Command Router

Component Design Document

1 Description

The Command Router component receives incoming commands and routes them to the appropriate component for execution. Commands IDs are registered by components connected to the command router on initialization. These registrations are used to populate a router table (binary tree) which is used to translate incoming Command IDs to their destination component. When a command is received by the Command Router, its ID is looked up in the table, which returns the connector index on which the destination component is attached. The Command Router will then forward the command out of the appropriate index. Event errors are thrown if a Command ID is not found in the table.

In addition to routing commands the Command Router also receives command responses from the downstream components who execute commands. These responses are used to report data products on the command success and failure counts. Responses can also be forwarded to the sourcing command components, allowing command sources to check command responses or wait until a command response is received before sending a subsequent command.

The Command Router also has some of its own internal NOOP commands, to which it responds with Events. These commands can be useful for testing system aliveness.

It is advised to connect one index of the Command_T_Send connectors to the Command Router's own Command_T_Recv_Async connector in order to utilize the NOOP commands to enable self testing of command routing. Likewise, it is advisable to connect the Command Router's Command_Response_T_Send to the Command_Response_T_Recv_Async connector and one index of the Command_Response_T_To_Forward_Send connector to the Command_Response_T_Recv_Async connector in order to fully utilize the component's ability to self test command response forwarding (see the Noop_Response command).

2 Requirements

The requirements for the Command Router component are specified below.

1. The component shall route incoming commands to the correct destination component for execution.
2. The component shall drop and report incoming commands that have an unrecognized identifier.
3. The component shall implement a NOOP command.
4. The component shall receive command responses and forward them to the source components who sent the commands.
5. The component shall produce a data product relating the the number of received commands and the identifier of the last received command.
6. The component shall produce a data product relating the the number of successful commands and the identifier of the last successful command.
7. The component shall produce a data product relating the the number of failed commands and the identifier of the last failed command.

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** - *active*
- **Number of Connectors** - 10
- **Number of Invokee Connectors** - 4
- **Number of Invoker Connectors** - 6
- **Number of Generic Connectors** - *None*
- **Number of Generic Types** - *None*
- **Number of Unconstrained Arrayed Connectors** - 2
- **Number of Commands** - 4
- **Number of Parameters** - *None*
- **Number of Events** - 18
- **Number of Faults** - *None*
- **Number of Data Products** - 7
- **Number of Data Dependencies** - *None*
- **Number of Packets** - *None*

3.2 Diagram

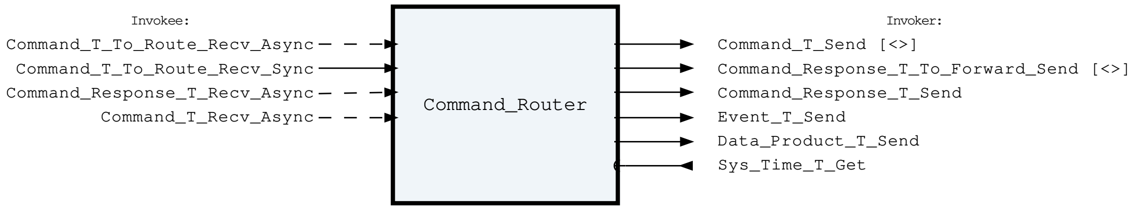

Figure 1: Command Router component diagram.

The Command Router is best understood when viewed in the context of an assembly. The following diagram shows a typical setup for the Command Router.

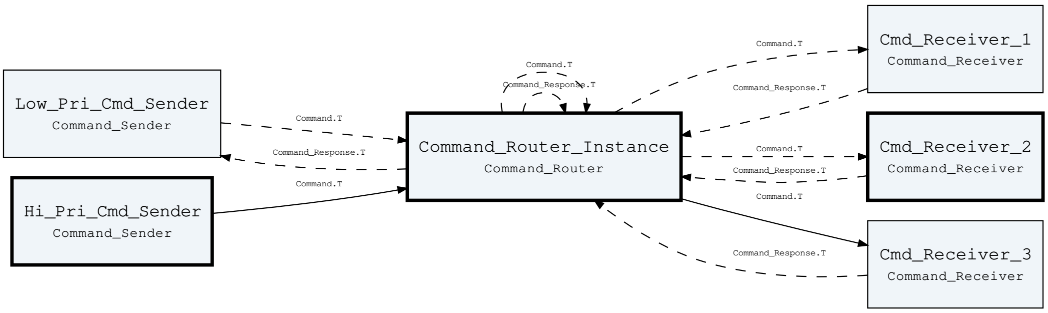

Figure 2: Example usage of the Command Router.

In the above context diagram the Command Router receives commands from two source components. One of these components sends commands at low priority on the Command Router's asynchronous connector. The other sends commands at high priority using the synchronous connector, bypassing the Command Router's internal queue. In most cases, using the asynchronous queue should be the preferred way to execute commands. An application for using the synchronous connector may include fault protection commands that need to execute before any currently queued commands.

After commands are sent to the Command Router they are forwarded to one of three downstream components to execute the commands. The Command Router's binary tree lookup algorithm is used to determine which connector, and therefore which downstream component, a command should be directed to. This binary tree is populated at startup using the command response connectors, which should always be connected from every downstream component back to the Command Router.

The command response connectors are also used to return the success/fail status of the command back to the router after execution. These command responses are tabulated by the Command Router and reported as data products. The Command Router can also be configured to forward the command response back to the component who sent the commands, although this is not required. In the diagram above the low priority sender component expects a response back, but the high priority sender component does not. Command responses can be used by sending components to make decisions based off of whether a command succeeded or not, or to simply meter out commands, not sending another command until the response from the previous command has been received. Both of these patterns are commonly utilized when implementing a command sequencing component.

Also seen in the diagram, the Command Router has loopback connections to itself for commands and command response. This allows the Command Router to self test its capabilities by routing and executing NOOP commands and returning and forwarding the command responses from those commands.

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: Command Router Invokee Connectors

Name	**Kind**	**Type**	**Return_Type**	**Count**
Command_T_To_Route_Recv_Async	recv_async	Command.T	-	1
Command_T_To_Route_Recv_Sync	recv_sync	Command.T	-	1
Command_Response_T_Recv_Async	recv_async	Command_Response.T	-	1
Command_T_Recv_Async	recv_async	Command.T	-	1

Connector Descriptions:

- **Command_T_To_Route_Recv_Async** - On this connector the Command Router recieves incoming commands that need to be routed to the correct destination component.
- **Command_T_To_Route_Recv_Sync** - On this connector the Command Router recieves incoming commands that need to be routed to the correct destination component. This connector is synchronous, and thus bypasses the internal queue that the Command_T_To_Route_Recv_Async uses. It should be used by components that need high priority command execution. It should only be called after command registration has occurred,

or a race condition is present.

- **Command_Response_T_Recv_Async** - Command registrations are received on this connector during initialization. Command responses from connected components are recieved on this connector during execution.
- **Command_T_Recv_Async** - This is the command recieve connector for the Command Router. The NOOP commands sent on this connector will be executed by the command router. This connector will usually be connected in loopback from the Command_T_Send connector in order to provide aliveness test capabilities, or disconnected completely.

3.3.2 Internal Queue

This component contains an internal first-in-first-out (FIFO) queue to handle asynchronous messages. This queue is sized at initialization as a configurable number of bytes. Determining the size of the component queue can be difficult. The following table lists the connectors that will put asynchronous messages onto the queue, and the maximum sizes of each of those messages on the queue. Note that each message put onto the queue also incurs an overhead on the queue of 5 additional bytes, which is included in the max message size below:

Table 2: Command Router Asynchronous Connectors

Name	Type	Max Size (bytes)
Command_T_To_Route_Recv_Async	Command.T	106
Command_Response_T_Recv_Async	Command_Response.T	12
Command_T_Recv_Async	Command.T	106

If you are unsure how to size the queue of this component, it is recommended that you make the queue size a multiple of the largest size found above.

3.3.3 Invoker Connectors

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

Table 3: Command Router Invoker Connectors

Name	Kind	Type	Return_Type	Count
Command_T_Send	send	Command.T	-	<>
Command_Response_T_To_Forward_Send	send	Command_Response.T	-	<>
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:

- **Command_T_Send** - This connector has an unconstrained size that is determined by the assembly in which the Command Router is instantiated. Each index of the connector should connect to different destination component that receives commands. The Command Router will route commands destined for each component on the appropriate index of this connector.
- **Command_Response_T_To_Forward_Send** - Command responses received from command

executing components are forwarded back to their command sources using this arrayed connector. One index of this connector can be connected in loopback to the Command_Response_T_Recv_Async connector in order to command forwarding self test capabilities (see the Noop_Response command).

- **Command_Response_T_Send** - This connector is used to register the Command Router's NOOP commands at initialization, and respond to NOOP commands during ececution. It is usually connected in loopback to the Command_Response_T_Recv_Async connector.
- **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 Interrupts

This component contains no interrupts.

3.5 Initialization

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

3.5.1 Component Instantiation

This component contains no instantiation parameters in its discriminant.

3.5.2 Component Base Initialization

This component achieves base class initialization using the `init_Base` subprogram. This subprogram requires the following parameters:

Table 4: Command Router Base Initialization Parameters

Name	Type
Queue_Size	Natural
Command_T_Send_Count	Connector_Count_Type
Command_Response_T_To_Forward_Send_Count	Connector_Count_Type

Parameter Descriptions:

- **Queue_Size** - The number of bytes that can be stored in the component's internal queue.
- **Command_T_Send_Count** - The size of the Command_T_Send invoker connector array.
- **Command_Response_T_To_Forward_Send_Count** - The size of the Command_Response_T_To_Forward_Send invoker connector array.

3.5.3 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 5: Command Router Set Id Bases Parameters

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

Parameter Descriptions:

- **`Event_Id_Base`** - The value at which the component's event identifiers begin.
- **`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.

3.5.4 Component Map Data Dependencies

This component contains no data dependencies.

3.5.5 Component Implementation Initialization

The calling of this implementation class initialization procedure is mandatory. This component requires the maximum number of unique commands that it expects to need to route. This number will be used to size the internal router table on the heap. Any attempted command registrations beyond this limit will be reported as an error event and will not be registered. The `init` subprogram requires the following parameters:

Table 6: Command Router Implementation Initialization Parameters

Name	**Type**	**Default Value**
`Max_Number_Of_Commands`	`Natural`	*None provided*

Parameter Descriptions:

- **`Max_Number_Of_Commands`** - The maximum number of unique commands that can be registered with the command router component.

3.6 Commands

These are the commands for the Command Router component. They are NOOP commands that produce events to facilitate unit testing and aliveness checks during flight.

Table 7: Command Router Commands

Local ID	**Command Name**	**Argument Type**
0	`Noop`	-
1	`Noop_Arg`	`Command_Router_Arg.T`
2	`Noop_Response`	-
3	`Reset_Data_Products`	-

Command Descriptions:

- **`Noop`** - Simple NOOP command which produces an event saying that it was triggered. This can be used to self test the command routing system and verify system aliveness.
- **`Noop_Arg`** - Simple NOOP command which produces an event saying that it was triggered with a certain Arg. This can be used to self test the command argument validation system. Sending a command with an Arg value of 868 will cause the component to Fail the command. Any other value will produce a successfully executed command.
- **`Noop_Response`** - A NOOP command which self tests the command response forwarding mechanism. The command handler itself acts as a command sender component, and sends out a NOOP command with a registered Source Id. The Command Router should then send out an event saying that the command response was forwarded and received again by the Command Router.

- **Reset_Data_Products** - This command resets the values of all the component's data product to the values at initialization.

3.7 Parameters

The Command Router component has no parameters.

3.8 Events

Below is a list of the events for the Command Router component.

Table 8: Command Router Events

Local ID	Event Name	Parameter Type
0	Command_Received	Command_Header.T
1	Command_Execution_Successful	Command_Response.T
2	Command_Execution_Failure	Command_Response.T
3	Command_Id_Not_Registered	Command_Header.T
4	Registration_Id_Conflict	Command_Id.T
5	Router_Table_Full	Command_Id.T
6	Outgoing_Command_Dropped	Command_Header.T
7	Incoming_Command_Dropped	Command_Header.T
8	Noop_Command_Dropped	Command_Header.T
9	Command_Response_Dropped	Command_Response.T
10	Noop_Received	-
11	Noop_Arg_Received	Command_Router_Arg.T
12	Noop_Response_Received	-
13	Noop_Response_Forwarding_Success	Command_Response.T
14	Forwarded_Command_Response_Dropped	Command_Response.T
15	Invalid_Command_Source_Id	Command_Response.T
16	Invalid_Command_Received	Invalid_Command_Info.T
17	Data_Products_Reset	-

Event Descriptions:

- **Command_Received** - A command was received by the command router to be routed.
- **Command_Execution_Successful** - A command was routed, executed, and returned a response saying it was executed successfully
- **Command_Execution_Failure** - A command execution failed.
- **Command_Id_Not_Registered** - A command was sent to the router, but it was not found in the router table.
- **Registration_Id_Conflict** - The command Id has already been registered.
- **Router_Table_Full** - Cannot add command Id to router table because it is full.
- **Outgoing_Command_Dropped** - A command was dropped because the recipient's queue was full.
- **Incoming_Command_Dropped** - A command was dropped because the command router's queue was full.
- **Noop_Command_Dropped** - A noop command was dropped because the command router's queue was full.
- **Command_Response_Dropped** - A command response was dropped because the command router's queue was full.
- **Noop_Received** - A Noop command was received.

- **Noop_Arg_Received** - A Noop command was received with an argument.
- **Noop_Response_Received** - A noop response self test command was received.
- **Noop_Response_Forwarding_Success** - If this event is sent then the noop response self test command succeeded.
- **Forwarded_Command_Response_Dropped** - A forwarded command response was dropped because the receiving component's queue overflowed.
- **Invalid_Command_Source_Id** - A command response contained an invalid source id. This is a software bug and should be corrected.
- **Invalid_Command_Received** - A command was received with invalid parameters.
- **Data_Products_Reset** - The component's data products have been reset to initialization values.

3.9 Data Products

Data products for the Command Router component.

Table 9: Command Router Data Products

Local ID	Data Product Name	Type
0x0000 (0)	Command_Receive_Count	Packed_U16.T
0x0001 (1)	Command_Success_Count	Packed_U16.T
0x0002 (2)	Command_Failure_Count	Packed_U16.T
0x0003 (3)	Last_Received_Command	Command_Id.T
0x0004 (4)	Last_Successful_Command	Command_Id.T
0x0005 (5)	Last_Failed_Command	Command_Id_Status.T
0x0006 (6)	Noop_Arg_Last_Value	Command_Router_Arg.T

Data Product Descriptions:

- **Command_Receive_Count** - The number of commands received by the component.
- **Command_Success_Count** - The number of commands that successfully executed.
- **Command_Failure_Count** - The number of commands that failed to execute.
- **Last_Received_Command** - The ID of the last received command by the command router.
- **Last_Successful_Command** - The ID of the last successful command routed by the command router.
- **Last_Failed_Command** - The ID and status of the last failed command routed by the command router.
- **Noop_Arg_Last_Value** - The last value sent with the Noop_Arg command. This data product can be useful for testing purposes.

3.10 Data Dependencies

The Command Router component has no data dependencies.

3.11 Packets

The Command Router component has no packets.

3.12 Faults

The Command Router component has no faults.

4 Unit Tests

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

4.1 *Command_ Router_ Tests* Test Suite

This is a unit test suite for the Command Router component

Test Descriptions:

- **`Test_Nominal_Routing`** - This unit test excersizes command routing via the Command Router internal commands.
- **`Test_Nominal_Registration`** - This unit test excersizes command registration from the external testing component, and then makes sure the command routing works.
- **`Test_Routing_Errors`** - This unit test makes sure errors are thrown when unknown commands are sent.
- **`Test_Registration_Errors`** - This unit test makes sure errors are thrown when command registration goes awry.
- **`Test_Full_Queue_Errors`** - This unit test makes sure errors are thrown when the command router queue gets full.
- **`Test_Invalid_Argument_Length`** - This unit test makes sure errors are thrown when a command is received with an invalid argument length.
- **`Test_Invalid_Argument_Value`** - This unit test makes sure errors are thrown when a command is received with an invalid value.
- **`Test_Failed_Command`** - This unit test makes sure that a failed command reports the correct data products and events.
- **`Test_Synchronous_Command`** - This unit test makes sure that the synchronous command connector works as expected, bypassing queue.
- **`Test_Command_Response_Forwarding`** - This unit test makes sure that the command response forwarding system and registration works as expected.
- **`Test_Command_Response_Forwarding_Dropped`** - This unit test makes sure that the component reports an event if a command response forward is dropped by a downstream component.
- **`Test_Outgoing_Command_Dropped`** - This unit test makes sure that the component reports an event if a command is dropped by a downstream component.

5 Appendix

5.1 Preamble

This component contains no preamble code.

5.2 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..

Command.T:

Generic command packet for holding arbitrary commands

Table 10: Command Packed Record : 808 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	-	768	40	807	Header.Arg_Buffer_Length

Field Descriptions:

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

Command_Header.T:

Generic command header for holding arbitrary commands

Table 11: 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 96	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_Id.T:

A packed record which holds a command identifier.

Table 12: Command_Id Packed Record : 16 bits

Name	Type	Range	Size (Bits)	Start Bit	End Bit
Id	Command_Types.Command_Id	0 to 65535	16	0	15

Field Descriptions:

- **Id** - The command identifier

Command_Id_Status.T:

Record for holding a command identifier and command response status.

Table 13: Command_Id_Status Packed Record : 24 bits

Name	Type	Range	Size (Bits)	Start Bit	End Bit
Id	Command_Types. Command_Id	0 to 65535	16	0	15
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	16	23

Field Descriptions:

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

Command_Response.T:

Record for holding command response data.

Table 14: 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.

Command_Router_Arg.T:

A 32-bit unsigned integer with range 0 to 999.

Preamble (inline Ada definitions):

```ada
subtype Value_Type is Natural range 0 .. 999;
```

Table 15: Command_Router_Arg Packed Record : 32 bits

Name	Type	Range	Size (Bits)	Start Bit	End Bit
Value	Value_Type	0 to 999	32	0	31

Field Descriptions:

- **Value** - The 32-bit unsigned integer with range 0 to 999.

Data_Product.T:

Generic data product packet for holding arbitrary data types

Table 16: 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 17: 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 18: 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 19: 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 20: 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 unkown 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.

Packed_U16.T:

Single component record for holding packed unsigned 16-bit value.

Table 21: Packed_U16 Packed Record : 16 bits

Name	Type	Range	Size (Bits)	Start Bit	End Bit
Value	Interfaces. Unsigned_16	0 to 65535	16	0	15

Field Descriptions:

- **Value** - The 16-bit unsigned integer.

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 22: 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.

5.3 Enumerations

The following section outlines any enumerations used in the component.

Command_Enums.Command_Response_Status.E:

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

Table 23: 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.