Parameter Store

Component Design Document

1 Description

The Parameter Store component is responsible for storing and managing access to a memory region holding a parameter table. The managed memory region is usually located in nonvolatile storage and can serve as the backup or the default parameter values to use at startup for the system.

2 Requirements

The requirements for the Parameter Store component are specified below.

1. The component shall manage a region of memory for parameter storage.
2. The component shall update the memory region through a parameter table upload.
3. The component shall return the memory region through a parameter table request.
4. The component shall produce a packet reflecting the current values stored in the memory region.
5. The component shall produce a parameter table packet upon command.
6. The component shall produce a parameter table packet whenever a new table has been uploaded.
7. The component shall reject parameter table requests to a memory region with an invalid length.
8. The component shall reject parameter table updates from a memory region with an invalid length.
9. The component shall reject parameter table updates from a memory region with an invalid CRC.

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** - 1

- **Number of Parameters** - *None*
- **Number of Events** - 9
- **Number of Faults** - *None*
- **Number of Data Products** - *None*
- **Number of Data Dependencies** - *None*
- **Number of Packets** - 1

3.2 Diagram

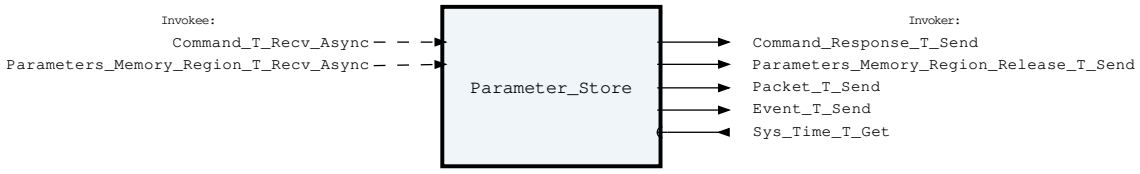

Figure 1: Parameter Store component diagram.

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: Parameter Store Invokee Connectors

Name	Kind	Type	Return_Type	Count
Command_T_Recv_Async	recv_async	Command.T	-	1
Parameters_Memory_Region_T_Recv_Async	recv_async	Parameters_Memory_Region.T	-	1

Connector Descriptions:

- **Command_T_Recv_Async** - This is the command receive connector.
- **Parameters_Memory_Region_T_Recv_Async** - When a memory region is received on this connector it is assumed that it contains a memory region that is the same size as the managed region. Based on the operation a new parameter table can be loaded into the store, or the current parameter data can be fetched from the store.

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: Parameter Store Asynchronous Connectors

Name	Type	Max Size (bytes)
Command_T_Recv_Async	Command.T	265
Parameters_Memory_Region_T_Recv_Async	Parameters_Memory_Region.T	18

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: Parameter Store Invoker Connectors

Name	Kind	Type	Return_Type	Count
Command_Response_T_Send	send	Command_Response.T	-	1
Parameters_Memory_Region_Release_T_Send	send	Parameters_Memory_Region_Release.T	-	1
Packet_T_Send	send	Packet.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.
- **Parameters_Memory_Region_Release_T_Send** - After a memory region is received on the Memory_Region_T_Recv_Async connector and then processed, it is released via a call to this connector. A status is also returned, so the downstream component can determine if the parameter table update was successful or not.
- **Packet_T_Send** - The parameter packet connector. A copy of the managed parameter table is dumped via 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 Initialization

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

3.4.1 Component Instantiation

This component contains no instantiation parameters in its discriminant.

3.4.2 Component Base Initialization

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

Table 4: Parameter Store 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.

3.4.3 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: Parameter Store Set Id Bases Parameters

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

Parameter Descriptions:

- **`Packet_Id_Base`** - The value at which the component's unresolved packet identifiers begin.
- **`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.

3.4.4 Component Map Data Dependencies

This component contains no data dependencies.

3.4.5 Component Implementation Initialization

The calling of this implementation class initialization procedure is mandatory. The component is initialized by providing the memory region it is to manage which holds the parameter table. The `init` subprogram requires the following parameters:

Table 6: Parameter Store Implementation Initialization Parameters

Name	Type	Default Value
`Bytes`	`Basic_Types.Byte_Array_Access`	*None provided*
`Dump_Parameters_On_Change`	`Boolean`	`False`

Parameter Descriptions:

- **`Bytes`** - A pointer to an allocation of bytes to be used for storing the parameter table. The size of this byte array MUST be the exact size of the parameter table to be stored, or updating or fetch the table will be rejected with a length error.
- **`Dump_Parameters_On_Change`** - If set to True, the component will dump the current parameter values any time a memory region is received to change the parameter table. If set to False, parameters will only be dumped when requested by command.

3.5 Commands

These are the commands for the Parameter Store component.

Table 7: Parameter Store Commands

Local ID	Command Name	Argument Type
0	Dump_Parameter_Store	-

Command Descriptions:

- **Dump_Parameter_Store** - Produce a packet with the currently stored parameter values.

3.6 Events

Events for the Parameter Store component.

Table 8: Parameter Store Events

Local ID	Event Name	Parameter Type
0	Memory_Region_Length_Mismatch	Invalid_Parameters_Memory_Region_Length.T
1	Memory_Region_Crc_Invalid	Invalid_Parameters_Memory_Region_Crc.T
2	Dumped_Parameters	-
3	Parameter_Table_Updated	Memory_Region.T
4	Parameter_Table_Fetched	Memory_Region.T
5	Invalid_Command_Received	Invalid_Command_Info.T
6	Command_Dropped	Command_Header.T
7	Memory_Region_Dropped	Parameters_Memory_Region.T
8	Table_Validation_Not_Supported	Memory_Region.T

Event Descriptions:

- **Memory_Region_Length_Mismatch** - A memory region was received with an invalid length. The length of the region must be the same size as the parameter table.
- **Memory_Region_Crc_Invalid** - A memory region parameter table was received with an invalid CRC. The computed CRC does not match the CRC found in the header.
- **Dumped_Parameters** - Produced a packet with the contents of the parameter store.
- **Parameter_Table_Updated** - Parameter table updated from a received memory region.
- **Parameter_Table_Fetched** - Starting fetching of the parameters into received memory region.
- **Invalid_Command_Received** - A command was received with invalid parameters.
- **Command_Dropped** - A command was dropped due to a full queue.
- **Memory_Region_Dropped** - A memory region was dropped due to a full queue.
- **Table_Validation_Not_Supported** - Produced a packet with the contents of the parameter store.

3.7 Packets

Packets for the Parameter Store component.

Table 9: Parameter Store Packets

Local ID	Packet Name	Type
0x0000 (0)	Stored_Parameters	*Undefined*

Packet Descriptions:

- **Stored_Parameters** - This packet contains a copy of all the parameter stored and managed by this component.

4 Unit Tests

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

4.1 *Parameter_Store_Tests* Test Suite

This is a unit test suite for the Parameter Store component.

Test Descriptions:

- **Test_Nominal_Dump_Parameters** - This unit test tests the nominal dumping of the parameter table by command.
- **Test_Nominal_Table_Upload** - This unit test tests the nominal updating of the parameter table by memory region upload.
- **Test_Nominal_Table_Fetch** - This unit test tests the nominal fetching of the parameter table by into a provided memory region.
- **Test_Table_Upload_Length_Error** - This unit test tests the behavior when updating the parameter table with a memory region of invalid length.
- **Test_Table_Upload_Crc_Error** - This unit test tests the behavior when updating the parameter table with a memory region that contains an invalid CRC.
- **Test_Table_Validate_Unimplemented** - This unit test tests the behavior when unimplemented validation of the parameter table by memory region upload is requested.
- **Test_Table_Fetch_Length_Error** - This unit test tests the behavior when fetching the parameter table with a memory region of invalid length.
- **Test_No_Dump_On_Change** - This unit test tests the no-dump-on-change configuration for the Init function and makes sure the component behaves as expected.
- **Test_Full_Queue** - This unit test tests a command or memory region being dropped due to a full queue.
- **Test_Invalid_Command** - This unit test exercises that an invalid command throws the appropriate event.

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..

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 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 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.

Event.T:

Generic event packet for holding arbitrary events

Table 13: 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 14: 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 15: 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.

Invalid_Parameters_Memory_Region_Crc.T:

A packed record which holds data related to an invalid parameter memory region CRC.

Table 16: Invalid_Parameters_Memory_Region_Crc Packed Record : 168 bits

Name	Type	Range	Size (Bits)	Start Bit	End Bit
Parameters_Region	Parameters_Memory_Region.T	-	104	0	103
Header	Parameter_Table_Header.T	-	48	104	151
Computed_Crc	Crc_16.Crc_16_Type	-	16	152	167

Field Descriptions:

- **Parameters_Region** - The memory region and operation.
- **Header** - The parameter table header stored in the memory region.

- **Computed_Crc** - The FSW computed CRC of the parameter table stored in the memory region.

Invalid_Parameters_Memory_Region_Length.T:

A packed record which holds data related to an invalid parameter memory region length.

Table 17: Invalid_Parameters_Memory_Region_Length Packed Record : 136 bits

Name	Type	Range	Size (Bits)	Start Bit	End Bit
Parameters_Region	Parameters_ Memory_Region.T	-	104	0	103
Expected_Length	Natural	0 to 2147483647	32	104	135

Field Descriptions:

- **Parameters_Region** - The memory region and operation.
- **Expected_Length** - The length bound that the memory region failed to meet.

Memory_Region.T:

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

Table 18: 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.

Packet.T:

Generic packet for holding arbitrary data

Table 19: 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 20: 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

Parameter_Table_Header.T:

A packed record which holds parameter table header data. This data is will be prepended to the table data upon upload. *Preamble (inline Ada definitions):*

```
-- Declare the start index at which to begin calculating the CRC. The
-- start index is dependent on this type, and so is declared here so that
-- it is easier to keep in sync.
Crc_Section_Length : constant Natural := Crc_16.Crc_16_Type'Length;
Version_Length : constant Natural := 4;
```

Table 21: Parameter_Table_Header Packed Record : 48 bits

Name	Type	Range	Size (Bits)	Start Bit	End Bit
Crc_Table	Crc_16.Crc_16_Type	-	16	0	15
Version	Short_Float	-3.40282e+38 to 3.40282e+38	32	16	47

Field Descriptions:

- **Crc_Table** - The CRC of the parameter table, as computed by a ground system, and uplinked with the table.
- **Version** - The current version of the parameter table.

Parameters_Memory_Region.T:

A packed record which holds the parameter memory region to operate on as well as an enumeration specifying the operation to perform.

Table 22: Parameters_Memory_Region Packed Record : 104 bits

Name	Type	Range	Size (Bits)	Start Bit	End Bit
Region	Memory_Region.T	-	96	0	95
Operation	Parameter_Enums. Parameter_Table_ Operation_Type.E	0 => Get 1 => Set 2 => Validate	8	96	103

Field Descriptions:

- **Region** - The memory region.
- **Operation** - The parameter table operation to perform.

Parameters_Memory_Region_Release.T:

A packed record which holds the parameter memory region to release as well as the status returned from the parameter update operation.

Table 23: Parameters_Memory_Region_Release Packed Record : 104 bits

Name	Type	Range	Size (Bits)	Start Bit	End Bit
Region	Memory_Region. T	-	96	0	95
Status	Parameter_ Enums. Parameter_ Table_Update_ Status.E	0 => Success 1 => Length_Error 2 => Crc_Error 3 => Parameter_Error 4 => Dropped	8	96	103

Field Descriptions:

- **Region** - The memory region.
- **Status** - The return status from the parameter update operation.

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 enumerations 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.

Parameter_Enums.Parameter_Table_Update_Status.E:

This status enumeration provides information on the success/failure of a parameter table update.

Table 26: Parameter_Table_Update_Status Literals:

Name	Value	Description
Success	0	Parameter was successfully staged.
Length_Error	1	Parameter table length was not correct.
Crc_Error	2	The computed CRC of the table does not match the stored CRC.
Parameter_Error	3	An individual parameter was found invalid due to a constraint error within a component, or failing component-specific validation.
Dropped	4	The operation could not be performed because it was dropped from a full queue.

Parameter_Enums.Parameter_Table_Operation_Type.E:

This enumeration lists the different parameter table operations that can be performed.

Table 27: Parameter_Table_Operation_Type Literals:

Name	Value	Description
Get	0	Retrieve the current values of the parameters.
Set	1	Set the current values of the parameters.
Validate	2	Validate the current values of the parameters.