Product Packetizer

Component Design Document

1 Description

The product packetizer requests data products from an external component and packetizes them into packets at a configurable rate. The packets that this component produces are configured via an autocoded table provided at instantiation.

2 Requirements

The requirements for the Product Packetizer component are specified below.

1. The component shall produce packets that contain a statically defined list of data product values and data product timestamps.
2. The component shall produce packets at a periodic rate.
3. The component shall modify packet rates by command.
4. The component shall provide a command that produces a packet of a given ID upon request.

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** - 7
- **Number of Invokee Connectors** - 2
- **Number of Invoker Connectors** - 5
- **Number of Generic Connectors** - *None*
- **Number of Generic Types** - *None*
- **Number of Unconstrained Arrayed Connectors** - *None*
- **Number of Commands** - 5
- **Number of Parameters** - *None*
- **Number of Events** - 10
- **Number of Faults** - *None*
- **Number of Data Products** - *None*
- **Number of Data Dependencies** - *None*

- **Number of Packets** - 1

3.2 Diagram

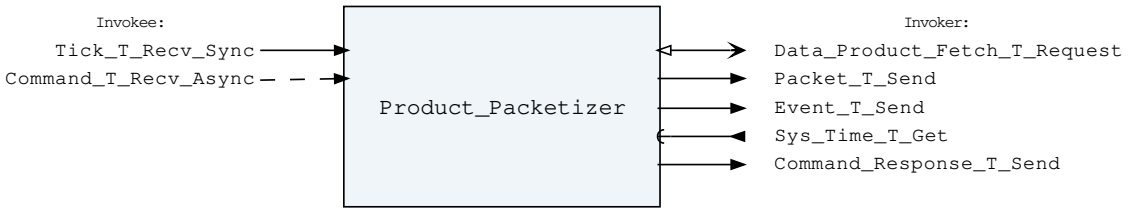

Figure 1: Product Packetizer component diagram.

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: Product Packetizer Invokee Connectors

Name	Kind	Type	Return_Type	Count
Tick_T_Recv_Sync	recv_sync	Tick.T	-	1
Command_T_Recv_Async	recv_async	Command.T	-	1

Connector Descriptions:

- **Tick_T_Recv_Sync** - This is the base tick for the component. It should be received at least as fast as the maximum desired product creation frequency.
- **Command_T_Recv_Async** - This is the command receive connector, used for configuring the packetizer during runtime.

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: Product Packetizer Asynchronous Connectors

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

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: Product Packetizer Invoker Connectors

Name	Kind	Type	Return_Type	Count
Data_Product_ Fetch_T_Request	request	Data_Product_ Fetch.T	Data_Product_ Return.T	1
Packet_T_Send	send	Packet.T	-	1
Event_T_Send	send	Event.T	-	1
Sys_Time_T_Get	get	-	Sys_Time.T	1
Command_ Response_T_Send	send	Command_ Response.T	-	1

Connector Descriptions:

- **Data_Product_Fetch_T_Request** - Fetch a data product item from the database.
- **Packet_T_Send** - Send a packet of data products.
- **Event_T_Send** - Events are sent out of this connector.
- **Sys_Time_T_Get** - The system time is retrieved via this connector.
- **Command_Response_T_Send** - This connector is used to register and respond to the component's commands.

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 requires a list of packet descriptions which outline the packets that the component is responsible for building. This list should be provided as an autocoded output from a product_packets.yaml file. This component contains the following instantiation parameters in its discriminant:

Table 4: Product Packetizer Instantiation Parameters

Name	Type
Packet_List	Product_Packet_Types.Packet_Description_List_Access_Type

Parameter Descriptions:

- **Packet_List** - The list of packets to packetize.

3.5.2 Component Base Initialization

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

Table 5: Product Packetizer 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 6: Product Packetizer Set Id Bases Parameters

Name	Type
Command_Id_Base	Command_Types.Command_Id_Base
Event_Id_Base	Event_Types.Event_Id_Base

Parameter Descriptions:
- **Command_Id_Base** - The value at which the component's command identifiers begin.
- **Event_Id_Base** - The value at which the component's event identifiers begin.

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. This initialization function is used to initialize the roll-over value for the packetizer's internal counter. It is calculated as the largest 32-bit multiple of all the provided periods in the packet_List. This ensures that no packets are skipped or sent too often when a rollover occurs. Note, that this only guarantees expected roll-over behavior if the period of the packets are not changed during runtime via command. If this happens, then the user accepts that a rollover may cause unwanted behavior. The init subprogram requires the following parameters:

Table 7: Product Packetizer Implementation Initialization Parameters

Name	Type	Default Value
Commands_Dispatched_Per_Tick	Positive	3

Parameter Descriptions:
- **Commands_Dispatched_Per_Tick** - The number of commands executed per tick, if any are in the queue.

3.6 Commands

These are the commands for the product packetizer component.

Table 8: Product Packetizer Commands

Local ID	Command Name	Argument Type
0	Set_Packet_Period	Packet_Period.T
1	Enable_Packet	Packet_Id.T
2	Disable_Packet	Packet_Id.T
3	Send_Packet	Packet_Id.T
4	Enable_Packet_On_Change	Packet_Id.T

Command Descriptions:

- **Set_Packet_Period** - Command to change the period of packet generation for a given packet id.
- **Enable_Packet** - Command to enable the emission of a packet from the packetizer.
- **Disable_Packet** - Command to disable the emission of a packet from the packetizer.
- **Send_Packet** - Command to build specific packet and send it out on the next available tick. The packet is built and sent regardless of the packet being enabled or disabled.
- **Enable_Packet_On_Change** - Command to enable the emission of a packet from the packetizer only when data products have changed since the last emission.

3.7 Parameters

The Product Packetizer component has no parameters.

3.8 Events

Below is a list of the events for the Product Packetizer component.

Table 9: Product Packetizer Events

Local ID	Event Name	Parameter Type
0	Invalid_Packet_Id_Commanded	Invalid_Packet_Id.T
1	Packet_Enabled	Packet_Period.T
2	Packet_Disabled	Packet_Period.T
3	Packet_Enabled_On_Change	Packet_Period.T
4	Packet_Period_Set	Packet_Period.T
5	Data_Product_Missing_On_Fetch	Packet_Data_Product_Ids.T
6	Packet_Period_Item_Bad_Id	Packet_Data_Product_Ids.T
7	Data_Product_Length_Mismatch	Invalid_Data_Product_Length.T
8	Invalid_Command_Received	Invalid_Command_Info.T
9	Dropped_Command	Command_Header.T

Event Descriptions:

- **Invalid_Packet_Id_Commanded** - An invalid packet id was commanded for a given command.
- **Packet_Enabled** - A packet was enabled.
- **Packet_Disabled** - A packet was disabled.
- **Packet_Enabled_On_Change** - A packet was enabled in on-change mode.
- **Packet_Period_Set** - A packet period was set.
- **Data_Product_Missing_On_Fetch** - A data product was missing when fetched for packet insertion.

- **Packet_Period_Item_Bad_Id** - A packet period packet item could not be formed because the ID is invalid.
- **Data_Product_Length_Mismatch** - A data product was fetched but contained an unexpected length.
- **Invalid_Command_Received** - A command was received with invalid parameters.
- **Dropped_Command** - A command was dropped due to a full queue.

3.9 Data Products

The Product Packetizer component has no data products.

3.10 Data Dependencies

The Product Packetizer component has no data dependencies.

3.11 Packets

Packets for the Product Packetizer. This packet list is populated based on the product packets model provided to the Product Packetizer component at initialization.

Table 10: Product Packetizer Packets

Global ID	Packet Name	Type
0x0309 (777)	Dummy	*Undefined*

Packet Descriptions:
- **Dummy** - A placeholder packet that will be overwritten with the packet list provided by the product packets model.

3.12 Faults

The Product Packetizer component has no faults.

4 Unit Tests

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

4.1 *Tests* Test Suite

This is a unit test suite for the Product Packetizer.

Test Descriptions:
- **Test_Nominal_Packetizing** - This unit test tests the packetizing of packets in a nominal situation.
- **Test_Packet_Enable_Disable** - This unit test tests enabling and disabling a packet via command.
- **Test_Packet_Set_Period** - This unit test tests changing a packet's period by command.
- **Test_Missing_Data_Product** - This unit test tests the component's response to receiving a missing data product.

- **Test_Bad_Id_Data_Product** - This unit test tests the component's response to receiving a bad ID status in response to a data product request.
- **Test_Data_Product_Size_Mismatch** - This unit test tests the component's response to receiving a data product with an unexpected size.
- **Test_Roll_Over** - This unit test tests the component's behavior when rolling over its internal count.
- **Test_Bad_Commands** - This unit test tests the component's behavior when sending commands with bad packet ids
- **Test_Send_Packet_Command** - This unit test tests the component's ability to respond to a Send Packet command.
- **Test_Offset** - This unit test tests a packet with an offset and makes sure packets come out at the correct time.
- **Test_Padding** - This unit test tests a packet with padding.
- **Test_Zero_Period** - This unit test tests a packet with a period of zero. It should behave just like disabled.
- **Test_Full_Queue** - This unit test tests a command being dropped due to a full queue.
- **Test_Packet_Period_Items** - This unit test tests the special packet period items that can be emitted inside a product packetizer packet. This unit test also tests a special packet period item with a bad ID.

4.2 *Tests* Test Suite

This is the on-change unit test suite for the Product Packetizer. This test suite specifically tests the emit packet on-change feature of the component.

Test Descriptions:

- **Test_On_Change_Nominal** - This unit test tests the basic on-change functionality where packets are only sent when data products have changed since the last emission.
- **Test_On_Change_Used_For_On_Change_False** - This unit test tests that data products with used_for_on_change set to False do not trigger packet emission in on-change mode.
- **Test_Enable_Packet_On_Change_Command** - This unit test tests the Enable_Packet_On_Change command to dynamically switch a packet to on-change mode and verifies the correct event is emitted.
- **Test_On_Change_Multiple_Changes** - This unit test tests multiple data product changes over time to verify emission time tracking works correctly.
- **Test_On_Change_With_Period** - This unit test tests that on-change packets respect their evaluation period and only check for changes on period boundaries.

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..

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_Header.T:

Generic command header for holding arbitrary commands

Table 12: 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 13: Command_Response Packed Record : 56 bits

Name	Type	Range	Size (Bits)	Start Bit	End Bit
Source_Id	Command_ Types.Command_ Source_Id	0 to 65535	16	0	15

Registration_ Id	Command_ Types.Command_ Registration_ Id	0 to 65535	16	16	31
Command_Id	Command_Types. Command_Id	0 to 65535	16	32	47
Status	Command_Enums. Command_ Response_ Status.E	0 => Success 1 => Failure 2 => Id_Error 3 => Validation_Error 4 => Length_Error 5 => Dropped 6 => Register 7 => Register_Source	8	48	55

Field Descriptions:

- **Source_Id** - The source ID. An ID assigned to a command sending component.
- **Registration_Id** - The registration ID. An ID assigned to each registered component at initialization.
- **Command_Id** - The command ID for the command response.
- **Status** - The command execution status.

Data_Product.T:

Generic data product packet for holding arbitrary data types

Table 14: 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 15: 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 16: 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 17: 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 1 => Not_Available 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.

Event.T:

Generic event packet for holding arbitrary events

Table 18: 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 19: 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 20: Invalid_Command_Info Packed Record : 112 bits

Name	Type	Range	Size (Bits)	Start Bit	End Bit
Id	Command_Types.Command_Id	0 to 65535	16	0	15
Errant_Field_Number	Interfaces.Unsigned_32	0 to 4294967295	32	16	47
Errant_Field	Basic_Types.Poly_Type	-	64	48	111

Field Descriptions:

- **Id** - The command Id received.
- **Errant_Field_Number** - The field that was invalid. 1 is the first field, 0 means unknown field, 2**32 means that the length field of the command was invalid.
- **Errant_Field** - A polymorphic type containing the bad field data, or length when Errant_Field_Number is 2**32.

Invalid_Data_Product_Length.T:

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

Table 21: Invalid_Data_Product_Length Packed Record : 96 bits

Name	Type	Range	Size (Bits)	Start Bit	End Bit
Header	Data_Product_Header.T	-	88	0	87
Expected_Length	Data_Product_Types. Data_Product_Buffer_ Length_Type	0 to 32	8	88	95

Field Descriptions:

- **Header** - The packet identifier
- **Expected_Length** - The packet length bound that the length failed to meet.

Invalid_Packet_Id.T:

A packed record which holds a packet identifier and data product identifier

Table 22: Invalid_Packet_Id Packed Record : 32 bits

Name	Type	Range	Size (Bits)	Start Bit	End Bit
Packet_Id	Packet_Types. Packet_Id	0 to 65535	16	0	15
Command_Id	Command_Types. Command_Id	0 to 65535	16	16	31

Field Descriptions:

- **Packet_Id** - The packet identifier
- **Command_Id** - The command id

Packet.T:

Generic packet for holding arbitrary data

Table 23: 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_Data_Product_Ids.T:

A packed record which holds a packet identifier and data product identifier.

Table 24: Packet_Data_Product_Ids Packed Record : 32 bits

Name	Type	Range	Size (Bits)	Start Bit	End Bit
Packet_Id	Packet_Types. Packet_Id	0 to 65535	16	0	15
Data_Product_Id	Data_Product_Types. Data_Product_Id	0 to 65535	16	16	31

Field Descriptions:

- **Packet_Id** - The packet identifier
- **Data_Product_Id** - The data product identifier

Packet_Header.T:

Generic packet header for holding arbitrary data

Table 25: 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

Packet_Id.T:

A packed record which holds a packet identifier.

Table 26: Packet_Id Packed Record : 16 bits

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

Field Descriptions:

- **Id** - The packet identifier

Packet_Period.T:

A packed record which holds a packet identifier and period.

Table 27: Packet_Period Packed Record : 48 bits

Name	Type	Range	Size (Bits)	Start Bit	End Bit
Id	Packet_Types. Packet_Id	0 to 65535	16	0	15
Period	Natural	0 to 2147483647	32	16	47

Field Descriptions:

- **Id** - The packet identifier
- **Period** - The packet period, in ticks

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 28: 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 29: 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.

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 30: 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.

Data_Product_Enums.Fetch_Status.E:

This status denotes whether a data product fetch was successful.

Table 31: 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.