Event Packetizer

Component Design Document

1 Description

The Event Packetizer component receives events synchronously and places them into a packet. This component receives a periodic tick. A packet is sent out upon receiving a tick if 1) the component has a full packet to send or 2) a partial packet timeout has occurred and the component has a packet with at least one event in it.

2 Requirements

No requirements have been specified for this component.

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** - 3
- **Number of Invoker Connectors** - 4
- **Number of Generic Connectors** - *None*
- **Number of Generic Types** - *None*
- **Number of Unconstrained Arrayed Connectors** - *None*
- **Number of Commands** - 1
- **Number of Parameters** - *None*
- **Number of Events** - *None*
- **Number of Faults** - *None*
- **Number of Data Products** - 2
- **Number of Data Dependencies** - *None*
- **Number of Packets** - 1

3.2 Diagram

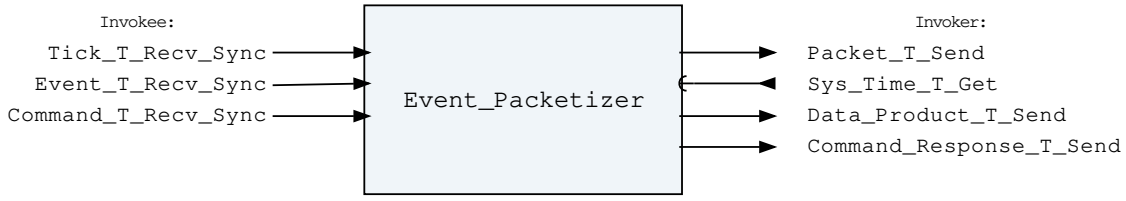

Figure 1: Event Packetizer component diagram.

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: Event Packetizer Invokee Connectors

Name	Kind	Type	Return_Type	Count
Tick_T_Recv_Sync	recv_sync	Tick.T	-	1
Event_T_Recv_Sync	recv_sync	Event.T	-	1
Command_T_Recv_Sync	recv_sync	Command.T	-	1

Connector Descriptions:

- **Tick_T_Recv_Sync** - This is the base tick for the component. Upon reception the component will send out one full packet, if a full packet is contained within the component. A partial packet will be sent out if the packet timeout occurs.
- **Event_T_Recv_Sync** - Events are received synchronously on this connector and stored into an internal packet.
- **Command_T_Recv_Sync** - This is the command receive connector.

3.3.2 Invoker Connectors

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

Table 2: Event Packetizer Invoker Connectors

Name	Kind	Type	Return_Type	Count
Packet_T_Send	send	Packet.T	-	1
Sys_Time_T_Get	get	-	Sys_Time.T	1
Data_Product_T_Send	send	Data_Product.T	-	1
Command_Response_T_Send	send	Command_Response.T	-	1

Connector Descriptions:

- **Packet_T_Send** - Send a packet of events.

- **Sys_Time_T_Get** - The system time is retrieved via this connector.
- **Data_Product_T_Send** - Data products are sent out of this connector.
- **Command_Response_T_Send** - This connector is used to register and respond to the component's commands.

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

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 3: Event Packetizer 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`
`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.
- **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 achieves implementation class initialization using the `init` subprogram. The `init` subprogram requires the following parameters:

Table 4: Event Packetizer Implementation Initialization Parameters

Name	Type	Default Value
`Num_Internal_Packets`	`Two_Or_More`	*None provided*
`Partial_Packet_Timeout`	`Natural`	*None provided*

Parameter Descriptions:

- **Num_Internal_Packets** - The number of packets that the component contains internally. This is the available buffer that the component has to store events. When all packets are exhausted, then the component begins dropping events. The component needs to be at least double buffered, meaning a minimum of two packets need to be allocated.
- **Partial_Packet_Timeout** - The number of ticks that can be received before a partial packet timeout occurs. When a partial packet timeout occurs, a packet containing at least one event is sent out, and then the timeout is reset. A value of zero passed for this parameter will disable the partial packet timeout, meaning only full packets are ever sent out of the component.

3.5 Commands

These are the commands for the event packetizer component.

Table 5: Event Packetizer Commands

Local ID	Command Name	Argument Type
0	Send_Packet	-

Command Descriptions:
- **Send_Packet** - Send a packet out next tick, unless there are no events stored within the component.

3.6 Data Products

Data products for the Event Packetizer component.

Table 6: Event Packetizer Data Products

Local ID	Data Product Name	Type
0x0000 (0)	Events_Dropped_Count	Packed_U32.T
0x0001 (1)	Bytes_Available	Packed_Natural.T

Data Product Descriptions:
- **Events_Dropped_Count** - The number of events dropped by the component.
- **Bytes_Available** - The current number of bytes available for event storage within the component.

3.7 Packets

Packets for the event packetizer

Table 7: Event Packetizer Packets

Local ID	Packet Name	Type
0x0000 (0)	Events_Packet	*Undefined*

Packet Descriptions:
- **Events_Packet** - This packet contains events as subpackets.

4 Unit Tests

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

4.1 *Event_ Packetizer_ Tests* Test Suite

This is a unit test suite for the Event Packetizer component

Test Descriptions:

- **`Test_Nominal_Packetization`** - This unit test exercises the nominal behavior of the event packetizer with 3 internal packets.
- **`Test_Partial_Packet_Timeout`** - This unit test exercises the partial packet timeout feature.
- **`Test_Partial_Packet_Timeout_Of_1`** - This unit test exercises the partial packet timeout feature with value set to 1, which means timeout should always occur.
- **`Test_Commanded_Packetization`** - This unit test tells the packetizer to packetize a partial packet via command.
- **`Test_Dropped_Events`** - This unit test exercises the behavior of the packetizer when it is so full that events begin getting dropped.
- **`Uninitialized`** - This unit test exercises the behavior of the packetizer when it is uninitialized.

5 Appendix

5.1 Preamble

This component contains the following preamble code. This is inline Ada code included in the component model that is usually used to define types or instantiate generic packages used by the component. Preamble code is inserted as the top line of the component base package specification.

```
1 subtype Two_Or_More is Positive range 2 .. Positive'Last;
```

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 8: 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 9: 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 10: 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 11: 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 12: 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 13: Event Packed Record : 344 bits *(maximum)*

Name	Type	Range	Size (Bits)	Start Bit	End Bit	Variable Length

Header	Event_Header.T	-	88	0	87	-
Param_Buffer	Event_Types. Parameter_ Buffer_Type	-	256	88	343	Header.Param_ Buffer_Length

Field Descriptions:

- **Header** - The event header
- **Param_Buffer** - A buffer that contains the event parameters

Event_Header.T:

Generic event packet for holding arbitrary events

Table 14: Event_Header Packed Record : 88 bits

Name	Type	Range	Size (Bits)	Start Bit	End Bit
Time	Sys_Time.T	-	64	0	63
Id	Event_Types.Event_ Id	0 to 65535	16	64	79
Param_Buffer_Length	Event_Types. Parameter_Buffer_ Length_Type	0 to 32	8	80	87

Field Descriptions:

- **Time** - The timestamp for the event.
- **Id** - The event identifier
- **Param_Buffer_Length** - The number of bytes used in the param buffer

Packed_Natural.T:

Single component record for holding packed Natural value.

Table 15: Packed_Natural Packed Record : 32 bits

Name	Type	Range	Size (Bits)	Start Bit	End Bit
Value	Natural	0 to 2147483647	32	0	31

Field Descriptions:

- **Value** - The 32-bit Natural Integer.

Packed_U32.T:

Single component record for holding packed unsigned 32-bit value.

Table 16: Packed_U32 Packed Record : 32 bits

Name	Type	Range	Size (Bits)	Start Bit	End Bit
Value	Interfaces. Unsigned_32	0 to 4294967295	32	0	31

Field Descriptions:

- **`Value`** - The 32-bit unsigned integer.

Packet.T:

Generic packet for holding arbitrary data

Table 17: 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 18: 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 19: 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 20: 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 21: 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.