Logger

Component Design Document

1 Description

The Logger component receives data of generic statically-sized, or variable-sized type. This data is synchronously added to an internal circular buffer. By default, the logging of this data can be disabled at component start and can be enabled via command. Various commands also exist to dump the internal circular buffer. The circular buffer of the logger can either be declared on the heap or in a static memory location, via the component's Init subprogram.

2 Requirements

The requirements for the Logger component are specified below.

1. The component shall store incoming data of a generic type into a memory buffer.
2. The component shall be enabled or disabled via command.
3. The default state of the component should be disabled upon initialization.
4. The component shall be able to dump its memory contents upon command.
5. The component shall publish a data product that includes its enabled/disabled status.

3 Design

3.1 At a Glance

Below is a list of useful parameters and statistics that give a quick look into the makeup of the component.

- **Execution** - *passive*
- **Number of Connectors** - 7
- **Number of Invokee Connectors** - 2
- **Number of Invoker Connectors** - 5
- **Number of Generic Connectors** - *None*
- **Number of Generic Types** - 2
- **Number of Unconstrained Arrayed Connectors** - *None*
- **Number of Commands** - 7
- **Number of Parameters** - *None*
- **Number of Events** - 6
- **Number of Faults** - *None*

- **Number of Data Products** - 1
- **Number of Data Dependencies** - *None*
- **Number of Packets** - 1

3.2 Diagram

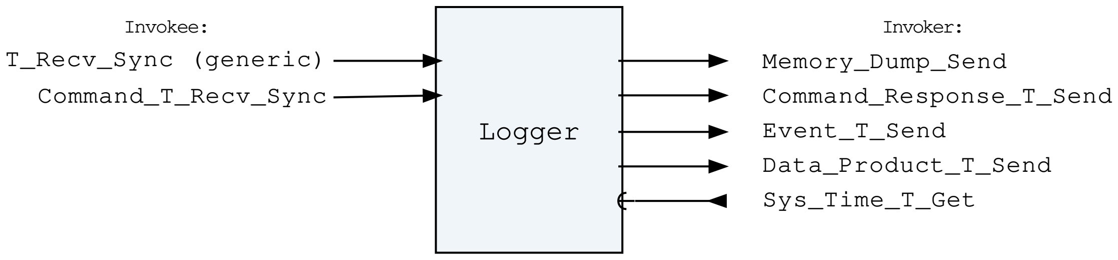

Figure 1: Logger component diagram.

3.3 Connectors

Below are tables listing the component's connectors.

3.3.1 Invokee Connectors

The following is a list of the component's *invokee* connectors:

Table 1: Logger Invokee Connectors

Name	Kind	Type	Return_Type	Count
`T_Recv_Sync`	`recv_sync`	`T` (generic)	-	1
`Command_T_Recv_Sync`	`recv_sync`	`Command.T`	-	1

Connector Descriptions:

- **`T_Recv_Sync`** - The generic log data connector.
- **`Command_T_Recv_Sync`** - This is the command receive connector.

3.3.2 Invoker Connectors

The following is a list of the component's *invoker* connectors:

Table 2: Logger Invoker Connectors

Name	Kind	Type	Return_Type	Count
`Memory_Dump_Send`	`send`	`Memory_Packetizer_Types.Memory_Dump`	-	1
`Command_Response_T_Send`	`send`	`Command_Response.T`	-	1
`Event_T_Send`	`send`	`Event.T`	-	1
`Data_Product_T_Send`	`send`	`Data_Product.T`	-	1
`Sys_Time_T_Get`	`get`	-	`Sys_Time.T`	1

Connector Descriptions:

- **`Memory_Dump_Send`** - The memory dump connector.
- **`Command_Response_T_Send`** - This connector is used to register and respond to the component's commands.
- **`Event_T_Send`** - Events are sent out of this connector.
- **`Data_Product_T_Send`** - Data products are sent out of this connector.
- **`Sys_Time_T_Get`** - The system time is retrieved via this connector.

3.4 Initialization

Below are details on how the component should be initialized in an assembly.

3.4.1 Generic Component Instantiation

The component is parameterized by the type that is stores in its internal log. To support variable length packed records, a subprogram is also provided which determines the length (in bytes) of the incoming type. This component contains generic formal types. These generic formal types must be instantiated with a valid actual type prior to component initialization. This is done by specifying types for the following generic formal parameters:

Table 3: Logger Generic Formal Types

Name	Formal Type Definition
`T`	`type T is private;`
`Serialized_Length`	`with function Serialized_Length (Src : in T; Num_Bytes_Serialized : out Natural) return Serializer_Types.Serialization_Status;`

Generic Formal Type Descriptions:

- **`T`** - The generic type of data passed in to be logged.
- **`Serialized_Length`** - A method that returns the serialized length of an item of type T. This is useful for serializing variable length packed types onto the log.

3.4.2 Component Instantiation

This component contains no instantiation parameters in its discriminant.

3.4.3 Component Base Initialization

This component contains no base class initialization, meaning there is no `init_Base` subprogram for this component.

3.4.4 Component Set ID Bases

This component contains commands, events, packets, faults, or data products that require a base identifier to be set at initialization. The `set_Id_Bases` procedure must be called with the following parameters:

Table 4: Logger Set Id Bases Parameters

Name	Type
Command_Id_Base	Command_Types.Command_Id_Base
Data_Product_Id_Base	Data_Product_Types.Data_Product_Id_Base
Event_Id_Base	Event_Types.Event_Id_Base
Packet_Id_Base	Packet_Types.Packet_Id_Base

Parameter Descriptions:

- **Command_Id_Base** - The value at which the component's command identifiers begin.
- **Data_Product_Id_Base** - The value at which the component's data product identifiers begin.
- **Event_Id_Base** - The value at which the component's event identifiers begin.
- **Packet_Id_Base** - The value at which the component's unresolved packet identifiers begin.

3.4.5 Component Map Data Dependencies

This component contains no data dependencies.

3.4.6 Component Implementation Initialization

The calling of this implementation class initialization procedure is mandatory. This init function provides memory allocation for the logger's internal memory. Preallocated memory can be provided via the "bytes" access type, in which case "size" must be negative and will be ignored. If you would like to allocate the internal memory on the heap then "bytes" must be set to null, and "size" must be a positive number representing the number of bytes you would like to allocate. The `init` subprogram requires the following parameters:

Table 5: Logger Implementation Initialization Parameters

Name	Type	Default Value
Bytes	Basic_Types.Byte_Array_Access	null
Meta_Data	Circular_Buffer_Meta.T_Access	null
Size	Integer	-1
Initial_Mode	Logger_Enums.Logger_Mode.E	Logger_Enums.Logger_Mode.Disabled

Parameter Descriptions:

- **Bytes** - A pointer to an allocation of bytes to be used for storing log data. If this is set to null, then memory will be allocated on the heap using the "size" parameter instead. Note: This must be set to null if the "size" parameter is positive below.
- **Meta_Data** - A pointer to an allocation of a meta data record for storing the log meta data. This can be used to place the meta data where desired in memory. This item must be set to null if "size" is positive, and non-null if "bytes" is non-null.
- **Size** - The number of bytes to allocate on the heap for memory storage. Note: This must be set to a negative value if the "bytes" parameters is not null.
- **Initial_Mode** - The initial mode of the logger (enabled/disabled) upon initialization

3.5 Commands

These are the commands for the logger component.

Table 6: Logger Commands

Local ID	Command Name	Argument Type
0	`Enable`	-
1	`Disable`	-
2	`Dump_Log`	-
3	`Dump_Newest_Data`	`Packed_Positive_Length.T`
4	`Dump_Oldest_Data`	`Packed_Positive_Length.T`
5	`Dump_Log_Memory`	-
6	`Send_Meta_Data_Event`	-

Command Descriptions:

- **`Enable`** - Enable the logger to start saving data.
- **`Disable`** - Disable the logger from saving received data.
- **`Dump_Log`** - Dump the entire log oldest to newest data.
- **`Dump_Newest_Data`** - Dump the newest X bytes of data from the log.
- **`Dump_Oldest_Data`** - Dump the oldest X bytes of data from the log.
- **`Dump_Log_Memory`** - Dump the entire region of memory associated with the logger from start to finish in memory byte order.
- **`Send_Meta_Data_Event`** - Send an event out with the meta data of the log.

3.6 Events

Below is a list of the events for the Logger component.

Table 7: Logger Events

Local ID	Event Name	Parameter Type
0	`Log_Attempt_Failed`	`Logger_Error.T`
1	`Log_Disabled`	`Circular_Buffer_Meta.T`
2	`Log_Enabled`	-
3	`Log_Info_Update`	`Logger_Info.T`
4	`Dumping_Log_Memory`	`Memory_Region.T`
5	`Invalid_Command_Received`	`Invalid_Command_Info.T`

Event Descriptions:

- **`Log_Attempt_Failed`** - A log attempt failed with the following status.
- **`Log_Disabled`** - The log was disabled. No more data will be stored.
- **`Log_Enabled`** - The log was enabled. Data will now be stored.
- **`Log_Info_Update`** - The current meta data of the log was requested.
- **`Dumping_Log_Memory`** - Currently dumping log memory from the following location.
- **`Invalid_Command_Received`** - A command was received with invalid parameters.

3.7 Data Products

Data products for the Logger component.

Table 8: Logger Data Products

Local ID	Data Product Name	Type
`0x0000 (0)`	`Mode`	`Logger_Status.T`

Data Product Descriptions:

- **`Mode`** - The current enabled/disabled mode of the component.

3.8 Packets

Packets for the logger.

Table 9: Logger Packets

Local ID	Packet Name	Type
`0x0000 (0)`	`Log_Packet`	*Undefined*

Packet Descriptions:

- **`Log_Packet`** - This packet contains log data.

4 Unit Tests

The following section describes the unit test suites written to test the component.

4.1 *Variable_ Tests* Test Suite

This is a unit test suite for the Logger component which logs a variable sized log type onto a log instantiated statically, not on the heap.

Test Descriptions:

- **`Test_Log_And_Dump`** - This unit test tests the storing of variable length log data and subsequent dumping by command.
- **`Test_Logger_Error`** - This unit test tests the behavior when the logger receives a poorly formatted variable type.
- **`Test_Invalid_Command`** - This unit test makes sure an invalid command is reported and ignored.

4.2 *Logger_ Tests* Test Suite

This is a unit test suite for the Logger component which logs a statically sized log type onto a log instantiated on the heap.

Test Descriptions:

- **`Test_Log_And_Dump_Enabled`** - This unit test tests the storing of log data and subsequent dumping by command when the log is enabled.
- **`Test_Log_And_Dump_Disabled`** - This unit test tests the storing of log data and subsequent dumping by command when the log is disabled.

- **`Test_Log_Overwrite_And_Dump`** - This unit test tests the storing of a lot of log data, such that the circular buffer overwrites, and subsequent dumping by command.
- **`Test_Enable_Disable`** - This unit test tests the enabled/disable commands to the logger to make sure they behave as expected.
- **`Test_Init`** - This unit test tests initializing the log with both valid and invalid values.
- **`Test_Invalid_Command`** - This unit test makes sure an invalid command is reported and ignored.

5 Appendix

5.1 Packed Types

The following section outlines any complex data types used in the component in alphabetical order. This includes packed records and packed arrays that might be used as connector types, command arguments, event parameters, etc..

Circular_Buffer_Meta.T:

This record holds meta data associated with a circular buffer data structure.

Table 10: Circular_Buffer_Meta Packed Record : 96 bits

Name	**Type**	**Range**	**Size (Bits)**	**Start Bit**	**End Bit**
`Head`	`Interfaces. Unsigned_32`	`0 to 4294967295`	32	0	31
`Count`	`Interfaces. Unsigned_32`	`0 to 4294967295`	32	32	63
`Size`	`Interfaces. Unsigned_32`	`0 to 4294967295`	32	64	95

Field Descriptions:

- **`Head`** - The head index of the buffer.
- **`Count`** - The number of bytes currently used in the buffer.
- **`Size`** - The total size of the buffer in bytes.

Command.T:

Generic command packet for holding arbitrary commands

Table 11: Command Packed Record : 2080 bits *(maximum)*

Name	**Type**	**Range**	**Size (Bits)**	**Start Bit**	**End Bit**	**Variable Length**
`Header`	`Command_ Header.T`	-	40	0	39	-
`Arg_Buffer`	`Command_ Types. Command_Arg_ Buffer_Type`	-	2040	40	2079	`Header.Arg_ Buffer_Length`

Field Descriptions:

- **`Header`** - The command header
- **`Arg_Buffer`** - A buffer that contains the command arguments

Command_Header.T:

Generic command header for holding arbitrary commands

Table 12: Command_Header Packed Record : 40 bits

Name	Type	Range	Size (Bits)	Start Bit	End Bit
`Source_Id`	`Command_Types. Command_Source_Id`	`0 to 65535`	16	0	15
`Id`	`Command_Types. Command_Id`	`0 to 65535`	16	16	31
`Arg_Buffer_Length`	`Command_Types. Command_Arg_Buffer_ Length_Type`	`0 to 255`	8	32	39

Field Descriptions:

- **`Source_Id`** - The source ID. An ID assigned to a command sending component.
- **`Id`** - The command identifier
- **`Arg_Buffer_Length`** - The number of bytes used in the command argument buffer

Command_Response.T:

Record for holding command response data.

Table 13: Command_Response Packed Record : 56 bits

Name	Type	Range	Size (Bits)	Start Bit	End Bit
`Source_Id`	`Command_ Types.Command_ Source_Id`	`0 to 65535`	16	0	15
`Registration_ Id`	`Command_ Types.Command_ Registration_ Id`	`0 to 65535`	16	16	31
`Command_Id`	`Command_Types. Command_Id`	`0 to 65535`	16	32	47

`Status`	`Command_Enums. Command_ Response_ Status.E`	`0 => Success`
		`1 => Failure`
		`2 => Id_Error`
		`3 => Validation_Error`
		`4 => Length_Error`
		`5 => Dropped`
		`6 => Register`
		`7 => Register_Source`
			8	48	55

Field Descriptions:

- **`Source_Id`** - The source ID. An ID assigned to a command sending component.
- **`Registration_Id`** - The registration ID. An ID assigned to each registered component at initialization.

- **Command_Id** - The command ID for the command response.
- **Status** - The command execution status.

Data_Product.T:

Generic data product packet for holding arbitrary data types

Table 14: Data_Product Packed Record : 344 bits *(maximum)*

Name	Type	Range	Size (Bits)	Start Bit	End Bit	Variable Length
Header	Data_Product_ Header.T	-	88	0	87	-
Buffer	Data_Product_ Types.Data_ Product_ Buffer_Type	-	256	88	343	Header.Buffer_ Length

Field Descriptions:

- **Header** - The data product header
- **Buffer** - A buffer that contains the data product type

Data_Product_Header.T:

Generic data_product packet for holding arbitrary data_product types

Table 15: Data_Product_Header Packed Record : 88 bits

Name	Type	Range	Size (Bits)	Start Bit	End Bit
Time	Sys_Time.T	-	64	0	63
Id	Data_Product_Types. Data_Product_Id	0 to 65535	16	64	79
Buffer_Length	Data_Product_ Types.Data_Product_ Buffer_Length_Type	0 to 32	8	80	87

Field Descriptions:

- **Time** - The timestamp for the data product item.
- **Id** - The data product identifier
- **Buffer_Length** - The number of bytes used in the data product buffer

Event.T:

Generic event packet for holding arbitrary events

Table 16: Event Packed Record : 344 bits *(maximum)*

Name	Type	Range	Size (Bits)	Start Bit	End Bit	Variable Length
Header	Event_Header.T	-	88	0	87	-
Param_Buffer	Event_Types. Parameter_ Buffer_Type	-	256	88	343	Header.Param_ Buffer_Length

Field Descriptions:

- **Header** - The event header
- **Param_Buffer** - A buffer that contains the event parameters

Event_Header.T:

Generic event packet for holding arbitrary events

Table 17: Event_Header Packed Record : 88 bits

Name	Type	Range	Size (Bits)	Start Bit	End Bit
Time	Sys_Time.T	-	64	0	63
Id	Event_Types.Event_Id	0 to 65535	16	64	79
Param_Buffer_Length	Event_Types.Parameter_Buffer_Length_Type	0 to 32	8	80	87

Field Descriptions:

- **Time** - The timestamp for the event.
- **Id** - The event identifier
- **Param_Buffer_Length** - The number of bytes used in the param buffer

Invalid_Command_Info.T:

Record for holding information about an invalid command

Table 18: Invalid_Command_Info Packed Record : 112 bits

Name	Type	Range	Size (Bits)	Start Bit	End Bit
Id	Command_Types.Command_Id	0 to 65535	16	0	15
Errant_Field_Number	Interfaces.Unsigned_32	0 to 4294967295	32	16	47
Errant_Field	Basic_Types.Poly_Type	-	64	48	111

Field Descriptions:

- **Id** - The command Id received.
- **Errant_Field_Number** - The field that was invalid. 1 is the first field, 0 means unknown field, 2**32 means that the length field of the command was invalid.
- **Errant_Field** - A polymorphic type containing the bad field data, or length when Errant_Field_Number is 2**32.

Logger_Error.T:

A packed record which holds status information about a failed log attempt.

Table 19: Logger_Error Packed Record : 40 bits

Name	Type	Range	Size (Bits)	Start Bit	End Bit
Num_Bytes_Logged	Natural	0 to 2147483647	32	0	31
Status	Logger_Enums.Log_Attempt_Status.E	0 => Success 1 => Serialization_Failure 2 => Too_Full	8	32	39

Field Descriptions:

- **`Num_Bytes_Logged`** - The number of bytes that was attempted to store.
- **`Status`** - The returned status from the log attempt.

Logger_Info.T:

A packed record which holds information about the internal status of the log.

Table 20: Logger_Info Packed Record : 104 bits

Name	Type	Range	Size (Bits)	Start Bit	End Bit
Meta_Data	Circular_Buffer_Meta.T	-	96	0	95
Current_Mode	Logger_Enums.Logger_Mode.E	0 => Disabled 1 => Enabled	8	96	103

Field Descriptions:

- **`Meta_Data`** - The current meta data of the internal circular buffer.
- **`Current_Mode`** - Is the log enabled or disabled?

Logger_Status.T:

A packed record which holds the enabled/disabled state of the logger

Table 21: Logger_Status Packed Record : 8 bits

Name	Type	Range	Size (Bits)	Start Bit	End Bit
Current_Mode	Logger_Enums.Logger_Mode.E	0 => Disabled 1 => Enabled	8	0	7

Field Descriptions:

- **`Current_Mode`** - Is the log enabled or disabled?

Memory_Region.T:

A memory region described by a system address and length (in bytes).

Table 22: Memory_Region Packed Record : 96 bits

Name	Type	Range	Size (Bits)	Start Bit	End Bit
Address	System.Address	-	64	0	63
Length	Natural	0 to 2147483647	32	64	95

Field Descriptions:

- **Address** - The starting address of the memory region.
- **Length** - The number of bytes at the given address to associate with this memory region.

Packed_Positive_Length.T:

Single component record for holding packed Positive value that represents a length.

Table 23: Packed_Positive_Length Packed Record : 32 bits

Name	Type	Range	Size (Bits)	Start Bit	End Bit
Length	Positive	1 to 2147483647	32	0	31

Field Descriptions:

- **Length** - The 32-bit Positive Integer that represents a length.

Sys_Time.T:

A record which holds a time stamp using GPS format including seconds and subseconds since epoch (1-5-1980 to 1-6-1980 midnight).

Table 24: Sys_Time Packed Record : 64 bits

Name	Type	Range	Size (Bits)	Start Bit	End Bit
Seconds	Interfaces. Unsigned_32	0 to 4294967295	32	0	31
Subseconds	Interfaces. Unsigned_32	0 to 4294967295	32	32	63

Field Descriptions:

- **Seconds** - The number of seconds elapsed since epoch.
- **Subseconds** - The number of 1/(2^32) sub-seconds.

5.2 Enumerations

The following section outlines any enumerations used in the component.

Command_Enums.Command_Response_Status.E:

This status enumeration provides information on the success/failure of a command through the command response connector.

Table 25: Command_Response_Status Literals:

Name	Value	Description

Success	0	Command was passed to the handler and successfully executed.
Failure	1	Command was passed to the handler not successfully executed.
Id_Error	2	Command id was not valid.
Validation_Error	3	Command parameters were not successfully validated.
Length_Error	4	Command length was not correct.
Dropped	5	Command overflowed a component queue and was dropped.
Register	6	This status is used to register a command with the command routing system.
Register_Source	7	This status is used to register command sender's source id with the command router for command response forwarding.

Logger_Enums.Logger_Mode.E:

This flag denotes whether the log is currently enabled or disabled.

Table 26: Logger_Mode Literals:

Name	Value	Description
Disabled	0	The log is disabled and not currently logging data.
Enabled	1	The log is enabled and currently logging data.

Logger_Enums.Log_Attempt_Status.E:

This enumerations returns the status of a log attempt.

Table 27: Log_Attempt_Status Literals:

Name	Value	Description
Success	0	Log action was successful.
Serialization_Failure	1	Logging failed due to a serialization error.
Too_Full	2	Logging failed because the log was too full to fit the data.