Ccsds Product Extractor

Component Design Document

1 Description

The product extractor is a component that extracts data from an incoming packet which it then creates into a data product and sends as its own data product for other component use. This is performed by using a list of types that include the offset and the corresponding APID to know which packets to extract from. The data is verified against the respective type at which point it will either send it on or create an event indicating there was an error. All of this information is derived from a user defined YAML input model that contains the information for each data product and the residing packet. See the generator documentation for more information.

2 Requirements

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

1. The component shall receive CCSDS packets and extract a data type if it is part of the initial list of products.
2. The component shall check the type of the extracted product and verify that it is valid.
3. The component shall forward data products when extracted and verified.
4. The component shall send errors when the extraction is invalid or out of the range of the packet.

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** - *passive*
- **Number of Connectors** - 4
- **Number of Invokee Connectors** - 1
- **Number of Invoker Connectors** - 3
- **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** - 2

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

3.2 Diagram

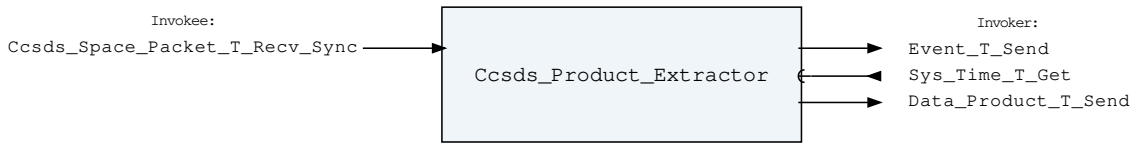

Figure 1: Ccsds Product Extractor component diagram.

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: Ccsds Product Extractor Invokee Connectors

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

Connector Descriptions:

- **Ccsds_Space_Packet_T_Recv_Sync** - The connector that will receive the CCSDS space packets and extract data products if necessary

3.3.2 Invoker Connectors

The following is a list of the component's *invoker* connectors:

Table 2: Ccsds Product Extractor Invoker Connectors

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

Connector Descriptions:

- **Event_T_Send** - The Event connector for sending events
- **Sys_Time_T_Get** - The system time is retrieved via this connector.
- **Data_Product_T_Send** - The connector for data products

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 contains no base class initialization, meaning there is no `init_Base` subprogram for this component.

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 3: Ccsds Product Extractor Set Id Bases Parameters

Name	Type
`Data_Product_Id_Base`	`Data_Product_Types.Data_Product_Id_Base`
`Event_Id_Base`	`Event_Types.Event_Id_Base`

Parameter Descriptions:

- **`Data_Product_Id_Base`** - The value at which the component's data product identifiers begin.
- **`Event_Id_Base`** - The value at which the component's event identifiers begin.

3.5.4 Component Map Data Dependencies

This component contains no data dependencies.

3.5.5 Component Implementation Initialization

The calling of this implementation class initialization procedure is mandatory. The component achieves implementation class initialization using the `init` subprogram. The `init` subprogram requires the following parameters:

Table 4: Ccsds Product Extractor Implementation Initialization Parameters

Name	Type	Default Value
`Data_Product_Extraction_List`	`Product_Extractor_Types.Extracted_Product_List_Access`	*None provided*

Parameter Descriptions:

- **`Data_Product_Extraction_List`** - The list of data products that will be extracted from packets.

3.6 Commands

The Ccsds Product Extractor component has no commands.

3.7 Parameters

The Ccsds Product Extractor component has no parameters.

3.8 Events

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

Table 5: Ccsds Product Extractor Events

Local ID	Event Name	Parameter Type
0	Invalid_Extracted_Product_Data	Invalid_Product_Data.T
1	Invalid_Extracted_Product_Length	Invalid_Product_Length.T

Event Descriptions:

- **Invalid_Extracted_Product_Data** - Event that is issued when the defined extracted product does not match the data that was read.
- **Invalid_Extracted_Product_Length** - The length and offset of the extracted product exceeded the length of the incoming packet.

3.9 Data Products

Data products for the Ccsds Product Extractor component.

Table 6: Ccsds Product Extractor Data Products

Local ID	Data Product Name	Type
0x0000 (0)	Dummy	Packed_Byte.T

Data Product Descriptions:

- **Dummy** - A dummy data product since this component doesn't have its own data products, this provides a base to start from. This will be removed and replaced with the extracted products that the user defines in the extracted_products YAML file.

3.10 Data Dependencies

The Ccsds Product Extractor component has no data dependencies.

3.11 Packets

The Ccsds Product Extractor component has no packets.

3.12 Faults

The Ccsds Product Extractor component has no faults.

4 Unit Tests

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

4.1 *Ccsds_Product_Extractor_Tests* Test Suite

This is a unit test suite for the Ccsds Product Extractor component

Test Descriptions:

- **`Test_Received_Data_Product_Packet`** - This unit test is used to test the logic of receiving a packet that contains a data product that needs to be extracted

5 Appendix

5.1 Preamble

This component contains no preamble code.

5.2 Packed Types

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

Ccsds_Primary_Header.T:

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

```
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 7: 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 8: 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

Data_Product.T:

Generic data product packet for holding arbitrary data types

Table 9: Data_Product Packed Record : 344 bits *(maximum)*

Name	Type	Range	Size (Bits)	Start Bit	End Bit	Variable Length
Header	Data_Product_ Header.T	-	88	0	87	-
Buffer	Data_Product_ Types.Data_ Product_ Buffer_Type	-	256	88	343	Header.Buffer_ Length

Field Descriptions:

- **Header** - The data product header
- **Buffer** - A buffer that contains the data product type

Data_Product_Header.T:

Generic data_product packet for holding arbitrary data_product types

Table 10: Data_Product_Header Packed Record : 88 bits

Name	Type	Range	Size (Bits)	Start Bit	End Bit
Time	Sys_Time.T	-	64	0	63
Id	Data_Product_Types. Data_Product_Id	0 to 65535	16	64	79
Buffer_Length	Data_Product_ Types.Data_Product_ Buffer_Length_Type	0 to 32	8	80	87

Field Descriptions:

- **Time** - The timestamp for the data product item.
- **Id** - The data product identifier
- **Buffer_Length** - The number of bytes used in the data product buffer

Event.T:

Generic event packet for holding arbitrary events

Table 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_Product_Data.T:

This record contains information for an event when a product is extracted from a CCSDS packet, but the data was invalid for the type.

Table 13: Invalid_Product_Data Packed Record : 112 bits

Name	Type	Range	Size (Bits)	Start Bit	End Bit
Id	Data_Product_Types.Data_Product_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 data product Id that was attempted to be extracted.
- **Errant_Field_Number** - The field that was invalid.
- **Errant_Field** - A polymorphic type containing the bad field data.

Invalid_Product_Length.T:

This record contains information for an event when a product could not be extracted from a packet because the offset and length of the data type exceeded the length of the packet.

Table 14: Invalid_Product_Length Packed Record : 48 bits

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

Apid	Ccsds_Primary_ Header.Ccsds_Apid_ Type	0 to 2047	16	16	31
Length	Interfaces. Unsigned_16	0 to 65535	16	32	47

Field Descriptions:
- **Id** - The data product Id that was attempted to be extracted.
- **Apid** - The Apid of the packet that the data product could not be extracted from
- **Length** - Length of the packet that failed due to the offset exceeding the length of the packet

Packed_Byte.T:

Single component record for holding a byte

Table 15: 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

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 16: 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.

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 17: 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 18: 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 19: 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