

# Application Note

## MCU Control Center Tool Developer's Guide

---



Ryan Ma, Delaney Woodward and Nima Eskandari

### ABSTRACT

The MCU Control Center Tool is a data logging and visualization tool for C2000™ microcontrollers. This tool is integrated into the Code Composer Studio™ (CCS) environment and the SysConfig tool for ease of use. This application note identifies common hardware setups that can be used with the tool set to enable data logging, describes each of the various features and use case scenarios, goes through adding support to an existing project, and highlights the available SDK examples. The features described in this document are:

- Application logging - generic data logging to a PC
  - Transfer bridge - data logging using the Fast Serial Interface (FSI) peripheral logs, capable of 200MBPS
  - Communication logging - enhanced data logging using FSI or communication peripheral application debugging
  - Rapid Time logging - minimally intrusive data logging for real-time applications
- 

### Table of Contents

<b>1 Introduction</b>	<b>3</b>
<b>2 Hardware Setup Options</b>	<b>4</b>
2.1 Setup #1	4
2.2 Setup #2	5
2.3 Setup #3	5
2.4 Setup #4	6
<b>3 Software Layers</b>	<b>6</b>
<b>4 GUI Creation</b>	<b>7</b>
<b>5 Application Logging</b>	<b>10</b>
5.1 Application Logging Walk-through	11
<b>6 Transfer Bridge</b>	<b>16</b>
6.1 Transfer Bridge Walk-through	17
<b>7 Communication Logger</b>	<b>21</b>
7.1 Communication Logger Walk-through	21
<b>8 Rapid-Time Logger</b>	<b>25</b>
8.1 Rapid Time Logging Walk-through	25
<b>9 Transfer Examples Overview</b>	<b>30</b>
<b>10 Summary</b>	<b>31</b>
<b>11 References</b>	<b>31</b>

### List of Figures

Figure 1-1. MCU Control Center High Level Diagram	3
Figure 2-1. Hardware Setup #1	4
Figure 2-2. Hardware Setup #2	5
Figure 2-3. Hardware Setup #3	5
Figure 2-4. Hardware Setup #4	6
Figure 4-1. SysCfg Generated GUI Layer Files	7
Figure 4-2. Open Project Properties	7
Figure 4-3. Add CCS Variable	8
Figure 4-4. GUI_SUPPORT Variable Added	8
Figure 4-5. Plugin Name	8
Figure 4-6. Reload Window To See Generated Plugins	9
Figure 4-7. View GUI	9
Figure 5-1. Application Logging Diagram	10

Figure 5-2. High Level Application Logger.....	11
Figure 5-3. Add MCU Control Center Module.....	11
Figure 5-4. Custom Export Logger Configuration.....	12
Figure 5-5. Export Submodule Configurations.....	12
Figure 5-6. Transfer Communication Link.....	13
Figure 5-7. Export Custom Logs Configuration.....	13
Figure 5-8. Export Log Support and Timestamp.....	14
Figure 5-9. Export Log Timestamp Configuration.....	14
Figure 5-10. Serial Port Settings.....	15
Figure 5-11. Logger Output.....	15
Figure 6-1. Transfer Bridge Diagram .....	16
Figure 6-2. High Level Transfer Bridge Project.....	17
Figure 6-3. Bridge Project Software Layer.....	18
Figure 6-4. Transfer Bridge Module.....	18
Figure 6-5. Transfer Bridge Communication Links.....	19
Figure 6-6. Bridge Buffer (Optional).....	19
Figure 6-7. Transfer Bridge Final Output.....	20
Figure 7-1. Communication Logger Diagram .....	21
Figure 7-2. Communication Logger Software Layer.....	22
Figure 7-3. MCU Control Center Module.....	22
Figure 7-4. Enable FSI Logger and Communication Logger .....	23
Figure 7-5. FSI Communication Logger Configurations .....	23
Figure 7-6. Final Output.....	24
Figure 8-1. Rapid-Time Logger Software Layer.....	25
Figure 8-2. Rapid-Time Logger SysConfig.....	25
Figure 8-3. Log Message Structure 0 Definitions.....	26
Figure 8-4. Log Message Structure 1 Definitions.....	27
Figure 8-5. Enable Rapid Time Logger.....	28
Figure 8-6. Navigate to JSON rt_log.json file.....	28
Figure 8-7. View the Rapid-Time Log Data in the PC GUI.....	29

## List of Tables

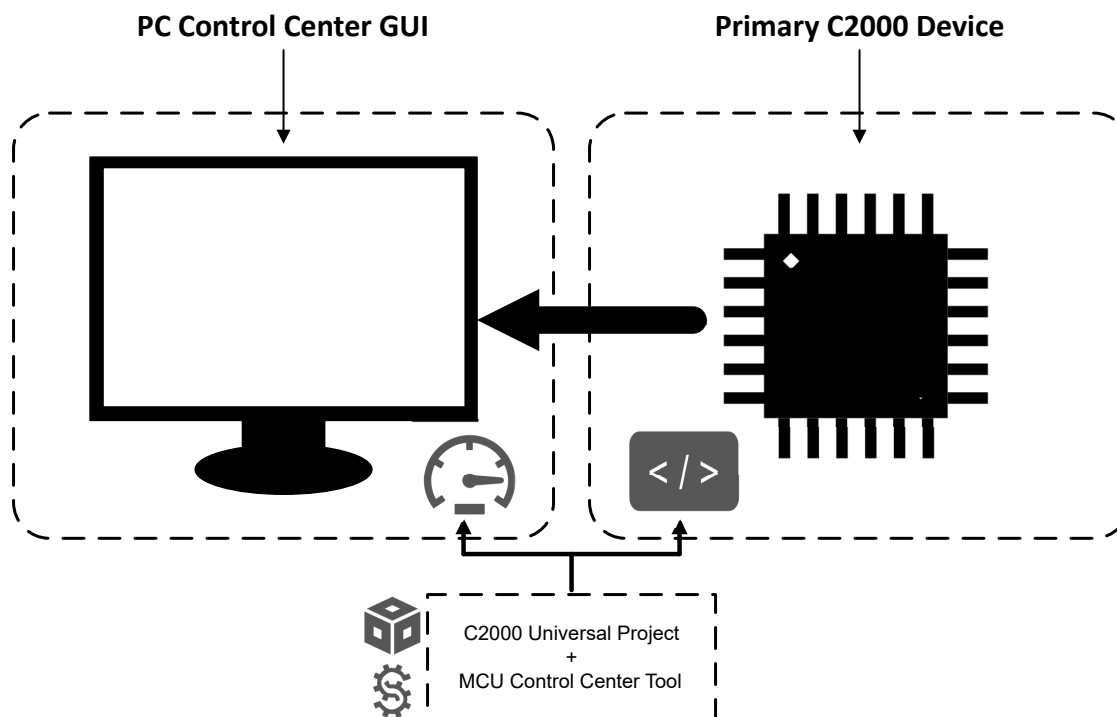
Table 3-1. SysConfig Generated Files.....	6
Table 6-1. Supported Communication Link Combinations.....	17
Table 8-1. Example Log Variable Settings.....	26

## Trademarks

C2000™, Code Composer Studio™, and LaunchPADs™ are trademarks of Texas Instruments.  
All trademarks are the property of their respective owners.

## 1 Introduction

The MCU Control Center is a SysConfig GUI-based tool that aids in exporting data out of a device through a communication peripheral and enables visualization of this data on a PC. The MCU Control Center generates application code that abstracts the communication peripheral configuration, data packaging and format configuration, and PC GUI creation code. The purpose of MCU Control Center is to provide an easy way for developers to incorporate data logging and visualization into an existing project by using SysConfig Tool. The GUI that is generated through this tool can be opened inside CCS.



### Note

The acronyms Universal Asynchronous Receiver/Transmitter (UART) and Serial Communication Interface (SCI) are often used interchangeably in this document. SCI refers to a peripheral present on most C2000 devices that uses a UART protocol.

**Figure 1-1. MCU Control Center High Level Diagram**

## 2 Hardware Setup Options

Depending on the specific C2000 device and hardware available to the user, there are four common hardware setups that can be used to integrate data logging into a project. The first step is to determine which peripherals are available on the logging (primary) device. To create a communication link between an MCU and a PC, the USB peripheral must be present somewhere in the hardware setup since this is the protocol commonly available for interfacing with a PC.

All C2000 devices have a SCI or UART peripheral which can log data using the standard UART protocol. The benefits of using this protocol on the logging device is that all controlCARDS and LaunchPADs™ sold by Texas Instruments have a flashed UART-to-USB bridge device on the board that can convert the UART messages to the necessary USB format for the PC.

The downside to using this protocol is that the SCI and UART peripherals have a relatively low maximum transmission speed, which interferes with performance in time-critical applications. In this scenario, a peripheral called Fast Serial Interface (FSI) can be utilized to log data at high speeds. Select C2000 devices have the FSI peripheral present and can be used for this purpose. Find the C2000 device in the [C2000 Real-Time Microcontrollers Peripherals User Guide](#) to determine the peripherals and maximum speed capabilities available. Note that the MCU Control Center requires the use of the Sysconfig GUI, which is only available on devices Generation 3 and newer, so these are the only devices that can be used with this tool.

### Note

While the F2838x does have a UART module, the module is only accessible via the CM core on the device. Sysconfig does not support the CM core, so MCU Control Center cannot be used with the UART peripheral on F2838x.

### 2.1 Setup #1

The first hardware setup is possible only if the data logging (primary) device has a USB peripheral present. In this setup, no bridge device is required since the device can send data directly to the PC using the USB protocol. Overall, this is the simplest hardware setup since there is no requirement for additional MCU's besides the primary device.

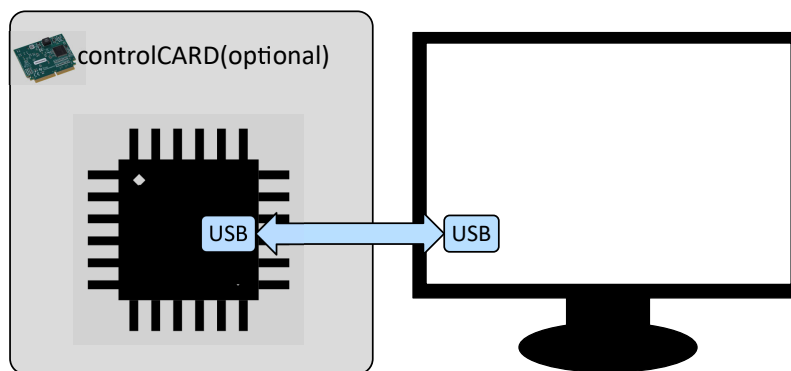


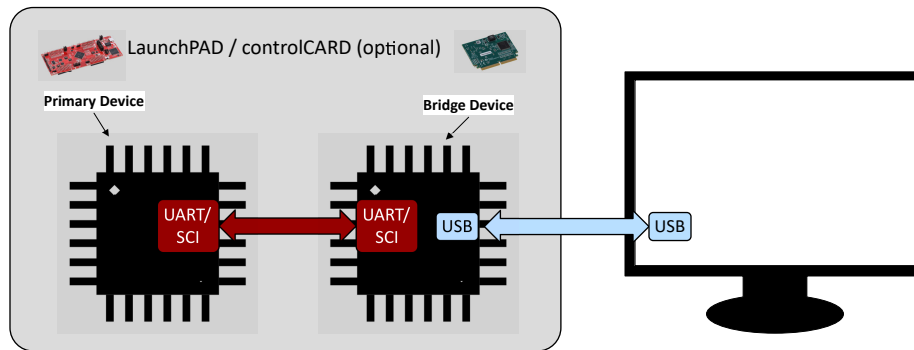
Figure 2-1. Hardware Setup #1

## 2.2 Setup #2

The second hardware setup is a very common use case and is possible on all Gen 3 and newer C2000 devices. This setup uses the SCI or UART peripheral to export data out of the device and a bridge device to convert these messages to a USB protocol that the PC can understand. If using a LaunchPAD or controlCARD, both the primary and bridge devices are already present and correctly wired. The bridge device on the board has also already been flashed with a program that does the necessary protocol conversion (in most cases this is the XDS110 program). This just leaves the SCI/UART code on the primary device to be configured, which can be done automatically by the MCU Control Center tool. See more details in later sections.

Note that if there is no LaunchPAD or controlCARD at the user's disposal, but only a standalone part (for example, the C2000 part is already soldered onto a custom board), an additional bridge device is needed. Since this type of bridge device is commonly used in embedded development, there are many options available for hardware that can be purchased at a low cost (whether from Texas Instruments or a third party vendor).

Another option here for interfacing with a standalone (non-USB-present) device, is to purchase a C2000 device with an USB module and use MCU Control Center to generate and flash the bridge application.

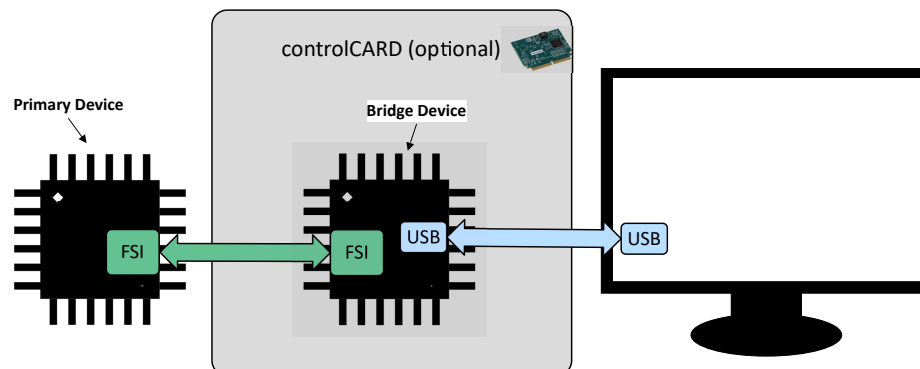


**Figure 2-2. Hardware Setup #2**

## 2.3 Setup #3

The third hardware setup is possible only if the data logging (primary) device has a FSI peripheral present and the bridge device has both an FSI and a USB peripheral present. This setup uses the FSI peripheral on a standalone part to export data out of the device and a bridge device to convert these messages to a USB protocol that the PC can understand. If using a controlCARD for the bridge device, the USB signals are already correctly wired to a USB connector that can be plugged into a PC. Both the FSI code on the primary device and the FSI-to-USB code on the bridge device need to be configured, which can both be done automatically by the MCU Control Center tool in two separate Sysconfig projects. See more details in later sections.

Note that if there is no controlCARD at the user's disposal, but only a standalone part that has the FSI and USB peripheral and can be used as the bridge device, USB connector circuitry is needed to connect to a USB port.



**Figure 2-3. Hardware Setup #3**

## 2.4 Setup #4

The fourth hardware setup is possible only if both the data logging (primary) device and first bridge device has an FSI peripheral present. This setup uses the FSI peripheral to export data out of a standalone device, one bridge device to convert these messages to a UART protocol, and a second bridge device to convert these messages from a UART protocol to a USB protocol that the PC can understand.

If using a LaunchPAD or controlCARD of a C2000 device with a FSI peripheral, both bridge devices are already present and correctly wired. The second bridge device on the board has also already been flashed with a program that does the necessary protocol conversion between UART and USB (in most cases this is the XDS110 debugger). This leaves the FSI code on the primary device to be configured, as well as the FSI-to-UART bridge code on the first bridge device to be configured. The configurations for both for both of these devices can be done automatically by the MCU Control Center tool (in two separate Sysconfig projects). More detail is provided in later in the document.

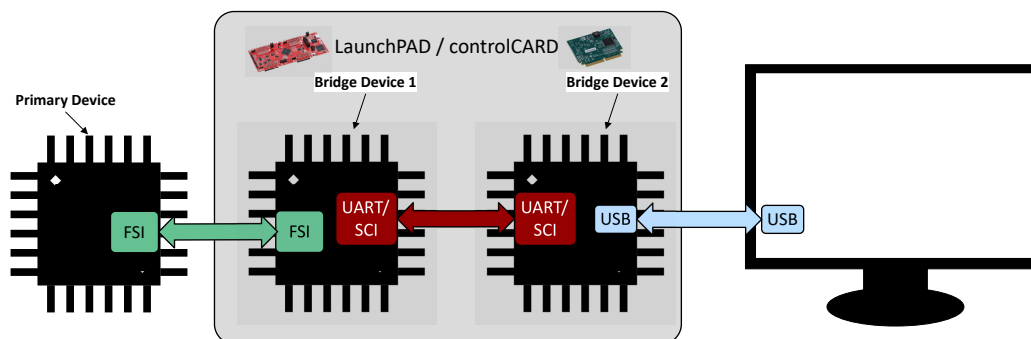


Figure 2-4. Hardware Setup #4

## 3 Software Layers












This section discusses the software abstraction layers for different feature implementations of the MCU Control Center tool. These files are autogenerated by Sysconfig when MCU Control Center modules are added and configured.

Table 3-1. SysConfig Generated Files

Logging Layer	Description	Header C Files Involved
User Application	This layer provides the highest abstraction level. This describes the top level API functions to call within the user application code for each feature. (application logging, communication logger, bridge, and rapid time logging)	logger/coms_logger.h export/export_log.h logger/rt_log.h bridge/bridge.h
Package	This layer packages the logs before sending them out of the device. The packaging layers available are JSON and START/END.	export/export_package.h
Buffer	This layer provides a buffer for the log messages before sending them out. This allows for the flexibility to capture and send data in two different areas of code.	export/export_buffer.h
Export	This layer provides the functionality to export the data via a communication peripheral by writing to the appropriate peripheral's buffers.	export/export.h

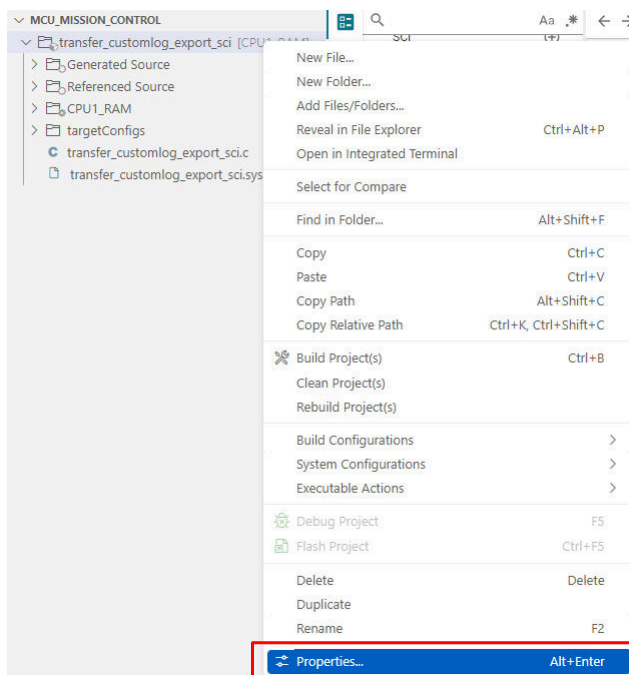
## 4 GUI Creation

The GUI layer leverages GUI Composer on the PC and can be run inside the CCS environment. For more information regarding GUI Composer, refer to the [GUI Composer Documentation](#). The following files are generated to create the GUI composer application once the MCU Control Center Sysconfig module is added to the project.

 <a href="#">transfer.opt</a>	Transfer	<input checked="" type="checkbox"/>
 <a href="#">gui/assets/icons.svg</a>	Transfer	<input checked="" type="checkbox"/>
 <a href="#">gui/index.gui</a>	Transfer	<input checked="" type="checkbox"/>
 <a href="#">gui/index.html</a>	Transfer	<input checked="" type="checkbox"/>
 <a href="#">gui/index.js</a>	Transfer	<input checked="" type="checkbox"/>
 <a href="#">gui/codec.js</a>	Transfer	<input checked="" type="checkbox"/>
 <a href="#">gui/hash_table.json</a>	Transfer	<input checked="" type="checkbox"/>
 <a href="#">gui/index.json</a>	Transfer	<input checked="" type="checkbox"/>
 <a href="#">gui/project.json</a>	Transfer	<input checked="" type="checkbox"/>
 <a href="#">gui/package.json</a>	Transfer	<input checked="" type="checkbox"/>
 <a href="#">gui_setup.bat</a>	Transfer	<input checked="" type="checkbox"/>

**Figure 4-1. SysCfg Generated GUI Layer Files**

The following post build step is required for generating the custom GUI application for CCS. Navigate to *Properties -> General -> Variables* for the specific project and add a variable called *GUI\_SUPPORT*.



**Figure 4-2. Open Project Properties**

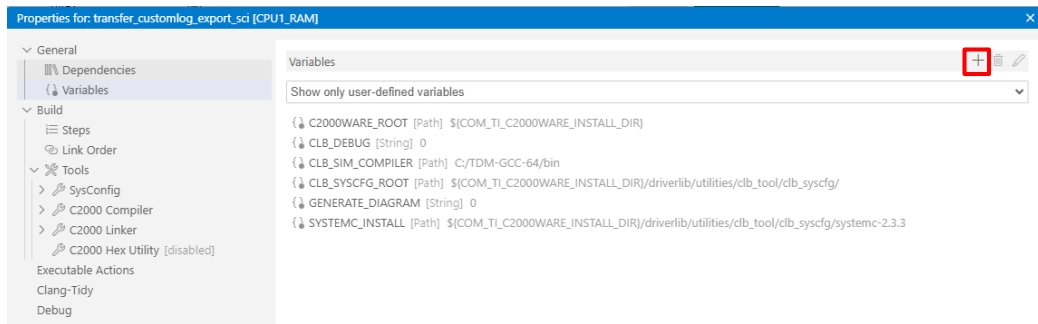


Figure 4-3. Add CCS Variable

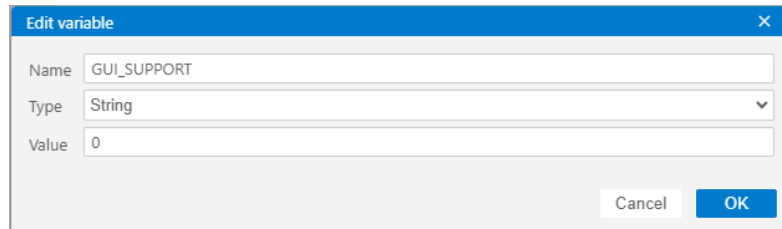


Figure 4-4. GUI\_SUPPORT Variable Added

Additionally, in the project properties, add the following line to the post build step by going to *Build* → *Steps*:

```
if ${GUI_SUPPORT} == 1 ${BuildDirectory}\syscfg\gui_setup.bat
```

Before building the project, make sure the GUI\_SUPPORT variable is set to 1. This copies the GUI files autogenerated by Sysconfig into the proper location in the CCS installation folder at the end of each project build. Once placed in that folder, the GUI application can be successfully launched straight from inside the CCS environment. Once built, the autogenerated GUI project name matches the name selected in the Sysconfig setting below.

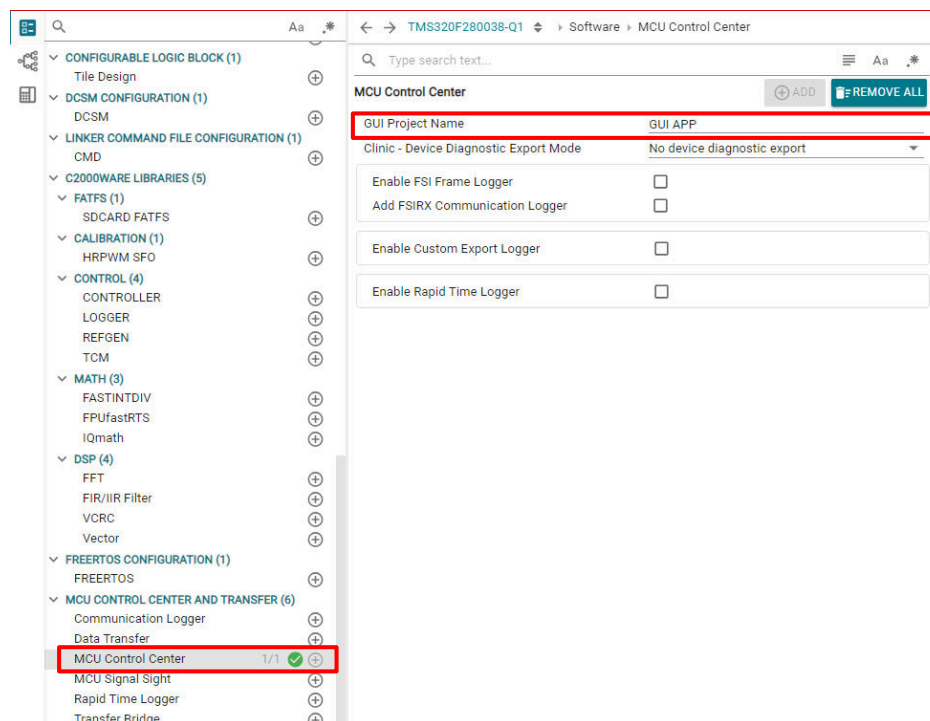
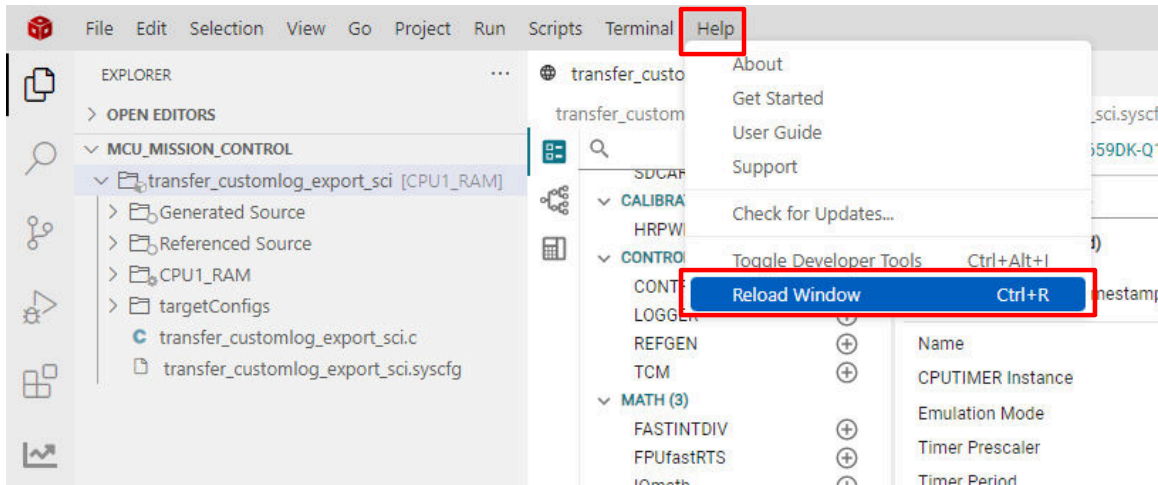


Figure 4-5. Plugin Name

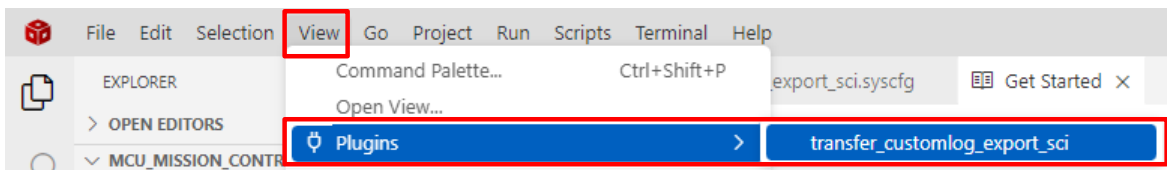


After building the project, refresh the CCS View to see the newly generated GUI in the CCS plugins dropdown. Go to **Help** → **Reload Window**.



**Figure 4-6. Reload Window To See Generated Plugins**

To open the integrated GUI application, go to **View** → **Plugins** → {NameOfGUI}. The GUI opens up in CCS once clicked on.

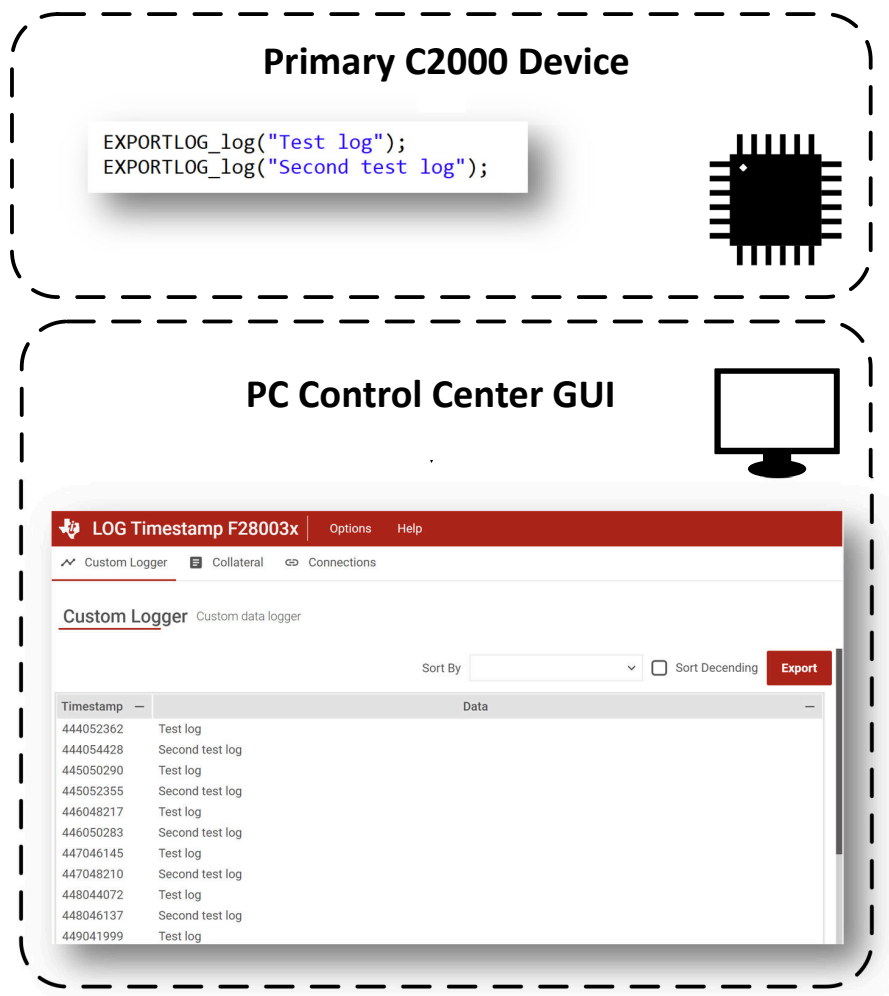


**Figure 4-7. View GUI**

## 5 Application Logging

Application logging can be implemented in applications where there is simply a requirement for the MCU to export data. This feature can export strings, variable values, or arrays, to a PC or bridge device. Application logging basically works as a *printf()* statement that uses a communication peripheral on the device (such as SCI/UART) to send print messages rather than JTAG. Compared to a regular *printf()* call, the *EXPORTLOG\_log()* function allows for a higher speed of transmission and removes the requirement for a debugger connection to view debug data from the device.

This feature can be used directly with the setups in [Section 2.1](#) or [Section 2.2](#) or for [Section 2.3](#) and [Section 2.4](#) with a paired bridge device using the module in [Section 6](#).



**Figure 5-1. Application Logging Diagram**

## 5.1 Application Logging Walk-through

A high-level overview of the Application Logging implementation and all the layers involved from Table 3-1 is shown below. Application Logging allows the user to print out strings and arrays of various data types such as signed, unsigned integers and floats.

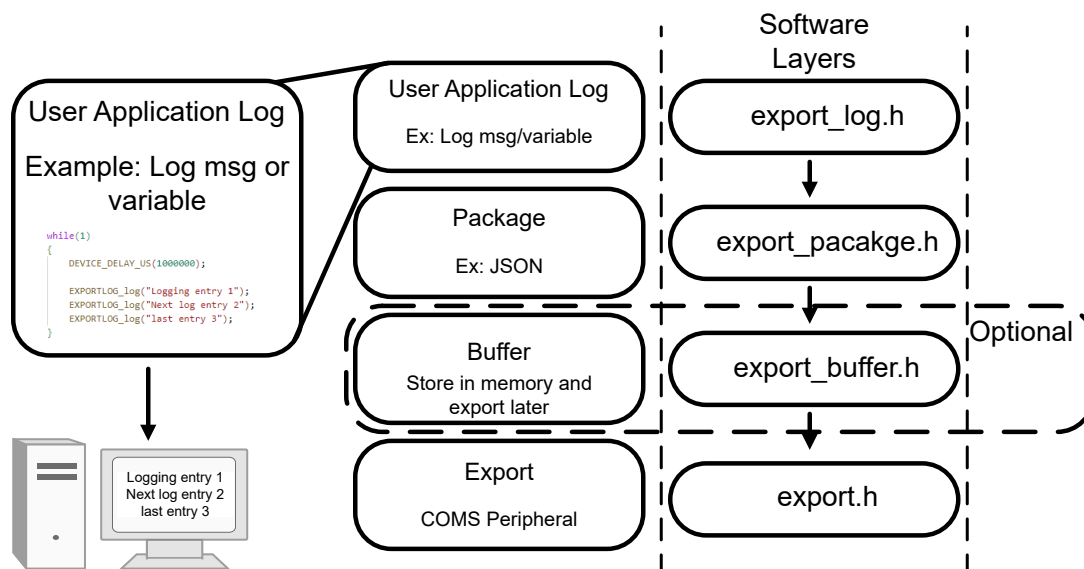


Figure 5-2. High Level Application Logger

The following steps can be done to easily add this feature into a CCS project.

### SysConfig Configurations

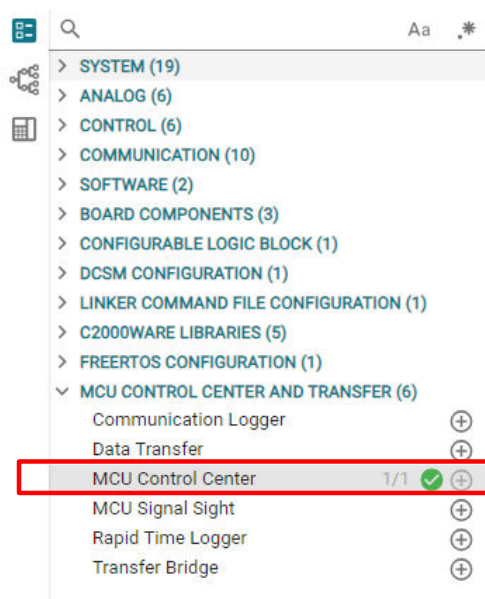
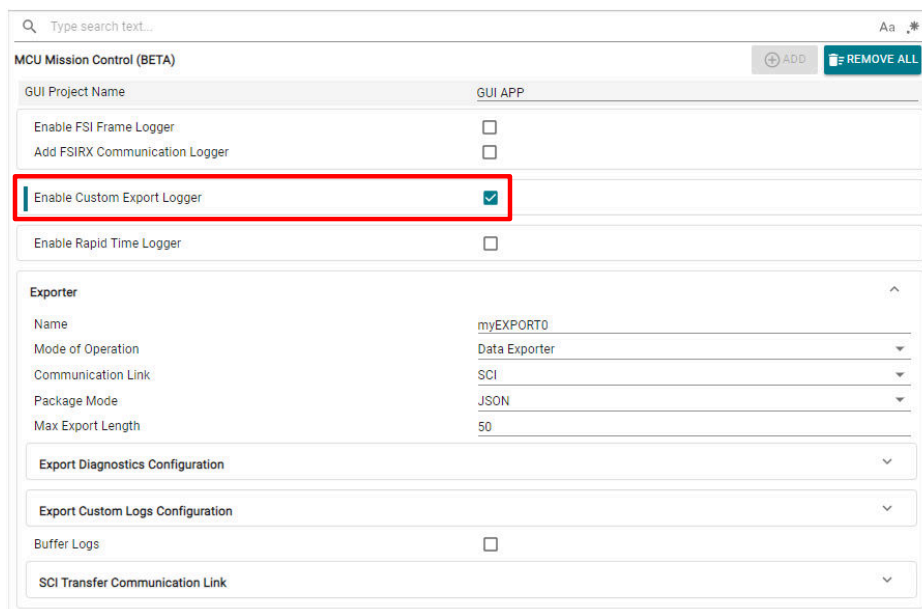


Figure 5-3. Add MCU Control Center Module



MCU Mission Control (BETA)

GUI Project Name      GUI APP

Enable FSI Frame Logger ☐

Add FSIRX Communication Logger ☐

**Enable Custom Export Logger ☒**

Enable Rapid Time Logger ☐

**Exporter**

Name: myEXPORT0

Mode of Operation: Data Exporter

Communication Link: SCI

Package Mode: JSON

Max Export Length: 50

Export Diagnostics Configuration

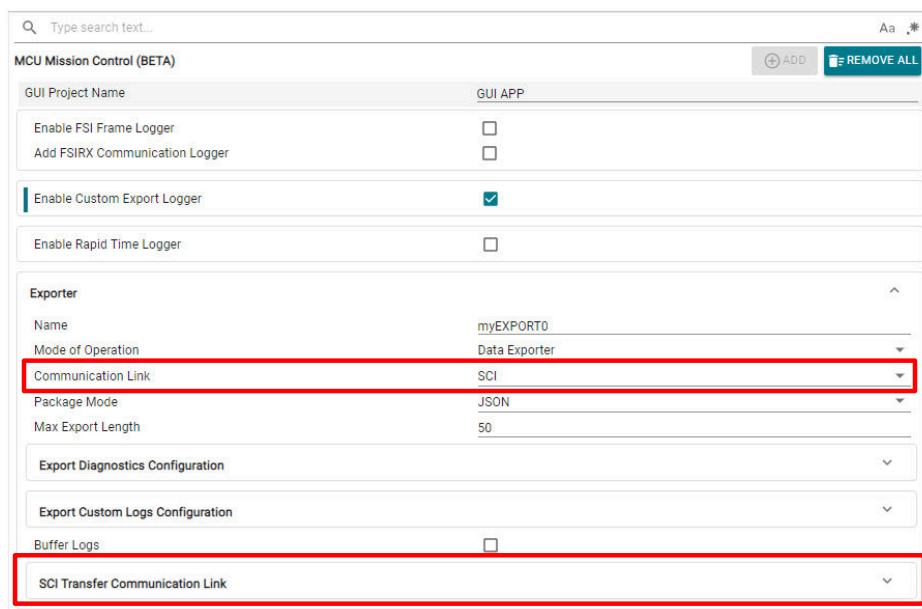
Export Custom Logs Configuration

Buffer Logs ☐

SCI Transfer Communication Link

**Figure 5-4. Custom Export Logger Configuration**

Configure the Exporter module settings for the required communication peripheral. For this walk-through example, SCI is used as the communication peripheral. Under the *Transfer Communication Link* setting, SysConfig automatically adds a communication peripheral available on the device with some default settings that can be edited.



MCU Mission Control (BETA)

GUI Project Name      GUI APP

Enable FSI Frame Logger ☐

Add FSIRX Communication Logger ☐

Enable Custom Export Logger ☒

Enable Rapid Time Logger ☐

**Exporter**

Name: myEXPORT0

Mode of Operation: Data Exporter

**Communication Link: SCI**

Package Mode: JSON

Max Export Length: 50

Export Diagnostics Configuration


Export Custom Logs Configuration

Buffer Logs ☐

**SCI Transfer Communication Link**

**Figure 5-5. Export Submodule Configurations**

All the configurable communication peripheral settings associated when adding the exporter submodule are shown below.



Name	myEXPORT0_SCI
Baud Rate	115200
Actual Baud Rate Value	115741
Baud Rate Error Percent [%]	0.47
Use Loopback Mode	<input type="checkbox"/>
Word Length	8
Stop Mode	1
Parity Mode	No parity
Use Interrupt	<input type="checkbox"/>
Use FIFO	<input checked="" type="checkbox"/>
Use Case	ALL

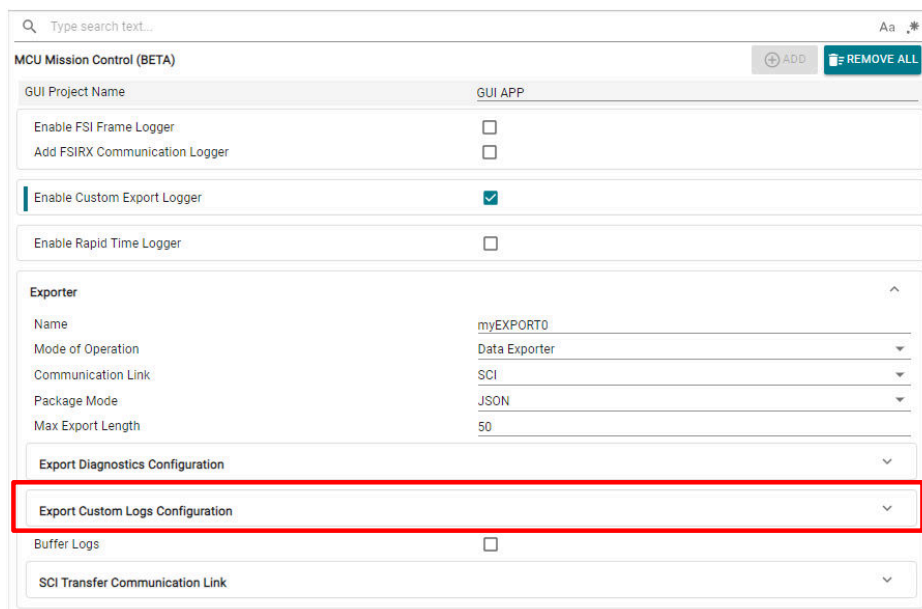
PinMux Qualification

PinMux Peripheral and Pin Configuration

SCI Peripheral	Any(SCIB)
SCL_RX	Any(C1, GPIO200/J1)
SCL_TX	Any(C0, GPIO199/H1)

**Figure 5-6. Transfer Communication Link**

Additional settings that can be added to enhance the data log captures are under the Export Custom Logs Configuration. Export Log Support check box generates functions that can be used under the export or export\_log.h generated files.



MCU Mission Control (BETA)

GUI Project Name

GUI APP

Enable FSI Frame Logger ☐

Add FSIRX Communication Logger ☐

Enable Custom Export Logger ☒

Enable Rapid Time Logger ☐

Exporter

Name	myEXPORT0
Mode of Operation	Data Exporter
Communication Link	SCI
Package Mode	JSON
Max Export Length	50

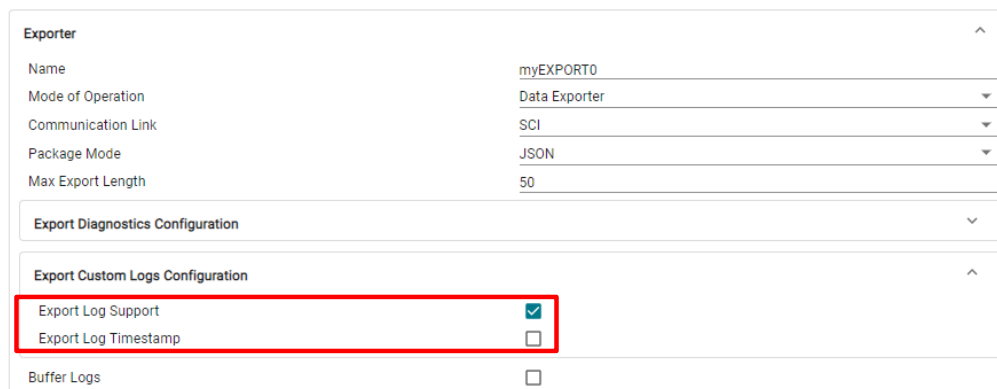
Export Diagnostics Configuration

Export Custom Logs Configuration

Buffer Logs ☐

SCI Transfer Communication Link

**Figure 5-7. Export Custom Logs Configuration**



Exporter	
Name	myEXPORT0
Mode of Operation	Data Exporter
Communication Link	SCI
Package Mode	JSON
Max Export Length	50

Export Diagnostics Configuration

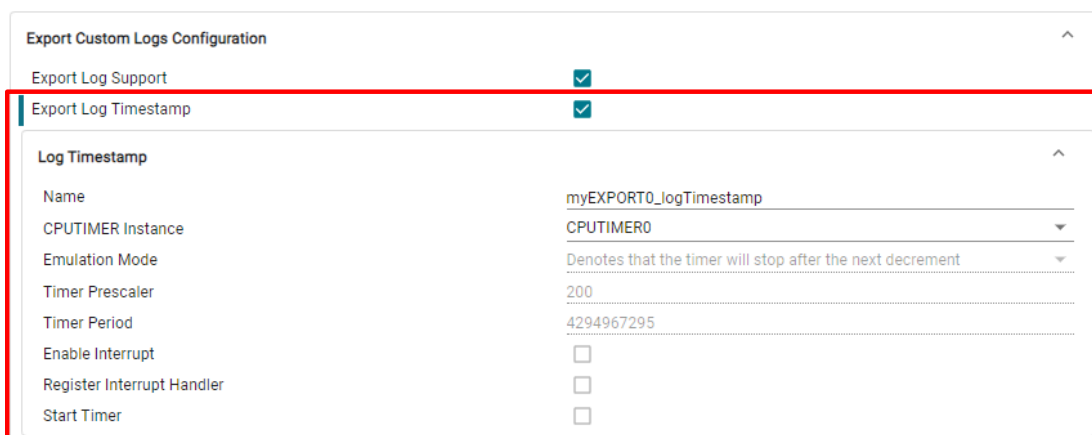
Export Custom Logs Configuration

Export Log Support	<input checked="" type="checkbox"/>
Export Log Timestamp	<input type="checkbox"/>

Buffer Logs ☐

**Figure 5-8. Export Log Support and Timestamp**

Export Log Timestamp is another feature that can be added to log the time that a log was captured. If Export Log Timestamp is enabled, SysConfig automatically adds a CPU Timer module to be configured.



Export Custom Logs Configuration

Export Log Support ☒

Export Log Timestamp ☒

Log Timestamp

Name	myEXPORT0_logTimestamp
CPUTIMER Instance	CPUTIMER0
Emulation Mode	Denotes that the timer will stop after the next decrement
Timer Prescaler	200
Timer Period	4294967295
Enable Interrupt	<input type="checkbox"/>
Register Interrupt Handler	<input type="checkbox"/>
Start Timer	<input type="checkbox"/>

**Figure 5-9. Export Log Timestamp Configuration**

## Application Code

These are all the modules required to be added for the GUI and data logging software layer. The only remaining steps needed to data log strings or variables is to call the export/export\_log.h layer helper functions from the application code. Make sure to include the export/export\_log.h file at the top of the main code.

## Include Files

```
//
// Included Files
//
#include "export/export_log.h"
```

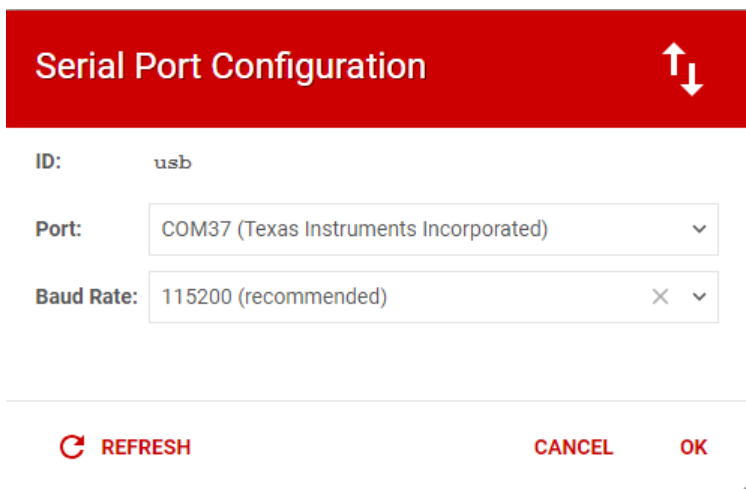
Before building the project, make sure to go over [Section 4](#) to enable GUI\_SUPPORT. Once this is enabled, build the project and open the GUI within CCS.

## Application Code Logging

```
EXPORTLOG_log("Logging entry 1");
EXPORTLOG_log("Next log entry 2");
EXPORTLOG_log("last entry 3");
```

## COM Port Settings for GUI

Load the .out file but do not run yet. Verify the correct COM port is selected for the GUI by going to Options → Serial Port Settings...→ <COM Port>

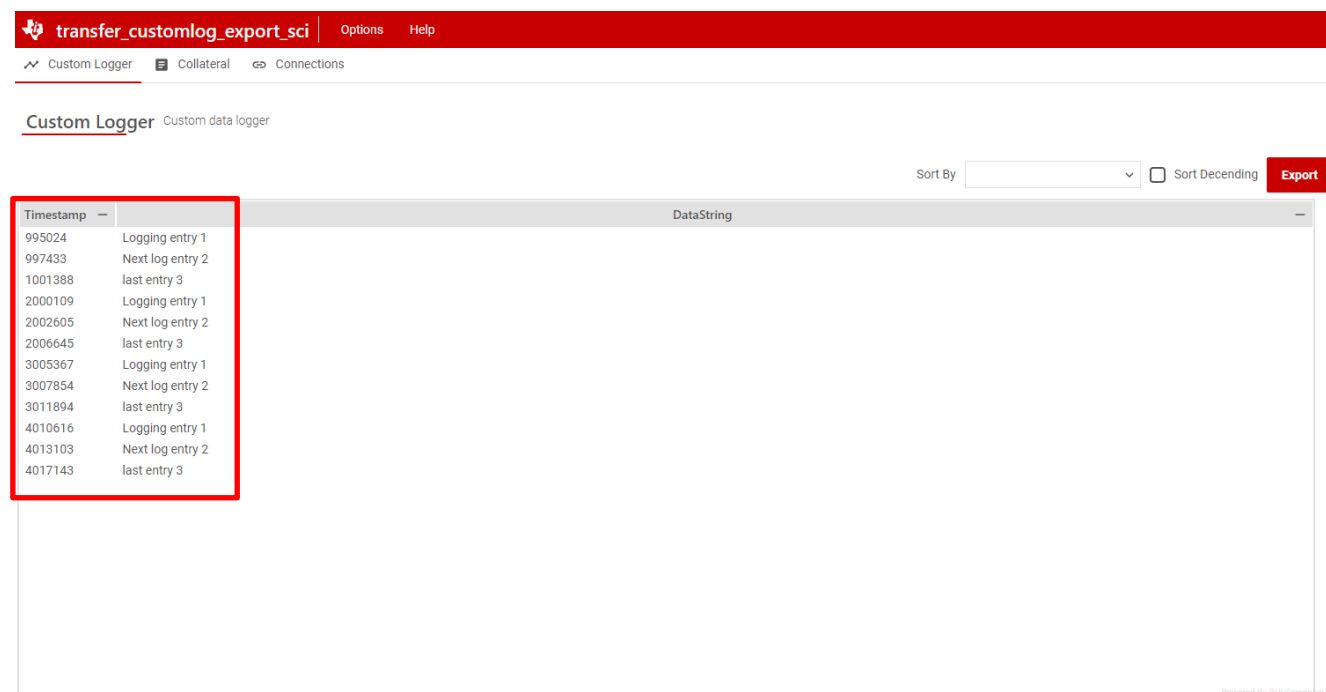


The dialog box is titled "Serial Port Configuration" with a red header bar. It contains three fields: "ID:" with the value "usb", "Port:" with a dropdown menu showing "COM37 (Texas Instruments Incorporated)", and "Baud Rate:" with a dropdown menu showing "115200 (recommended)". At the bottom, there are three buttons: "REFRESH" (with a circular arrow icon), "CANCEL", and "OK".

Figure 5-10. Serial Port Settings

## Final Output

Once the correct COM Port is selected, run the application code. The below image shows the final output. The user can modify these print messages as well as move where in the application the EXPORTLOG\_log() functions are called however needed for the specific application and debugging scenario. See [Section 9](#) for a description of other Application Logging examples provided by the C2000ware SDK.



The screenshot shows the "transfer\_customlog\_export\_sci" application window. The "Custom Logger" tab is active, displaying a table of log entries. The table has two columns: "Timestamp" and "DataString". The entries are grouped into three sets of three, each starting with "Logging entry 1", followed by "Next log entry 2", and "last entry 3". The timestamps are 995024, 997433, 1001388, 2000109, 2002605, 2006645, 3005367, 3007854, 3011894, 4010616, 4013103, and 4017143. The "Export" button is visible in the top right corner.

Timestamp	DataString
995024	Logging entry 1
997433	Next log entry 2
1001388	last entry 3
2000109	Logging entry 1
2002605	Next log entry 2
2006645	last entry 3
3005367	Logging entry 1
3007854	Next log entry 2
3011894	last entry 3
4010616	Logging entry 1
4013103	Next log entry 2
4017143	last entry 3

Figure 5-11. Logger Output

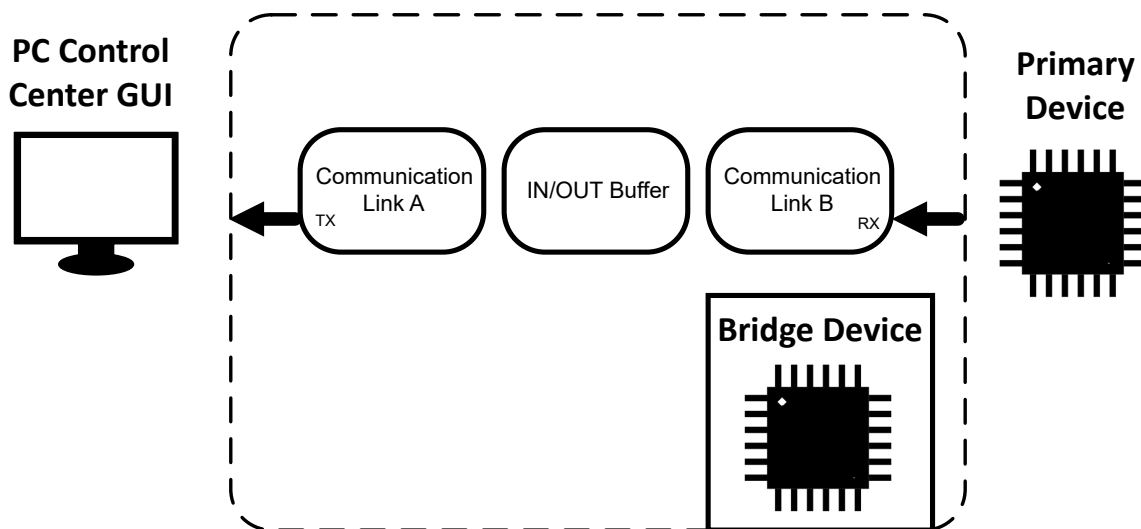
## 6 Transfer Bridge

When data being exported out the primary device is not in a communication protocol supported by the PC (USB), a bridge device (or series of bridge devices) can be used to convert the data packets to the USB protocol. Bridge devices converting a UART protocol to a USB protocol are commonly found on LaunchPADs and controlCARDS, as well as many third party vendors. However, for the setups shown in [Section 2.3](#) or [Section 2.4](#), a different type of bridge firmware is needed; either a FSI-to-USB or a FSI-to-UART bridge firmware. This is where the Transfer Bridge module provided by the MCU Control Center tool can be used to generate code that receives fast communication peripheral packets, buffers the data, then transmits the data through another communication peripheral (UART or USB). The buffering of packets on the bridge device is what allows the data to be quickly exported out of the primary device (without a significant performance hit to the primary application), while still transmitting the data to a PC at a slower speed (either by UART then USB, or through USB).

See [Figure 6-1](#) for a visualization of this concept. Communication Link B represents the communication peripheral used to receive data with the bridge device (often FSI), and Communication Link A represents the communication peripheral used to transmit this data back out of the bridge device to the PC (often UART or USB). The buffer is optionally placed in between the two communication links. Note that in this feature, only the payload of the data packets received by Communication Link B is extracted and forwarded directly to Communication Link A; the Transfer Bridge does not perform any extra data processing.

The below diagram shows the Transfer Bridge feature being used with the setup shown in [Section 2.3](#). If using the setup shown in [Section 2.4](#), then an additional UART-to-USB bridge device is placed between the bridge device and the PC.

This feature can be used with the setup shown in [Section 2.2](#), [Section 2.3](#) or [Section 2.4](#), as well as many other custom hardware setups in a variety of different applications.



**Figure 6-1. Transfer Bridge Diagram**

While the FSI-to-UART bridge is a common use case, the Transfer Bridge feature can be used to generate firmware for other bridge types as well, which can be useful for a variety of applications and hardware setup options. The table below describes all of the different communication link combinations currently supported by the tool. Some are used in the hardware setups highlighted in this document, and others are more application-specific.

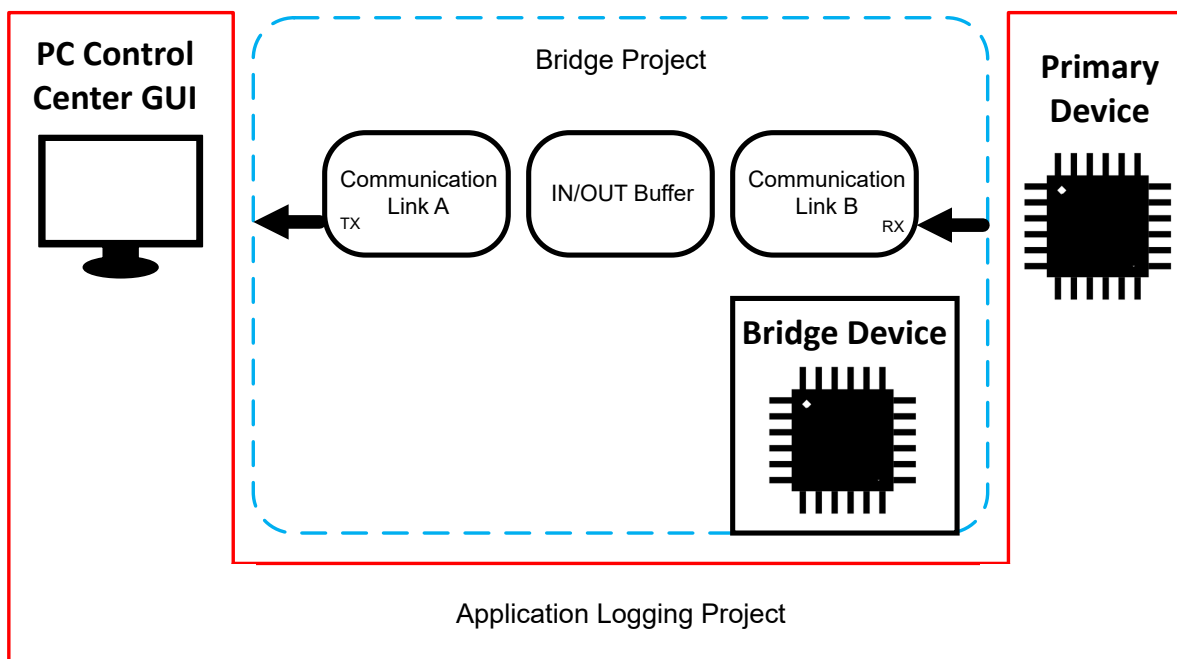


**Table 6-1. Supported Communication Link Combinations**

Communication Link B (RX)	Communication Link A (TX)	Corresponding Hardware Setup
FSI	SCI (UART)	#4
FSI	USB	#3
FSI	SPI	N/A - application specific
SCI (UART)	USB	#2
SCI (UART)	SPI	N/A - application specific
SPI	SCI (UART)	N/A - application specific
SPI	USB	#2 but with SPI messages

## 6.1 Transfer Bridge Walk-through

A high-level view of a bridge project and all the layers involved from [Table 3-1](#) is shown below. The Transfer Bridge module allows the user to receive data by one communication peripheral and transmit this data by a different communication peripheral. The intended purpose is for a bridge device to convert the communication protocol of the data sent (for example, convert FSI frames to UART) between two end points. In certain use cases, the bridge device acts as a buffer for devices using a fast serial communication protocol to a slower communication protocol. The steps needed to add this feature are described in the following walk-through.

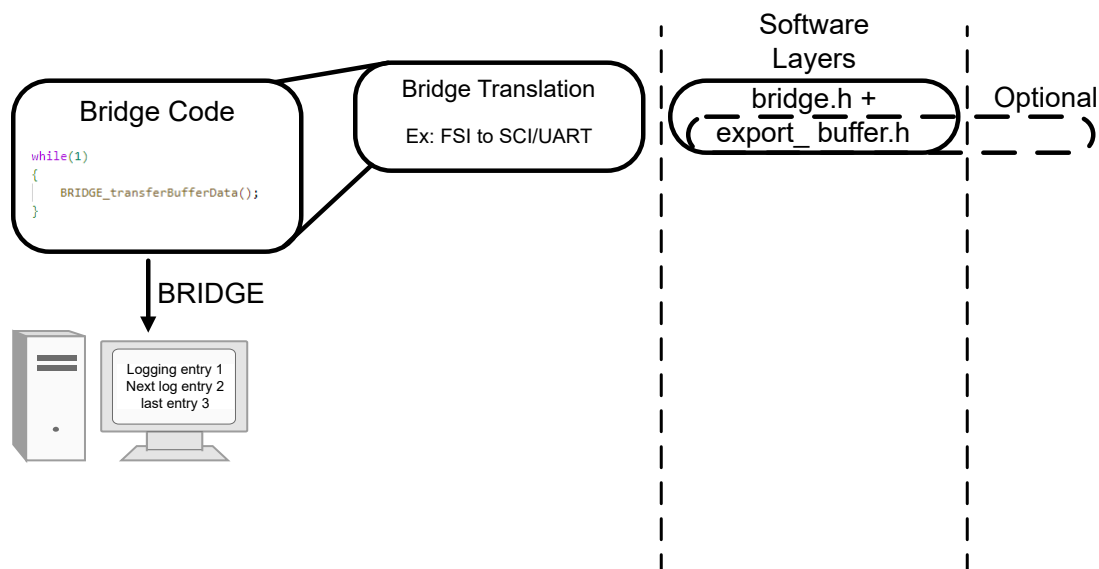


In this figure, the dotted blue line is the Transfer Bridge project, which is used to convert the data from one communication protocol to another. Communication Link B receives data and then transmits the data through Communication Link A. The portion of the diagram outlined in red is the primary device project, which uses the same software implementation as shown in [Section 5.1](#), except for that FSI is selected as the communication link rather than SCI. This walk-through focuses on how to configure just the bridge project portion.

**Figure 6-2. High Level Transfer Bridge Project**

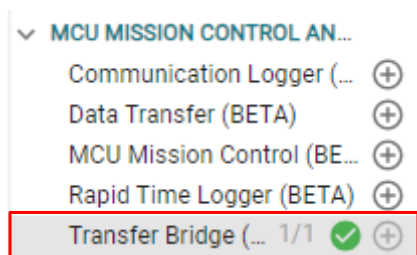
### Software Layer

The following are the software layers required for the bridge project. Note that a buffer layer can optionally be added to the bridge project to increase the memory that is used to store the received data on the device before being exported out.



**Figure 6-3. Bridge Project Software Layer**

## SysConfig Configurations



**Figure 6-4. Transfer Bridge Module**

The figure below shows the Communication Links A and B that are added automatically inside the Transfer Bridge Sysconfig module. Communication Link B receives incoming data using Protocol B and (optionally) stores this data in a buffer. Communication Link A reads the received data (from the Communication Link B or from the buffer) and transmits the data out with Protocol A. For this walk-through, a common use case is shown: the bridge device receives FSI protocol packets and transmits them with the SCI peripheral (UART protocol) packets.

The screenshot shows the 'Transfer Bridge (BETA) (1 of 1 Added)' configuration window. It features a search bar at the top, a list of added bridges (myBridge0), and a table of settings. The 'Communication Link A' is set to 'SCI' and 'Communication Link B' is set to 'FSI'. Below these, there are checkboxes for 'Buffer Received Data' and 'Add Error Handler', both of which are checked. The interface also shows expandable sections for 'SCI Communication Link A', 'FSI RX Communication Link B', and 'Communication Link B Buffer'.

**Figure 6-5. Transfer Bridge Communication Links**

The settings below are optional, and can be used to add a buffer on the bridge device. This option allows the device to accumulate more data before sending the data out when the buffer is full. This feature is useful when the logging device is sending data to the bridge device at a higher speed than the bridge device can process.

This screenshot shows the same configuration window as Figure 6-5, but with additional settings highlighted. The 'Buffer Received Data' checkbox is checked, and the 'Communication Link B Buffer' section is expanded, showing a dropdown menu for selecting the buffer size. The 'Add Error Handler' checkbox is also checked.

**Figure 6-6. Bridge Buffer (Optional)**

## Include Files

```
//
// Included Files
//
...
#include "bridge/bridge.h"
```

## Transfer Bridge Initialization

This step is only required if the buffer is enabled. Add this in the main initialization sequence after *Board\_init()*.

```
//
// Logging Inits
//
BRIDGE_init();
```

## Transfer Bridge Application Logging Code

The final step is to add the application code that handles the receiving of the data packets and transfers the converted packets out of the device.

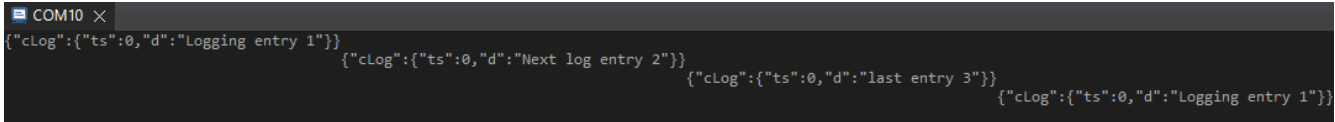
```
while(1)
{
    BRIDGE_transferBufferData();
}
```

## Testing the Bridge Device

To test if the bridge device is functioning correctly, the following steps can be followed. These steps assume there is a different device logging or transmitting data to the bridge device.

1. Flash the bridge device with the Transfer Bridge application program
2. Connect the FSITX pins of the primary device to the bridge device FSIRX pins
  - a. Connect primary device FSITX\_CLK to bridge device FSIRX\_CLK
  - b. Connect primary device FSITX\_D0 to bridge device FSIRX\_D0
  - c. Connect primary device FSITX\_D1 to bridge device FSIRX\_D1 (optional - if dual data lane is configured in *Frame Configuration*)
3. Connect the bridge device to a PC by the USB connector
4. Run the Application Logging project code on the primary device
5. Open a serial port application and configure the port settings to match the SCI module configurations

The final output appears in the serial terminal port.



```
COM10 x
{"cLog":{"ts":0,"d":"Logging entry 1"}}
{"cLog":{"ts":0,"d":"Next log entry 2"}}
{"cLog":{"ts":0,"d":"last entry 3"}}
{"cLog":{"ts":0,"d":"Logging entry 1"}}
```

**Figure 6-7. Transfer Bridge Final Output**

## 7 Communication Logger

Many communication protocols include meaningful data other than the payload. For example, the FSI protocol includes the frame type, user data, CRC byte, and frame tag fields in every frame. The Communication Logger feature provides bridge software that additionally extracts all the non-payload data from each received data packet. This feature allows the bridge device to receive communication peripheral data packets in any packaging format and export the data to a PC GUI to be analyzed. The packets sent to the GUI are formatted such that all parts of the message is designated for display to the user.

Although use cases vary, one application of this feature is debugging of an FSI implementation in an application. In this case, the application is already using FSI to communicate with another MCU, but has some communication errors or issues that need to be debugged. The Communication Logger can display every part of the FSI frames transmitted by the primary device in a human-readable format inside of the generated GUI.

Another use case is for data logging with a fast peripheral, similar to the use case of the [Section 6](#) feature, but with more message details logged in the GUI. The Communication Logger feature is used to receive and (optionally) buffer high-speed FSI messages coming from the primary device, extract all elements of the message contents, and send them via UART or USB to the PC for visualization in a GUI.

The diagram below shows the Communication Logger feature being used with the setup shown in [Section 2.3](#). Note that for this feature, the bridge device Sysconfig project is used to generate the Control Center PC GUI since the data packing/unpacking scheme between the bridge device and the GUI need to be commonly understood. If using [Section 2.4](#), then an additional UART-to-USB bridge device is placed between the bridge device and the PC.

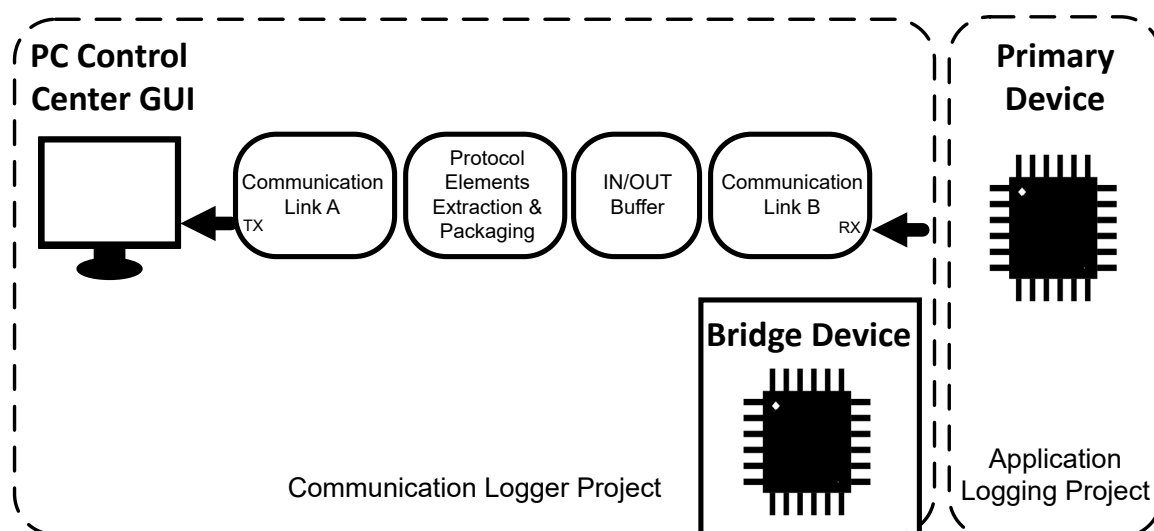


Figure 7-1. Communication Logger Diagram

### Note

Currently, this feature only supports the FSI protocol as the incoming protocol (Protocol B).

### 7.1 Communication Logger Walk-through

A high level view of the communication logger feature and all the layers involved from [Table 3-1](#) is shown below. The Communication Logger feature displays each element of the received FSI frame in a separate column of the logging table in the Control Center GUI. The steps needed to add this functionality to a CCS project are described in the following walk-through.

## Software Layers

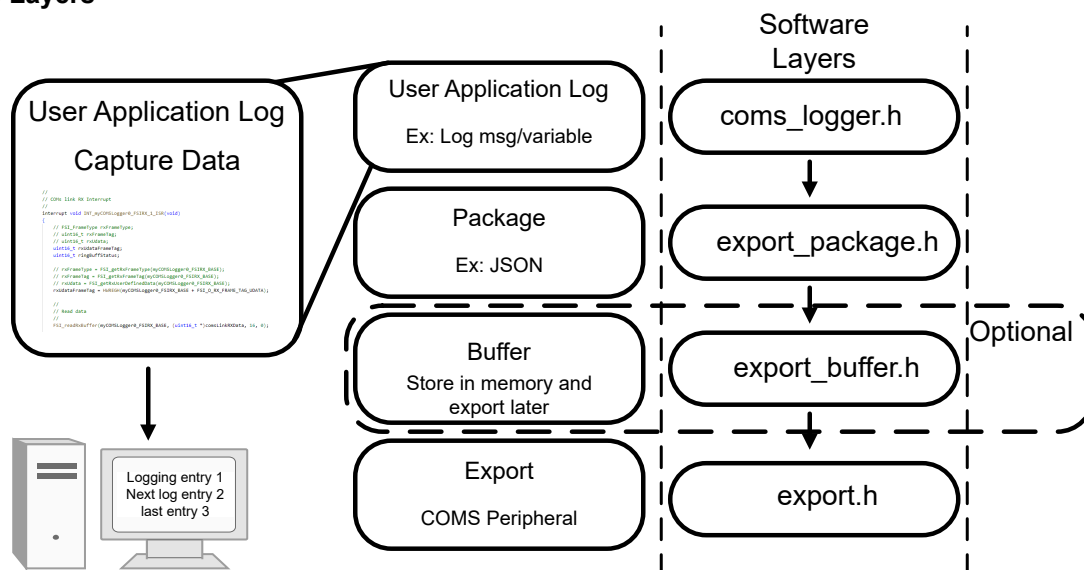


Figure 7-2. Communication Logger Software Layer

## Sysconfig Configurations

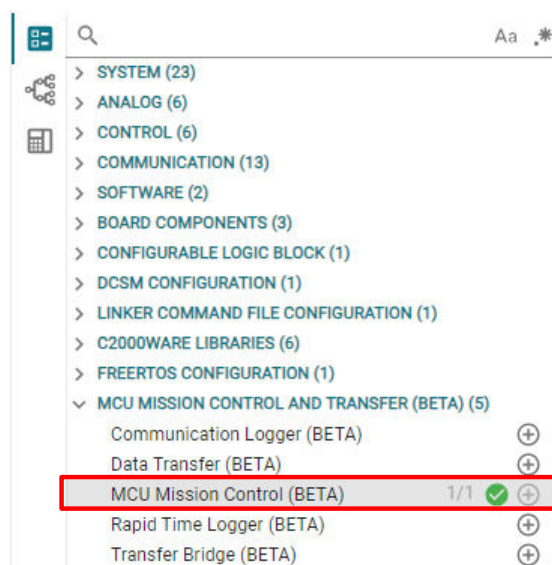
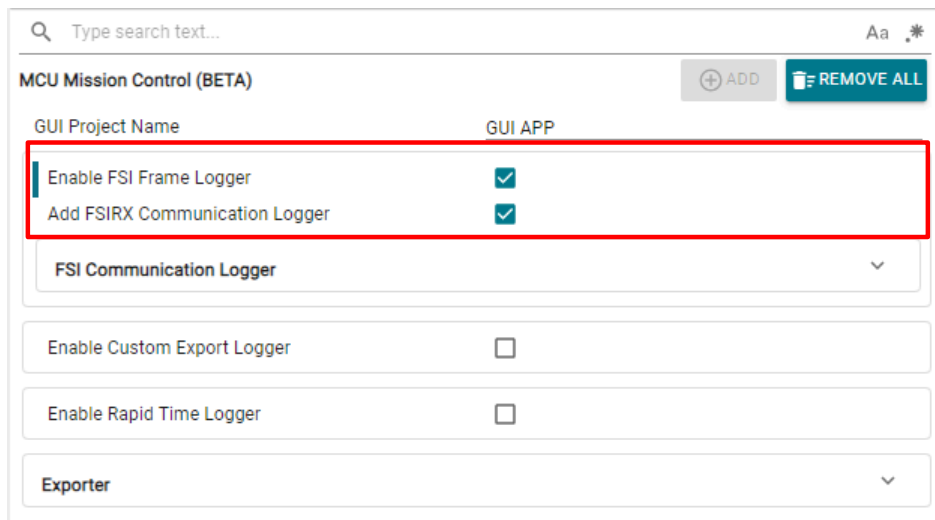


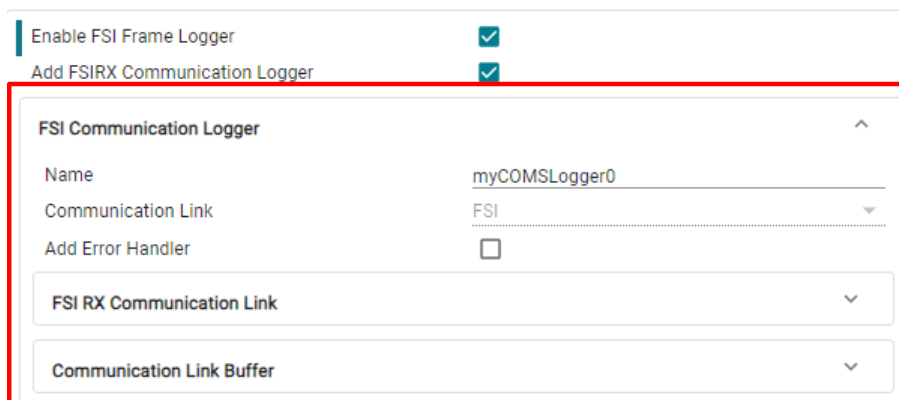
Figure 7-3. MCU Control Center Module



MCU Mission Control (BETA) [ADD] [REMOVE ALL]

GUI Project Name	GUI APP
Enable FSI Frame Logger	<input checked="" type="checkbox"/>
Add FSIRX Communication Logger	<input checked="" type="checkbox"/>
FSI Communication Logger	
Enable Custom Export Logger	<input type="checkbox"/>
Enable Rapid Time Logger	<input type="checkbox"/>
Exporter	

**Figure 7-4. Enable FSI Logger and Communication Logger**



Enable FSI Frame Logger ☒  
Add FSIRX Communication Logger ☒

FSI Communication Logger

Name: myCOMSLogger0  
Communication Link: FSI  
Add Error Handler: ☐  
FSI RX Communication Link  
Communication Link Buffer

**Figure 7-5. FSI Communication Logger Configurations**

## Include Files

```
//  
// Included Files  
//  
...  
#include "export/export.h"  
#include "logger/coms_logger.h"
```

## Communication Logger Initialization

```
//  
// Logging Inits  
//  
EXPORT_init();  
COMSLOG_init();
```

## Communication Logger Application Code

```
while(1)  
{  
    COMSLOG_transferBufferData();  
}
```

## Communication Logger Error Handling (Optional)

```
void COMSLOG_transferBufferOverflow() {
    //
    // Received too much data too quickly. The transfer buffer overflowed
    // Make the buffer larger
    //
    ESTOP0;
}

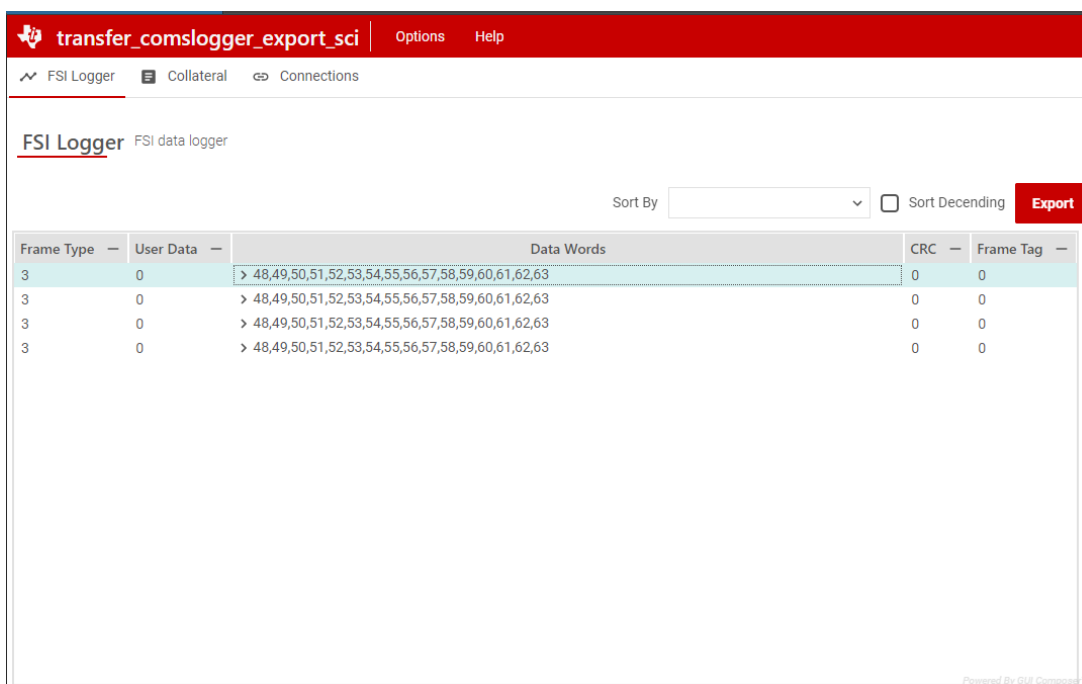
void COMSLOG_comsLinkError(uint16_t status) {
    //
    // FSI receive error occurred
    // Bad frames received
    //
    ESTOP0;
}
```

## Build the Project

Build the communication logger application code, and make sure to follow the steps in [Section 4](#) to generate the GUI inside CCS.

## Testing the Communication Logger Tool

- Flash the Communication Logger application to the bridge device
- Connect the bridge device's FSIRX pins to the primary device FSITX pins
  - Connect primary device FSITX\_CLK to bridge device FSIRX\_CLK
  - Connect primary device FSITX\_D0 to bridge device FSIRX\_D0
  - Connect primary device FSITX\_D1 to bridge device FSIRX\_D1 (optional - if dual data lane is configured in *Frame Configuration*)
- Connect the bridge device to the PC by using the USB connector
- Run the Application Logger application project on the primary device
- Open the generated GUI from inside CCS
- The final output appears as shown below



The screenshot shows the 'transfer\_comslogger\_export\_sci' application window. The 'FSI Logger' tab is active, displaying a table of captured data frames. The table has columns for Frame Type, User Data, Data Words, CRC, and Frame Tag. The data shows four frames, all with Frame Type 3, User Data 0, and Data Words starting with 48,49,50,51,52,53,54,55,56,57,58,59,60,61,62,63. The CRC and Frame Tag are 0 for all frames.

Frame Type	User Data	Data Words	CRC	Frame Tag
3	0	> 48,49,50,51,52,53,54,55,56,57,58,59,60,61,62,63	0	0
3	0	> 48,49,50,51,52,53,54,55,56,57,58,59,60,61,62,63	0	0
3	0	> 48,49,50,51,52,53,54,55,56,57,58,59,60,61,62,63	0	0
3	0	> 48,49,50,51,52,53,54,55,56,57,58,59,60,61,62,63	0	0

Figure 7-6. Final Output



## 8 Rapid-Time Logger

Previously, with the Communication Logger feature, viewing of the additional data elements provided in each FSI packet was enabled. The Rapid-Time Logger takes advantage of these additional elements by using the User Data element to encode the type of logging message and the Frame Type element to store which variable from the application is being sent. This form of data logging is the fastest because this form leverages the FSI packaging features unique to the FSI protocol as the packaging layer of the application. With the setups shown in [Section 2.3](#) or [Section 2.4](#), the Rapid-Time Logger feature can be used on the primary device in conjunction with a special implementation of the Communication Logger feature on the bridge device.

This feature is commonly used for debugging or viewing of application variables when the primary application has tight timing requirements. The full utilization of the FSI protocol features on the primary device allow for fast and optimized transmission of data. In conjunction, the bridge device extracts the data from every FSI element and transmits to the PC (either directly by the USB peripheral or through the secondary UART-to-USB bridge device using UART). A JSON file containing the encodings for each application variable and message type is generated by the Rapid-Time Logger Sysconfig module for use by the GUI application. This allows each data packet to display meaningfully in the GUI without the need to transmit extra characters in the FSI packet.

### 8.1 Rapid Time Logging Walk-through

A high level overview of the Rapid Time Logger feature and all the layers involved from [Table 3-1](#) is shown below. The steps to enable the Rapid Time Logger feature on the primary device as well as the enhance Communication Logger feature on the bridge device are described in the following walk-through.

#### Software Layers

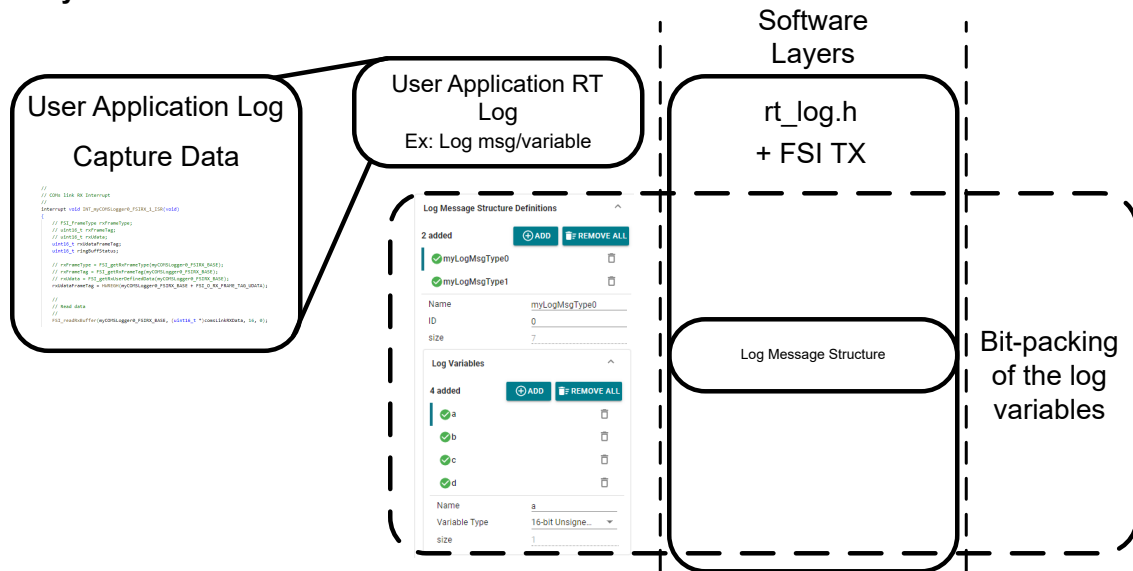


Figure 8-1. Rapid-Time Logger Software Layer

#### Sysconfig Configurations

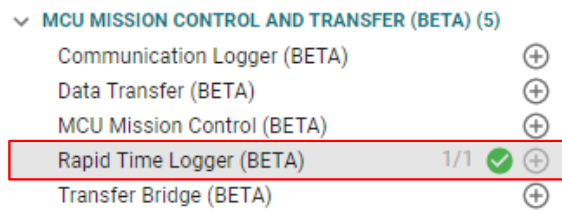
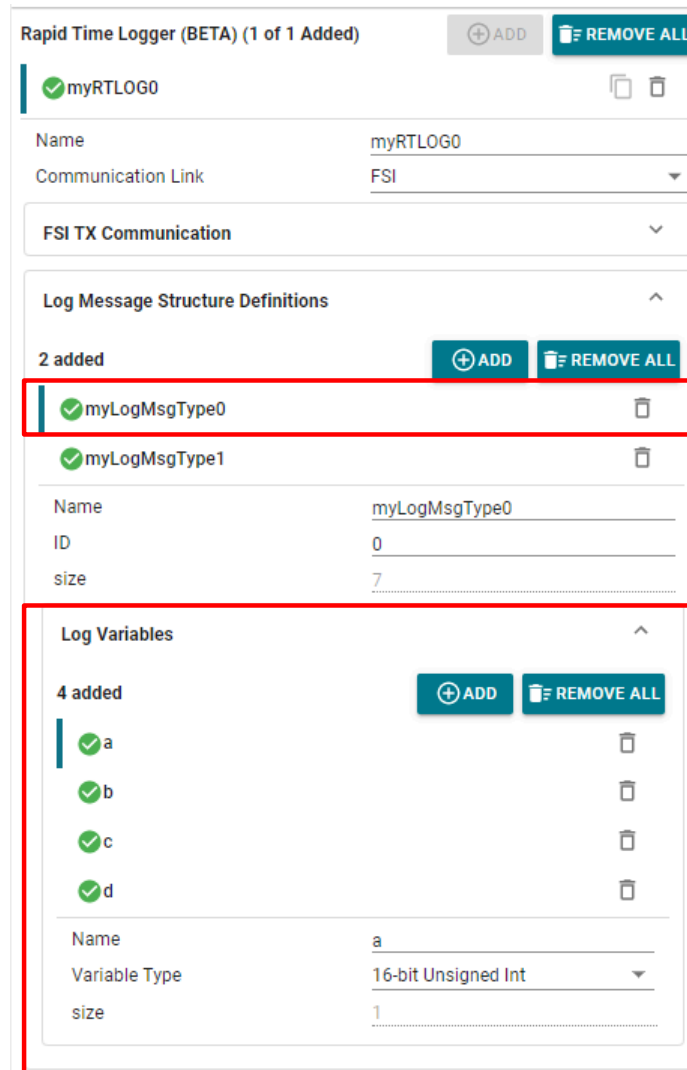


Figure 8-2. Rapid-Time Logger SysConfig

For this walk-through, two example message structures have been added in the GUI. Under the *Log Message Structure* view, add two instances. Each instance represents a specific message structure that is designated in the packet. Different message structure definitions can be used for various application specific purposes (for example: message type 0 is sent before some event and message type 1 is sent after some event in the application). For this example, add four variables named *a*, *b*, *c*, and *d* in the first log message structure definition. In the second log message structure definition, add another variable named *e*.

**Table 8-1. Example Log Variable Settings**

Variable	Variable Type
a	16-bit unsigned integer
b	32-bit unsigned integer
c	32-bit floating point
d	Array of 16-bit unsigned integers Length of array : 2
e	Array of 32-bit floating points Length of array: 8



The screenshot shows the 'Rapid Time Logger (BETA) (1 of 1 Added)' window. It features a list of log message structures, with 'myRTLOG0' selected. Below this, the 'Log Message Structure Definitions' section shows two added structures: 'myLogMsgType0' and 'myLogMsgType1'. The 'myLogMsgType0' structure is highlighted with a red box. Its details are shown below: Name 'myLogMsgType0', ID '0', and size '7'. The 'Log Variables' section for 'myLogMsgType0' is also highlighted with a red box, showing four added variables: 'a', 'b', 'c', and 'd'. Each variable has a unique name, variable type, and size. The details for variable 'a' are shown below: Name 'a', Variable Type '16-bit Unsigned Int', and size '1'.

In the above figure, each log variable can have a unique name, variable type, and size. The size is automatically calculated based on the variable type. For this walk-through configure *a*, *b*, *c*, *d*, and *e* according to the table for each structure type.

**Figure 8-3. Log Message Structure 0 Definitions**

Log Message Structure Definitions

2 added

+ ADD
REMOVE ALL

myLogMsgType0

myLogMsgType1

Name
myLogMsgType1

ID
1

size
16

Log Variables

1 added

+ ADD
REMOVE ALL

e

Name
e

Variable Type
Array of 32-bit Floating Poi...

Length of Array
8

size
16

Figure 8-4. Log Message Structure 1 Definitions

## Include Files and Global Variables

```
//
// Included Files
//
...
#include "logger/rt_log.h"
uint16_t a = 0;
uint32_t b = 6798004;
float c = -189.4934;
uint16_t d[2] = {19872, 290};
float e[8] = {
    1243.43, -4399.24, -23.392, 0.0213,
    -2093, 238.4993, -2390.300, 329.401
};
volatile uint16_t toggle = 0;
```

## Rapid Time Logger Initialization

```
//
// Logging Inits
//
RTLOG_init();
```

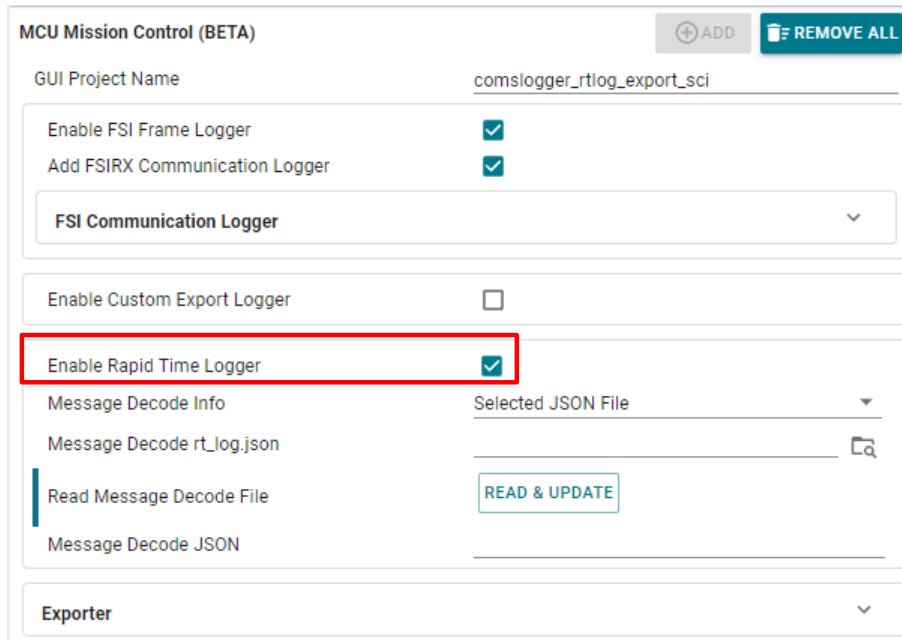
## Add Rapid Time Logs in Application Code

```
// Insert delay if required for debugging purposes
DEVICE_DELAY_US(1000000);
if (toggle == 0)
{
    RTLOG_writeLog_0(a, b, c, d);
}
else {
    RTLOG_writeLog_1(e);
}
toggle ^= 1;
```

## Communication Logger Additional Steps

The only steps left are setting up the communication logger feature for the bridge device as described in [Section 7.1](#) and add some additional steps. TI provides a JSON file that contains encodings for all the variables sent over the FSI TX frame that the communication logger uses to decode the Rapid Time Logger messages. The steps below are needed after the primary project has been configured and the steps from the [Section 5.1](#) have been followed on the bridge project.

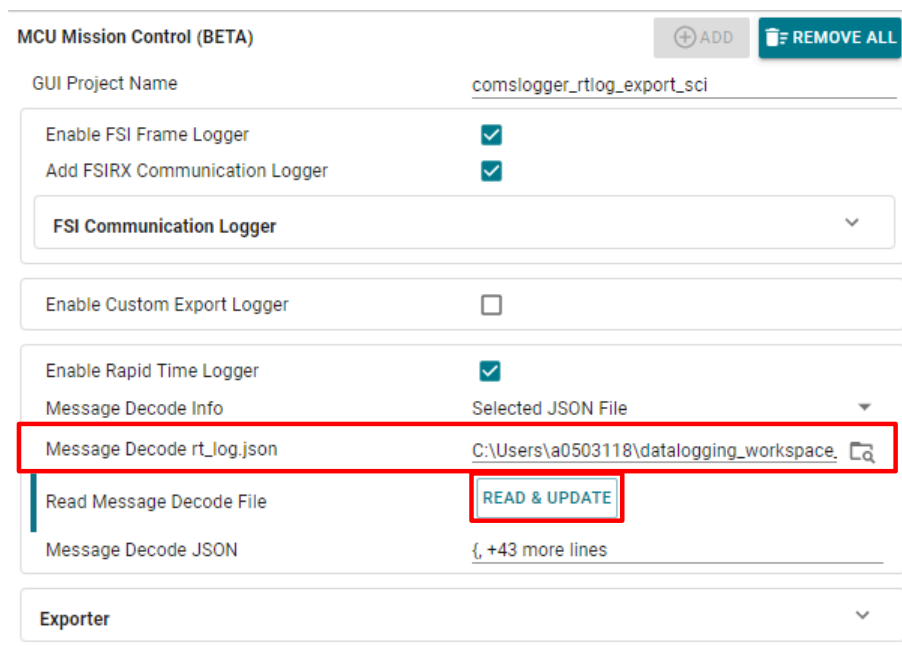
1. Select *Enable Rapid Time Logger* in the MCU Control Center Sysconfig module.



The screenshot shows the 'MCU Mission Control (BETA)' interface. At the top, there are 'ADD' and 'REMOVE ALL' buttons. Below, the 'GUI Project Name' is 'comslogger\_rtlog\_export\_sci'. Under the 'FSI Communication Logger' section, 'Enable FSI Frame Logger' and 'Add FSIRX Communication Logger' are both checked. In the 'Enable Custom Export Logger' section, the 'Enable Rapid Time Logger' checkbox is checked and highlighted with a red rectangle. Below this, the 'Message Decode Info' section shows 'Message Decode rt\_log.json' with a 'Selected JSON File' dropdown and a 'READ & UPDATE' button. The 'Message Decode JSON' field is empty. At the bottom, there is an 'Exporter' dropdown.

**Figure 8-5. Enable Rapid Time Logger**

2. Search for the `rt_log.json` file located in the build folder of the Rapid Time Logger project.
  - a. For example, `<workspace_ccs>/<name_of_project>/<build_folder>/syscfg/logger/rt_log.json`



This screenshot is similar to Figure 8-5, but the 'Message Decode rt\_log.json' field is now populated with the file path 'C:\Users\ao503118\datalogging\_workspace\'. This field and the 'READ & UPDATE' button are highlighted with a red rectangle. The 'Message Decode JSON' field now shows '{, +43 more lines'.

**Figure 8-6. Navigate to JSON `rt_log.json` file**

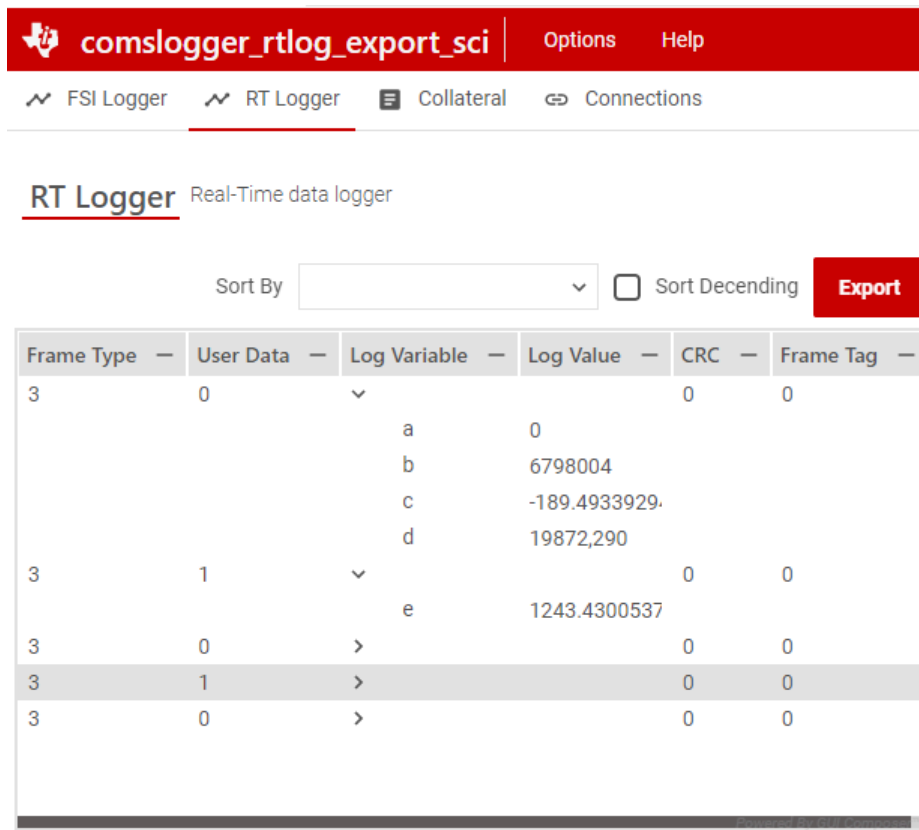
## Build the Project

Build the communication logger application code, and make sure to follow the steps in [Section 4](#) to generate the GUI inside CCS.

## Testing the Real Time Logger Feature

1. Flash the Communication Logger application project on the bridge device
2. Connect the communication logging device's FSIRX pins to the real time logging device's FSITX pins
  - a. Connect primary device FSITX\_CLK to bridge device FSIRX\_CLK
  - b. Connect primary device FSITX\_D0 to bridge device FSIRX\_D0
  - c. Connect primary device FSITX\_D1 to bridge device FSIRX\_D1 (Optional - if dual data lane is configured in *Frame Configuration*)
3. Connect the bridge device to the PC by using the USB connector
4. Run the Rapid Time Logger application project on the primary device
5. Open the generated GUI inside CCS
6. The final output appears as shown below

## Final Output



The screenshot shows the 'comslogger\_rtlog\_export\_sci' application window. The 'RT Logger' tab is active, displaying 'Real-Time data logger'. Above the table, there is a 'Sort By' dropdown menu, a 'Sort Descending' checkbox, and an 'Export' button. The table has columns: Frame Type, User Data, Log Variable, Log Value, CRC, and Frame Tag. The data is as follows:

Frame Type	User Data	Log Variable	Log Value	CRC	Frame Tag
3	0	▼		0	0
		a	0		
		b	6798004		
		c	-189.4933929		
		d	19872,290		
3	1	▼		0	0
		e	1243.4300537		
3	0	>		0	0
3	1	>		0	0
3	0	>		0	0

**Figure 8-7. View the Rapid-Time Log Data in the PC GUI**

## 9 Transfer Examples Overview

The walk-through previously described in this document gave instructions for how to add data logging support to an existing CCS project. Additionally, C2000 example code for each of the MCU Control Center features can be referenced in the [C2000Ware SDK](#) in the following path: `C2000Ware_VERSION#/driverlib/[DEVICE_GPN]/examples/[CORE_IF_MULTICORE]/transfer/`. Descriptions of each example, along with how examples can be used together for an exporting device and a bridge device are shown below.

Primary Device Example	Bridge Device Example	Description	Hardware Setup
transfer_customlog_export_usb	N/A	<ul style="list-style-type: none"> <li><i>Feature:</i> Application Logging</li> <li><i>Primary Communication Protocol:</i> USB</li> <li><i>Primary Packaging Format:</i> JSON</li> <li>Logs simple text-based messages in a GUI every 1 second (basically a printf using USB).</li> </ul>	<a href="#">Setup #1</a>
transfer_customlog_export_sci	(1)	<ul style="list-style-type: none"> <li><i>Feature:</i> Application Logging</li> <li><i>Primary Communication Protocol:</i> UART</li> <li><i>Primary Packaging Format:</i> JSON</li> <li>Logs simple text-based messages in a GUI every 1 second (basically a printf using UART).</li> </ul>	<a href="#">Setup #2</a>
transfer_customlog_export_sci_buffer	(1)	<ul style="list-style-type: none"> <li><i>Feature:</i> Application Logging</li> <li><i>Primary Communication Protocol:</i> UART</li> <li><i>Primary Packaging Format:</i> JSON</li> <li>Buffers and logs simple text-based messages in a GUI every 1 second using separate processes (basically a printf using UART for a system with tight timings).</li> </ul>	<a href="#">Setup #2</a>
transfer_customlog_export_sci_logArrays	(1)	<ul style="list-style-type: none"> <li><i>Feature:</i> Application Logging</li> <li><i>Primary Communication Protocol:</i> UART</li> <li><i>Primary Packaging Format:</i> JSON</li> <li>Logs text-based and numerical data arrays in a GUI every 1 second.</li> </ul>	<a href="#">Setup #2</a>
transfer_raw_fsi_tx	transfer_comslogger_export_sci	<ul style="list-style-type: none"> <li><i>Feature:</i> Communication Logger</li> <li><i>Primary Communication Protocol:</i> FSI</li> <li><i>Primary Packaging Format:</i> None (only FSI frames)</li> <li>Log raw data using FSI - Extract all raw FSI data and transmit via SCI to display in a GUI</li> </ul>	<a href="#">Setup #4</a>
transfer_customlog_export_fsi	transfer_bridge_sci	<ul style="list-style-type: none"> <li><i>Feature:</i> Application Logger and Transfer Bridge</li> <li><i>Primary Communication Protocol:</i> FSI</li> <li><i>Primary Packaging Format:</i> JSON</li> <li>Log data using FSI (JSON format) - Receive JSON formatted FSI data and transmit payload via SCI to display in a GUI</li> </ul>	<a href="#">Setup #4</a>

Primary Device Example	Bridge Device Example	Description	Hardware Setup
transfer_rtlog	transfer_comslogger_rtlog_export_sci	<ul style="list-style-type: none"> <li><i>Feature:</i> Real-Time Logger and Communication Logger</li> <li><i>Primary Communication Protocol:</i> FSI</li> <li><i>Primary Packaging Format:</i> Custom (according to frame structures)</li> <li>Log real-time data using FSI in a custom byte packing format - Extract received FSI data elements and transmit by SCI to display in a GUI intelligently.</li> </ul>	<a href="#">Setup #4</a>

- (1) This example requires a UART-to-USB bridge device. All C2000 LaunchPADs and controlCARDs have this bridge device incorporated on the board already and require no extra hardware for viewing the data on a PC. When using a custom board for the logging device, an external UART-to-USB bridge device is required.

## 10 Summary

MCU Control Center provides a data logging implementation with a custom GUI application to capture and analyze logged data. The features available in this tool can be added to any project to implement a variety of use cases. The MCU Control Center tool makes collecting data from a C2000 device seamless through autogenerated target code support, an intuitive graphical user interface GUI, and efficient use of the communication peripherals available on a given C2000 device.

## 11 References

- Texas Instruments, [MCU Signal Sight Tool Developer's Guide](#), application note
- Texas Instruments, [C2000 Real-Time Microcontrollers Peripherals](#), user's guide
- Texas Instruments, [C2000 Sysconfig](#), application note
- Texas Instruments, [DLT Developer's Guide With Tooling](#), application note

## IMPORTANT NOTICE AND DISCLAIMER

TI PROVIDES TECHNICAL AND RELIABILITY DATA (INCLUDING DATA SHEETS), DESIGN RESOURCES (INCLUDING REFERENCE DESIGNS), APPLICATION OR OTHER DESIGN ADVICE, WEB TOOLS, SAFETY INFORMATION, AND OTHER RESOURCES "AS IS" AND WITH ALL FAULTS, AND DISCLAIMS ALL WARRANTIES, EXPRESS AND IMPLIED, INCLUDING WITHOUT LIMITATION ANY IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE OR NON-INFRINGEMENT OF THIRD PARTY INTELLECTUAL PROPERTY RIGHTS.

These resources are intended for skilled developers designing with TI products. You are solely responsible for (1) selecting the appropriate TI products for your application, (2) designing, validating and testing your application, and (3) ensuring your application meets applicable standards, and any other safety, security, regulatory or other requirements.

These resources are subject to change without notice. TI grants you permission to use these resources only for development of an application that uses the TI products described in the resource. Other reproduction and display of these resources is prohibited. No license is granted to any other TI intellectual property right or to any third party intellectual property right. TI disclaims responsibility for, and you will fully indemnify TI and its representatives against, any claims, damages, costs, losses, and liabilities arising out of your use of these resources.

TI's products are provided subject to [TI's Terms of Sale](#) or other applicable terms available either on [ti.com](https://www.ti.com) or provided in conjunction with such TI products. TI's provision of these resources does not expand or otherwise alter TI's applicable warranties or warranty disclaimers for TI products.

TI objects to and rejects any additional or different terms you may have proposed.

Mailing Address: Texas Instruments, Post Office Box 655303, Dallas, Texas 75265  
Copyright © 2025, Texas Instruments Incorporated