

Limiter

Component Design Document

1 Description

This is the limiter component. This component receives a generic type of data and queues that data. It then meters the output of the data through a “send” connector at a commandable rate. The rate is set upon initialization, can be changed by command, or by parameter. The command or parameter connections may be omitted if one or both of these features are not used. The packet rate is in units of the periodic tick rate that drives the component.

2 Requirements

The requirements for the Limiter component are specified below.

1. The component shall be able to receive and queue a generic data type.
2. The component shall send queued data at a configurable periodic rate.
3. The rate at which queued data is sent shall be configurable by command.
4. The rate at which queued data is sent shall be configurable by parameter.

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** - 9
- **Number of Invokee Connectors** - 4
- **Number of Invoker Connectors** - 5
- **Number of Generic Connectors** - *None*
- **Number of Generic Types** - 2
- **Number of Unconstrained Arrayed Connectors** - *None*
- **Number of Commands** - 1
- **Number of Parameters** - 1
- **Number of Events** - 4
- **Number of Faults** - *None*
- **Number of Data Products** - 1

- **Number of Data Dependencies** - *None*
- **Number of Packets** - *None*

3.2 Diagram

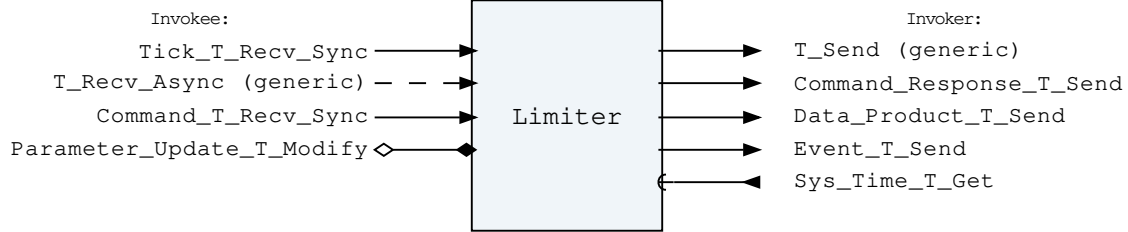


Figure 1: Limiter component diagram.

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: Limiter Invokee Connectors

Name	Kind	Type	Return_Type	Count
Tick_T_Recv_Sync	recv_sync	Tick.T	-	1
T_Recv_Async	recv_async	T (generic)	-	1
Command_T_Recv_Sync	recv_sync	Command.T	-	1
Parameter_Update_T_Modify	modify	Parameter_Update.T	-	1

Connector Descriptions:

- **Tick_T_Recv_Sync** - This is the base tick for the component.
- **T_Recv_Async** - The generic asynchronous invokee connector.
- **Command_T_Recv_Sync** - This is the command receive connector. This does not need to be connected if the command for this component will not be used.
- **Parameter_Update_T_Modify** - The parameter update connector. This does not need to be connected if the parameter for this component will not be used.

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: Limiter Asynchronous Connectors

Name	Type	Max Size (bytes)
T_Recv_Async	T (generic)	<i>Unconstrained</i>

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: Limiter Invoker Connectors

Name	Kind	Type	Return_Type	Count
T_Send	send	T (generic)	-	1
Command_Response_T_Send	send	Command_Response.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:

- **T_Send** - The generic invoker connector.
- **Command_Response_T_Send** - This connector is used to register and respond to the component's commands. This does not need to be connected if the command for this component will not be used.
- **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 Initialization

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

3.4.1 Generic Component Instantiation

The limiter is generic in that it can be instantiated to queue and meter the sending out of any type at compile time. This component contains generic formal types. These generic formal types must be instantiated with a valid actual type prior to component initialization. This is done by specifying types for the following generic formal parameters:

Table 4: Limiter Generic Formal Types

Name	Formal Type Definition
T	type T is private;
Serialized_Length	with function Serialized_Length (Src : in T; Num_Bytes_Serialized : out Natural) return Serializer_Types.Serialization_Status;

Generic Formal Type Descriptions:

- **T** - The generic type of data passed in and out of the limiter.

- **Serialized_Length** - A method that returns the serialized length of an item of type T. This is useful for serializing variable length packed types onto the queue.

3.4.2 Component Instantiation

This component contains no instantiation parameters in its discriminant.

3.4.3 Component Base Initialization

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

Table 5: Limiter 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.

3.4.4 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: Limiter 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
Command_Id_Base	Command_Types.Command_Id_Base
Parameter_Id_Base	Parameter_Types.Parameter_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.
- **Command_Id_Base** - The value at which the component's command identifiers begin.
- **Parameter_Id_Base** - The value at which the component's parameter identifiers begin.

3.4.5 Component Map Data Dependencies

This component contains no data dependencies.

3.4.6 Component Implementation Initialization

The calling of this implementation class initialization procedure is mandatory. This initialization function is used to set a threshold for the maximum number of data sends that the component will produce when a tick is received. The `init` subprogram requires the following parameters:

Table 7: Limiter Implementation Initialization Parameters

Name	Type	Default Value
Max_Sends_Per_Tick	Interfaces.Unsigned_16	<i>None provided</i>

Parameter Descriptions:

- **Max_Sends_Per_Tick** - The maximum number of sends that this component will produce when a tick is received. The component will stop producing packets if the threshold is met or when the queue is empty, whichever happens first.

3.5 Commands

These are the commands for the limiter component.

Table 8: Limiter Commands

Local ID	Command Name	Argument Type
0	Sends_Per_Tick	Packed_U16.T

Command Descriptions:

- **Sends_Per_Tick** - Set a new value for the maximum number of sends this component will produce per tick.

3.6 Parameters

Parameters for the Limiter component.

Table 9: Limiter Parameters

Local ID	Parameter Name	Type	Default Value
0x0000 (0)	Max_Sends_Per_Tick	Packed_U16.T	(Value=>1)

Parameter Descriptions:

- **Max_Sends_Per_Tick** - The maximum number of packets to send out in one tick.

3.7 Events

Below is a list of the events for the Limiter component.

Table 10: Limiter Events

Local ID	Event Name	Parameter Type
0	Max_Send_Per_Tick_Set	Packed_U16.T
1	Data_Dropped	-
2	Invalid_Command_Received	Invalid_Command_Info.T
3	Invalid_Parameter_Received	Invalid_Parameter_Info.T

Event Descriptions:

- **Max_Send_Per_Tick_Set** - A new maximum sends per tick rate has been set for the limiter.

- **Data_Dropped** - The queue for data overflowed and an incoming data was dropped.
- **Invalid_Command_Received** - A command was received with invalid parameters.
- **Invalid_Parameter_Received** - A parameter was received with invalid values.

4 Unit Tests

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

4.1 *Limiter_Tests* Test Suite

This is a unit test suite for the Limiter component

Test Descriptions:

- **Test_Nominal_Limiting** - This unit test excersizes the limiter component, making sure it behaves as expected.
- **Test_Change_Rate_Command** - This unit test excersizes the change rate command of the limiter and makes sure it works.
- **Test_Change_Rate_Parameter** - This unit test excersizes the change rate parameter of the limiter and makes sure it works.
- **Test_Queue_Overflow** - This unit test exercises that a queue overflow results in the appropriate event.
- **Test_Invalid_Command** - This unit test exercises that an invalid command throws the appropriate event.
- **Test_Invalid_Parameter** - This unit test exercises that an invalid parameter throws the appropriate event.

4.2 *Variable_Tests* Test Suite

This is unit test suite tests out the variable queueing of a generic component. The purpose of this unit test is not to test the limiter specifically, but the autocoding logic of a generic asynchronous connector of variable length type.

Test Descriptions:

- **Test_Queueing_Variable_Length** - This unit test excersizes the limiter component while queueing variable length packets.

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..

Command.T:

Generic command packet for holding arbitrary commands

Table 11: Command Packed Record : 808 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	-	768	40	807	Header.Arg_Buffer_Length

Field Descriptions:

- **Header** - The command header
- **Arg_Buffer** - A buffer to 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 96	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_Header.T:

Generic data_product packet for holding arbitrary data_product types

Table 15: 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 16: 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 17: 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 18: 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_Parameter_Info.T:

Record for holding information about an invalid parameter

Table 19: Invalid_Parameter_Info Packed Record : 112 bits

Name	Type	Range	Size (Bits)	Start Bit	End Bit
Id	Parameter_Types. Parameter_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 parameter 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 parameter was invalid.
- **Errant_Field** - A polymorphic type containing the bad field data, or length when Errant_Field_Number is 2**32.

Packed_U16.T:

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

Table 20: 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.

Parameter.T:

Generic parameter packet for holding a generic parameter

Table 21: Parameter Packed Record : 280 bits (*maximum*)

Name	Type	Range	Size (Bits)	Start Bit	End Bit	Variable Length
Header	Parameter_ Header.T	-	24	0	23	-

Buffer	Parameter_Types. Parameter_Buffer_Type	-	256	24	279	Header.Buffer_Length
--------	---	---	-----	----	-----	----------------------

Field Descriptions:

- **Header** - The parameter header
- **Buffer** - A buffer to that contains the parameter type

Parameter_Header.T:

Generic parameter header for holding arbitrary parameters

Table 22: Parameter_Header Packed Record : 24 bits

Name	Type	Range	Size (Bits)	Start Bit	End Bit
Id	Parameter_Types. Parameter_Id	0 to 65535	16	0	15
Buffer_Length	Parameter_Types. Parameter_Buffer_Length_Type	0 to 32	8	16	23

Field Descriptions:

- **Id** - The parameter identifier
- **Buffer_Length** - The number of bytes used in the parameter type buffer

Parameter_Update.T:

A record intended to be used as a provide/modify connector type for updating/fetching parameters.

Table 23: Parameter_Update Packed Record : 296 bits (*maximum*)

Name	Type	Range	Size (Bits)	Start Bit	End Bit	Variable Length
Operation	Parameter_Enums. Parameter_Operation_Type.E	0 => Stage 1 => Update 2 => Fetch	8	0	7	-
Status	Parameter_Enums. Parameter_Update_Status.E	0 => Success 1 => Id_Error 2 => Validation_Error 3 => Length_Error	8	8	15	-
Param	Parameter.T	-	280	16	295	-

Field Descriptions:

- **Operation** - The parameter operation to perform.

- **Status** - The parameter return status.
- **Param** - The parameter that has been updated or fetched.

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 24: 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 25: 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.

5.2 Enumerations

The following section outlines any enumerations used in the component.

Command_Enums.Command_Response_Status.E:

This status enumerations provides information on the success/failure of a command through the command response connector.

Table 26: 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.

Parameter_Enums.Parameter_Operation_Type.E:

This enumeration lists the different parameter operations that can be performed.

Table 27: Parameter_Operation_Type Literals:

Name	Value	Description
Stage	0	Stage the parameter.
Update	1	All parameters are staged, it is ok to update all parameters now.
Fetch	2	Fetch the parameter.

Parameter_Enums.Parameter_Update_Status.E:

This status enumerations provides information on the success/failure of a parameter operation.

Table 28: Parameter_Update_Status Literals:

Name	Value	Description
Success	0	Parameter was successfully staged.
Id_Error	1	Parameter id was not valid.
Validation_Error	2	Parameter values were not successfully validated.
Length_Error	3	Parameter length was not correct.