

Renesas Synergy™ Platform

Console 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 Console Framework is a complete API implementation for a menu-driven console command line interface (CLI) using the ThreadX® RTOS. The Console Framework module uses a lower-level communications interface which connects to a hardware option for either UART, USB, or Ethernet Telnet connectivity. The Console Framework module has a user-defined menu of commands and various APIs to present a prompt, identify and issue a callback for menu commands, and read, write and parse input strings.

Contents

1. Console Framework Module Features	3
2. Console Framework Module APIs Overview	3
3. Console Framework Module Operational Overview	5
3.1 Console Framework Module Initialization	5
3.2 Console Framework Module Input Processing	5
3.3 Creating Console Framework Module Required Structures – The Menu	5
3.4 Console Framework Module Important Operational Notes and Limitations.....	6
3.4.1 Console Framework Module Operational Notes	6
3.4.2 Console Framework Module Limitations	6
4. Including the Console Framework Module in an Application	7
5. Configuring the Console Framework Module	9
5.1 Configuration Settings for the Console Framework Stack Modules.....	10
5.1.1 Configuration Settings for the Telnet Stack Modules	10
5.1.2 Configuration Settings for the USB Stack Modules.....	12
5.1.3 Configuration Settings for the UART Stack Modules	15
5.1.4 Configuration Settings for the USB Stack Modules (Deprecated)	19
6. Using the Console Framework Module in an Application	20
7. Console Framework Module Application Project.....	21
8. Customizing the Console Framework Module for a Target Application	24
8.1 Menu Structures	24
8.2 Thread Entry Name	24
8.3 Instance Name	24

8.4 Communications Framework Module Selection and Configuration 24

9. Running the Console Framework Module Application Example25

10. Console Framework Module Conclusion.....25

11. Console Framework Module Next Steps.....25

12. Console Framework Module Reference Information26

Revision History28

1. Console Framework Module Features

The console framework supports the following features:

- Creation of a menu-based command-line interface
- Submenus and navigation through multiple menus in a single call
- Menu navigation to go up to the parent menu or back to the root
- A help menu for each menu
- Writing NULL terminated strings and reading until return character is received
- An API to help parse arguments to the command line
- Case-insensitive inputs

The Console Framework module organization, as depicted in the thread stack window in the SSP Configurator, is shown in the following figure. Each implementation choice, Ethernet, UART, and USB has its own lower-level modules that are added automatically based on your implementation choice. In most cases, all the needed configuration information is automatically added to the modules leaving you with just a few important configuration settings that need to be selected.

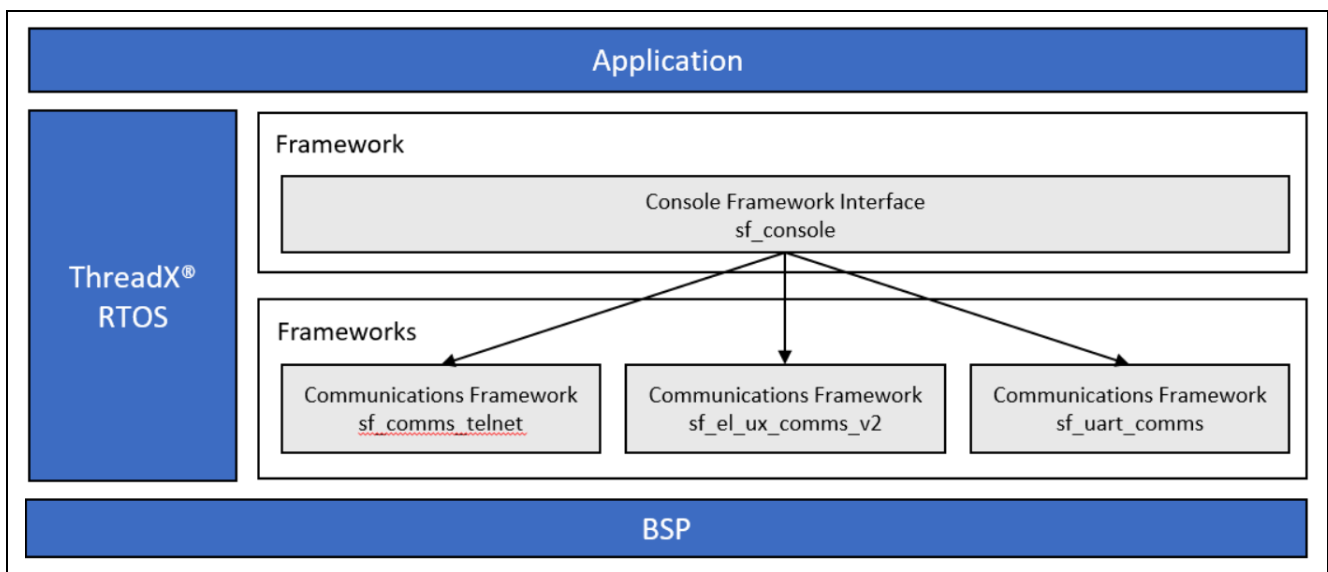


Figure 1. Console Framework Module Block Diagram

2. Console Framework Module APIs Overview

The Console Framework defines APIs for opening, closing, reading, writing, and issuing an input prompt. It also provides some additional functions, such as `parse` and `argumentFind`, to assist in processing more complex commands. A complete list of the available APIs, an example API call, a short description of each can be found in the following tables. A table of status return values follows the API summary table.

Table 1. Console Framework Module API Summary

Function Name	Example API Call and Definition
.open	<pre>g_sf_console0.p_api->open(g_sf_console0.p_ctrl, g_sf_console0.p_cfg)</pre> <p>The <code>open</code> API configures the console. This function must be called before any other console functions.</p> <p>Note: This call is made automatically during system initialization, prior to entering the user thread. Unless the user closes the console, <code>open</code> will not need to be called.</p>
.close	<pre>g_sf_console0.p_api->close(g_sf_console0.p_ctrl);</pre> <p>The <code>close</code> API handles the clean-up of internal driver data.</p>
.prompt	<pre>g_sf_console0.p_api->prompt(g_sf_console0.p_ctrl, NULL, TX_WAIT_FOREVER);</pre> <p>The Console Framework Module API prints the prompt string from the menu, waits for input, parses the input based on the menu, and calls the appropriate callback function if a command is identified.</p>
.parse	<pre>g_sf_console0.p_api->parse(g_sf_console0.p_ctrl, commands, input, s_length);</pre> <p>The <code>parse</code> API looks for an input string in the command menu and, if one is found, calls the appropriate callback function.</p>
.read	<pre>g_sf_console0.p_api->read(g_sf_console0.p_ctrl, ch, 1, TX_WAIT_FOREVER);</pre> <p>The <code>read</code> API puts data into the destination, byte-by-byte and echoes the input to the console. Backspace, delete, and left/right arrow keys are supported. Read completes when a line ending with CR, CR+LF, or CR+NULL is received, or when the input exceeds the number of bytes allowed. If the buffer overflows <code>SF_CONSOLE_MAX_INPUT_LENGTH</code>, the read will return an error code.</p>
.write	<pre>g_sf_console0.p_api->write(g_sf_console0.p_ctrl, (uint8_t*)data_string, TX_WAIT_FOREVER);</pre> <p>The <code>write</code> API gets the buffer mutex object and handles data transmission at the HAL layer. It obtains the event flag to synchronize the completion of a data transfer.</p>
.argumentFind	<pre>g_sf_console0.p_api->argumentFind("LED", p_args- >p_remaining_string, NULL, &led_num);</pre> <p>The <code>argumentFind</code> API locates a command line argument in an input string and returns the index of the character immediately following the argument. Any string numbers are converted to integers.</p>
.versionGet	<pre>g_sf_console0.p_api->versionGet(&version);</pre> <p>Retrieve the API version with the version pointer.</p>

Note: For more complete descriptions of operation and definitions for the function data structures, typedefs, defines, API data, API structures, and function variables, review the *SSP User's Manual API References* for the associated module.

Table 2. Status Return Values

Name	Description
SSP_SUCCESS	API call successful.
SSP_ERR_ASSERTION	p_ctrl is NULL.
SSP_ERR_UNSUPPORTED	Command not found in the current menu.

Note: Lower-level drivers may return common error codes. Refer to the *SSP User's Manual*, API References for the associated module for a definition of all relevant status return values.

3. Console Framework Module Operational Overview

The Console Framework module is a ThreadX®-aware Command Line Interface (CLI). The module uses ThreadX objects like mutex for blocking and synchronization techniques like event flags for the completion of a transaction. The key operational elements of the Console Framework are initialization and input processing, each of which are described in the following sections.

3.1 Console Framework Module Initialization

The `open` call is automatically generated by the ISDE and is in the `src/synergy_gen/toggle_thread.c` file where the module was added. The `open` call requires the application to define a root menu with a variable name that matches the one in the configurator (`g_sf_console_root_menu`) by default. By the time execution reaches `src/toggle_thread_entry.c`, the module is ready to use, provided the necessary hardware connection is established.

3.2 Console Framework Module Input Processing

The Console Framework module requires a set of menus, command structures, and callbacks. The Console Framework module typically operates from the prompt, often located within a while loop in the entry thread. The framework `prompt` API will print the current menu as a prompt, then read input and echo it back to the console (unless echo is disabled in the properties).

The following operations are performed against your input:

While the console is accepting input,

- Backspace will remove characters before the cursor.
- Delete will remove characters after the cursor.
- The left and right arrow keys move the cursor.
- The up-arrow key will fill in the last command only when nothing else has been entered.

Note: There is no history beyond the last command; if the up-arrow key is pressed twice, the console does not know what command was entered prior to the last command and it will continue to display the last command.

When the console sees a return character on the read input, it parses the input string and calls the associated callback or switches to the next menu if `SF_CONSOLE_CALLBACK_NEXT_FUNCTION` is used in place of the callback for the command. The console will continue parsing until a callback function is called. If the `prompt` API is called again, it will prompt using the menu that contains the callback function. To navigate up to the parent menu, enter '^'. To navigate to the root menu from any submenu, enter '~'.

3.3 Creating Console Framework Module Required Structures – The Menu

The Console Framework requires a menu and it is up to you to create the structure that is used by the Console Framework to implement the menu. The Console Menu structure (depicted in the following figure) includes a pointer to the previous menu, (for creating multi-level menus) a name for the menu, the number of commands in the menu, and a pointer to an array of command structures. As seen in the following figure, each entry in the array of commands includes pointers to the command name string, the `help` command description string, the associated command callback function, and a context parameter provided to the callback.

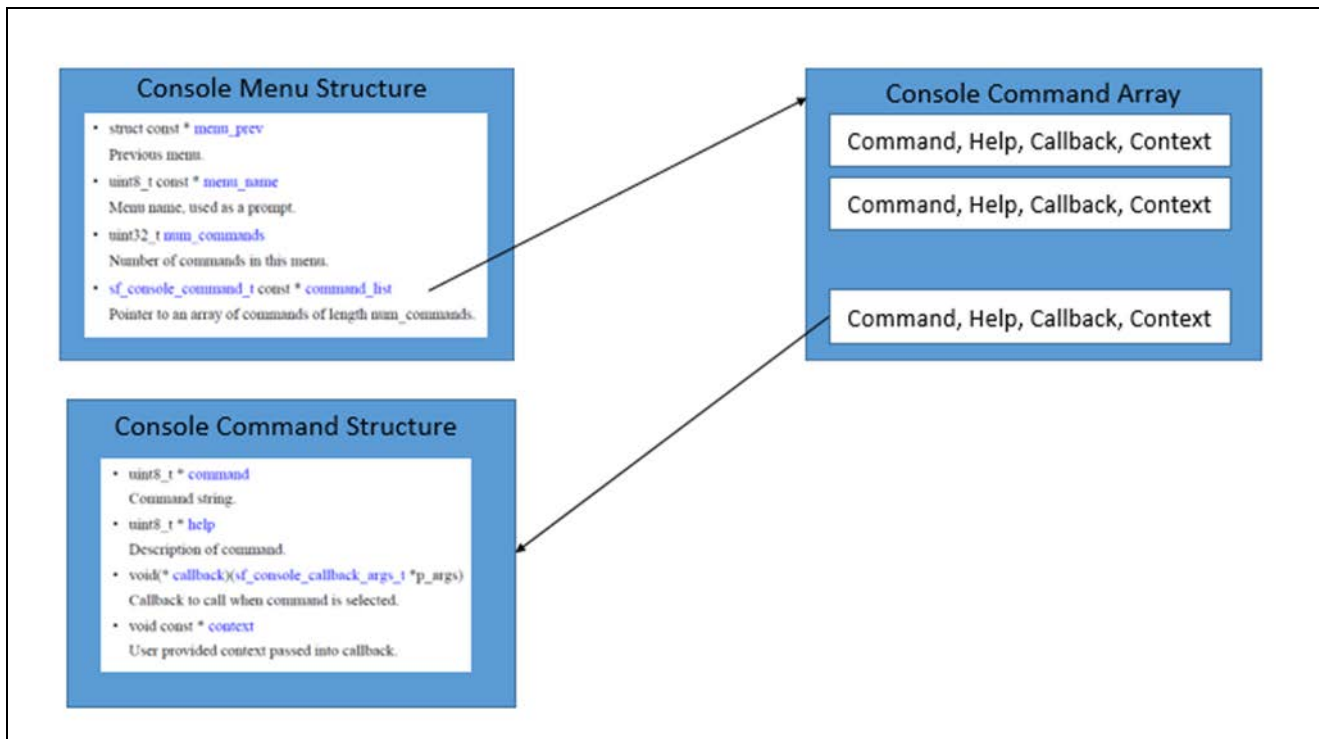


Figure 2. Console Framework Module Menu Structures

The application project example illustrates a Console Framework with a single menu; it controls the toggle of an LED from the CLI. The console command array (`g_sf_console_commands`, seen on the right in the following figure) stores the array of commands (in this case, just a single command.) The command structure defines the command as `LED TOGGLE`, the help description as `Toggle an LED`, the callback as `led_toggle_callback`, and the context as `NULL`, since it is unused for this example.

The root menu, seen on the left side of the following figure, is identified by the `g_sf_console_root_menu` structure. The structure defines the `menu_prev` entry as `NULL`, since there is only the single menu, the `menu_name` as **Root**, the `num_commands` as the size of the array divided by the size of an entry (to determine the total number of entries) as **1**, and the `command_list` starting address as **address**, the location of the first entry in the command array.

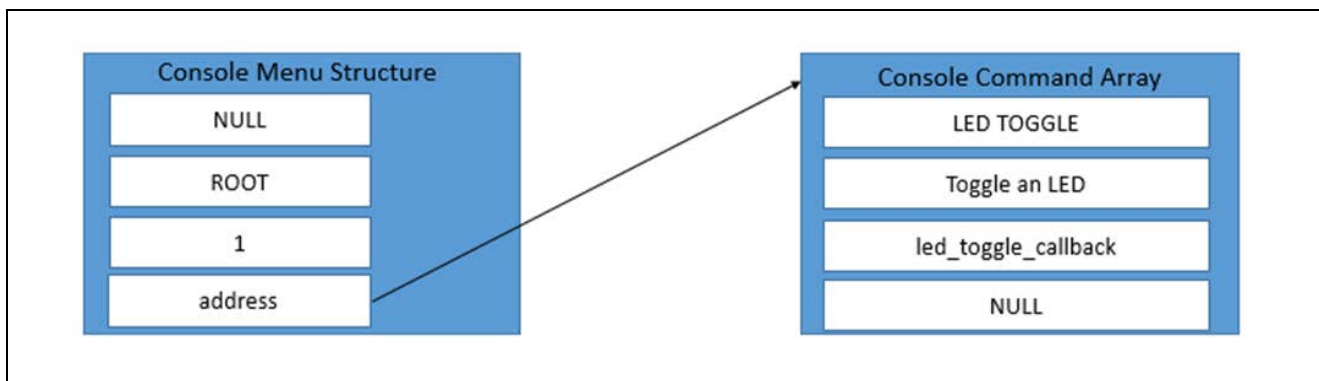


Figure 3. Menu Structure Example Diagram

3.4 Console Framework Module Important Operational Notes and Limitations

3.4.1 Console Framework Module Operational Notes

To use the Console Framework module `prompt` API, first set up the menu, command structures, and callbacks.

3.4.2 Console Framework Module Limitations

There are no known limitations for using this module.

Refer to the most recent *SSP Release Notes* for any additional operational limitations for this module.

4. Including the Console Framework Module in an Application

This section describes how to include the Console Framework module in an application using the SSP Configurator.

Note: This section assumes 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 Console Framework module to your application, simply add it to a project thread using the stacks selection sequence provided in the following table. (The default name for the console framework module is `g_sf_console0`. This name can be changed in the associated **Properties** window.)

Table 3. Console Framework Module Selection Sequence

Resource	ISDE Tab	Stacks Selection Sequence
g_sf_console0 Console Framework on sf_console	Threads	New Stack > Framework > Services > Console Framework on sf_console

When the console framework on `sf_console` is added to the thread stack as shown in the following figure, the configurator reports (via the Add Communications Framework block) that at least one Communications Framework is required to complete the Console Framework. The Communications Framework determines the type of communications interface (and the underlying hardware implementation) the Console Framework will use. Any low-level modules that need additional configuration information will have box 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 modules; these are either optional or recommended (this is indicated in the block with the inclusion of this text.) If the addition of lower-level modules is required, the module description will include **Add** in the text. Clicking on any **Pink** banded modules will bring up the **New** icon and then display the possible choices.

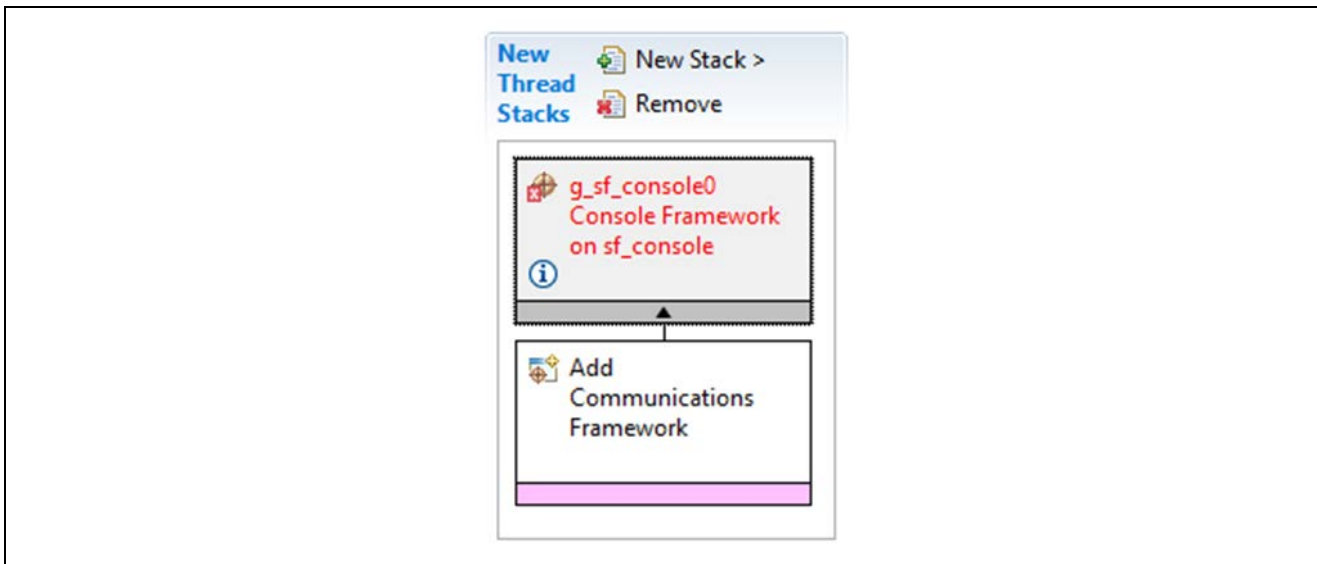


Figure 4. Console Framework Module Stack

Currently there are three possible Communications Frameworks available that the Console Framework can select: UART, USB, or Telnet. Configurations for each are shown in the following thread stack illustrations and they can be easily imported by right clicking the **Add Communications Framework** block and selecting the desired Communications Framework. (Note that the Telnet option uses a deprecated implementation.) This will function correctly, but is indicated as deprecated since it will be replaced in a future release. Designs using the deprecated module will need to be updated to the new implementation after that release.

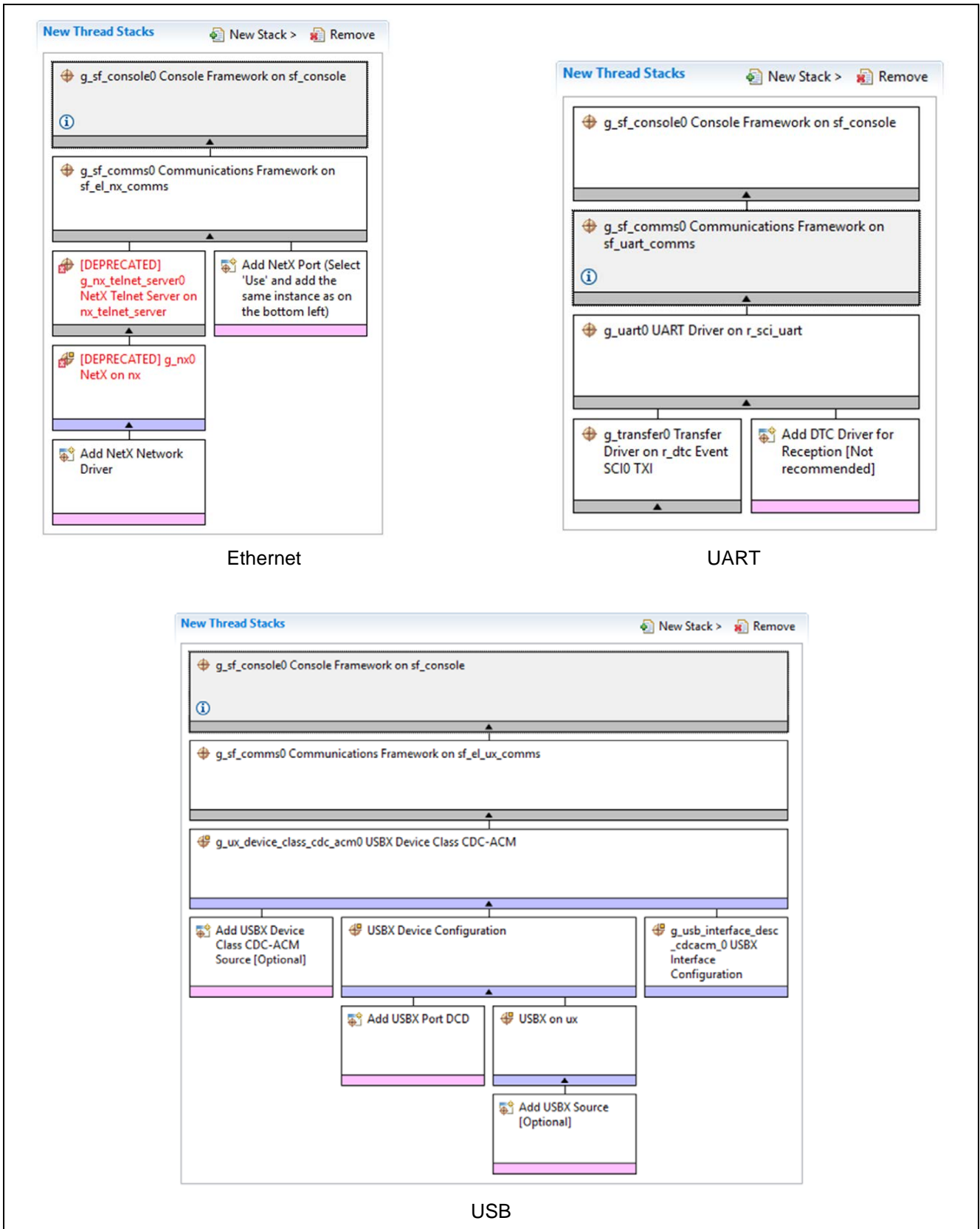


Figure 5. Communications Framework Module Options

Note: Other communications frameworks may be offered in future SSP releases. Refer to the current *SSP User Manual* for details.

5. Configuring the Console Framework Module

The Console Framework module must be configured by the user for the desired operation. The SSP configuration window will automatically identify (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 are 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. 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 will include an indication as to the validity of the setting based on the targeted MCU (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 configuration table settings in the following table. This will help orient you and can be a useful hands-on approach to learning the ins and outs of developing with SSP.

Table 4. Configuration Settings for the Console Framework Module on sf_console

Parameter	Value	Description
Parameter Checking	BSP, Enable, Disable Default: BSP	Selects if code for parameter checking is to be included in the build.
Maximum Input String Length	128	Input string length.
Maximum Write String Length	128	Output string length.
Name	g_sf_console0	Console Framework module name.
Name of Initial Menu (Application Defined)	g_sf_console_root_menu	Name of starting menu.
Echo	True, False Default: True	Enable or disable echo to terminal from the prompt.
Autostart	True, False Default: False	If True, the prompt will occur using the top-level menu after initialization. If False, the prompt needs to be called in the application.
Name of the sf_console Initialization Function	sf_console_init0	Name of the sf_console initialization function selection.
Auto sf_console Initialization	Enable, Disable Default: Enable	Auto sf_console initialization selection.

Note: The example values and defaults are for a project using the S7G2 Synergy MCU. 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 IP address for the Telnet connection or the baud rate of the UART might need to be modified. The configurable properties for the lower-level stack modules are displayed for completeness and as a reference.

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

5.1 Configuration Settings for the Console Framework Stack Modules

Typically, only a small number of settings must be modified from the default for lower-level modules as indicated via the 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 will be locked to prevent user modification. The following tables identify all the settings within the properties section for the module.

5.1.1 Configuration Settings for the Telnet Stack Modules

When the Telnet option is selected for the low-level implementation, the following modules are automatically added to the stack. The following tables show the detailed configuration settings available, many of which are populated with default settings that can be used in most common applications.

Table 5. Configuration Settings for the Telnet Option (sf_el_nx_comms)

ISDE Property	Value	Description
Parameter Checking	BSP, Enabled, Disabled Default: BSP	Enable or disable the parameter checking.
Name	g_sf_comms0	Module name.
Channel	0	Underlying channel used by Ethernet driver.
IP Address Byte 1	192	IP Address Byte 1 selection
IP Address Byte 2	168	IP Address Byte 2 selection
IP Address Byte 3	0	IP Address Byte 3 selection
IP Address Byte 4	0	IP Address Byte 4 selection
Subnet Mask Byte 1	255	Subnet Mask Byte 1 selection
Subnet Mask Byte 2	255	Subnet Mask Byte 2 selection
Subnet Mask Byte 3	255	Subnet Mask Byte 3 selection
Subnet Mask Byte 4	0	Subnet Mask Byte 4 selection

Note: Information and a description of valid and common settings for IP Addresses and the associated masks are available in the IP Address Limitations Synergy Platform Knowledge Base article available as described at the end of this document. The example values and defaults are for a project using the S7G2 Synergy MCU Group. Other MCUs may have different default values and available configuration settings.

In some cases, settings other than the defaults for stack modules can be desirable. For example, it might be useful to have a different IP address. The configurable properties for the lower-level stack modules are given in the following sections for completeness and as a reference.

Note: The following deprecated configurations of the NetX Telnet server module and the NetX module are required when using the Communications Framework module on sf_el_nx_comms. The Communications Framework module on sf_el_nx_comms will be updated in a future release to avoid using deprecated configurations.

Table 6. Configuration Settings for the NetX Telnet Server Module (nx_telnet_server)

ISDE Property	Value	Description
Name	g_nx_telnet_server0	Module name
Show deprecation warning	Enabled, Disabled Default: Enabled	Show deprecation warning selection

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

Table 7. Configuration Settings for the NetX Module (nx)

ISDE Property	Value	Description
Name	Default: g_nx0	NetX module name.
Show deprecation warning	Enabled, Disabled Default: Enabled	Show deprecation warning selection.

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

Table 8. Configuration Settings for NetX Port ETHER (sf_el_nx)

ISDE Property	Value	Description
Parameter Checking	BSP, Enabled, Disabled Default: BSP	Enable or disable the parameter checking
Channel 0 Phy Reset Pin	IOPORT_PORT_09_PIN_03	Channel 0 Phy reset pin selection
Channel 0 MAC Address High Bits	0x00002E09	Channel 0 MAC address high bits selection
Channel 0 MAC Address Low Bits	0x0A0076C7	Channel 0 MAC address low bits selection
Channel 1 Phy Reset Pin	IOPORT_PORT_07_PIN_06	Channel 1 Phy reset pin selection
Channel 1 MAC Address High Bits	0x00002E09	Channel 1 MAC address high bits selection
Channel 1 MAC Address Low Bits	0x0A0076C8	Channel 1 MAC address low bits selection
Number of Receive Buffer Descriptors	8	Number of receive buffer descriptors selection
Number of Transmit Buffer Descriptors	32	Number of transmit buffer descriptors selection
Ethernet Interrupt Priority	Priority 0 (highest), Priority 1:2, Priority 3 (CM4: valid, CM0+: lowest- not valid if using ThreadX), Priority 4:14 (CM4: valid, CM0+: invalid), Priority 15 (CM4 lowest - not valid if using ThreadX, CM0+: invalid) Default: Disabled	Ethernet interrupt priority selection
Name	g_sf_el_nx	Module name
Channel	0	Channel selection
Callback	NULL	Callback selection

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

5.1.2 Configuration Settings for the USB Stack Modules

When the USB option is selected for low-level implementation, the following modules are automatically added to the stack. The following tables show the detailed configuration settings available, many of which are populated with default settings that can be used in most common applications.

Table 9. Configuration Settings for USB Communications Framework Module (sf_el_ux_comms)

ISDE Property	Value	Description
Parameter Checking	BSP, Enabled, Disabled Default: BSP	Enables or disables the parameter checking.
Read Input Buffer Size (Bytes)	128	Maximum number of bytes that can be received at a time in the <code>read</code> API.
Timeout in ticks	1000	Timeout value to suspend a USBX CDC instance creation in the <code>open</code> API.
Name	g_sf_comms0	Module name.
Name of the sf_comms initialization function	sf_comms_init0	Name of helper function to initialize Communications Framework. The function will be presented in the auto-generated code in the <code><xxx_thread>.c</code> , where <code><xxx_thread></code> is the name of your thread symbol given to the Thread property. The function is to be called in the auto-generated code if Auto sf_comms Initialization property is Enabled. If Disabled, the function can be called in the user application.
Auto sf-comms Initialization	Enable, Disable Default: Enable	Auto Initialization support of Communications Framework. The helper function above will be called in the auto-generated code if this configuration is enabled. Else, the function will not be called automatically and user can call it sometime later.

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

Table 10. Configuration for the USBX Device Class CDC ACM Module (g_ux_device_class_cdc_acm0)

ISDE Property	Value	Description
Name	g_ux_device_class_cdc_acm0	Module name
USBX CDC-ACM instance_activate Function Callback	ux_cdc_device0_instance_activate	USBX CDC-ACM instance_activate Function Callback selection
USBX CDC-ACM instance_deactivate Function Callback	ux_cdc_device0_instance_deactivate	USBX CDC-ACM instance_deactivate Function Callback selection

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

Table 11. Configuration Settings for the USBX Device Configuration

ISDE Property	Value	Description
Vendor ID	0x045B	Vendor ID selection
Product ID	0x0000	Product ID selection
Device Release Number	0x0000	Device Release Number selection
Index of Manufacturing String Descriptor	0x00	Index of Manufacturing String Descriptor selection

ISDE Property	Value	Description
Index of Product String Descriptor	0x00	Index of Product String Descriptor selection
Index of Serial Number String Descriptor	0x00	Index of Serial Number String Descriptor selection
Class Code	Device, Communications(CDC), HID, Mass Storage, Miscellaneous, Vendor Specific Default: Communications	Class Code selection
Index of String Descriptor describing this configuration	0x00	Index of String Descriptor describing this configuration selection
Size of USB Descriptor in bytes for this configuration (Modify this value only for Vendor-specific Class, otherwise set zero)	0x00	Size of USB Descriptor in bytes for this configuration (Modify this value only for Vendor-specific Class, otherwise set zero) selection
Number of Interfaces (Modify this value only for Vendor-specific Class, otherwise set zero)	0x00	Number of Interfaces (Modify this value only for Vendor-specific Class, otherwise set zero) selection
Self-Powered	Enable, Disable Default: Enable	Self-Powered selection
Remote Wakeup	Enable, Disable Default: Disable	Remote Wakeup selection
Maximum Power Consumption (in 2mA units)	50	Maximum Power Consumption (in 2mA units) selection
Supported Language Code	0x0409	Supported Language Code selection
Name of USBX String Framework	NULL	Name of USBX String Framework selection
Total index number of USB String Descriptors in USB String Framework	0	Total index number of USB String Descriptors in USB String Framework selection
Name of USBX Language Framework	NULL	Name of USBX Language Framework selection
Number of Languages to support (US English is applied if zero is set)	0	Number of Languages to support (US English is applied if zero is set) selection

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

Table 12. Configuration Settings for the USBX Interface Configuration

ISDE Property	Value	Description
Name	g_usb_interface_desc_cdcacm_0	Module name
Interface Number of Communications Class interface	0x00	Interface Number of Communications Class interface selection
Interrupt Transfer endpoint to use for Communications Class	Endpoint 1-9 Default: Endpoint 3	Interrupt Transfer endpoint to use for Communications Class selection
Polling period for Interrupt Endpoint (in mS/125 us units for FS/HS)	0x0F	Polling period for Interrupt Endpoint (in mS/125us units for FS/HS) selection
Interface Number of Data Class interface	0x01	Interface Number of Data Class interface selection
Bulk In Transfer endpoint to use for Data Class	Endpoint 1-9 Default: Endpoint 1	Bulk In Transfer endpoint to use for Data Class selection
Bulk Out Transfer endpoint to use for Data Class	Endpoint 1-9 Default: Endpoint 2	Bulk Out Transfer endpoint to use for Data Class selection
Index of String Descriptor Describing Communications Class interface (Interface Descriptor: Interface)	0x00	Index of String Descriptor Describing Communications Class interface (Interface Descriptor: Interface) selection
Index of String Descriptor Describing Data Class interface (Interface Descriptor: Interface)	0x00	Index of String Descriptor Describing Data Class interface (Interface Descriptor: Interface) selection

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

Table 13. Configuration Settings for the USBX Port DCD on sf_el_ux for USBFS

ISDE Property	Value	Description
Full Speed Interrupt Priority	Priority 0 (highest), Priority 1:2, Priority 3 (CM4: valid, CM0+: lowest- not valid if using ThreadX), Priority 4:14 (CM4: valid, CM0+: invalid), Priority 15 (CM4 lowest - not valid if using ThreadX, CM0+: invalid) Default: Disabled	Full speed interrupt priority selection.
Name	g_sf_el_ux_dcd_fs_0	Module name.
USB Controller Selection	USBFS	USB controller selection.

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

Table 14. Configuration Settings for the USBX Port DCD on sf_el_ux for USBHS

ISDE Property	Value	Description
High Speed Interrupt Priority	Priority 0 (highest), Priority 1:2, Priority 3 (CM4: valid, CM0+: lowest- not valid if using ThreadX), Priority 4:14 (CM4: valid, CM0+: invalid), Priority 15 (CM4 lowest - not valid if using ThreadX, CM0+: invalid) Default: Disabled	High speed interrupt priority selection.
Name	g_sf_el_ux_dcd_hs_0	Module name.
USB Controller Selection	USBHS	USB controller selection.

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

Table 15. Configuration Settings for the USBX on ux

ISDE Property	Value	Description
USBX Pool Memory Name	g_ux_pool_memory	USBX pool memory name selection.
USBX Pool Memory Size	18432	USBX pool memory size selection.
User Callback for Host Event Notification (Only valid for USB Host)	NULL	User callback for host event notification (only valid for USB host)

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

5.1.3 Configuration Settings for the UART Stack Modules

When the UART option is selected for low-level implementation, the following modules are automatically added to the stack. The following tables show the detailed configuration settings available, many are populated with default settings that can be used in most common applications.

Table 16. Configuration Settings for the UART Communications Framework Module (sf_uart_comms)

ISDE Property	Value	Description
Parameter Checking	BSP, Enabled, Disabled Default: BSP	Selects if parameter checking is included.
Read Input Queue Size (4-Byte Words)	15	Buffer size for data reception queue. sf_uart_comms utilizes the ThreadX Queue for the queue management.
Name	g_sf_comms0	Name of UART communications framework module.

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

Table 17. Configuration Settings for the UART Driver Module (r_sci_uart)

ISDE Property	Value	Description
External RTS Operation	Enable, Disable Default: Disable	Enable an IOPORT pin to be used as RTS signal. For RTS functionality set this configuration parameter to "Enable" and specify the configuration "Name of UART callback function for the RTS external pin control".
Reception	Enable, Disable Default: Enable	Enable or disable UART reception for all UART channels on SCI. Setting this configuration parameter to "Disable" reduces code size because the portion of code for UART reception is not

ISDE Property	Value	Description
		compiled. You cannot set this parameter for individual UART channels.
Transmission	Enable, Disable Default: Enable	Enable or disable UART transmission for all UART channels on SCI. Setting "Disable" to this configuration allows to get smaller code size due to the portion of code for UART transmission is compiled out, however, you can only set "Disable" to this configuration if any other SCI channels which work as UART ports do not perform the transmission.
Parameter Checking	BSP, Enabled, Disabled Default: BSP	Enable or disable the parameter error checking.
Name	g_uart0	The name to be used for UART on SCI module control block instance. This name is also used as the prefix of the other variable instances.
Channel	0	SCI channel number.
Baud Rate	9600	Baud rate selection.
Data Bits	7 bits, 8, bits, 9 bits Default: 8 bits	UART data bits.
Parity	None, Odd, Even Default: None	UART parity bits.
Stop Bits	1 bit, 2 bits Default: 1 bit	UART stop bits.
CTS/RTS Selection	CTS (Note that RTS is available when enabling External RTS Operation mode which uses 1 GPIO pin), RTS (CTS is disabled) Default: RTS (CTS is disabled)	Select CTS or RTS for the CTSn/RTSn pin of SCI channel n. The SCI hardware supports either the CTS or the RTS control signal on this pin but not both. For an application that uses both CTS and RTS, select "CTS" for this configuration parameter and enable the configuration "External RTS Operation" specifying the configuration "Name of UART callback function for the RTS external pin control".
Name of UART callback function to be defined by user	NULL	Name must be a valid C symbol.
Name of UART callback function for the RTS external pin control to be defined by user	NULL	Name must be a valid C symbol.
Clock Source	Internal Clock, External Clock 8x baudrate, External Clock 16x baudrate Default: Internal Clock	Selection of the clock source to be used in the baud-rate clock generator block.
Baudrate Clock Output from SCK pin	Enable, Disable Default: Disable	Optional setting to output the baud-rate clock on the SCKn pin for the selected channel n.
Start bit detection	Falling Edge, Low Level Default: Falling Edge	Start bit detection mode in the reception, usually set "Falling Edge" to this configuration.

ISDE Property	Value	Description
Noise Cancel	Enable, Disable Default: Disable	Enable the digital noise cancellation on RXDn pin. The digital noise filter block in SCI consists of two-stage flip-flop circuits. For detail, refer to the Noise cancellation section in the Renesas Synergy hardware manual.
Bit Rate Modulation Enable	Enable, Disable Default: Enable	Bit rate modulation enable selection.
Receive FIFO Trigger Level	One or Max Default: Max	Trigger level for FIFO
Receive Interrupt Priority	Priority 0 (highest), Priority 1:2, Priority 3 (CM4: valid, CM0+: lowest- not valid if using ThreadX), Priority 4:14 (CM4: valid, CM0+: invalid), Priority 15 (CM4 lowest - not valid if using ThreadX, CM0+: invalid) Default: Disabled	Receive interrupt priority selection.
Transmit Interrupt Priority	Priority 0 (highest), Priority 1:2, Priority 3 (CM4: valid, CM0+: lowest- not valid if using ThreadX), Priority 4:14 (CM4: valid, CM0+: invalid), Priority 15 (CM4 lowest - not valid if using ThreadX, CM0+: invalid) Default: Disabled	Transmit interrupt priority selection.
Transmit End Interrupt Priority	Priority 0 (highest), Priority 1:2, Priority 3 (CM4: valid, CM0+: lowest- not valid if using ThreadX), Priority 4:14 (CM4: valid, CM0+: invalid), Priority 15 (CM4 lowest - not valid if using ThreadX, CM0+: invalid) Default: Disabled	Transmit end interrupt priority selection.
Error Interrupt Priority	Priority 0 (highest), Priority 1:2, Priority 3 (CM4: valid, CM0+: lowest- not valid if using ThreadX), Priority 4:14 (CM4: valid, CM0+: invalid), Priority 15 (CM4 lowest - not valid if using ThreadX, CM0+: invalid) Default: Disabled	Error interrupt priority selection.

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

Table 18. Transfer Driver Module on r_dtc Event SCI0 TXI

ISDE Property	Value	Description
Parameter Checking	BSP, Enabled, Disabled (Default: BSP)	Selects if code for parameter checking is to be included in the build
Software Start	Enabled, Disabled Default: Disabled	Software start selection

ISDE Property	Value	Description
Linker section to keep DTC vector table	.ssp_dtc_vector_table	Linker section to keep DTC vector table selection
Name	g_transfer0	Module name
Mode	Normal	Mode selection
Transfer Size	1 Byte	Transfer size selection
Destination Address Mode	Fixed	Destination address mode selection
Source Address Mode	Incremented	Source address mode selection
Repeat Area (Unused in Normal Mode)	Source	Repeat area selection
Interrupt Frequency	After all transfers have completed	Interrupt frequency selection
Destination Pointer	NULL	Destination pointer selection
Source Pointer	NULL	Source pointer selection
Number of Transfers	0	Number of transfers selection
Number of Blocks (Valid only in Block Mode)	0	Number of blocks selection
Activation Source (Must enable IRQ)	Event SCI0 TXI	Activation source selection
Auto Enable	FALSE	Auto enable selection
Callback (Only valid with Software start)	NULL	Callback selection
ELC Software Event Interrupt Priority	Priority 0 (highest), Priority 1:2, Priority 3 (CM4: valid, CM0+: lowest- not valid if using ThreadX), Priority 4:14 (CM4: valid, CM0+: invalid), Priority 15 (CM4 lowest - not valid if using ThreadX, CM0+: invalid) Default: Disabled	ELC Software Event interrupt priority selection.

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

Table 19. Transfer Driver Module on r_dtc Event SCI0 RXI

ISDE Property	Value	Description
Parameter Checking	BSP, Enabled, Disabled Default: BSP	Selects if code for parameter checking is to be included in the build
Name	g_transfer1	Module name
Mode	Normal	Mode selection
Transfer Size	1 Byte	Transfer size selection
Destination Address Mode	Incremented	Destination address mode selection
Source Address Mode	Fixed	Source address mode selection
Repeat Area (Unused in Normal Mode)	Destination	Repeat area selection
Interrupt Frequency	After all transfers have completed	Interrupt frequency selection
Destination Pointer	NULL	Destination pointer selection
Source Pointer	NULL	Source pointer selection
Number of Transfers	0	Number of transfers selection
Number of Blocks (Valid only in Block Mode)	0	Number of blocks selection
Activation Source (Must enable IRQ)	Event SPI0 RXI	Activation source selection

ISDE Property	Value	Description
Auto Enable	FALSE	Auto enable selection
Callback (Only valid with Software start)	NULL	Callback selection
ELC Software Event Interrupt Priority	Priority 0 (highest), Priority 1:2, Priority 3 (CM4: valid, CM0+: lowest- not valid if using ThreadX), Priority 4:14 (CM4: valid, CM0+: invalid), Priority 15 (CM4 lowest - not valid if using ThreadX, CM0+: invalid) Default: Disabled	ELC Software Event interrupt priority selection.

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

5.1.4 Configuration Settings for the USB Stack Modules (Deprecated)

When the USB option is selected for low-level implementation, the following modules are automatically added to the stack. The following tables show the detailed configuration settings available, many of which are populated with default settings that can be used in most common applications.

Table 20. Configuration Settings for USB Communications Framework Module (sf_el_ux_comms)

ISDE Property	Value	Description
Parameter Checking	BSP, Enable, Disable Default: BSP	Selects if code for parameter checking is to be included in the build.
Memory Size (Bytes)	65536	Memory size selection.
Timeout in ticks	1000	Timeout value to suspend a USBX CDC instance creation in the <code>open()</code> API.
Read Input Buffer Size (Bytes)	128	This is the maximum number of bytes that can be received at a time in the <code>read()</code> API.
Name	g_sf_comms0	Name for USB communications framework module.
Name of the sf_comms Initialization Function	sf_comms_init0	Name of the sf_comms initialization function selection.
Auto sf_comms Initialization	Enable, Disable Default: Enable	Auto sf_comms initialization selection.

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

Table 21. Configuration for the USBX Device Class CDC ACM Module (g_ux_device_class_cdc_acm0)

ISDE Property	Value	Description
Name	Default: g_ux_device_class_cdc_acm0	USBX device class <code>cdc-acm</code> module name.
Show Deprecation Warning	Enabled, Disabled Default: Enabled	Show deprecation warning selection.

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

Table 22. Configuration Settings for the USXB on ux

ISDE Property	Value	Description
Name	Default: g_ux0	USBX on ux module name.
Show Deprecation Warning	Enabled, Disabled Default: Enabled	Show deprecation warning selection.

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

Table 23. Configuration Settings for the USBX Ports HS and FS on sf_el_ux

ISDE Property	Value	Description
VBUSEN Pin Signal Logic	Active Low, Active High Default: Active High	VBUSEN pin signal logic selection.
High Speed Interrupt Priority	Priority 0 (highest), Priority 1:2, Priority 3 (CM4: valid, CM0+: lowest- not valid if using ThreadX), Priority 4:14 (CM4: valid, CM0+: invalid), Priority 15 (CM4 lowest - not valid if using ThreadX, CM0+: invalid) Default: Disabled	High speed interrupt priority selection.
Full Speed Interrupt Priority	Priority 0 (highest), Priority 1:2, Priority 3 (CM4: valid, CM0+: lowest- not valid if using ThreadX), Priority 4:14 (CM4: valid, CM0+: invalid), Priority 15 (CM4 lowest - not valid if using ThreadX, CM0+: invalid) Default: Disabled	Full speed interrupt priority selection.
Name	Default: g_sf_el_ux	Name for the USBX Ports HS and FS.
Show deprecation warning	Enabled, Disabled Default: Enabled	Show deprecation warning selection.

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

6. Using the Console Framework Module in an Application

The key elements in constructing a simple Console Framework are selecting and configuring the stack, defining the menu structure, writing the code for the menu command callbacks, and using the API calls to implement the desired CLI functions within the target application. The typical steps in using the Console Framework module in an application are:

1. Create menu and command structures.
2. Implement needed callbacks.
3. Initialize the SF_CONSOLE using the `open` API.
4. Use the `prompt` API to generate the prompt and process commands.
5. Use other APIs (`read`, `write`, `parse` or `argumentFind`) as needed to process commands.
6. Use the `close` API to close the module if desired.

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

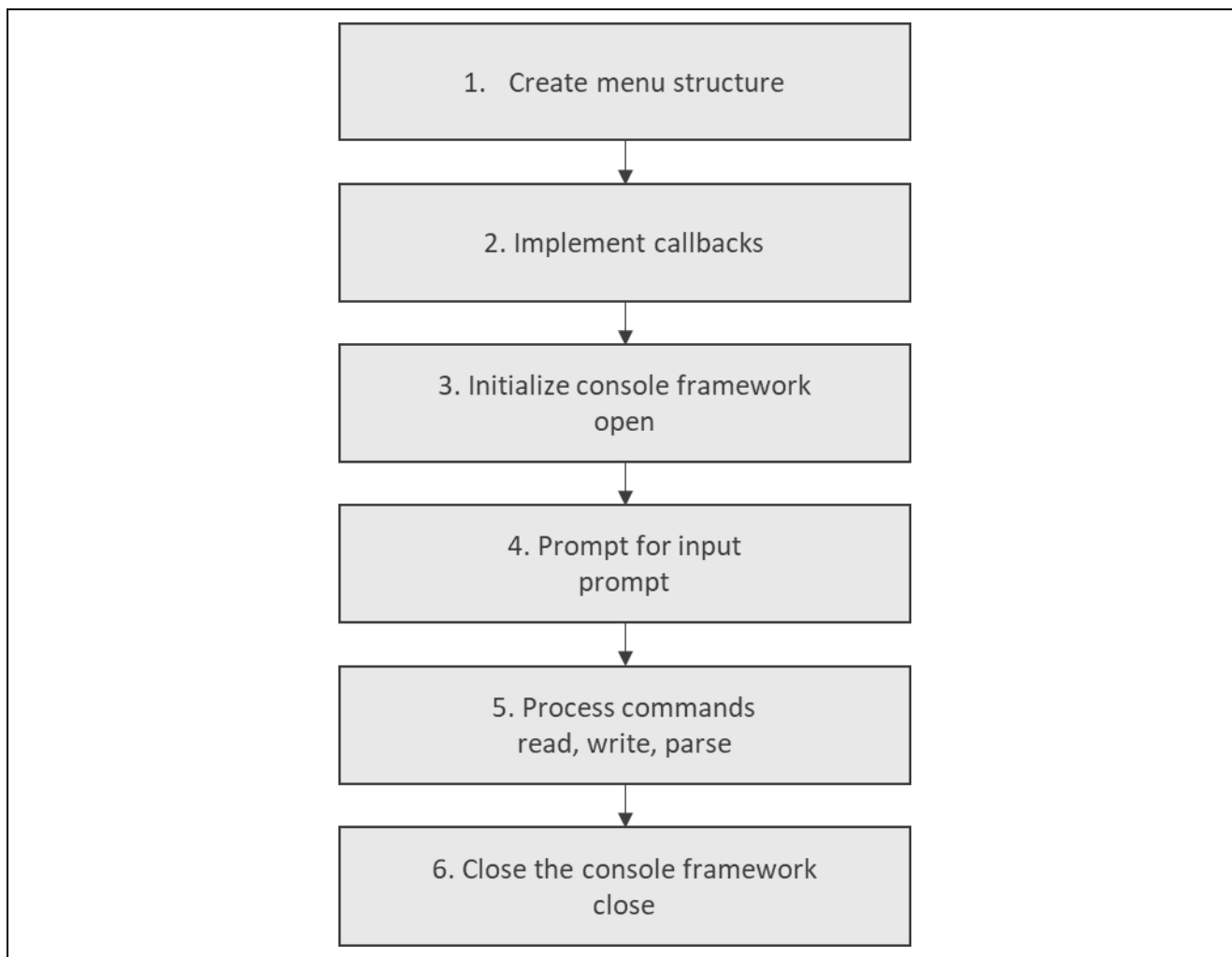


Figure 6. Typical Console Framework Module Application

7. Console Framework Module Application Project

The application project associated with this module guide demonstrates the aforementioned steps in an example application. You may want to import and open the application project within the ISDE and view the configuration settings for the Console Framework module. You can also read over the code (in `console_framework_callback.h` and `console_framework_mg_api.h`) which is used to illustrate the Console Framework module APIs in a complete design.

The application project demonstrates the typical use of the Console Framework module. (The application project Console Framework stack is illustrated in the following figure.) The Console Framework stack has been added to a thread called `g_sf_console0` and the USB implementation has been selected for the lower-level driver. The SK-S7G2 kit is being used as the BSP so all the various interconnect resources are already configured; you only need to enable the USB interrupt as directed by the ISDE prompt and change Device Class to Miscellaneous if on Windows 10. The following table shows the required configuration setting:

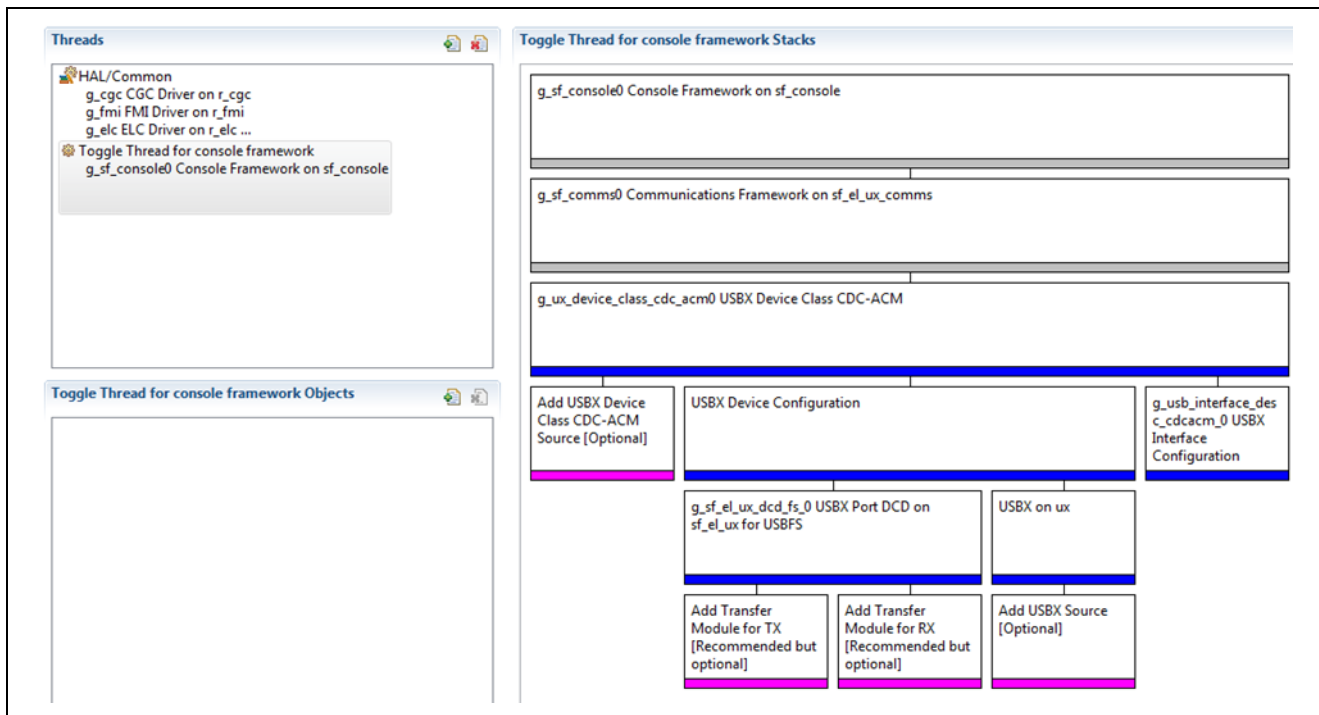


Figure 7. Application Project Console Framework Stack: USB Communications Option Selected

Table 24. Adding the USBX Port DCD HS or FS on sf_el_ux Configuration Settings for the Application Project

Resource	ISDE Tab	Property / Configuration Setting
g_sf_el_ux_dcd_fs0 USBX Port DCD on sf_el_ux orUSBFS	Threads Stack/ Properties	Full Speed Interrupt Priority / Priority 4

A simple use of the Console Framework to toggle an LED on and off fits on a single page of code. In this simple design, when the TOGGLE command is detected by the `prompt` API, it calls a callback associated with the TOGGLE command and toggles an LED on the target kit. The following table identifies the target versions for the associated software and hardware used by the application project and a simple block diagram of the LED toggle console framework implementation is displayed in the figure which follows the following table:

Table 25. Software and Hardware Resources Used by the Application Project

Resource	Revision	Description
e ² studio	5.3.1 or later	Integrated Solution Development Environment
SSP	1.2.0 or later	Synergy Software Platform
IAR EW for Synergy	7.71.2 or later	IAR Embedded Workbench® for Renesas Synergy™
SSC	5.3.1 or later	Synergy Standalone Configurator
SK-S7G2	v3.0 to v3.1	Starter Kit

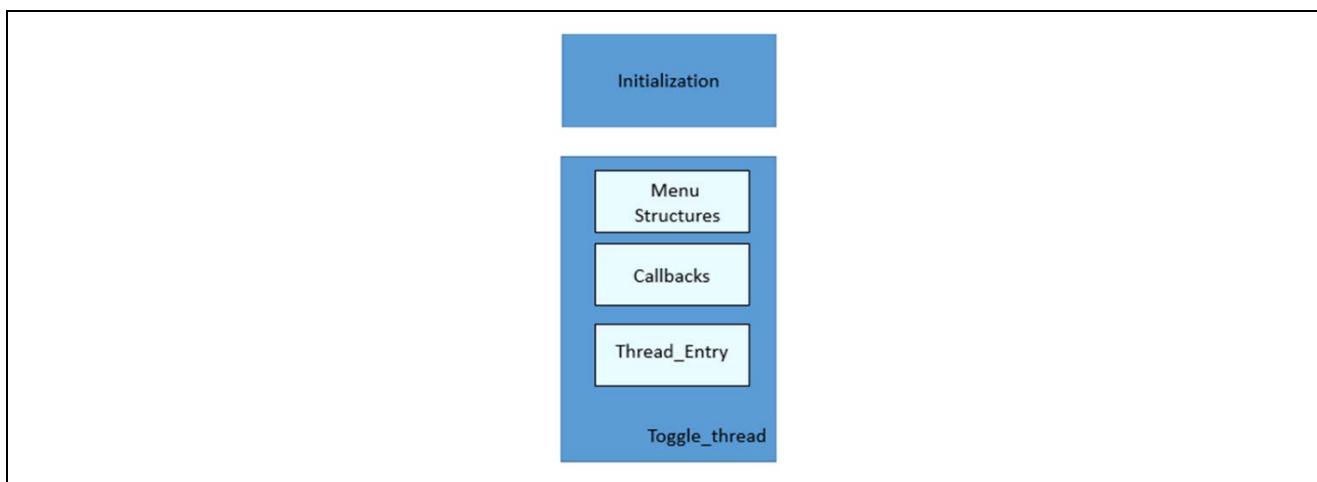


Figure 8. Console Framework Toggle Command Implementation Block Diagram

The `toggle_thread_entry.c` file is in the project once it has been imported into the ISDE. You can open this file within the ISDE and follow along with the following description to help identify key uses of APIs. The toggle command Console Framework-based implementation begins with system initialization where the `open` command and other housekeeping functions are executed. The toggle-thread routine implements the remainder of the toggle Console Framework functions, and these elements will appear in any Console Framework implementation. In the example project, the menu structures (to define the commands and the menu) are implemented at the beginning of the `toggle_thread.c` file. These are followed by the command-related callback. In the example, there is a single toggle command-related callback that toggles the output signal that drives the on-board LED. Finally, the `thread_entry` routine is created. In this example, a single call to the `prompt` API is all that is needed.

The first section of the code in `toggle_thread_entry.c` has the header file and the call to the `prompt` API. The callback functions, menu, and commands are defined in the `console_framework_callback.h` file. For this simple example, the command structure has only a single entry that includes the TOGGLE command, the help description text, the callback function definition, and a NULL context entry. More menu items could simply be appended to the structure in more complex systems.

The next section of code in the `console_framework_callback.h` has the root menu structure. It has a NULL entry for the previous menu, since the example has only a single menu, with no sub menus. The menu name is **Command**, and the number of menu entries is simply the size of the entire command array divided by the size of a single array element. The menu structure has a pointer to the first element of the command structure array. The menu is now fully defined for this simple example.

Note: A more complex menu structure is used in the application projects, and is available as described in the references section at the end of this document. Once you are familiar with the simple example basic menu structures, review the menus used in the developer example code to see the implementation of multi-level menus, uses of command arguments, uses of distributed command structures throughout a project, and the use of read and write APIs.

The next section of code defines the callback function that implements the associated toggle command. All Console Framework commands are implemented as callback functions. The toggle command gets a list of LEDs via the `LedsGet` BSP call; it then reads the value of the first LEDs driver pin and writes out the inverted version to the same pin, toggling the LED.

The last section of code is the thread entry and implements the entire simple Console Framework in a single line of code. The `prompt` API is called with the instance control structure (`g_sf_console0.p_ctrl`) as the main argument. The instance structure is created by the `open` API, automatically called during thread initialization, and does not appear in this code sub-section. The other arguments are a pointer to the valid menu commands for the prompt, in this example it isn't required and is NULL since you have just the single command. For more complex, context-sensitive menus, this pointer would identify the valid subset of possible commands and a wait time. (The wait time is FOREVER in this example since there is no timeout on a prompt reply. Automated test systems, for example, might use a known maximum delay time, between commands, as an error checking mechanism.)

8. Customizing the Console Framework Module for a Target Application

The Console Framework can be customized for different menus, different thread entry names, different instance names, and different configuration settings for the Communications Framework stack modules (or even a completely different Communications Framework entirely.)

8.1 Menu Structures

The Console Framework is primarily customized using the menu and command structures, and it is easy to modify these as required by a different application. The only other changes possibly needed to the example code to get it working in a different system are to possibly modify the header file name, the instance name and, if a different Communications Framework is used, to configure the lower-level modules.

8.2 Thread Entry Name

Make sure the thread entry name is consistent with the header file. In this simple example, the thread name defined in the associated properties entry is `toggle_thread` and the required include file definition is:

```
#include toggle_thread.h.
```

8.3 Instance Name

The instance name used for API calls needs to match the name given during configuration. In this example, the instance name was defined as `g_sf_console0` and the API call uses `g_sf_console0` for the API definition and the instance definition for `p_ctrl`:

```
g_sf_console0.p_api->prompt(g_sf_console0.p_ctrl, NULL, TX_WAIT_FOREVER);
```

8.4 Communications Framework Module Selection and Configuration

Since the Console Framework uses a standard interface to the Communications Framework, a custom application can select different options (Telnet, USB or UART) when adding a Communications Framework, as illustrated in the Console Framework stack shown in the following figure. If the Communications Framework is changed, none of the toggle command Console Framework code needs to be edited and any configuration differences required by the new Communications Framework will be filled in using the configurator.

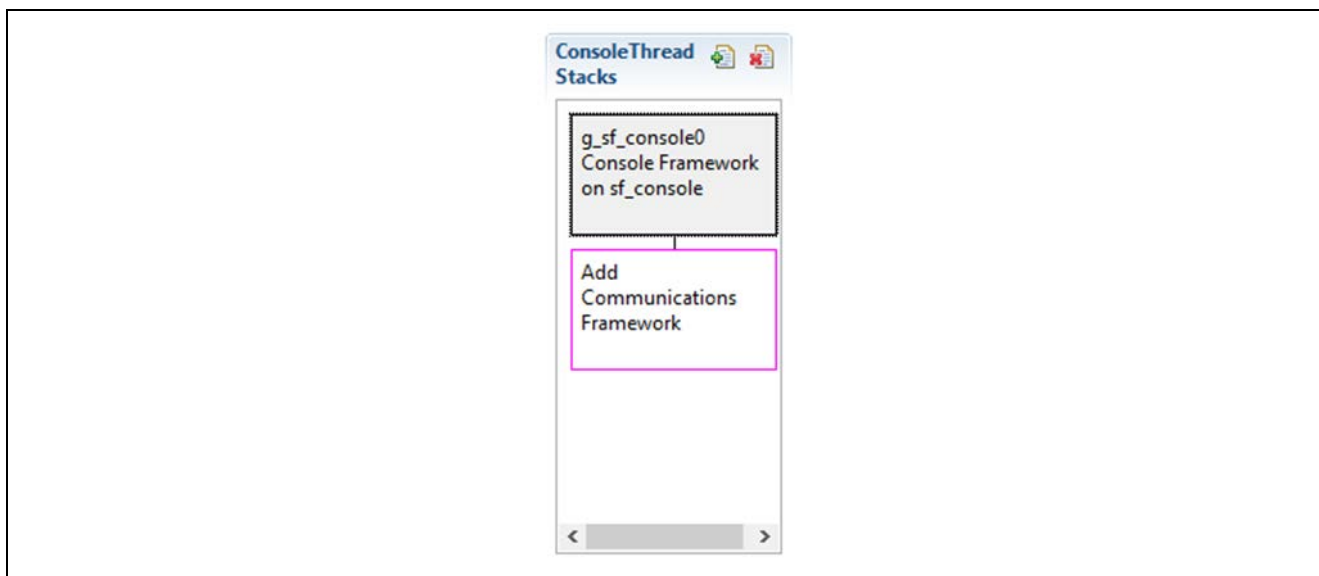


Figure 9. Adding Communications Framework Options Under the Console Framework

View the application frameworks video, available as described in the References section of this document, to see how easy it is to swap the console physical connection from USB to Telnet. The complete console swap is done entirely within the configuration window and takes only a couple of minutes.

9. Running the Console Framework Module Application Example

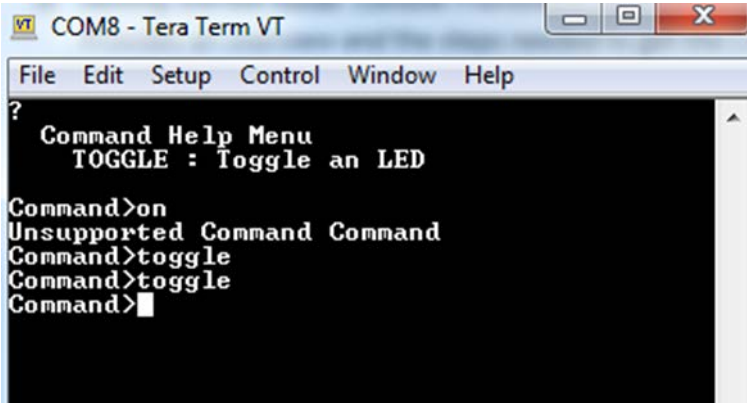
To run the console framework module application project and to see it executed on a target kit, you can simply import it into your ISDE, compile, and run debug. Refer to the *SSP Import Guide* (11an0023eu0116-synergy-ssp-import-guide.pdf, included in this package) for instructions on importing the project into e² studio or IAR embedded workbench and building/running the application.

To implement the Console Framework module 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.

To establish the connection through the CDC, the CDC driver must be installed first. The link to the driver-installation process is available as described in the References section in this document.

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 of the *SSP User's Manual* for a description on how to accomplish these steps.

1. Refer to the *Renesas Synergy™ Project Import Guide* (11an0023eu0121-synergy-ssp-import-guide.pdf, included in this package) for instructions on importing the project into e² studio or IAR EW for Synergy and building/running the application.
2. Connect to the host PC via a micro USB cable to J19 on SK-S7G2 board and connect another micro USB cable form host to J5 connector of the SK-S7G2 board.
3. Start to debug the application.
4. Open Terra Term application and connect to serial port for serial communication with USB CDC.
5. Write ? and press enter to see available commands.
6. Write **TOGGLE** to toggle an LED on the board.
7. As output, LED will toggle when user enters command on serial console.



```
VT COM8 - Tera Term VT
File Edit Setup Control Window Help
?
  Command Help Menu
  TOGGLE : Toggle an LED

Command>on
Unsupported Command Command
Command>toggle
Command>toggle
Command>
```

Figure 10. Example Output from Console Framework Application Project

10. Console Framework Module Conclusion

This module guide has provided all the background information needed to select, add, configure, and use the component 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 incorrect selection of lower-level drivers. The use of high level APIs as demonstrated in the application project illustrate additional development time savings by allowing work to begin at a high level, avoiding the time required in older development environments to use or, in some cases, create, lower-level drivers.

11. Console Framework Module Next Steps

After you have mastered a simple Console Framework example you may want to review a more complex example. The developer examples project provides a more complex example that demonstrates several of the more complex features of the Console Framework. Refer to the *Getting Started Guide for the Developer Examples*, available as described in the References section at the end of this document.

12. Console Framework Module Reference Information

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

Links to all the most up-to-date sf_console module reference materials and resources are available on the Synergy Knowledge Base: <https://en-support.renesas.com/knowledgeBase/16977533> or https://en-support.renesas.com/search/sf_console%20Module%20Guide%20Resources

Links to application projects which uses advanced console framework for reference are as follows:

<https://www.renesas.com/us/en/search/keyword-search.html#q=r11an0337eu>

<https://www.renesas.com/us/en/search/keyword-search.html#q=r11an0336eu>

<https://www.renesas.com/us/en/search/keyword-search.html#q=r11an0335eu>

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	May.15.17	-	Initial document release
1.01	Aug.31.17	-	Update to the Hardware and Software Resources table
1.02	Jun.03.19	-	Update to the reference links to use the latest

Notice

1. Descriptions of circuits, software and other related information in this document are provided only to illustrate the operation of semiconductor products and application examples. You are fully responsible for the incorporation or any other use of the circuits, software, and information in the design of your product or system. Renesas Electronics disclaims any and all liability for any losses and damages incurred by you or third parties arising from the use of these circuits, software, or information.
2. Renesas Electronics hereby expressly disclaims any warranties against and liability for infringement or any other claims involving patents, copyrights, or other intellectual property rights of third parties, by or arising from the use of Renesas Electronics products or technical information described in this document, including but not limited to, the product data, drawings, charts, programs, algorithms, and application examples.
3. No license, express, implied or otherwise, is granted hereby under any patents, copyrights or other intellectual property rights of Renesas Electronics or others.
4. You shall not alter, modify, copy, or reverse engineer any Renesas Electronics product, whether in whole or in part. Renesas Electronics disclaims any and all liability for any losses or damages incurred by you or third parties arising from such alteration, modification, copying or reverse engineering.
5. Renesas Electronics products are classified according to the following two quality grades: "Standard" and "High Quality". The intended applications for each Renesas Electronics product depends on the product's quality grade, as indicated below.
 - "Standard": Computers; office equipment; communications equipment; test and measurement equipment; audio and visual equipment; home electronic appliances; machine tools; personal electronic equipment; industrial robots; etc.
 - "High Quality": Transportation equipment (automobiles, trains, ships, etc.); traffic control (traffic lights); large-scale communication equipment; key financial terminal systems; safety control equipment; etc.

Unless expressly designated as a high reliability product or a product for harsh environments in a Renesas Electronics data sheet or other Renesas Electronics document, Renesas Electronics products are not intended or authorized for use in products or systems that may pose a direct threat to human life or bodily injury (artificial life support devices or systems; surgical implantations; etc.), or may cause serious property damage (space system; undersea repeaters; nuclear power control systems; aircraft control systems; key plant systems; military equipment; etc.). Renesas Electronics disclaims any and all liability for any damages or losses incurred by you or any third parties arising from the use of any Renesas Electronics product that is inconsistent with any Renesas Electronics data sheet, user's manual or other Renesas Electronics document.
6. When using Renesas Electronics products, refer to the latest product information (data sheets, user's manuals, application notes, "General Notes for Handling and Using Semiconductor Devices" in the reliability handbook, etc.), and ensure that usage conditions are within the ranges specified by Renesas Electronics with respect to maximum ratings, operating power supply voltage range, heat dissipation characteristics, installation, etc. Renesas Electronics disclaims any and all liability for any malfunctions, failure or accident arising out of the use of Renesas Electronics products outside of such specified ranges.
7. Although Renesas Electronics endeavors to improve the quality and reliability of Renesas Electronics products, semiconductor products have specific characteristics, such as the occurrence of failure at a certain rate and malfunctions under certain use conditions. Unless designated as a high reliability product or a product for harsh environments in a Renesas Electronics data sheet or other Renesas Electronics document, Renesas Electronics products are not subject to radiation resistance design. You are responsible for implementing safety measures to guard against the possibility of bodily injury, injury or damage caused by fire, and/or danger to the public in the event of a failure or malfunction of Renesas Electronics products, such as safety design for hardware and software, including but not limited to redundancy, fire control and malfunction prevention, appropriate treatment for aging degradation or any other appropriate measures. Because the evaluation of microcomputer software alone is very difficult and impractical, you are responsible for evaluating the safety of the final products or systems manufactured by you.
8. Please contact a Renesas Electronics sales office for details as to environmental matters such as the environmental compatibility of each Renesas Electronics product. You are responsible for carefully and sufficiently investigating applicable laws and regulations that regulate the inclusion or use of controlled substances, including without limitation, the EU RoHS Directive, and using Renesas Electronics products in compliance with all these applicable laws and regulations. Renesas Electronics disclaims any and all liability for damages or losses occurring as a result of your noncompliance with applicable laws and regulations.
9. Renesas Electronics products and technologies shall not be used for or incorporated into any products or systems whose manufacture, use, or sale is prohibited under any applicable domestic or foreign laws or regulations. You shall comply with any applicable export control laws and regulations promulgated and administered by the governments of any countries asserting jurisdiction over the parties or transactions.
10. It is the responsibility of the buyer or distributor of Renesas Electronics products, or any other party who distributes, disposes of, or otherwise sells or transfers the product to a third party, to notify such third party in advance of the contents and conditions set forth in this document.
11. This document shall not be reprinted, reproduced or duplicated in any form, in whole or in part, without prior written consent of Renesas Electronics.
12. Please contact a Renesas Electronics sales office if you have any questions regarding the information contained in this document or Renesas Electronics products.

(Note1) "Renesas Electronics" as used in this document means Renesas Electronics Corporation and also includes its directly or indirectly controlled subsidiaries.

(Note2) "Renesas Electronics product(s)" means any product developed or manufactured by or for Renesas Electronics.

(Rev.4.0-1 November 2017)

Corporate Headquarters

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

Trademarks

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

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/.