Ccsds Subpacket Extractor

Component Design Document

1 Description

This component extracts CCSDS formatted subpackets from a larger CCSDS formatted packet. Incoming CCSDS packets are assumed to contain smaller CCSDS subpackets in their data section. Static offsets can be provided at initialization to skip bytes at the beginning or end of the incoming packet data. A maximum number of subpackets to extract can also be provided during initialization. If extracting a subpacket fails because the subpacket length is too large, the bytes are dropped and reported as an error packet. Note that this component can receive packets either synchronously, or asynchronously and can be made either active or passive depending on the desired use case. Different use cases are presented in the Design section.

2 Requirements

The requirements for the CCSDS Subpacket Extractor component are specified below.

1. The component shall extract CCSDS subpackets from a received CCSDS packet.
2. The component shall drop unsuccessfully extracted CCSDS subpackets and report the occurrence in an event.
3. The component shall package and send unsuccessfully extracted CCSDS subpackets as error packets.

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** - *either*
- **Number of Connectors** - 6
- **Number of Invokee Connectors** - 2
- **Number of Invoker Connectors** - 4
- **Number of Generic Connectors** - *None*
- **Number of Generic Types** - *None*
- **Number of Unconstrained Arrayed Connectors** - *None*
- **Number of Commands** - *None*
- **Number of Parameters** - *None*
- **Number of Events** - 4

- **Number of Faults** - *None*
- **Number of Data Products** - *None*
- **Number of Data Dependencies** - *None*
- **Number of Packets** - 1

3.2 Diagram

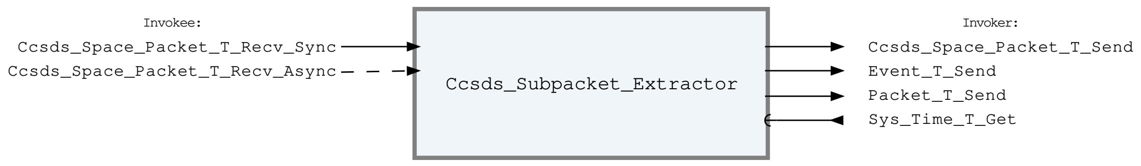
Invokee:
Ccsds_Space_Packet_T_Recv_Sync
Ccsds_Space_Packet_T_Recv_Async
Ccsds_Subpacket_Extractor
Invoker:
Ccsds_Space_Packet_T_Send
Event_T_Send
Packet_T_Send
Sys_Time_T_Get

Figure 1: Ccsds Subpacket Extractor component diagram.

The CCSDS Subpacket Extractor can be used in a variety of different ways, each of which supports a different execution context.

The following diagram shows the most common way that the CCSDS Subpacket Extractor component might operate in the context of an assembly.

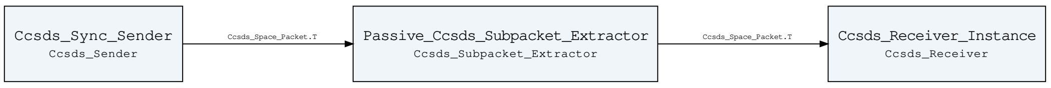

Figure 2: Example usage of a passive CCSDS Subpacket Extractor which extracts subpackets on the thread of its caller, and passes them along to a receiving component.

In the above context diagram the CCSDS Subpacket Extractor is made passive. This means that any CCSDS packet passed to it will be subpacket extracted on the thread of the calling component, since the calling component uses the `Ccsds_Space_Packet_Recv_Sync` connector.

Sometimes it is desirable to do the extracting on a different thread of execution to decouple this processing from work going on upstream of the component. The most obvious way to accomplish this is to make the CCSDS Subpacket Extractor component active, so it has its own thread of execution. The context diagram below shows this setup.

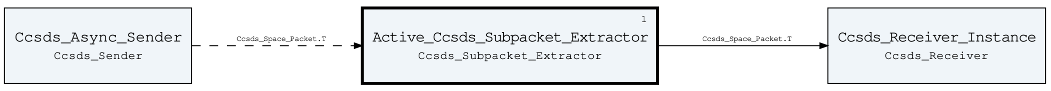

Figure 3: Example usage of an active CCSDS Subpacket Extractor which extracts any subpackets from packets found on its queue using its own internal thread of execution.

In this case, the calling component passes CCSDS packets to the CCSDS Subpacket Extractor via the `Ccsds_Space_Packet_Recv_Async` connector, which puts the CCSDS packets on the CCSDS Subpacket Extractor's internal queue. When the CCSDS Subpacket Extractor is given execution time, it pops the CCSDS packets off of the queue and performs extraction.

It is also possible to achieve both synchronous and asynchronous routing using the same CCSDS Subpacket Extractor component. This might be useful if you have a normal data path which takes the asynchronous path, but a time critical data path which requires synchronous extraction. This might be the case when decoding time critical packets related to control or fault protection. The following context diagram shows this case.

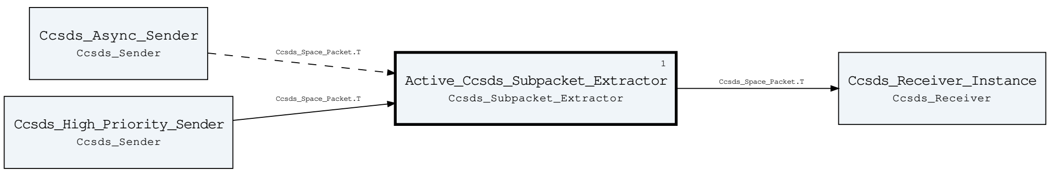

Figure 4: Example usage of a CCSDS Subpacket Extractor which extracts subpackets on the thread of its caller or its own thread depending on which connector is invoked.

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: Ccsds Subpacket Extractor Invokee Connectors

Name	Kind	Type	Return_Type	Count
Ccsds_Space_Packet_T_Recv_Sync	recv_sync	Ccsds_Space_Packet.T	-	1
Ccsds_Space_Packet_T_Recv_Async	recv_async	Ccsds_Space_Packet.T	-	1

Connector Descriptions:

- **Ccsds_Space_Packet_T_Recv_Sync** - The synchronous ccsds packet receive connector.
- **Ccsds_Space_Packet_T_Recv_Async** - The asynchronous ccsds packet receive connector.

3.3.2 Internal Queue

This component contains an internal first-in-first-out (FIFO) queue to handle asynchronous messages. This queue is sized at initialization as a configurable number of bytes. Determining the size of the component queue can be difficult. The following table lists the connectors that will put asynchronous messages onto the queue, and the maximum sizes of each of those messages on the queue. Note that each message put onto the queue also incurs an overhead on the queue of 5 additional bytes, which is included in the max message size below:

Table 2: Ccsds Subpacket Extractor Asynchronous Connectors

Name	Type	Max Size (bytes)
Ccsds_Space_Packet_T_Recv_Async	Ccsds_Space_Packet.T	1285

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: Ccsds Subpacket Extractor Invoker Connectors

Name	Kind	Type	Return_Type	Count
Ccsds_Space_ Packet_T_Send	send	Ccsds_Space_ Packet.T	-	1
Event_T_Send	send	Event.T	-	1
Packet_T_Send	send	Packet.T	-	1
Sys_Time_T_Get	get	-	Sys_Time.T	1

Connector Descriptions:

- **`Ccsds_Space_Packet_T_Send`** - The ccsds packet send connector.
- **`Event_T_Send`** - Events are sent out of this connector.
- **`Packet_T_Send`** - Error packets are sent out of this connector.
- **`Sys_Time_T_Get`** - The system time is retrieved via this connector.

3.4 Initialization

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

3.4.1 Component Instantiation

This component contains no instantiation parameters in its discriminant.

3.4.2 Component Base Initialization

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

Table 4: Ccsds Subpacket Extractor Base Initialization Parameters

Name	Type
Queue_Size	Natural

Parameter Descriptions:

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

3.4.3 Component Set ID Bases

This component contains commands, events, packets, faults, or data products that require a base identifier to be set at initialization. The `set_Id_Bases` procedure must be called with the following parameters:

Table 5: Ccsds Subpacket Extractor Set Id Bases Parameters

Name	Type
Event_Id_Base	Event_Types.Event_Id_Base
Packet_Id_Base	Packet_Types.Packet_Id_Base

Parameter Descriptions:

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

3.4.4 Component Map Data Dependencies

This component contains no data dependencies.

3.4.5 Component Implementation Initialization

The calling of this implementation class initialization procedure is mandatory. The component extracts CCSDS subpackets from the data section of a larger CCSDS packet. The init function allows the component to ignore the first, Start_Offset, or last, Stop_Offset, number of bytes during extraction. This might be useful to ignore a leading secondary header or a trailing checksum. The `init` subprogram requires the following parameters:

Table 6: Ccsds Subpacket Extractor Implementation Initialization Parameters

Name	**Type**	**Default Value**
`Start_Offset`	`Natural`	`0`
`Stop_Offset`	`Natural`	`0`
`Max_Subpackets_To_Extract`	`Integer`	`-1`

Parameter Descriptions:

- **`Start_Offset`** - The number of bytes past the primary CCSDS header to start extracting subpackets from.
- **`Stop_Offset`** - The number of bytes at the end of CCSDS packet to not attempt to extract subpackets from. This value should be used to ignore Stop_Offset number of bytes at the end of a packet.
- **`Max_Subpackets_To_Extract`** - The maximum number of subpackets to attempt to extract from an incoming packet. A negative number indicates that there is no upper limit to the amount of subpackets that can be extracted. A value of zero disables any subpacketization, which might be useful to disable this component during testing.

3.5 Events

Below is a list of the events for the Ccsds Subpacket Extractor component.

Table 7: Ccsds Subpacket Extractor Events

Local ID	**Event Name**	**Parameter Type**
0	`Invalid_Received_Packet_Length`	`Invalid_Packet_Length.T`
1	`Invalid_Extracted_Packet_Length`	`Invalid_Packet_Length.T`
2	`Dropped_Trailing_Bytes`	`Packed_Byte.T`
3	`Dropped_Packet`	`Ccsds_Primary_Header.T`

Event Descriptions:

- **`Invalid_Received_Packet_Length`** - A packet was received with a length that is too large or too small.
- **`Invalid_Extracted_Packet_Length`** - A packet was extracted with a length that is too large.
- **`Dropped_Trailing_Bytes`** - Some remaining bytes were found at the end of a packet that are too small to be a CCSDS packet.
- **`Dropped_Packet`** - The component's queue overflowed and a packet with the following header was dropped.

3.6 Packets

Packets for the CCSDS Subpacket Extractor component.

Table 8: Ccsds Subpacket Extractor Packets

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

Packet Descriptions:

- **`Error_Packet`** - This packet contains a CCSDS packet that was dropped due to error.

4 Unit Tests

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

4.1 *Ccsds_Subpacket_Extractor_Tests* Test Suite

This is a unit test suite for the CCSDS Subpacket Extractor.

Test Descriptions:

- **`Nominal_Extraction`** - This unit test tests the nominal subpacket extraction from a larger CCSDS packet.
- **`Test_Invalid_Length`** - This unit test tests that the proper events are sent out when a CCSDS packet with an incorrect size is received by the component.
- **`Test_Invalid_Subpacket_Length`** - This unit test tests that the proper events are sent out when a CCSDS packet with an incorrect size is extracted by the component.
- **`Test_Remaining_Bytes`** - This unit test tests that the proper events are sent out when a CCSDS packet with left over bytes is encountered.
- **`Test_Dropped_Packet`** - This unit test tests that the proper events are sent out when a packet is dropped due to an overflowed queue.
- **`Test_Offsets`** - This unit test tests the component with nonzero start and stop offsets for CCSDS extraction.
- **`Test_Max_Subpackets_To_Extract`** - This unit test tests the component with a zero and positive Max_Subpackets_To_Extract configuration and looks for appropriate behavior.

5 Appendix

5.1 Packed Types

The following section outlines any complex data types used in the component in alphabetical order. This includes packed records and packed arrays that might be used as connector types, command arguments, event parameters, etc..

Ccsds_Primary_Header.T:

Record for the CCSDS Packet Primary Header *Preamble (inline Ada definitions):*

```ada
subtype Three_Bit_Version_Type is Interfaces.Unsigned_8 range 0 .. 7;
type Ccsds_Apid_Type is mod 2**11;
type Ccsds_Sequence_Count_Type is mod 2**14;
```

Table 9: Ccsds_Primary_Header Packed Record : 48 bits

Name	**Type**	**Range**	**Size (Bits)**	**Start Bit**	**End Bit**
`Version`	`Three_Bit_Version_Type`	`0 to 7`	3	0	2

`Packet_Type`	`Ccsds_Enums.Ccsds_Packet_Type.E`	`0 => Telemetry`
		`1 => Telecommand`
			1	3	3

`Secondary_Header`	`Ccsds_Enums.Ccsds_Secondary_Header_Indicator.E`	`0 => Secondary_Header_Not_Present`
		`1 => Secondary_Header_Present`
			1	4	4

`Apid`	`Ccsds_Apid_Type`	`0 to 2047`	11	5	15

`Sequence_Flag`	`Ccsds_Enums.Ccsds_Sequence_Flag.E`	`0 => Continuationsegment`
		`1 => Firstsegment`
		`2 => Lastsegment`
		`3 => Unsegmented`
			2	16	17

`Sequence_Count`	`Ccsds_Sequence_Count_Type`	`0 to 16383`	14	18	31
`Packet_Length`	`Interfaces.Unsigned_16`	`0 to 65535`	16	32	47

Field Descriptions:

- **`Version`** - Packet Version Number
- **`Packet_Type`** - Packet Type
- **`Secondary_Header`** - Does packet have CCSDS secondary header
- **`Apid`** - Application process identifier
- **`Sequence_Flag`** - Sequence Flag
- **`Sequence_Count`** - Packet Sequence Count
- **`Packet_Length`** - This is the packet data length. One added to this number corresponds to the number of bytes included in the data section of the CCSDS Space Packet.

Ccsds_Space_Packet.T:

Record for the CCSDS Space Packet *Preamble (inline Ada definitions):*

```
use Basic_Types;
subtype Ccsds_Data_Type is Byte_Array (0 ..
↪  Configuration.Ccsds_Packet_Buffer_Size - 1);
```

Table 10: Ccsds_Space_Packet Packed Record : 10240 bits *(maximum)*

Name	Type	Range	Size (Bits)	Start Bit	End Bit	Variable Length
Header	Ccsds_Primary_Header.T	-	48	0	47	-
Data	Ccsds_Data_Type	-	10192	48	10239	Header.Packet_Length

Field Descriptions:

- **Header** - The CCSDS Primary Header
- **Data** - User Data Field

Event.T:

Generic event packet for holding arbitrary events

Table 11: 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 12: 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_Packet_Length.T:

A packed record which holds data related to an invalid command packet length.

Table 13: Invalid_Packet_Length Packed Record : 112 bits

Name	**Type**	**Range**	**Size (Bits)**	**Start Bit**	**End Bit**
Ccsds_Header	Ccsds_ Primary_ Header.T	-	48	0	47
Length	Integer	-2147483648 to 2147483647	32	48	79
Length_Bound	Integer	-2147483648 to 2147483647	32	80	111

Field Descriptions:

- **Ccsds_Header** - The packet identifier
- **Length** - The packet length
- **Length_Bound** - The packet length bound that the length failed to meet.

Packed_Byte.T:

Single component record for holding a byte

Table 14: Packed_Byte Packed Record : 8 bits

Name	**Type**	**Range**	**Size (Bits)**	**Start Bit**	**End Bit**
Value	Basic_Types.Byte	0 to 255	8	0	7

Field Descriptions:

- **Value** - The byte

Packet.T:

Generic packet for holding arbitrary data

Table 15: 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 16: Packet_Header Packed Record : 112 bits

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

Field Descriptions:

- **`Time`** - The timestamp for the packet item.
- **`Id`** - The packet identifier
- **`Sequence_Count`** - Packet Sequence Count
- **`Buffer_Length`** - The number of bytes used in the packet buffer

Sys_Time.T:

A record which holds a time stamp using GPS format including seconds and subseconds since epoch (1-5-1980 to 1-6-1980 midnight).

Table 17: Sys_Time Packed Record : 64 bits

Name	**Type**	**Range**	**Size (Bits)**	**Start Bit**	**End Bit**
Seconds	Interfaces. Unsigned_32	0 to 4294967295	32	0	31
Subseconds	Interfaces. Unsigned_32	0 to 4294967295	32	32	63

Field Descriptions:

- **`Seconds`** - The number of seconds elapsed since epoch.
- **`Subseconds`** - The number of 1/(2^32) sub-seconds.

5.2 Enumerations

The following section outlines any enumerations used in the component.

Ccsds_Enums.Ccsds_Packet_Type.E:

This single bit is used to identify that this is a Telecommand Packet or a Telemetry Packet. A Telemetry Packet has this bit set to value 0; therefore, for all Telecommand Packets Bit 3 shall be set to value 1.

Table 18: Ccsds_Packet_Type Literals:

Name	**Value**	**Description**
`Telemetry`	0	`Indicates a telemetry packet`
`Telecommand`	1	`Indicates a telecommand packet`

Ccsds_Enums.Ccsds_Secondary_Header_Indicator.E:

This one bit flag signals the presence (Bit 4 = 1) or absence (Bit 4 = 0) of a Secondary Header data structure within the packet.

Table 19: Ccsds_Secondary_Header_Indicator Literals:

Name	**Value**	**Description**
`Secondary_Header_Not_Present`	0	`Indicates that the secondary header is not present within the packet`
`Secondary_Header_Present`	1	`Indicates that the secondary header is present within the packet`

Ccsds_Enums.Ccsds_Sequence_Flag.E:

This flag provides a method for defining whether this packet is a first, last, or intermediate component of a higher layer data structure.

Table 20: Ccsds_Sequence_Flag Literals:

Name	**Value**	**Description**
`Continuationsegment`	0	`Continuation component of higher data structure`
`Firstsegment`	1	`First component of higher data structure`
`Lastsegment`	2	`Last component of higher data structure`
`Unsegmented`	3	`Standalone packet`