Command Protector

Component Design Document

1 Description

This component is initialized with a list of protected commands. A protected command is a command that could be potentially hazardous to the system or its environment. These commands should only be sent with the utmost care, and should never be executed by accident. This component implements a protection mechanism to prevent inadvertent execution of these commands.
The component receives commands on the Command_T_To_Forward_Recv_Sync connector, and checks their IDs against the protected command list. If a command is found in the list, then it is only forwarded if the component has been first 'armed', otherwise the command is dropped and an error packet is produced with the rejected command data. Commands that are not found in the protected command list are always forwarded. To 'arm' the component, a special 'arm' command must be sent to the component to transition it to the 'armed' state. At this point, a protected command may be successfully forwarded. Note that after the receipt of any command, the component will transition back to the 'unarmed' state, rejecting any subsequently received protected commands until another 'arm' command is received. The component will also transition back to the 'unarmed' state after a timeout expires, which is set as an argument in the 'arm' command itself.
The protected command list is stored internally as a binary tree data structure that can determine if a command is protected or not in O(log(n)) time, where n is the number of protected commands. Since most systems only manage a handful of protected commands, the performance of this component should be acceptable for most missions.

2 Requirements

The requirements for the Command Protector component are specified below.

1. The component shall enter the armed state upon command.
2. The component shall reject protected commands if not in the armed state.
3. The component shall report the armed state in telemetry.
4. The component shall exit the armed state upon receipt of any command, unless it is another arm command.
5. The component shall exit the armed state after a timeout.
6. The component shall set the armed state timeout through ground command.
7. The component shall accept armed state timeouts in the range of 0 to 255 seconds (in 1 second intervals). A value of zero implies infinite timeout.
8. The component shall report the armed state timeout in telemetry.

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** - 9
- **Number of Invokee Connectors** - 3
- **Number of Invoker Connectors** - 6
- **Number of Generic Connectors** - *None*
- **Number of Generic Types** - *None*
- **Number of Unconstrained Arrayed Connectors** - *None*
- **Number of Commands** - 1
- **Number of Parameters** - *None*
- **Number of Events** - 6
- **Number of Faults** - *None*
- **Number of Data Products** - 4
- **Number of Data Dependencies** - *None*
- **Number of Packets** - 1

3.2 Diagram

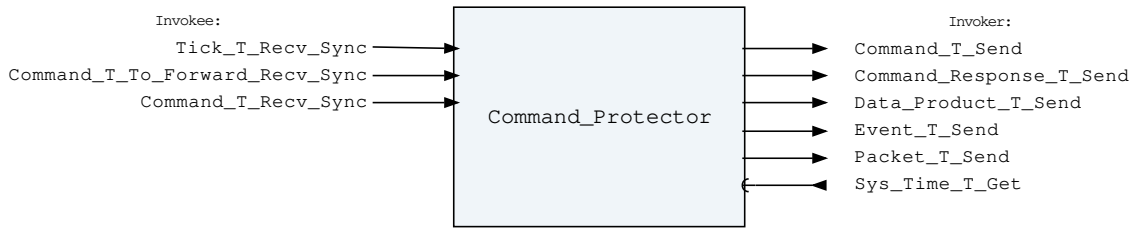

Figure 1: Command Protector component diagram.

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: Command Protector Invokee Connectors

Name	Kind	Type	Return_Type	Count
Tick_T_Recv_Sync	recv_sync	Tick.T	-	1
Command_T_To_ Forward_Recv_ Sync	recv_sync	Command.T	-	1
Command_T_Recv_ Sync	recv_sync	Command.T	-	1

Connector Descriptions:

- **Tick_T_Recv_Sync** - This tick is used to keep track of the armed state timeout and send the data product relating the current timeout value.
- **Command_T_To_Forward_Recv_Sync** - Commands received on this connector will be checked against the protected command list and rejected if the system is 'unarmed'. Commands not found in the protected command list will be forwarded.
- **Command_T_Recv_Sync** - The command receive connector for this component's specific commands.

3.3.2 Invoker Connectors

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

Table 2: Command Protector Invoker Connectors

Name	Kind	Type	Return_Type	Count
Command_T_Send	send	Command.T	-	1
Command_Response_T_Send	send	Command_Response.T	-	1
Data_Product_T_Send	send	Data_Product.T	-	1
Event_T_Send	send	Event.T	-	1
Packet_T_Send	send	Packet.T	-	1
Sys_Time_T_Get	get	-	Sys_Time.T	1

Connector Descriptions:

- **Command_T_Send** - Commands that are received on the Command_T_To_Forward_Recv_Sync connector are forwarded out this connector.
- **Command_Response_T_Send** - This connector is used to register and respond to the component's specific commands.
- **Data_Product_T_Send** - Data products are sent out of this connector.
- **Event_T_Send** - Events are sent out of this connector.
- **Packet_T_Send** - The packet send connector, used for sending error packets filled with protected commands that are rejected.
- **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 contains no base class initialization, meaning there is no `init_Base` subprogram for this component.

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 3: Command Protector Set Id Bases Parameters

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

Parameter Descriptions:

- **`Event_Id_Base`** - The value at which the component's event identifiers begin.
- **`Data_Product_Id_Base`** - The value at which the component's data product identifiers begin.
- **`Command_Id_Base`** - The value at which the component's command identifiers begin.
- **`Packet_Id_Base`** - The value at which the component's unresolved packet 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 a list of protected command IDs at initialization. The `init` subprogram requires the following parameters:

Table 4: Command Protector Implementation Initialization Parameters

Name	Type	Default Value
`Protected_Command_Id_List`	`Command_Id_List`	*None provided*

Parameter Descriptions:

- **`Protected_Command_Id_List`** - The list of command IDs to consider as protected commands.

3.6 Commands

These are the commands for the Command Protector component.

Table 5: Command Protector Commands

Local ID	Command Name	Argument Type
0	`Arm`	`Packed_Arm_Timeout.T`

Command Descriptions:

- **`Arm`** - Transition the component to the 'arm' state. A timeout is provided, which when expires, will transition the component back to the 'unarmed' state, unless a command is received first.

A timeout value of zero implies infinite timeout, meaning the component will only transition to the 'unarmed' when a command is received.

3.7 Parameters

The Command Protector component has no parameters.

3.8 Events

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

Table 6: Command Protector Events

Local ID	Event Name	Parameter Type
0	Rejected_Protected_Command	Command_Header.T
1	Accepted_Protected_Command	Command_Header.T
2	Armed	Packed_Arm_Timeout.T
3	Unarmed	-
4	Unarmed_Timeout	-
5	Invalid_Command_Received	Invalid_Command_Info.T

Event Descriptions:

- **Rejected_Protected_Command** - A protected command was rejected (dropped) because the component was unarmed.
- **Accepted_Protected_Command** - A protected command was accepted because the component was armed.
- **Armed** - The component received the arm command an is now armed.
- **Unarmed** - The component received a command and is now unarmed.
- **Unarmed_Timeout** - The component armed state timed out and is now unarmed.
- **Invalid_Command_Received** - A command was received with invalid parameters.

3.9 Data Products

Data products for the Command Protector.

Table 7: Command Protector Data Products

Local ID	Data Product Name	Type
0x0000 (0)	Armed_State	Packed_Arm_State.T
0x0001 (1)	Armed_State_Timeout	Packed_Arm_Timeout.T
0x0002 (2)	Protected_Command_Reject_Count	Packed_U16.T
0x0003 (3)	Protected_Command_Forward_Count	Packed_U16.T

Data Product Descriptions:

- **Armed_State** - The current armed/unarmed state of the component.
- **Armed_State_Timeout** - The time remaining (in ticks) until the armed state expires.
- **Protected_Command_Reject_Count** - The number of protected commands rejected because the component was unarmed.
- **Protected_Command_Forward_Count** - The number of protected commands forwarded be-

cause the component was armed.

3.10 Data Dependencies

The Command Protector component has no data dependencies.

3.11 Packets

Packets for the Command Protector component.

Table 8: Command Protector Packets

Local ID	Packet Name	Type
0x0000 (0)	Error_Packet	Command.T

Packet Descriptions:

- **Error_Packet** - This packet contains a protected command that was dropped due the component not being 'armed'.

4 Unit Tests

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

4.1 *Command_Protector_Tests* Test Suite

This is a unit test suite for the Command Protector.

Test Descriptions:

- **Test_Initialization** - This unit test tests all permutations of initializing the component and makes sure improper initialization results in a runtime assertion.
- **Test_Unprotected_Command_Accept** - This unit test tests that an unprotected command is passed along no matter the state of the component.
- **Test_Protected_Command_Accept** - This unit test tests that a protected command is accepted if the component is armed.
- **Test_Protected_Command_Reject** - This unit test tests that a protected command is rejected if the component is unarmed.
- **Test_Protected_Command_Reject_Timeout** - This unit test tests that a protected command is rejected if the component is unarmed due to timeout.
- **Test_Invalid_Command** - This unit test makes sure that an invalid command is handled gracefully.

5 Appendix

5.1 Preamble

This component contains the following preamble code. This is inline Ada code included in the component model that is usually used to define types or instantiate generic packages used by the component. Preamble code is inserted as the top line of the component base package specification.

```
type Command_Id_List is array (Natural range <>) of Command_Types.Command_Id;
```

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 9: 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 10: 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_Response.T:

Record for holding command response data.

Table 11: 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 12: 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 13: 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 14: 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 15: 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 16: 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_Arm_State.T:

Holds the armed state.

Table 17: Packed_Arm_State Packed Record : 8 bits

Name	Type	Range	Size (Bits)	Start Bit	End Bit
State	Command_Protector_Enums.Armed_State.E	0 => Unarmed 1 => Armed	8	0	7

Field Descriptions:

- **State** - The armed/unarmed status.

Packed_Arm_Timeout.T:

Holds the armed state timout.

Preamble (inline Ada definitions):

```
1 type Arm_Timeout_Type is new Natural range 0 .. 255;
```

Table 18: Packed_Arm_Timeout Packed Record : 8 bits

Name	Type	Range	Size (Bits)	Start Bit	End Bit
Timeout	Arm_Timeout_Type	0 to 255	8	0	7

Field Descriptions:

- **Timeout** - The timeout value (in ticks).

Packed_U16.T:

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

Table 19: 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.

Packet.T:

Generic packet for holding arbitrary data

Table 20: Packet Packed Record : 10080 bits *(maximum)*

Name	Type	Range	Size (Bits)	Start Bit	End Bit	Variable Length
Header	Packet_ Header.T	-	112	0	111	-
Buffer	Packet_ Types.Packet_ Buffer_Type	-	9968	112	10079	Header. Buffer_Length

Field Descriptions:

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

Packet_Header.T:

Generic packet header for holding arbitrary data

Table 21: Packet_Header Packed Record : 112 bits

Name	Type	Range	Size (Bits)	Start Bit	End Bit
Time	Sys_Time.T	-	64	0	63
Id	Packet_Types. Packet_Id	0 to 65535	16	64	79
Sequence_Count	Packet_Types. Sequence_Count_Mod_ Type	0 to 16383	16	80	95
Buffer_Length	Packet_Types. Packet_Buffer_ Length_Type	0 to 1246	16	96	111

Field Descriptions:

- **Time** - The timestamp for the packet item.
- **Id** - The packet identifier
- **Sequence_Count** - Packet Sequence Count

- **Buffer_Length** - The number of bytes used in the packet buffer

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.

Tick.T:

The tick datatype used for periodic scheduling. Included in this type is the Time associated with a tick and a count.

Table 23: Tick Packed Record : 96 bits

Name	Type	Range	Size (Bits)	Start Bit	End Bit
Time	Sys_Time.T	-	64	0	63
Count	Interfaces. Unsigned_32	0 to 4294967295	32	64	95

Field Descriptions:

- **Time** - The timestamp associated with the tick.
- **Count** - The cycle number of the tick.

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 24: 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.

Command_Protector_Enums.Armed_State.E:

This type enumerates the armed state for the component.

Table 25: Armed_State Literals:

Name	Value	Description
Unarmed	0	The component is unarmed. Any protected commands received will be rejected.
Armed	1	The component is armed. If the next command received is a protected command, it will be forwarded.