# Command Sequencer

*Component Design Document*

## 1 Description

The Command Sequencer component executes command sequences with a configurable number of engines. The sequence engines execute sequences in the LASP Awesome Sequence Engine Language (LASEL) compiled by the LASP SEQ tool. Documentation on LASEL is included in this component's doc/ directory.
This component runs a configurable number of sequence engines using a single Adamant task. The task runs each engine in priority order, where lower numbered engines take precedence over higher numbered engines. Each engine contains a configurable-sized stack that allows sequences to call subsequences. This component adheres to the property that commands are only executed after previous commands have completed (ie. a command response has been received). In this way the sequences are largely event driven, waiting on the execution of previous commands to finish prior to executing subsequent ones. A periodic tick is supplied to the component to provide timing control for sequences that need to execute relative or absolute waits, or check until a telemetry condition has been met before proceeding.
The sequence engine and sequence runtime (LASEL interpreter) is located in the seq/ directory.

## 2 Requirements

The requirements for the Command Sequencer component are specified below.

1. The component shall run sequences compiled to the SEQ instruction language opcodes.
2. The component shall contain a compile-time configurable number of sequence engines.
3. The component shall contain a compile-time configurable sequence stack depth in each engine.
4. The component shall contain a compile-time configurable tick which determines the frequency of sequence waits and telemetry fetches.
5. The component shall prevent an executing sequence from executing a compile-time configurable number of instructions without yielding the CPU.
6. The component shall execute commands within a running sequence in order, not running a subsequent command before a previous command has completed execution.
7. The component shall report the internal state, sequence ID, and sequence position of each sequence engine in telemetry.
8. The component shall provide a command to kill a sequence engine.
9. The component shall not execute a sequence if its CRC or length cannot be validated prior to load.
10. The component shall report sequence print statements as events.

## 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** - 12
- **Number of Invokee Connectors** - 4
- **Number of Invoker Connectors** - 8
- **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** - 38
- **Number of Faults** - *None*
- **Number of Data Products** - 1
- **Number of Data Dependencies** - *None*
- **Number of Packets** - 2

### 3.2 Diagram

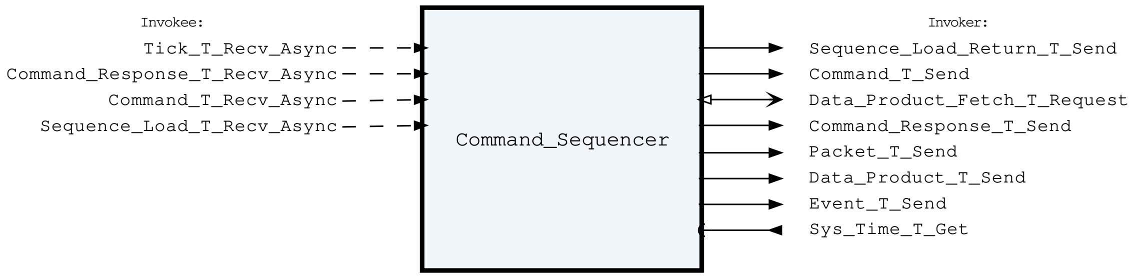

Figure 1: Command Sequencer component diagram.

### 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 Sequencer Invokee Connectors

| **Name** | **Kind** | **Type** | **Return_Type** | **Count** |
|---|---|---|---|---|
| Tick_T_Recv_Async | recv_async | Tick.T | - | 1 |

| | | | | |
|---|---|---|---|---|
| Command_ Response_T_ Recv_Async | recv_async | Command_ Response.T | - | 1 |
| Command_T_Recv_ Async | recv_async | Command.T | - | 1 |
| Sequence_Load_ T_Recv_Async | recv_async | Sequence_Load.T | - | 1 |

Connector Descriptions:

- **Tick_T_Recv_Async** - The schedule invokee connector. This is used to detect sequence timeout errors, meter out the checking of telemetry for sequence conditionals, and determine when to resume a sequence after a relative or absolute wait.
- **Command_Response_T_Recv_Async** - Command responses from sent commands are received on this connector, allowed subsequent commands in a sequence to be sent out. Note that during initialization a Register_Source message needs to be sent via this connector for each engine, providing a unique source identifier. This source identifier will be used for each engine when sending out commands after initialization. If using the command router, you should set up the command router to include a unique arrayed connector entry for each engine that needs to be registered.
- **Command_T_Recv_Async** - The command receive connector. Commands received on this connector are executed by the sequencer itself, ie. halting a sequence.
- **Sequence_Load_T_Recv_Async** - This connector is used to load a sequence into the command sequencer via memory region. Sequences are not copied to this component's memory, they are run directly from the address provided in the given sequence load memory region.

### 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 Sequencer Asynchronous Connectors

| **Name** | **Type** | **Max Size (bytes)** |
|---|---|---|
| Tick_T_Recv_Async | Tick.T | 17 |
| Command_Response_T_Recv_ Async | Command_Response.T | 12 |
| Command_T_Recv_Async | Command.T | 265 |
| Sequence_Load_T_Recv_Async | Sequence_Load.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: Command Sequencer Invoker Connectors

| Name | Kind | Type | Return_ Type | Count |
|---|---|---|---|---|
| Sequence_Load_Return_T_Send | send | Sequence_Load_Return.T | - | 1 |
| Command_T_Send | send | Command.T | - | 1 |
| Data_Product_Fetch_T_Request | request | Data_Product_Fetch.T | Data_Product_Return.T | 1 |
| Command_Response_T_Send | send | Command_Response.T | - | 1 |
| Packet_T_Send | send | Packet.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:

- **`Sequence_Load_Return_T_Send`** - This connector is used to send the return status from a sequence load operation.
- **`Command_T_Send`** - The command send connector. Commands originating from sequences are sent out of this connector.
- **`Data_Product_Fetch_T_Request`** - Fetch a data product item from the database. This is used to check telemetry during conditionals in a sequence.
- **`Command_Response_T_Send`** - This connector is used to register the components commands with the command router component.
- **`Packet_T_Send`** - Packets are sent out of this connector.
- **`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: Command Sequencer 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: Command Sequencer 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` |
| `Packet_Id_Base` | `Packet_Types.Packet_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.
- **`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 initialization subprogram creates a sequencer with the desired number of engines and internal stack sizes. The `init` subprogram requires the following parameters:

Table 6: Command Sequencer Implementation Initialization Parameters

| **Name** | **Type** | **Default Value** |
|---|---|---|
| `Num_Engines` | `Seq_Types.Num_Engines_Type` | *None provided* |
| `Stack_Size` | `Seq_Types.Stack_Depth_Type` | *None provided* |
| `Create_Sequence_Load_Command_Function` | `Create_Sequence_Load_Command_Access` | *None provided* |
| `Packet_Period` | `Interfaces.Unsigned_16` | *None provided* |
| `Continue_On_Command_Failure` | `Boolean` | *None provided* |
| `Timeout_Limit` | `Natural` | *None provided* |
| `Instruction_Limit` | `Positive` | *None provided* |

Parameter Descriptions:

- **`Num_Engines`** - The number of engines allocated in the sequencer. This determines the number of sequences the component can run in parallel.

- **`Stack_Size`** - The size of the stack allocated for each engine in entries. Each stack entry contains a single running sequence, and additional stack entries can be used for subsequence calls. A value of 5 here would allow a sequence to call subsequences up to 5 levels deep.
- **`Create_Sequence_Load_Command_Function`** - When a sequence loads or spans or calls another sequence, the command sequencer will call this function to formulate the correct sequence load command for the assembly. Since the specifics of sequence loading often varies on a mission by mission basis, this function allows the encoding of that mission specific behavior by the user.
- **`Packet_Period`** - The initial packet rate for the sequencer summary packet in ticks. A value of 0 disabled the packet.
- **`Continue_On_Command_Failure`** - If set to True, then the sequence engines will continue to execute even if a sent command fails. If set to False, then the engines will halt with an error status if a sent command fails.
- **`Timeout_Limit`** - The number of ticks to wait before timing out sequencer operations such as waiting on a command response or subsequence load. If a timeout of this type occurs the engine will transition to an error state. A value of zero disables these timeouts.
- **`Instruction_Limit`** - The maximum number of sequence instructions we allow the sequence to execute without hitting a pausing action such as sending a command, waiting on telemetry, or waiting for a relative or absolute time. The purpose of this parameter is to prevent a sequence from entering an infinite execution loop which would cause the entire component task to hang indefinitely. You should set the value to some maximum number of instructions that you never expect any of your compiled sequences to hit.

## 3.6 Commands

These are the commands for the Command Sequencer.

Table 7: Command Sequencer Commands

| Local ID | Command Name | Argument Type |
|---|---|---|
| 0 | `Kill_All_Engines` | - |
| 1 | `Kill_Engine` | `Packed_Sequence_Engine_Id.T` |
| 2 | `Set_Summary_Packet_Period` | `Packed_U16.T` |
| 3 | `Issue_Details_Packet` | `Packed_Sequence_Engine_Id.T` |
| 4 | `Set_Engine_Arguments` | `Packed_Variable_Array.T` |

Command Descriptions:

- **`Kill_All_Engines`** - This command halts all currently running engines.
- **`Kill_Engine`** - This command halts an engine with the provided engine number.
- **`Set_Summary_Packet_Period`** - Set the period of the summary packet. A period of zero disables the sending of the packet.
- **`Issue_Details_Packet`** - The sequence details packet for a particular engine is issued when this command is received.
- **`Set_Engine_Arguments`** - If a sequence requires arguments to be run correctly at the parent level, this command can be used to set the arguments into the engine prior to loading the sequence. This command will only be executed if there is no other sequence loaded in this engine. Arguments can only be set for a sequence that is going to be loaded into the parent stack position. If this command is not run prior to running a sequence in an engine, then the arguments will default to values of zero. If a sequence does not require some or all of the 16 arguments, then those arguments will never be read, and thus do not need to be set by this command.

## 3.7 Parameters

The Command Sequencer component has no parameters.

## 3.8 Events

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

Table 8: Command Sequencer Events

| Local ID | Event Name | Parameter Type |
|---|---|---|
| 0 | Starting_Sequence | Sequence_Load_Info.T |
| 1 | Finished_Sequence | Packed_Sequence_Engine_Id.T |
| 2 | Summary_Packet_Period_Set | Packed_U16.T |
| 3 | Details_Packet_Sent | Packed_Sequence_Engine_Id.T |
| 4 | Invalid_Engine_Id | Packed_Sequence_Engine_Id.T |
| 5 | Invalid_Sequence_Crc | Sequence_Crc_Error.T |
| 6 | Invalid_Sequence_Length | Sequence_Length_Error.T |
| 7 | Invalid_Sequence_Id | Sequence_Id_Error.T |
| 8 | Load_To_Uninitialized_Engine | Sequence_Load.T |
| 9 | Load_To_Invalid_Engine_Id | Sequence_Load.T |
| 10 | No_Engine_Available | Sequence_Load.T |
| 11 | No_Engine_Available_For_Load | Packed_Sequence_Engine_Id.T |
| 12 | Engine_In_Use | Sequence_In_Use_Error.T |
| 13 | Sequence_Load_Error | Sequence_Load_Error_Info.T |
| 14 | Killed_All_Engines | - |
| 15 | Killed_Engine | Packed_Sequence_Engine_Id.T |
| 16 | Dropped_Command | Command_Header.T |
| 17 | Dropped_Command_Response | Command_Response.T |
| 18 | Dropped_Tick | Tick.T |
| 19 | Dropped_Sequence_Load | Sequence_Load.T |
| 20 | Invalid_Command_Received | Invalid_Command_Info.T |
| 21 | Unexpected_Command_Response | Command_Response.T |
| 22 | Unexpected_Register_Source | Command_Response.T |
| 23 | Sequence_Execution_Error | Engine_Error_Type.T |
| 24 | Sequence_Timeout_Error | Engine_Error_Type.T |
| 25 | Unexpected_Command_Response_Id | Unexpected_Command_Response_Info.T |
| 26 | Sequence_Command_Failure | Command_Fail_Error_Type.T |
| 27 | Engine_Id_Out_Of_Range_Error | Engine_Id_Out_Of_Range.T |
| 28 | Engine_Unavailable_For_Load | Engine_Id_Out_Of_Range.T |
| 29 | Data_Product_Id_Out_Of_Range_Error | Engine_Error_Type.T |
| 30 | Data_Product_Extraction_Error | Engine_Error_Type.T |
| 31 | Execute_Recursion_Limit_Exceeded | Packed_Sequence_Engine_Id.T |
| 32 | Invalid_Engine_Kill_Range | Packed_Engine_Kill_Params.T |
| 33 | Engines_Killed | Packed_Engine_Kill_Params.T |
| 34 | Print | Seq_Print_Event_Record.T |

| | | |
|---|---|---|
| 35 | Loaded_Engine_Arguments | Packed_Sequence_Engine_Id.T |
| 36 | Unable_To_Load_Engine_Arguments | Unexpected_Engine_State.T |
| 37 | Unhandled_Telemetry_Type | Engine_Error_Type.T |

Event Descriptions:

- **Starting_Sequence** - Starting a sequence with the following information.
- **Finished_Sequence** - The sequence engine as finished its execution of the parent sequence.
- **Summary_Packet_Period_Set** - A command was received to change the packet period of the summary packet.
- **Details_Packet_Sent** - The sequencer engine details packet was sent for the request engine.
- **Invalid_Engine_Id** - The operation could not be completed because the engine ID provided is invalid.
- **Invalid_Sequence_Crc** - The sequence could not be run due to a bad CRC.
- **Invalid_Sequence_Length** - The sequence could not be run due to a bad length.
- **Invalid_Sequence_Id** - The sequence could not be run due to unexpected ID.
- **Load_To_Uninitialized_Engine** - The sequence could not be run because the engine has not yet been initialized.
- **Load_To_Invalid_Engine_Id** - The sequence could not be run due to unexpected engine id.
- **No_Engine_Available** - No engine is available to take a sequence load.
- **No_Engine_Available_For_Load** - No engine is available to take a sequence load from another currently running sequence.
- **Engine_In_Use** - The sequence could not be run because the current engine is in use.
- **Sequence_Load_Error** - A sequence could not be loaded due to an internal sequence runner error.
- **Killed_All_Engines** - A command was executed to kill all running sequences.
- **Killed_Engine** - A command was executed to kill a sequence running in a specific engine
- **Dropped_Command** - A command was dropped due to a full queue.
- **Dropped_Command_Response** - A command response was dropped due to a full queue.
- **Dropped_Tick** - A tick was dropped due to a full queue.
- **Dropped_Sequence_Load** - A sequence load was dropped due to a full queue.
- **Invalid_Command_Received** - A command was received with invalid parameters.
- **Unexpected_Command_Response** - A command response was found with an unrecognized source ID.
- **Unexpected_Register_Source** - An extra source registration was received, but all engines have a source ID already.
- **Sequence_Execution_Error** - An error occurred while executing a sequence.
- **Sequence_Timeout_Error** - A sequence timed out waiting on a command response of subsequence load.
- **Unexpected_Command_Response_Id** - A command response was received with an unexpected command ID.
- **Sequence_Command_Failure** - A command from a sequence failed to execute successfully.
- **Engine_Id_Out_Of_Range_Error** - During a sequence load from an engine, the destination

engine ID was found to be out of range.

- **`Engine_Unavailable_For_Load`** - During a sequence load from an engine, the destination engine ID was found to be unavailable.
- **`Data_Product_Id_Out_Of_Range_Error`** - A data product was fetched with an ID that was not recognized.
- **`Data_Product_Extraction_Error`** - Data could not be parsed out of the fetched data product.
- **`Execute_Recursion_Limit_Exceeded`** - The recursion limit was exceeded in a call to execute. This likely means a malformed sequence was executed.
- **`Invalid_Engine_Kill_Range`** - The engine kill range is invalid and cannot be executed.
- **`Engines_Killed`** - The provided engines were killed by another engine.
- **`Print`** - A sequence is sending the following print statement.
- **`Loaded_Engine_Arguments`** - Arguments were successfully loaded into an engine by command.
- **`Unable_To_Load_Engine_Arguments`** - Arguments were not successfully loaded into an engine because the engine is currently busy.
- **`Unhandled_Telemetry_Type`** - The telemetry type specified in the sequence is not handled by this implementation.

## 3.9 Data Products

Data products for the Command Sequencer component.

Table 9: Command Sequencer Data Products

| **Local ID** | **Data Product Name** | **Type** |
|---|---|---|
| `0x0000 (0)` | `Summary_Packet_Period` | `Packed_U16.T` |

Data Product Descriptions:

- **`Summary_Packet_Period`** - The current packet period for the summary packet.

## 3.10 Data Dependencies

The Command Sequencer component has no data dependencies.

## 3.11 Packets

Packets for the Command Sequencer.

Table 10: Command Sequencer Packets

| **Local ID** | **Packet Name** | **Type** |
|---|---|---|
| `0x0000 (0)` | `Summary_Packet` | *Undefined* |
| `0x0001 (1)` | `Details_Packet` | *Undefined* |

Packet Descriptions:

- **`Summary_Packet`** - This packet contains a brief summary of what sequences are running in the sequencer's engines. Its type is determined dynamically based on the number of engines declared in the component.
- **`Details_Packet`** - This packet contains all useful information about the current state a single engine within the sequencer. Its type is determined dynamically based on the stack size allocated to the engines.

### 3.12 Faults

The Command Sequencer component has no faults.

## 4 Unit Tests

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

### 4.1 *Command_ Sequencer_ Tests* Test Suite

This is a unit test suite for the Command Sequencer component.

Test Descriptions:

- **`Test_Nominal_Load_And_Run_Sequence`** - This unit test tests loading and running a simple sequence.
- **`Test_Nominal_Subsequence_Load`** - This unit test tests loading and running a sequence that loads a subsequence.
- **`Test_Nominal_Sequence_Spawn`** - This unit test tests loading and running a sequence that loads a sequence in another engine.
- **`Test_Nominal_Sequence_Replace`** - This unit test tests loading and running a sequence that loads a sequence into the same engine.
- **`Test_Nominal_Sequence_Telemetry_Compare`** - This unit test tests loading and running a sequence that executes a conditional on a data product.
- **`Test_Nominal_Sequence_Wait_New_Value`** - This unit test tests loading and running a sequence that executes a conditional on a data product with the waitnewvalue keyword.
- **`Test_Nominal_Fetch_Data_Product`** - This unit test tests loading and running a sequence that grabs a data product and sets it as a local variable.
- **`Test_Sequence_Telemetry_Compare_Error`** - This unit test tests loading and running a sequence that executes a conditional on a data product that is malformed.
- **`Test_Sequence_Telemetry_Compare_Corner_Cases`** - This unit test tests a few corner cases related to telemetry comparisons and makes sure they behave as intended.
- **`Test_Sequence_Spawn_Invalid_Engine`** - This unit test tests loading and running a sequence that loads a sequence into another engine that does not exist.
- **`Test_Sequence_Spawn_Unavailable_Engine`** - This unit test tests loading and running a sequence that loads a sequence into an engine that is currently busy.
- **`Test_Sequence_Spawn_Any_Unavailable`** - This unit test tests loading and running a sequence that loads a sequence into any engine when no engines are available.
- **`Test_Data_Product_Fetch_Error`** - This unit test tests loading and running a sequence that executes a conditional on a data product that is not available or has an unknown ID.
- **`Test_Relative_And_Absolute_Wait_Sequence`** - This unit test tests loading and running a simple sequence with relative and absolute waits.

- **Test_Sequence_Load_Error** - This unit test tests all the error conditions associated with a bad sequence load.
- **Test_Sequence_Execution_Error** - This unit test tests how the component responds to a sequence that has a failed command.
- **Test_Sequence_Timeouts** - This unit test tests sequence command and subsequence load timeouts.
- **Test_Issue_Details_Packet** - This unit test tests issuing the details packet by command.
- **Test_Set_Summary_Packet_Period** - This unit test tests changing the summary packet period by command.
- **Test_Command_Invalid_Engine** - This unit test exercises commands send to invalid engine IDs and makes sure they do not execute.
- **Test_Invalid_Command** - This unit test exercises that an invalid command throws the appropriate event.
- **Test_Queue_Overflow** - This unit test exercises that a queue overflow results in the appropriate event.
- **Test_Sequence_Internal_Execution_Error** - This unit test tests how the component responds to a sequence that is corrupted and fails internally.
- **Test_Kill_Engine_Command** - This unit test tests the kill engine command.
- **Test_Kill_Opcode** - This unit test tests the kill engine sequence opcode.
- **Test_Recursion_Error** - This unit test tests the sequence recursion error, where the execute recursion limit is exceeded.
- **Test_Print** - This unit test tests the sequencer print opcode which produces an event from the command sequencer.
- **Test_Set_Engine_Arguments** - This unit test tests the set engine arguments command.
- **Test_Return_Val** - This unit test tests the return value feature.
- **Test_Bad_Float** - This unit test tests what happens when a floating point sequence variable overflows.
- **Test_Complex_Command** - This unit test tests to make sure commands with complex arguments are formed correctly by the sequencer.
- **Test_Signed_Integer_Handling** - This unit test tests a signed integer corner case that needs to be handled.
- **Test_Set_Telemetry_Timeout** - This unit test tests a telemetry fetch when the telemetry item never becomes available.
- **Test_Sub_Seq_Load_Timeout** - This unit test tests when a sub sequence load fails due to timeout.

# 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.

```ada
-- Create a function type that takes in sequence load information and returns a
↪  command that performs the loading and running of
-- a sequence. The way a sequence is fetched and loaded into the sequencer may
↪  vary from mission to mission. This allows the user
```

```
-- of the sequencer to encode this mission specific behavior within a function
↪  that is passed at initialization.
--
-- When a sequence loads or spans or calls another sequence, the command
↪  sequencer will call this function to formulate the correct
-- sequence load command for the assembly. Note that the Source_Id in the
↪  command header does not need to be set by this function,
-- the sequencer will set it correctly prior to sending this command out.
type Create_Sequence_Load_Command_Access is access function (Id : in
↪  Sequence_Types.Sequence_Id; Engine_Number : in Seq_Types.Sequence_Engine_Id;
↪  Engine_Request : in
↪  Command_Sequencer_Enums.Sequence_Load_Engine_Request_Type.E) return
↪  Command.T;
```

## 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 11: 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_Fail_Error_Type.T:

A packed record which holds information about a command failure error from a sequence engine.

Table 12: Command_Fail_Error_Type Packed Record : 152 bits

| Name | Type | Range | Size (Bits) | Start Bit | End Bit |
|---|---|---|---|---|---|
| Response | Command_Response.T | - | 56 | 0 | 55 |
| Error_Report | Engine_Error_Type.T | - | 96 | 56 | 151 |

Field Descriptions:

- **Response** - The command response reporting the failure.
- **Error_Report** - The error report of the engine.

## Command_Header.T:

Generic command header for holding arbitrary commands

Table 13: 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 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`<br>`1 => Failure`<br>`2 => Id_Error`<br>`3 => Validation_Error`<br>`4 => Length_Error`<br>`5 => Dropped`<br>`6 => Register`<br>`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 15: 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_Fetch.T:

A packed record which holds information for a data product request.

Table 16: Data_Product_Fetch Packed Record : 16 bits

| Name | Type | Range | Size (Bits) | Start Bit | End Bit |
|---|---|---|---|---|---|
| Id | Data_Product_Types. Data_Product_Id | 0 to 65535 | 16 | 0 | 15 |

Field Descriptions:

- **Id** - The data product identifier

## 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

## Data_Product_Return.T:

This record holds data returned from a data product fetch request.

Table 18: Data_Product_Return Packed Record : 352 bits *(maximum)*

| Name | Type | Range | Size (Bits) | Start Bit | End Bit | Variable Length |
|---|---|---|---|---|---|---|
| `The_Status` | `Data_Product_Enums.Fetch_Status.E` | `0 => Success`<br>`1 => Not_Available`<br>`2 => Id_Out_Of_Range` | 8 | 0 | 7 | - |
| `The_Data_Product` | `Data_Product.T` | - | 344 | 8 | 351 | - |

Field Descriptions:

- **`The_Status`** - A status relating whether or not the data product fetch was successful or not.
- **`The_Data_Product`** - The data product item returned.

## Engine_Error_Type.T:

A packed record which holds error information for a particular sequence engine.

Table 19: Engine_Error_Type Packed Record : 96 bits

| Name | Type | Range | Size (Bits) | Start Bit | End Bit |
|---|---|---|---|---|---|
| `Engine_Id` | `Seq_Types.Sequence_Engine_Id` | `0 to 255` | 8 | 0 | 7 |
| `Sequence_Id` | `Sequence_Types.Sequence_Id` | `0 to 65535` | 16 | 8 | 23 |
| `Engine_State` | `Seq_Enums.Seq_Engine_State.E` | `0 => Uninitialized`<br>`1 => Inactive`<br>`2 => Reserved`<br>`3 => Active`<br>`4 => Waiting`<br>`5 => Engine_Error` | 3 | 24 | 26 |

| | | | | | |
|---|---|---|---|---|---|
| Sequence_ State | Seq_ Enums. Seq_ Runtime_ State.E | 0 => Unloaded<br>1 => Ready<br>2 => Done<br>3 => Wait_Relative<br>4 => Wait_Absolute<br>5 => Wait_Command<br>6 => Wait_Telemetry_Set<br>7 => Wait_Telemetry_Value<br>8 => Wait_Telemetry_Relative<br>9 => Telemetry_Set<br>10 => Timeout<br>11 => Kill_Engine<br>12 => Wait_Load_New_Seq_Overwrite<br>13 => Wait_Load_New_Sub_Seq<br>14 => Wait_Load_New_Seq_Elsewhere<br>15 => Print<br>16 => Error | 5 | 27 | 31 |
| Stack_ Level | Seq_ Types. Max_Seq_ Num | 0 to 255 | 8 | 32 | 39 |
| Program_ Counter | Seq_ Types. Seq_ Position | 0 to 65535 | 16 | 40 | 55 |
| Error_ Type | Seq_ Enums. Seq_ Error.E | 0 => None<br>1 => Parse<br>2 => Opcode<br>3 => Command_Parse<br>4 => Command_Length<br>5 => Command_Fail<br>6 => Update_Bit_Pattern<br>7 => Command_Argument<br>8 => Telemetry_Fail<br>9 => Variable<br>10 => Jump<br>11 => Cast<br>12 => Limit<br>13 => Eval<br>14 => Float_Value<br>15 => Execute<br>16 => Wait<br>17 => Load<br>18 => Spawn<br>19 => Load_Header<br>20 => Load_Length<br>21 => Invalid_Op<br>22 => Kill<br>23 => Recursion<br>24 => Command_Timeout<br>25 => Load_Timeout<br>26 => Telemetry_Timeout<br>27 => Unimplemented | 8 | 56 | 63 |
| Errant_ Field_ Number | Interfaces. Unsigned_ 32 | 0 to 4294967295 | 32 | 64 | 95 |

Field Descriptions:

- **`Engine_Id`** - The sequence engine identifier.
- **`Sequence_Id`** - The sequence ID of the lowest stack level child sequence.
- **`Engine_State`** - The sequence engine state.
- **`Sequence_State`** - The running sequence state.
- **`Stack_Level`** - How deep is the current stack usage? This reports the stack level of the currently running sequence.
- **`Program_Counter`** - The current program counter (relative address pointing to the current sequence instruction) of the lowest level child sequence.
- **`Error_Type`** - The sequence error type.
- **`Errant_Field_Number`** - This indicates which field of a packed record failed validation. The field that was invalid. 1 is the first field, 0 means no invalid field found.

## Engine_Id_Out_Of_Range.T:

A packed record which holds a sequence engine identifier for an engine that was out of range.

Table 20: Engine_Id_Out_Of_Range Packed Record : 16 bits

| Name | Type | Range | Size (Bits) | Start Bit | End Bit |
|---|---|---|---|---|---|
| `Engine_Id` | `Seq_Types.Sequence_Engine_Id` | `0 to 255` | 8 | 0 | 7 |
| `Engine_Id_To_Load` | `Seq_Types.Sequence_Engine_Id` | `0 to 255` | 8 | 8 | 15 |

Field Descriptions:

- **`Engine_Id`** - The source sequence engine.
- **`Engine_Id_To_Load`** - The destination sequence engine.

## Event.T:

Generic event packet for holding arbitrary events

Table 21: 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 22: 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 23: 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.

## Memory_Region.T:

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

Table 24: 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_Engine_Kill_Params.T:

A packed record which holds parameters for the engine kill operation.

Table 25: Packed_Engine_Kill_Params Packed Record : 24 bits

| **Name** | **Type** | **Range** | **Size (Bits)** | **Start Bit** | **End Bit** |
|---|---|---|---|---|---|
| `Executing_Engine` | `Seq_Types.Sequence_Engine_Id` | `0 to 255` | 8 | 0 | 7 |
| `First_Engine` | `Seq_Types.Sequence_Engine_Id` | `0 to 255` | 8 | 8 | 15 |
| `Num_Engines` | `Seq_Types.Sequence_Engine_Id` | `0 to 255` | 8 | 16 | 23 |

Field Descriptions:

- **`Executing_Engine`** - The engine which encountered this kill operation.
- **`First_Engine`** - The first engine ID to kill.
- **`Num_Engines`** - The number of engines to kill.

## Packed_Sequence_Engine_Id.T:

A packed record which holds a sequence engine identifier.

Table 26: Packed_Sequence_Engine_Id Packed Record : 8 bits

| **Name** | **Type** | **Range** | **Size (Bits)** | **Start Bit** | **End Bit** |
|---|---|---|---|---|---|
| `Engine_Id` | `Seq_Types.Sequence_Engine_Id` | `0 to 255` | 8 | 0 | 7 |

Field Descriptions:

- **`Engine_Id`** - The sequence engine.

## Packed_U16.T:

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

Table 27: 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_Variable_Array.T:

This record holds sequence engine arguments that can be provided to an engine prior to load of a sequence.

Table 28: Packed_Variable_Array Packed Record : 520 bits

| Name | Type | Range | Size (Bits) | Start Bit | End Bit |
|---|---|---|---|---|---|
| Engine_Id | Seq_Types. Sequence_Engine_ Id | 0 to 255 | 8 | 0 | 7 |
| Argument_01 | Interfaces. Unsigned_32 | 0 to 4294967295 | 32 | 8 | 39 |
| Argument_02 | Interfaces. Unsigned_32 | 0 to 4294967295 | 32 | 40 | 71 |
| Argument_03 | Interfaces. Unsigned_32 | 0 to 4294967295 | 32 | 72 | 103 |
| Argument_04 | Interfaces. Unsigned_32 | 0 to 4294967295 | 32 | 104 | 135 |
| Argument_05 | Interfaces. Unsigned_32 | 0 to 4294967295 | 32 | 136 | 167 |
| Argument_06 | Interfaces. Unsigned_32 | 0 to 4294967295 | 32 | 168 | 199 |
| Argument_07 | Interfaces. Unsigned_32 | 0 to 4294967295 | 32 | 200 | 231 |
| Argument_08 | Interfaces. Unsigned_32 | 0 to 4294967295 | 32 | 232 | 263 |
| Argument_09 | Interfaces. Unsigned_32 | 0 to 4294967295 | 32 | 264 | 295 |
| Argument_10 | Interfaces. Unsigned_32 | 0 to 4294967295 | 32 | 296 | 327 |
| Argument_11 | Interfaces. Unsigned_32 | 0 to 4294967295 | 32 | 328 | 359 |
| Argument_12 | Interfaces. Unsigned_32 | 0 to 4294967295 | 32 | 360 | 391 |
| Argument_13 | Interfaces. Unsigned_32 | 0 to 4294967295 | 32 | 392 | 423 |
| Argument_14 | Interfaces. Unsigned_32 | 0 to 4294967295 | 32 | 424 | 455 |
| Argument_15 | Interfaces. Unsigned_32 | 0 to 4294967295 | 32 | 456 | 487 |
| Argument_16 | Interfaces. Unsigned_32 | 0 to 4294967295 | 32 | 488 | 519 |

Field Descriptions:

- **Engine_Id** - The sequence engine.
- **Argument_01** - A 32-bit unsigned integer sequence argument. If the sequence expects this argument to be a type other than an unsigned integer, the unsigned integer value will be interpreted as that type without changing the underlying bit representation.
- **Argument_02** - A 32-bit unsigned integer sequence argument. If the sequence expects this argument to be a type other than an unsigned integer, the unsigned integer value will be interpreted as that type without changing the underlying bit representation.
- **Argument_03** - A 32-bit unsigned integer sequence argument. If the sequence expects this argument to be a type other than an unsigned integer, the unsigned integer value will be interpreted as that type without changing the underlying bit representation.
- **Argument_04** - A 32-bit unsigned integer sequence argument. If the sequence expects this argument to be a type other than an unsigned integer, the unsigned integer value will be interpreted as that type without changing the underlying bit representation.
- **Argument_05** - A 32-bit unsigned integer sequence argument. If the sequence expects this

argument to be a type other than an unsigned integer, the unsigned integer value will be interpreted as that type without changing the underlying bit representation.

- **`Argument_06`** - A 32-bit unsigned integer sequence argument. If the sequence expects this argument to be a type other than an unsigned integer, the unsigned integer value will be interpreted as that type without changing the underlying bit representation.
- **`Argument_07`** - A 32-bit unsigned integer sequence argument. If the sequence expects this argument to be a type other than an unsigned integer, the unsigned integer value will be interpreted as that type without changing the underlying bit representation.
- **`Argument_08`** - A 32-bit unsigned integer sequence argument. If the sequence expects this argument to be a type other than an unsigned integer, the unsigned integer value will be interpreted as that type without changing the underlying bit representation.
- **`Argument_09`** - A 32-bit unsigned integer sequence argument. If the sequence expects this argument to be a type other than an unsigned integer, the unsigned integer value will be interpreted as that type without changing the underlying bit representation.
- **`Argument_10`** - A 32-bit unsigned integer sequence argument. If the sequence expects this argument to be a type other than an unsigned integer, the unsigned integer value will be interpreted as that type without changing the underlying bit representation.
- **`Argument_11`** - A 32-bit unsigned integer sequence argument. If the sequence expects this argument to be a type other than an unsigned integer, the unsigned integer value will be interpreted as that type without changing the underlying bit representation.
- **`Argument_12`** - A 32-bit unsigned integer sequence argument. If the sequence expects this argument to be a type other than an unsigned integer, the unsigned integer value will be interpreted as that type without changing the underlying bit representation.
- **`Argument_13`** - A 32-bit unsigned integer sequence argument. If the sequence expects this argument to be a type other than an unsigned integer, the unsigned integer value will be interpreted as that type without changing the underlying bit representation.
- **`Argument_14`** - A 32-bit unsigned integer sequence argument. If the sequence expects this argument to be a type other than an unsigned integer, the unsigned integer value will be interpreted as that type without changing the underlying bit representation.
- **`Argument_15`** - A 32-bit unsigned integer sequence argument. If the sequence expects this argument to be a type other than an unsigned integer, the unsigned integer value will be interpreted as that type without changing the underlying bit representation.
- **`Argument_16`** - A 32-bit unsigned integer sequence argument. If the sequence expects this argument to be a type other than an unsigned integer, the unsigned integer value will be interpreted as that type without changing the underlying bit representation.

## Packet.T:

Generic packet for holding arbitrary data

Table 29: 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 30: 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

## Seq_Print_Event_Header.T:

A packed record that holds the header for a string to be printed via command sequencer event.

Table 31: Seq_Print_Event_Header Packed Record : 32 bits

| **Name** | **Type** | **Range** | **Size (Bits)** | **Start Bit** | **End Bit** |
|---|---|---|---|---|---|
| `Engine_Id` | `Seq_Types.Sequence_Engine_Id` | `0 to 255` | `8` | `0` | `7` |
| `Sequence_Id` | `Sequence_Types.Sequence_Id` | `0 to 65535` | `16` | `8` | `23` |
| `Print_Type` | `Seq_Enums.Seq_Print_Type.E` | `0 => Debug`<br>`1 => Info`<br>`2 => Critical`<br>`3 => Error` | `8` | `24` | `31` |

Field Descriptions:

- **`Engine_Id`** - The sequence engine that is running the sequence.
- **`Sequence_Id`** - The sequence id that sent the print.
- **`Print_Type`** - The type of print message (debug, info, critical, error).

## Seq_Print_Event_Record.T:

A packed record that holds a string to be printed via command sequencer event. *Preamble (inline Ada definitions):*

```
subtype Print_String_Type is Basic_Types.Byte_Array
↪   (Event_Types.Parameter_Buffer_Type'First ..
↪   Event_Types.Parameter_Buffer_Type'Last -
↪   Seq_Print_Event_Header.Size_In_Bytes);
```

Table 32: Seq_Print_Event_Record Packed Record : 256 bits

| Name | Type | Range | Size (Bits) | Start Bit | End Bit |
|---|---|---|---|---|---|
| Header | Seq_Print_Event_Header.T | - | 32 | 0 | 31 |
| Print_String | Print_String_Type | - | 224 | 32 | 255 |

Field Descriptions:

- **Header** - Basic information about the printed event.
- **Print_String** - The print string.

## Sequence_Crc_Error.T:

This is a packed record that holds information about sequence with bad CRC.

Table 33: Sequence_Crc_Error Packed Record : 224 bits

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

Field Descriptions:

- **Load** - The sequence load info.
- **Header** - The header of the sequence.
- **Computed_Crc** - The computed CRC of the sequence.

## Sequence_Header.T:

The header for a command sequence.

Table 34: 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_Id_Error.T:

This is a packed record that holds information about sequence with bad ID.

Table 35: Sequence_Id_Error Packed Record : 224 bits

| Name | Type | Range | Size (Bits) | Start Bit | End Bit |
|---|---|---|---|---|---|
| `Load` | `Sequence_Load.T` | - | 112 | 0 | 111 |
| `Header` | `Sequence_Header.T` | - | 96 | 112 | 207 |
| `Expected_Id` | `Sequence_Types. Sequence_Id` | `0 to 65535` | 16 | 208 | 223 |

Field Descriptions:

- **`Load`** - The sequence load info.
- **`Header`** - The header of the sequence.
- **`Expected_Id`** - The expected sequence id.

## Sequence_In_Use_Error.T:

This is a packed record that holds information about sequence being loaded to sequence that is busy.

Table 36: Sequence_In_Use_Error Packed Record : 216 bits

| Name | Type | Range | Size (Bits) | Start Bit | End Bit |
|---|---|---|---|---|---|
| `Load` | `Sequence_ Load.T` | - | 112 | 0 | 111 |
| `Header` | `Sequence_ Header.T` | - | 96 | 112 | 207 |

| | | | | | |
|---|---|---|---|---|---|
| State | Seq_<br>Enums.<br>Seq_<br>Runtime_<br>State.E | 0 => Unloaded<br>1 => Ready<br>2 => Done<br>3 => Wait_Relative<br>4 => Wait_Absolute<br>5 => Wait_Command<br>6 => Wait_Telemetry_Set<br>7 => Wait_Telemetry_Value<br>8 => Wait_Telemetry_Relative<br>9 => Telemetry_Set<br>10 => Timeout<br>11 => Kill_Engine<br>12 => Wait_Load_New_Seq_Overwrite<br>13 => Wait_Load_New_Sub_Seq<br>14 => Wait_Load_New_Seq_Elsewhere<br>15 => Print<br>16 => Error | 8 | 208 | 215 |

Field Descriptions:

- **`Load`** - The sequence load info.
- **`Header`** - The header of the sequence.
- **`State`** - The current sequencer state.

## Sequence_Length_Error.T:

This is a packed record that holds information about an invalid sequence length.

Table 37: Sequence_Length_Error Packed Record : 208 bits

| **Name** | **Type** | **Range** | **Size (Bits)** | **Start Bit** | **End Bit** |
|---|---|---|---|---|---|
| Load | Sequence_Load.T | - | 112 | 0 | 111 |
| Header | Sequence_Header.T | - | 96 | 112 | 207 |

Field Descriptions:

- **`Load`** - The sequence load info.
- **`Header`** - The header of the sequence.

## Sequence_Load.T:

A packed record which holds a sequence to load into a specific engine.

Table 38: Sequence_Load Packed Record : 112 bits

| **Name** | **Type** | **Range** | **Size (Bits)** | **Start Bit** | **End Bit** |
|---|---|---|---|---|---|
| Engine_Request | Command_<br>Sequencer_<br>Enums.<br>Sequence_<br>Load_Engine_<br>Request_Type.E | 0 => Specific_Engine<br>1 => Any_Engine | 8 | 0 | 7 |

| | | | | | |
|---|---|---|---|---|---|
| Engine_Id | Seq_Types. Sequence_ Engine_Id | 0 to 255 | 8 | 8 | 15 |
| Sequence_ Region | Memory_Region. T | - | 96 | 16 | 111 |

Field Descriptions:

- **Engine_Request** - Load the sequence into any available engine or a specific engine.
- **Engine_Id** - The destination engine in which to run the sequence.
- **Sequence_Region** - The memory region that holds the sequence.

## Sequence_Load_Error_Info.T:

This is a packed record that holds information about a sequence load error.

Table 39: Sequence_Load_Error_Info Packed Record : 232 bits

| Name | Type | Range | Size (Bits) | Start Bit | End Bit |
|---|---|---|---|---|---|
| Load | Sequence_ Load.T | - | 112 | 0 | 111 |
| Header | Sequence_ Header.T | - | 96 | 112 | 207 |
| Stack_ Level | Seq_ Types. Max_Seq_ Num | 0 to 255 | 8 | 208 | 215 |
| State | Seq_ Enums. Seq_ Runtime_ State.E | 0 => Unloaded<br>1 => Ready<br>2 => Done<br>3 => Wait_Relative<br>4 => Wait_Absolute<br>5 => Wait_Command<br>6 => Wait_Telemetry_Set<br>7 => Wait_Telemetry_Value<br>8 => Wait_Telemetry_Relative<br>9 => Telemetry_Set<br>10 => Timeout<br>11 => Kill_Engine<br>12 => Wait_Load_New_Seq_Overwrite<br>13 => Wait_Load_New_Sub_Seq<br>14 => Wait_Load_New_Seq_Elsewhere<br>15 => Print<br>16 => Error | 8 | 216 | 223 |

| | | | | | |
|---|---|---|---|---|---|
| Sequence_ Error_ Code | Seq_ Enums. Seq_ Error.E | 0 => None<br>1 => Parse<br>2 => Opcode<br>3 => Command_Parse<br>4 => Command_Length<br>5 => Command_Fail<br>6 => Update_Bit_Pattern<br>7 => Command_Argument<br>8 => Telemetry_Fail<br>9 => Variable<br>10 => Jump<br>11 => Cast<br>12 => Limit<br>13 => Eval<br>14 => Float_Value<br>15 => Execute<br>16 => Wait<br>17 => Load<br>18 => Spawn<br>19 => Load_Header<br>20 => Load_Length<br>21 => Invalid_Op<br>22 => Kill<br>23 => Recursion<br>24 => Command_Timeout<br>25 => Load_Timeout<br>26 => Telemetry_Timeout<br>27 => Unimplemented | 8 | 224 | 231 |

Field Descriptions:

- **`Load`** - The sequence load info.
- **`Header`** - The header of the sequence.
- **`Stack_Level`** - How deep is the current stack usage? This reports the stack level of the currently running sequence.
- **`State`** - The running sequence state.
- **`Sequence_Error_Code`** - The error code of the last encountered sequence error.

## Sequence_Load_Info.T:

This is a packed record that holds information about a sequence load.

Table 40: Sequence_Load_Info Packed Record : 224 bits

| Name | Type | Range | Size (Bits) | Start Bit | End Bit |
|---|---|---|---|---|---|
| Load | Sequence_Load.T | - | 112 | 0 | 111 |
| Header | Sequence_Header.T | - | 96 | 112 | 207 |
| Engine_Id | Seq_Types.Sequence_ Engine_Id | 0 to 255 | 8 | 208 | 215 |
| Stack_Level | Seq_Types.Max_Seq_ Num | 0 to 255 | 8 | 216 | 223 |

Field Descriptions:

- **`Load`** - The sequence load info.
- **`Header`** - The header of the sequence.
- **`Engine_Id`** - The sequence engine identifier.
- **`Stack_Level`** - How deep is the current stack usage? This reports the stack level of the currently running sequence.

## Sequence_Load_Return.T:

A packed record which holds a sequence load and the return status of that sequence load.

Table 41: Sequence_Load_Return Packed Record : 120 bits

| **Name** | **Type** | **Range** | **Size (Bits)** | **Start Bit** | **End Bit** |
|---|---|---|---|---|---|
| `Load` | `Sequence_ Load.T` | - | 112 | 0 | 111 |
| `Status` | `Command_ Sequencer_ Enums. Sequence_ Load_ Status.E` | `0 => Success`<br>`1 => Engine_Uninitialized`<br>`2 => Invalid_Engine_Number`<br>`3 => Engine_In_Use`<br>`4 => Unexpected_Sequence_Id`<br>`5 => Crc_Error`<br>`6 => Length_Error`<br>`7 => Load_Error`<br>`8 => Dropped` | 8 | 112 | 119 |

Field Descriptions:

- **`Load`** - The sequence load record.
- **`Status`** - The status of the sequence load 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 42: 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 43: 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.

## Unexpected_Command_Response_Info.T:

A packed record which holds information for an unexpected command response due to unexpected command id.

Table 44: Unexpected_Command_Response_Info Packed Record : 72 bits

| Name | Type | Range | Size (Bits) | Start Bit | End Bit |
|---|---|---|---|---|---|
| Response | Command_Response.T | - | 56 | 0 | 55 |
| Last_Sent_Command_ Id | Command_Types. Command_Id | 0 to 65535 | 16 | 56 | 71 |

Field Descriptions:

- **`Response`** - The command response reporting the failure.
- **`Last_Sent_Command_Id`** - The ID of the command we were expecting to receive a response for.

## Unexpected_Engine_State.T:

A packed record which holds an unexpected state for an engine.

Table 45: Unexpected_Engine_State Packed Record : 16 bits

| Name | Type | Range | Size (Bits) | Start Bit | End Bit |
|---|---|---|---|---|---|
| Engine_Id | Seq_Types. Sequence_ Engine_Id | 0 to 255 | 8 | 0 | 7 |
| Engine_State | Seq_Enums.Seq_ Engine_State.E | 0 => Uninitialized<br>1 => Inactive<br>2 => Reserved<br>3 => Active<br>4 => Waiting<br>5 => Engine_Error | 8 | 8 | 15 |

Field Descriptions:

- **`Engine_Id`** - The source sequence engine.
- **`Engine_State`** - The sequence engine state.

## 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 46: 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_Sequencer_Enums.Sequence_Load_Engine_Request_Type.E:

This enumeration provides the sequence load to any available engine or the specified engine.

Table 47: Sequence_Load_Engine_Request_Type Literals:

| Name | Value | Description |
|---|---|---|
| Specific_Engine | 0 | The sequence must be loaded to a specific engine number. |
| Any_Engine | 1 | The sequence can be loaded to any available engine. |

### Command_Sequencer_Enums.Sequence_Load_Status.E:

This status enumerations provides information on the success/failure of a sequence load and run operation.

Table 48: Sequence_Load_Status Literals:

| Name | Value | Description |
|---|---|---|
| Success | 0 | Sequence was successfully loaded into an engine and started running. |
| Engine_Uninitialized | 1 | The destination engine has not yet been initialized. |
| Invalid_Engine_Number | 2 | The engine number does not exist within the component. |

| | | |
|---|---|---|
| Engine_In_Use | 3 | The destination engine is currently busy and not able to run a new sequence in its current state. |
| Unexpected_Sequence_Id | 4 | The sequence could not be run because the ID does not match the expected sequence ID to be loaded. |
| Crc_Error | 5 | The computed CRC of the sequence does not match the stored CRC in its header so cannot be run. |
| Length_Error | 6 | The received sequence memory region is too small to hold the size of the sequence specified in the header. |
| Load_Error | 7 | The load of the sequence into the engine failed. This is a generic error that is clarified by an error type provided by the engine. |
| Dropped | 8 | The operation could not be performed because it was dropped from a full queue. |

### Data_Product_Enums.Fetch_Status.E:

This status denotes whether a data product fetch was successful.

Table 49: Fetch_Status Literals:

| **Name** | **Value** | **Description** |
|---|---|---|
| Success | 0 | The data product was returned successfully. |
| Not_Available | 1 | No data product is yet available for the provided id. |
| Id_Out_Of_Range | 2 | The data product id was out of range. |

### Seq_Enums.Seq_Engine_State.E:

The set of states that the sequence engine can be in.

Table 50: Seq_Engine_State Literals:

| **Name** | **Value** | **Description** |
|---|---|---|
| Uninitialized | 0 | The engine has not been initialized. Requires both a source id and an engine id. |
| Inactive | 1 | The engine does not have a running sequence loaded. |
| Reserved | 2 | Expecting a sequence to get loaded, but no sequence is currently loaded. |
| Active | 3 | The engine is currently in use and actively running a sequence. |
| Waiting | 4 | The sequence loaded in this engine is currently waiting on a time value. |
| Engine_Error | 5 | The engine has encountered an error of some sort and requires corrective action. |

### Seq_Enums.Seq_Runtime_State.E:

The set of states that a sequence runtime may be in.

Table 51: Seq_Runtime_State Literals:

| Name | Value | Description |
|---|---|---|
| Unloaded | 0 | No sequence is currently loaded in the sequence runner. |
| Ready | 1 | The sequence can safely be executed without any external work.<br>This state happens after being awoken from sleeping, or after a successful load, etc. |
| Done | 2 | The loaded sequence has successfully finished executing and will not execute until another sequence is loaded (or reloaded). |
| Wait_Relative | 3 | The sequence is waiting on a relative time value. |
| Wait_Absolute | 4 | The sequence is waiting on an absolute time value. |
| Wait_Command | 5 | The sequence is asking to send a command (could also be waiting on a command response). |
| Wait_Telemetry_Set | 6 | The sequence is waiting for a piece of telemetry, no timeout, and does not care about the value. |
| Wait_Telemetry_Value | 7 | The sequence is waiting for a piece of telemetry to be a certain value. Also waiting on an absolute timeout value. |
| Wait_Telemetry_Relative | 8 | The sequence is waiting for a piece of telemetry to be a certain value. Also waiting on a relative timeout value. |
| Telemetry_Set | 9 | The sequence has received valid telemetry and should check act upon it's value. |
| Timeout | 10 | This is a non-blocking state the means the sequence runtime has a wait that timed out. |
| Kill_Engine | 11 | The sequence is requesting to kill a range of engines. |
| Wait_Load_New_Seq_Overwrite | 12 | The current sequence has asked for a sequence to be loaded in the same engine as the current sequence. This will overwrite the current sequence. |
| Wait_Load_New_Sub_Seq | 13 | The current sequence has asked for a sequence to be loaded as a subsequence. When the subsequence finishes execution, control will be returned to the calling sequence. |
| Wait_Load_New_Seq_Elsewhere | 14 | The current sequence has asked for a sequence to be loaded into another engine. |
| Print | 15 | The current sequence has asked for a print statement to be issued. |

| | | |
|---|---|---|
| Error | 16 | The current sequence has encountered a known error. |

### Seq_Enums.Seq_Error.E:

The set of error states that a single sequence could be in.

Table 52: Seq_Error Literals:

| **Name** | **Value** | **Description** |
|---|---|---|
| None | 0 | This means that the sequence is currently not in an error state. |
| Parse | 1 | An instruction parse error occurred. This can happen when an instruction has a constrained field (i.e. eval operations, fetch, store, etc.) and the read value does not conform to the constraint. |
| Opcode | 2 | This occurs when an instruction opcode is read and it is not within the set of valid opcodes. Potential causes include misaligned memory, invalid jumps, compiler versions, bit corruption, etc. |
| Command_Parse | 3 | This occurs when there is an issue deserializing the bytes of an in-sequence command. If it failed for any reason, or if it read a different number of bytes than the instruction said it should. |
| Command_Length | 4 | This occurs when a sequence attempts to deserialize a command, but reads off the end of the sequence. |
| Command_Fail | 5 | Something upstream failed to send or resolve a command. Will not receive a command. |
| Update_Bit_Pattern | 6 | Something went wrong while parsing an update bit pattern. |
| Command_Argument | 7 | Something went wrong while updating a command bit pattern argument. |
| Telemetry_Fail | 8 | Something about the requested telemetry failed upstream. Will not receive telemetry. |
| Variable | 9 | This occurs when seq attempts to read or store a variable in an invalid spot. Seq maintains variables in a variable map and four internal variables, either of these could be miss-indexed. |
| Jump | 10 | This occurs if a sequence position jump is outside of the range of the sequence. All jumps are validated internally before they occur. |
| Cast | 11 | This occurs if a sequence attempts to cast one variable to another and it fails (i.e. Cast_F_To_U -1.0 -> ?). |
| Limit | 12 | This occurs when a sequence has executed too many instructions in a row without blocking (instruction limit is mission configurable, and should be relatively high). |

| | | |
|---|---|---|
| Eval | 13 | An exception occurred while performing an arithmetic, bitwise, or logical operation on two values. |
| Float_Value | 14 | The runtime attempted to read an internal as a float, but the float was read as NaN/inf/-inf/some invalid value. |
| Execute | 15 | The sequence was forced into an error because it finished execution in an invalid state. |
| Wait | 16 | The sequence is attempting to wait on an absolute value (seconds) that is in the past. |
| Load | 17 | The engine encountered an error while trying to load a sequence. |
| Spawn | 18 | The engine attempted to spawn a sequence on top of itself. Since 'start' exists, this was determined to be erroneous. |
| Load_Header | 19 | The sequence the was being loaded was not long enough to contain a valid header. |
| Load_Length | 20 | The sequence header and the memory region length are in disagreement. |
| Invalid_Op | 21 | The sequence attempted to perform an operation on a type that does not have that operation defined. |
| Kill | 22 | The sequence tried to execute a kill opcode with invalid parameters. |
| Recursion | 23 | The sequence has exceeded its in-component recursion limit and is not executable. |
| Command_Timeout | 24 | A command response was not received within the timeout limit. |
| Load_Timeout | 25 | A sequence load was not received within the timeout limit. |
| Telemetry_Timeout | 26 | A telemetry value was not received within the timeout limit. |
| Unimplemented | 27 | The opcode found has not been implemented in this runtime. |

## Seq_Enums.Seq_Print_Type.E:

The type of message associated with a print statement.

Table 53: Seq_Print_Type Literals:

| Name | Value | Description |
|---|---|---|
| Debug | 0 | A debug print statement. |
| Info | 1 | An informational print statement. |
| Critical | 2 | A critical print statement. |
| Error | 3 | An error print statement. |