Fault Correction

Component Design Document

1 Description

The Fault Correction component receives faults asynchronously. When it processes a fault, it determines the correct command response to send and sends it.

2 Requirements

The requirements for the Fault Correction component are specified below.

1. The component shall send out a predefined command in response to receiving a fault of a particular ID.
2. The component shall be configured with a table of fault IDs and their associated command responses at startup.
3. The component shall accept commands to enable and disable fault responses.
4. The component shall report the status of each fault response in its table 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** - *active*
- **Number of Connectors** - 7
- **Number of Invokee Connectors** - 2
- **Number of Invoker Connectors** - 5
- **Number of Generic Connectors** - *None*
- **Number of Generic Types** - *None*
- **Number of Unconstrained Arrayed Connectors** - *None*
- **Number of Commands** - 5
- **Number of Parameters** - *None*
- **Number of Events** - 11
- **Number of Faults** - *None*
- **Number of Data Products** - 4
- **Number of Data Dependencies** - *None*

- **Number of Packets** - *None*

3.2 Diagram

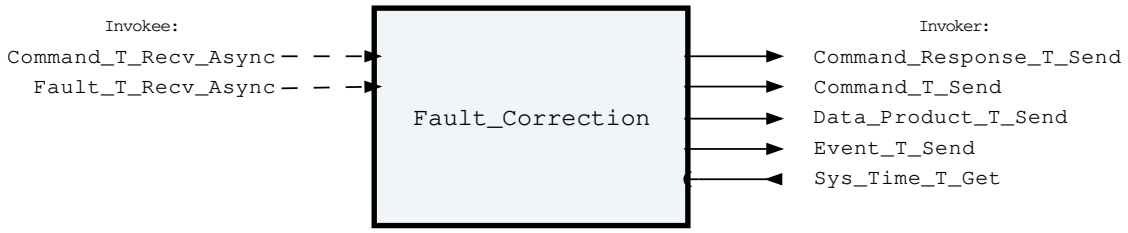

Figure 1: Fault Correction component diagram.

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: Fault Correction Invokee Connectors

Name	Kind	Type	Return_Type	Count
Command_T_Recv_Async	recv_async	Command.T	-	1
Fault_T_Recv_Async	recv_async	Fault.T	-	1

Connector Descriptions:

- **Command_T_Recv_Async** - This is the command receive connector.
- **Fault_T_Recv_Async** - Faults are received asynchronously on this connector.

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: Fault Correction Asynchronous Connectors

Name	Type	Max Size (bytes)
Command_T_Recv_Async	Command.T	265
Fault_T_Recv_Async	Fault.T	24

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: Fault Correction Invoker Connectors

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

Connector Descriptions:

- **Command_Response_T_Send** - This connector is used to send command responses.
- **Command_T_Send** - The command send connector, for sending correction commands for faults.
- **Data_Product_T_Send** - Data products are sent out of this connector.
- **Event_T_Send** - Events 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: Fault Correction Base Initialization Parameters

Name	Type
Queue_Size	Natural

Parameter Descriptions:

- **Queue_Size** - The number of bytes that can be stored in the component's internal queue.

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: Fault Correction 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

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.

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. The component is initialized by providing an access to a list of fault response configuration records. The `init` subprogram requires the following parameters:

Table 6: Fault Correction Implementation Initialization Parameters

Name	Type	Default Value
Fault_Response_Configurations	Fault_Correction_Types.Fault_Response_Config_List	*None provided*

Parameter Descriptions:

- **Fault_Response_Configurations** - An access to a list of fault response configurations.

3.6 Commands

These are the commands for the Fault Correction component.

Table 7: Fault Correction Commands

Local ID	Command Name	Argument Type
0	Enable_Fault_Response	Packed_Fault_Id.T
1	Disable_Fault_Response	Packed_Fault_Id.T
2	Clear_Fault_Response	Packed_Fault_Id.T
3	Clear_All_Fault_Responses	-
4	Reset_Data_Products	-

Command Descriptions:

- **Enable_Fault_Response** - Enable a fault response for the provided ID. This will only succeed if another response with the same Fault ID is not already enabled.
- **Disable_Fault_Response** - Disable a fault response for the provided ID.

- **Clear_Fault_Response** - Resets a fault response to the Enabled state of the provided ID. If the fault is latched, it unlatches the fault.
- **Clear_All_Fault_Responses** - Resets all fault responses to the Enabled state. Unlatches all latched fault responses.
- **Reset_Data_Products** - This command resets the values of all the component's data product to the values at initialization, except for the Fault_Response_Statuses data product which can be reset by the Clear_All_Fault_Responses command.

3.7 Parameters

The Fault Correction component has no parameters.

3.8 Events

Events for the Fault Correction component.

Table 8: Fault Correction Events

Local ID	Event Name	Parameter Type
0	Fault_Received	Fault_Static.T
1	Fault_Response_Sent	Command_Header.T
2	Fault_Response_Cleared	Packed_Fault_Id.T
3	Fault_Response_Disabled	Packed_Fault_Id.T
4	Fault_Response_Enabled	Packed_Fault_Id.T
5	All_Fault_Responses_Cleared	-
6	Unrecognized_Fault_Id	Packed_Fault_Id.T
7	Invalid_Command_Received	Invalid_Command_Info.T
8	Command_Dropped	Command_Header.T
9	Fault_Dropped	Fault_Header.T
10	Data_Products_Reset	-

Event Descriptions:

- **Fault_Received** - A fault was received.
- **Fault_Response_Sent** - A fault response was sent with the included command header.
- **Fault_Response_Cleared** - A fault response was cleared.
- **Fault_Response_Disabled** - A fault response has been disabled
- **Fault_Response_Enabled** - A fault response has been enabled.
- **All_Fault_Responses_Cleared** - Any latched faults have been unlatched by command.
- **Unrecognized_Fault_Id** - A fault response entry with the included fault ID was not found in the table.
- **Invalid_Command_Received** - A command was received with invalid parameters.
- **Command_Dropped** - A command was dropped due to a full queue.
- **Fault_Dropped** - A fault was dropped due to a full queue.
- **Data_Products_Reset** - The component's data products have been reset to initialization values.

3.9 Data Products

Data products for the Fault Correction component.

Table 9: Fault Correction Data Products

Local ID	Data Product Name	Type
0x0000 (0)	Fault_Counter	Packed_U16.T
0x0001 (1)	Last_Fault_Id_Received	Packed_Fault_Id.T
0x0002 (2)	Time_Of_Last_Fault_Received	Sys_Time.T
0x0003 (3)	Fault_Response_Statuses	Packed_U32.T

Data Product Descriptions:

- **Fault_Counter** - The number of faults received by the component.
- **Last_Fault_Id_Received** - The ID of the last fault received.
- **Time_Of_Last_Fault_Received** - The system time of the last fault received.
- **Fault_Response_Statuses** - 2-bits of status for each fault response that this component is managing. Note that Packed_U32.T is just a placeholder type for this data product. The actual type of this data product will be autocoded and at assembly model ingest time.

3.10 Data Dependencies

The Fault Correction component has no data dependencies.

3.11 Packets

The Fault Correction component has no packets.

3.12 Faults

The Fault Correction component has no faults.

4 Unit Tests

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

4.1 *Fault_ Correction_ Tests* Test Suite

This is a unit test suite for the Fault Correction component.

Test Descriptions:

- **Test_Initialization** - This unit test tests permutations of initializing the component and makes sure improper initialization results in a runtime assertion.
- **Test_Fault_Handling** - This unit test tests that fault handling is done appropriately when a fault is received.
- **Test_Enable_Disable_Fault_Response** - This unit test makes sure that the fault response enable and disable commands function properly.
- **Test_Clear_Fault_Response** - This unit test makes sure that the fault response clear commands function properly.
- **Test_Reset_Data_Products** - This unit test tests the reset data products command.

- **Test_Unrecognized_Fault_Id** - This unit test makes sure that commanding a change to a response with an unknown fault ID fails.
- **Test_Full_Queue** - This unit test tests that appropriate actions are taken when items are dropped off a full queue.
- **Test_Invalid_Command** - This unit test makes sure that an invalid command is handled gracefully.

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 : 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 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 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 12: 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 13: 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 14: 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 15: 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 16: 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

Fault.T:

Generic fault packet for holding arbitrary faults.

Table 17: Fault Packed Record : 152 bits *(maximum)*

Name	Type	Range	Size (Bits)	Start Bit	End Bit	Variable Length
Header	Fault_Header.T	-	88	0	87	-
Param_Buffer	Fault_Types.Parameter_Buffer_Type	-	64	88	151	Header.Param_Buffer_Length

Field Descriptions:

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

Fault_Header.T:

Generic fault header.

Table 18: Fault_Header Packed Record : 88 bits

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

Field Descriptions:

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

Fault_Static.T:

Generic fault packet for holding arbitrary faults. This is the same as the Fault.T type, except that it is not variable sized, it is always maximum sized. This can be useful for sending events with faults in them.

Table 19: Fault_Static Packed Record : 152 bits

Name	Type	Range	Size (Bits)	Start Bit	End Bit
Header	Fault_Header.T	-	88	0	87
Param_Buffer	Fault_Types.Parameter_Buffer_Type	-	64	88	151

Field Descriptions:

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

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 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.

Packed_Fault_Id.T:

A packed record which holds a fault identifier.

Table 21: Packed_Fault_Id Packed Record : 16 bits

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

Field Descriptions:

- **Id** - The fault identifier

Packed_U16.T:

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

Table 22: 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.

Packed_U32.T:

Single component record for holding packed unsigned 32-bit value.

Table 23: Packed_U32 Packed Record : 32 bits

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

Field Descriptions:
- **Value** - The 32-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 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.

5.3 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.