select, add, configure, and use the module in an example project. Many of these steps were time consuming and error-prone activities in previous generations of embedded systems. The Renesas Synergy Platform makes these steps much less time consuming and removes the common errors like conflicting configuration settings or the incorrect selection of lower-level modules. The use of high level APIs (as demonstrated in the application project) illustrates additional development time savings by allowing work to begin at a high level and avoiding the time required in older development environments to use, or, in some cases, create, lower-level drivers.

12. FileX Port Block Media Framework Module Next Steps

After you have mastered a simple FileX Port Block Media Framework project, you may want to review a more complex example. You may find that the USBX Device Class Mass Storage is a better fit for your target application. The USBX Device Class Mass Storage Module Guide illustrates the use of the mass storage device and can be easily adopted to work with FileX. This guide is available at the link shown in the References section at the end of this document.

13. FileX Port Block Media Framework Module Reference Information

SSP User Manual: Available in html format in the SSP distribution package and as a pdf from the Synergy Gallery.

Links to all the most up-to-date sf_el_fx module reference materials and resources are available on the Synergy Knowledge Base: https://en-us.knowledgebase.renesas.com/English_Content/Renesas_Synergy%E2%84%A2_Platform/Renesas_Synergy_Knowledge_Base/SF_EL_FX_Module_Guide_Resources.

Website and Support

Visit the following vanity URLs to learn about key elements of the Synergy Platform, download components and related documentation, and get support.

Synergy Software	www.renesas.com/synergy/software
Synergy Software Package	www.renesas.com/synergy/ssp
Software add-ons	www.renesas.com/synergy/addons
Software glossary	www.renesas.com/synergy/softwareglossary
Development tools	www.renesas.com/synergy/tools
Synergy Hardware	www.renesas.com/synergy/hardware
Microcontrollers	www.renesas.com/synergy/mcus
MCU glossary	www.renesas.com/synergy/mcuglossary
Parametric search	www.renesas.com/synergy/parametric
Kits	www.renesas.com/synergy/kits
Synergy Solutions Gallery	www.renesas.com/synergy/solutionsgallery
Partner projects	www.renesas.com/synergy/partnerprojects
Application projects	www.renesas.com/synergy/applicationprojects
Self-service support resources:	
Documentation	www.renesas.com/synergy/docs
Knowledgebase	www.renesas.com/synergy/knowledgebase
Forums	www.renesas.com/synergy/forum
Training	www.renesas.com/synergy/training
Videos	www.renesas.com/synergy/videos
Chat and web ticket	www.renesas.com/synergy/resourcelibrary

Revision History

Rev.	Date	Description	
		Page	Summary
1.00	Sep.12.17	-	Initial Release
1.01	Apr.30.19	-	Added switch information for new S7G2 v4.1 board. Updated for SSP v1.6.0.

Corporate Headquarters

TOYOSU FORESIA, 3-2-24 Toyosu,
Koto-ku, Tokyo 135-0061, Japan
www.renesas.com

Contact information

For further information on a product, technology, the most up-to-date version of a document, or your nearest sales office, please visit:
www.renesas.com/contact/.

Trademarks

Renesas and the Renesas logo are trademarks of Renesas Electronics Corporation. All trademarks and registered trademarks are the property of their respective owners.