

Sequence Store

Component Design Document

1 Description

The Sequence Store component is responsible for storing and managing access to a set of memory regions (slots) which each hold a single sequence. The managed memory regions are usually located in nonvolatile storage and can be read or written to via this component. Sequences are uploaded to slots and then accessed via sequence ID. Sequences can be activated (allowed to be fetched) or deactivated (not allowed to be fetched) by command. The component enforces that all activated sequences have unique IDs. The component will disallow the activation of a sequence if it shares an ID with an already activated sequence.

2 Requirements

The requirements for the Sequence Store component are specified below.

1. The component shall manage regions of memory (slots) for sequence storage.
2. The component shall write slots with sequence data upon sequence upload.
3. The component shall return pointers to sequences when sequence is requested for loading or running.
4. The component shall produce a packet reflecting the sequence IDs stored in the sequence slots.
5. The component shall produce a sequence store packet upon command.
6. The component shall produce a sequence store packet whenever a new sequence has been uploaded.
7. The component shall reject storing a new sequence if it has an invalid length.
8. The component shall reject storing a new sequence if it has 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** - 8
- **Number of Invokee Connectors** - 3
- **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 - 16
- Number of Faults - *None*
- Number of Data Products - *None*
- Number of Data Dependencies - *None*
- Number of Packets - 1

3.2 Diagram

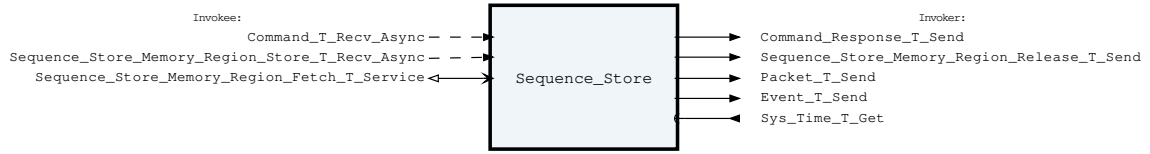


Figure 1: Sequence Store component diagram.

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: Sequence Store Invokee Connectors

Name	Kind	Type	Return_Type	Count
Command_T_Recv_Async	recv_async	Command.T	-	1
Sequence_Store_Memory_Region_Store_T_Recv_Async	recv_async	Sequence_Store_Memory_Region_Store.T	-	1
Sequence_Store_Memory_Region_Fetch_T_Service	service	Packed_Sequence_Id.T	Sequence_Store_Memory_Region_Fetch.T	1

Connector Descriptions:

- **Command_T_Recv_Async** - This is the command receive connector.
- **Sequence_Store_Memory_Region_Store_T_Recv_Async** - This connector is used to load a new sequence into a slot in the store.
- **Sequence_Store_Memory_Region_Fetch_T_Service** - This connector is used to fetch a pointer to a sequence found in the store given its ID.

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: Sequence Store Asynchronous Connectors

Name	Type	Max Size (bytes)
Command_T_Recv_Async	Command.T	265
Sequence_Store_Memory_Region_Store_T_Recv_Async	Sequence_Store_Memory_Region_Store.T	19

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: Sequence Store Invoker Connectors

Name	Kind	Type	Return_Type	Count
Command_Response_T_Send	send	Command_Response.T	-	1
Sequence_Store_Memory_Region_Release_T_Send	send	Sequence_Store_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.
- **Sequence_Store_Memory_Region_Release_T_Send** - After a memory region is stored, it is released via a call to this connector. A status is also returned, so the downstream component can determine if the sequence store operation was successful or not.
- **Packet_T_Send** - The packet connector. This produces a summary of the sequence store slots and what is contained within each.
- **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: Sequence 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.

3.5.3 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: Sequence Store Set Id Bases Parameters

Name	Type
Command_Id_Base	Command_Types.Command_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.
- **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.

3.5.4 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 the memory regions (slots) it is to manage. The `init` subprogram requires the following parameters:

Table 6: Sequence Store Implementation Initialization Parameters

Name	Type	Default Value
Sequence_Slots	Sequence_Slot_Array_Access	<i>None provided</i>
Check_Slots_At_Startup	Boolean	<i>None provided</i>
Dump_Slot_Summary_At_Startup	Boolean	<i>None provided</i>

Parameter Descriptions:

- **Sequence_Slots** - A list of memory regions. Each represents a slot to hold a single sequence of equal or smaller size (including header). This list will be copied into the component at initialization onto a heap allocated memory. The memory regions must not overlap in any way, must be large enough to at least hold a sequence header, and the list must not be empty. These properties will be enforced by the component via assertions when Init is called.
- **Check_Slots_At_Startup** - If True, then check the validity of the sequences in all slots by computing CRCs over them at startup.
- **Dump_Slot_Summary_At_Startup** - If True, then the slot summaries will be dumped at startup.

3.6 Commands

These are the commands for the Sequence Store component.

Table 7: Sequence Store Commands

Local ID	Command Name	Argument Type
0	Activate_Slot	Packed_Slot_Number.T
1	Deactivate_Slot	Packed_Slot_Number.T
2	Check_Slot	Packed_Slot_Number.T
3	Check_All_Slots	-
4	Dump_Summary	-

Command Descriptions:

- **Activate_Slot** - Activate a sequence slot so that its contents can be fetched.
- **Deactivate_Slot** - Deactivate a sequence slot so that its contents can no longer be fetched.
- **Check_Slot** - Check the CRC of a sequence in a particular slot to see if it matches the CRC found in the header.
- **Check_All_Slots** - Check the CRC of sequences in all sequence store slots to see if they match the CRCs found in their headers.
- **Dump_Summary** - Produce a packet with the currently stored slot summary information.

3.7 Parameters

The Sequence Store component has no parameters.

3.8 Events

Events for the Sequence Store component.

Table 8: Sequence Store Events

Local ID	Event Name	Parameter Type
0	Sequence_Slot_Active	Sequence_Store_Memory_Region_Store.T

1	Invalid_Sequence_Crc	Invalid_Sequence_Crc_Info.T
2	Invalid_Sequence_Length	Invalid_Sequence_Length_Info.T
3	Writing_Sequence_To_Slot	Sequence_Store_Memory_Region_Store.T
4	Wrote_Sequence_To_Slot	Slot_Written_Summary.T
5	Checked_Slot_Validity	Packed_Slot_Validity.T
6	Checking_All_Slot_Validity	-
7	Cannot_Activate_Duplicate_Sequence_Id	Packed_Sequence_Id.T
8	Invalid_Slot_Number	Packed_Slot_Number.T
9	Activated_Slot	Packed_Slot_Number.T
10	Deactivated_Slot	Packed_Slot_Number.T
11	Dumping_Slot_Summary	-
12	Dumped_Slot_Summary	-
13	Invalid_Command_Received	Invalid_Command_Info.T
14	Command_Dropped	Command_Header.T
15	Region_Store_Dropped	Sequence_Store_Memory_Region_Store.T

Event Descriptions:

- **Sequence_Slot_Active** - The destination slot is currently active so cannot be written to.
- **Invalid_Sequence_Crc** - The sequence to store has an invalid CRC and cannot be stored.
- **Invalid_Sequence_Length** - The sequence to store has a length that will not fit within the destination slot.
- **Writing_Sequence_To_Slot** - A command sequence memory region was received. It will be validated and written to the destination slot.
- **Wrote_Sequence_To_Slot** - A command sequence was successfully written to the destination slot.
- **Checked_Slot_Validity** - The validity of a particular slot was checked and the results are included in this event.
- **Checking_All_Slot_Validity** - The validity for all slots within the sequence store.
- **Cannot_Activate_Duplicate_Sequence_Id** - A duplicate sequence ID cannot be activated. All active sequence IDs must be unique.
- **Invalid_Slot_Number** - Slot number does not exist and is out of range for the available slots in the component.
- **Activated_Slot** - Slot was activated.
- **Deactivated_Slot** - Slot was deactivated.
- **Dumping_Slot_Summary** - Starting to produce a packet with a summary of the contents of the sequence store slots.
- **Dumped_Slot_Summary** - Produced a packet with a summary of the contents of the sequence store slots.
- **Invalid_Command_Received** - A command was received with invalid parameters.
- **Command_Dropped** - A command was dropped due to a full queue.
- **Region_Store_Dropped** - A sequence store memory region was dropped due to a full queue.

3.9 Data Products

The Sequence Store component has no data products.

3.10 Data Dependencies

The Sequence Store component has no data dependencies.

3.11 Packets

Packets for the Sequence Store component.

Table 9: Sequence Store Packets

Local ID	Packet Name	Type
0x0000 (0)	Slot_Summaries	<i>Undefined</i>

Packet Descriptions:

- **Slot_Summaries** - This packet contains a summary of what is contained in each Sequence Store slot. Since the number of slots the component manages is determined at initialization the type for the FSW is left undefined.

4 Unit Tests

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

4.1 *Sequence_Store_Tests* Test Suite

This is a unit test suite for the Sequence Store.

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_Dump_Summary** - This unit test tests the dump summary packet and command.
- **Test_Nominal_Activate_Deactivate_Slot** - This unit test tests the slot activate/deactivate commands.
- **Test_Activate_Deactivate_Slot_Fail** - This unit test tests the slot activate/deactivate command failure conditions.
- **Test_Check_Slot** - This unit test tests the check slot command.
- **Test_Check_All_Slots** - This unit test tests the check all slots command.
- **Test_Nominal_Write_Sequence** - This unit test tests writing a sequence to a slot in the nominal case.
- **Test_Write_Sequence_Fail** - This unit test tests writing a sequence to a slot when the operation fails for different reasons.
- **Test_Sequence_Fetch** - This unit test tests fetching a sequence by ID when it exists in the store, when it does not exist in the store, and when it is in the store but marked inactive.
- **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 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.

```

1  -- Type to hold an array of memory regions. Each represents a slot to hold
2  -- a single sequence of equal or smaller size (including header and meta data).
3  type Sequence_Slot_Array is array (Sequence_Store_Types.Slot_Number range <>) of
4  ↪ Memory_Region.T;
5  type Sequence_Slot_Array_Access is access all Sequence_Slot_Array;
```

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.

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_Sequence_Crc_Info.T:

This is a packed record that holds information about a sequence to store with invalid CRC.

Table 16: Invalid_Sequence_Crc_Info Packed Record : 224 bits

Name	Type	Range	Size (Bits)	Start Bit	End Bit
Store	Sequence_Store_Memory_Region_Store.T	-	112	0	111
Header	Sequence_Header.T	-	96	112	207
Computed_Crc	Crc_16.Crc_16_Type	-	16	208	223

Field Descriptions:

- **Store** - The to-store info.
- **Header** - The sequence header.
- **Computed_Crc** - The FSW computed CRC of the sequence stored in the memory region.

Invalid_Sequence_Length_Info.T:

This is a packed record that holds information about a sequence to store with invalid length.

Table 17: Invalid_Sequence_Length_Info Packed Record : 240 bits

Name	Type	Range	Size (Bits)	Start Bit	End Bit
Store	Sequence_Store_Memory_Region_Store.T	-	112	0	111
Header	Sequence_Header.T	-	96	112	207
Length_Bound	Natural	0 to 2147483647	32	208	239

Field Descriptions:

- **Store** - The to-store info.
- **Header** - The sequence header.
- **Length_Bound** - The length of the slot.

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.

Packed_Sequence_Id.T:

A packed record which holds a sequence identifier.

Table 19: Packed_Sequence_Id Packed Record : 16 bits

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

Field Descriptions:

- **Id** - The sequence identifier.

Packed_Slot_Number.T:

This is a packed record that holds a slot number.

Table 20: Packed_Slot_Number Packed Record : 16 bits

Name	Type	Range	Size (Bits)	Start Bit	End Bit
Slot	Sequence_Store_ Types.Slot_Number	0 to 65535	16	0	15

Field Descriptions:

- **Slot** - The slot number.

Packed_Slot_Validity.T:

This is a packed record that holds information about the validity of a particular slot.

Table 21: Packed_Slot_Validity Packed Record : 24 bits

Name	Type	Range	Size (Bits)	Start Bit	End Bit
Slot	Sequence_Store_ Types.Slot_Number	0 to 65535	16	0	15
Validity	Sequence_Store_ Enums.Slot_Valid_ Type.E	0 => Unchecked 1 => Valid 2 => Invalid 3 => Undefined	8	16	23

Field Descriptions:

- **Slot** - The slot number.
- **Validity** - The validity of the slot, based on a CRC computation against the CRC in the sequence's header.

Packet.T:

Generic packet for holding arbitrary data

Table 22: 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 23: 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

Sequence_Header.T:

The header for a command sequence.

Table 24: Sequence_Header Packed Record : 96 bits

Name	Type	Range	Size (Bits)	Start Bit	End Bit
Crc	Crc_16.Crc_16_ Type	-	16	0	15
Version	Interfaces. Unsigned_32	0 to 4294967295	32	16	47
Category	Interfaces. Unsigned_16	0 to 65535	16	48	63
Id	Sequence_Types. Sequence_Id	0 to 65535	16	64	79

Length	Sequence_Types. Sequence_Length_ Type	0 to 65535	16	80	95
--------	---	------------	----	----	----

Field Descriptions:

- **Crc** - The CRC of the sequence as computed by the ground.
- **Version** - The version of the compiler that the sequence was compiled with.
- **Category** - The category for this sequence. This field is currently unused by Adamant.
- **Id** - The sequence identifier.
- **Length** - The length of the sequence data in bytes (including the header).

Sequence_Store_Memory_Region_Fetch.T:

A packed record which holds a sequence fetch return.

Table 25: Sequence_Store_Memory_Region_Fetch Packed Record : 104 bits

Name	Type	Range	Size (Bits)	Start Bit	End Bit
Sequence_Region	Memory_Region.T	-	96	0	95
Status	Sequence_Store_Enums.Sequence_Fetch_Status.E	0 => Success 1 => Id_Not_Found	8	96	103

Field Descriptions:

- **Sequence_Region** - The memory region that holds the sequence.
- **Status** - The return status for the memory region fetch operation.

Sequence_Store_Memory_Region_Release.T:

A packed record which holds a sequence store release.

Table 26: Sequence_Store_Memory_Region_Release Packed Record : 104 bits

Name	Type	Range	Size (Bits)	Start Bit	End Bit
Sequence_Region	Memory_Region.T	-	96	0	95
Status	Sequence_Store_Enums.Sequence_Store_Status.E	0 => Success 1 => Invalid_Slot_Number 2 => Slot_In_Use 3 => Crc_Error 4 => Length_Error 5 => Dropped	8	96	103

Field Descriptions:

- **Sequence_Region** - The memory region that holds the sequence, to be released.
- **Status** - The return status for the memory region store operation.

Sequence_Store_Memory_Region_Store.T:

A packed record which holds a sequence identifier.

Table 27: Sequence_Store_Memory_Region_Store Packed Record : 112 bits

Name	Type	Range	Size (Bits)	Start Bit	End Bit
Slot	Sequence_Store_Types.Slot_Number	0 to 65535	16	0	15
Sequence_Region	Memory_Region.T	-	96	16	111

Field Descriptions:

- **Slot** - The slot number to store the sequence in.
- **Sequence_Region** - The memory region that holds the sequence.

Sequence_Store_Slot_Header.T:

This is the sequence header (including the slot meta data) contained at the beginning of each sequence store slot.

Table 28: Sequence_Store_Slot_Header Packed Record : 104 bits

Name	Type	Range	Size (Bits)	Start Bit	End Bit
Slot_Info	Sequence_Store_Slot_Metadata.T	-	8	0	7
Seq_Header	Sequence_Header.T	-	96	8	103

Field Descriptions:

- **Slot_Info** - The slot metadata.
- **Seq_Header** - The header for the command sequence.

Sequence_Store_Slot_Metadata.T:

Sequence store slot metadata, included at the beginning of each slot before the sequence. *Preamble*

(inline Ada definitions):

```
1 type Five_Bit_Pad_Type is mod 2**5;
```

Table 29: Sequence_Store_Slot_Metadata Packed Record : 8 bits

Name	Type	Range	Size (Bits)	Start Bit	End Bit
Reserved	Five_Bit_Pad_Type	0 to 31	5	0	4
State	Sequence_Store_Enums.Slot_State_Type.E	0 => Inactive 1 => Active	1	5	5

Validity	Sequence_Store_Enums.Slot_Valid_Type.E	0 => Unchecked 1 => Valid 2 => Invalid 3 => Undefined	2	6	7
----------	--	--	---	---	---

Field Descriptions:

- **Reserved** - Unused bits.
- **State** - The state of the slot.
- **Validity** - The validity of the slot, based on a CRC computation against the CRC in the sequence's header.

Slot_Written_Summary.T:

This is a packed record that holds information about a recently stored sequence.

Table 30: Slot_Written_Summary Packed Record : 216 bits

Name	Type	Range	Size (Bits)	Start Bit	End Bit
Store	Sequence_Store_Memory_Region_Store.T	-	112	0	111
Header	Sequence_Store_Slot_Header.T	-	104	112	215

Field Descriptions:

- **Store** - The to-store info.
- **Header** - The slot header.

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 31: 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 32: 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.

Sequence_Store_Enums.Slot_State_Type.E:

This enumeration determines the state of a sequence slot in the store.

Table 33: Slot_State_Type Literals:

Name	Value	Description
Inactive	0	The slot is inactive and the contained sequence cannot be fetched.
Active	1	The slot is active and the contained sequence can be fetched.

Sequence_Store_Enums.Slot_Valid_Type.E:

This enumeration determines the valid state of a sequence slot in the store based on a calculated CRC of the sequence.

Table 34: Slot_Valid_Type Literals:

Name	Value	Description
Unchecked	0	The CRC of the sequence in the slot has not been checked for validity.
Valid	1	The CRC of the sequence in the slot has been checked for validity and matches the stored CRC in the sequence header.
Invalid	2	The CRC of the sequence in the slot has been checked for validity and does not match the stored CRC in the sequence header.
Undefined	3	This is an invalid enumeration value. However, since this is represented in 2-bits and read from MRAM, it is possible that this could occur should the memory be corrupted, or never properly initialized. To keep the type unconstrained we need to include this value.

Sequence_Store_Enums.Sequence_Store_Status.E:

This status enumeration provides information on the success/failure of a sequence store operation.

Table 35: Sequence_Store_Status Literals:

Name	Value	Description
Success	0	Sequence was successfully stored.
Invalid_Slot_Number	1	The slot number does not exist within the component.
Slot_In_Use	2	The operation could not be completed because the destination slot is currently active.
Crc_Error	3	The computed CRC of the sequence does not match the stored CRC.
Length_Error	4	The sequence length was too large to store in the selected slot.
Dropped	5	The operation could not be performed because it was dropped from a full queue.

Sequence_Store_Enums.Sequence_Fetch_Status.E:

This status enumeration provides information on the success/failure of a sequence fetch operation.

Table 36: Sequence_Fetch_Status Literals:

Name	Value	Description
Success	0	Sequence was successfully returned.
Id_Not_Found	1	A sequence with the provided ID was not found in the store.