PID Controller

Component Design Document

1 Description

This component is a generic component for PID control that uses proportional, integral, and derivative gains. The component input is the measured and commanded positions which is used to find an error, as well as a feed-forward value to overcome friction and jitter. The component uses the error with the PID gains that are set by the user in the parameter table to perform the correct control for the particular system. Any one of the gains can be set to 0 to turn off that particular term. The component also has the ability to limit the integral term to prevent wind-up of that term and potential kickback in the physical system. There are also optional statistics for the mean, variance, and max of the error which is disabled by setting the Moving_Average_Max_Samples initialization parameter to 0. Lastly, the component also has the ability to produce diagnostics over a particular amount of time set by command, which contains the error and reference positions.

2 Requirements

These are the requirements for the PID Controller component.

1. The PID controller component shall take a measured error input to determine the P, I, and D components of the control.
2. The PID controller component shall include a limit to the integral term to avoid controller wind-up.
3. The PID controller component shall include a feed-forward term to the controller.
4. The PID controller component shall calculate the mean, variance, and max of the error over a specified sample count.
5. The PID controller component shall produce diagnostic packets with subpackets that contain the measured, reference, and the current angle.
6. The PID controller component shall output the current control error for control of hardware by other components.

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** - 3
- **Number of Invoker Connectors** - 6

- **Number of Generic Connectors** - *None*
- **Number of Generic Types** - *None*
- **Number of Unconstrained Arrayed Connectors** - *None*
- **Number of Commands** - 3
- **Number of Parameters** - 6
- **Number of Events** - 6
- **Number of Faults** - *None*
- **Number of Data Products** - 8
- **Number of Data Dependencies** - *None*
- **Number of Packets** - 1

3.2 Diagram

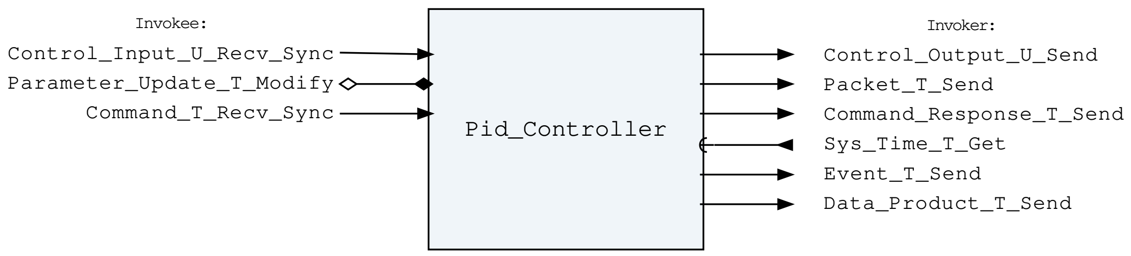

Figure 1: Pid Controller component diagram.

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: Pid Controller Invokee Connectors

Name	Kind	Type	Return_Type	Count
`Control_Input_U_Recv_Sync`	`recv_sync`	`Control_Input.U`	-	1
`Parameter_Update_T_Modify`	`modify`	`Parameter_Update.T`	-	1
`Command_T_Recv_Sync`	`recv_sync`	`Command.T`	-	1

Connector Descriptions:

- **`Control_Input_U_Recv_Sync`** - The connector for receiving the desired control location.
- **`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.
- **`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: Pid Controller Invoker Connectors

Name	Kind	Type	Return_Type	Count
Control_Output_U_Send	send	Control_Output.U	-	1
Packet_T_Send	send	Packet.T	-	1
Command_Response_T_Send	send	Command_Response.T	-	1
Sys_Time_T_Get	get	-	Sys_Time.T	1
Event_T_Send	send	Event.T	-	1
Data_Product_T_Send	send	Data_Product.T	-	1

Connector Descriptions:

- **Control_Output_U_Send** - The connector for sending the calculated PID controller output.
- **Packet_T_Send** - Packet for sending diagnostic packets.
- **Command_Response_T_Send** - This connector is used to register and respond to the component's commands.
- **Sys_Time_T_Get** - The system time is retrieved via this connector.
- **Event_T_Send** - The Event connector
- **Data_Product_T_Send** - The connector for data products

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: Pid Controller 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
Event_Id_Base	Event_Types.Event_Id_Base
Packet_Id_Base	Packet_Types.Packet_Id_Base
Parameter_Id_Base	Parameter_Types.Parameter_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.
- **`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.
- **`Parameter_Id_Base`** - The value at which the component's parameter 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: Pid Controller Implementation Initialization Parameters

Name	**Type**	**Default Value**
`Control_Frequency`	`Short_Float`	*None provided*
`Database_Update_Period`	`Unsigned_16`	*None provided*
`Moving_Average_Max_Samples`	`Natural`	*None provided*
`Moving_Average_Init_Samples`	`Integer`	`-1`

Parameter Descriptions:

- **`Control_Frequency`** - The frequency in Hz at which the PID controller is being driven. This determines the time step for the PID controller to use in the algorithm.
- **`Database_Update_Period`** - The period in which to update the data products
- **`Moving_Average_Max_Samples`** - The number of diagnostic samples to keep to perform the mean, variance, and max for the maximum duration
- **`Moving_Average_Init_Samples`** - The number of samples to initialize the object with. Must be less than the max, and is optional to set to the max with -1

3.5 Commands

These are the commands for the PID Controller component.

Table 5: Pid Controller Commands

Local ID	**Command Name**	**Argument Type**
0	`Start_Diagnostics`	`Packed_Natural_Duration.T`
1	`Set_Database_Update_Period`	`Packed_U16.T`
2	`Set_Controller_Statistic_Duration`	`Packed_Positive.T`

Command Descriptions:

- **`Start_Diagnostics`** - Set the PID controller diagnostic packet's duration to capture samples. Duration is a function of the controller frequency.
- **`Set_Database_Update_Period`** - Change the database update period, in units of the resolver acquisition period.
- **`Set_Controller_Statistic_Duration`** - Resets and changes the duration that the rolling statistics of the controller are measured, up to a max value set at compile time. This command will fail if the desired sample duration is greater than the max number of samples defined at compile time.

3.6 Parameters

The set of parameters for the gains in the pid controller

Table 6: Pid Controller Parameters

Local ID	Parameter Name	Type	Default Value
`0x0000 (0)`	`P_Gain`	`Packed_F32.T`	`(Value=>0.0)`
`0x0001 (1)`	`I_Gain`	`Packed_F32.T`	`(Value=>0.0)`
`0x0002 (2)`	`D_Gain`	`Packed_F32.T`	`(Value=>0.0)`
`0x0003 (3)`	`N_Filter`	`Packed_F32.T`	`(Value=>0.0)`
`0x0004 (4)`	`I_Min_Limit`	`Packed_F32.T`	`(Value=>-1.0*Short_Float'Large)`
`0x0005 (5)`	`I_Max_Limit`	`Packed_F32.T`	`(Value=>Short_Float'Large)`

Parameter Descriptions:

- **`P_Gain`** - The proportional gain used in the PID controller. Uses the error to determine the first step of control
- **`I_Gain`** - The integral gain used in the PID controller. Uses previous errors to help smoothly reach the desired location as well as determine overshoot and settling time.
- **`D_Gain`** - The derivative gain used in the PID controller. Determines how quickly the controller will attempt to reach the commanded position.
- **`N_Filter`** - The derivative filter used in the PID controller. Used in the control law to help dampen the derivative gain.
- **`I_Min_Limit`** - The minimum (negative direction) integral windup limit used in the PID controller. If the integrator goes below this limit then the integrator is capped at this limit. This prevents runaway integral windup in the negative direction. The negative and positive limits are separated to allow configuration of asymmetrical windup limits, which might be needed for control systems that cannot control in both directions, ie. a heater controller.
- **`I_Max_Limit`** - The maximum (positive direction) integral windup limit used in the PID controller. If the integrator goes above this limit then the integrator is capped at this limit. This prevents runaway integral windup in the positive direction. The negative and positive limits are separated to allow configuration of asymmetrical windup limits, which might be needed for control systems that cannot control in both directions, ie. a heater controller.

3.7 Events

Below is a list of the events for the Pid Controller component.

Table 7: Pid Controller Events

Local ID	Event Name	Parameter Type
0	`Invalid_Command_Received`	`Invalid_ Command_Info.T`
1	`Invalid_Parameter_Received`	`Invalid_ Parameter_Info. T`
2	`Database_Update_Period_Set`	`Packed_U16.T`
3	`Diagnostics_Started`	`Packed_Natural_ Duration.T`
4	`Set_Controller_Statistics_Duration`	`Packed_ Positive.T`
5	`Set_Controller_Statistics_Duration_Too_Large`	`Packed_ Positive.T`

Event Descriptions:

- **`Invalid_Command_Received`** - A command was received with invalid parameters.
- **`Invalid_Parameter_Received`** - Invalid parameter update
- **`Database_Update_Period_Set`** - The event to indicate that the database update period was commanded.
- **`Diagnostics_Started`** - This event indicates that the diagnostic packet request command has been sent.
- **`Set_Controller_Statistics_Duration`** - This event indicates that the command to change the duration that statistics are collected was received and changed.
- **`Set_Controller_Statistics_Duration_Too_Large`** - This event indicates that the command to change the duration that statistics are collected was received but failed to change the length.

3.8 Data Products

Data products for the pid controller component.

Table 8: Pid Controller Data Products

Local ID	Data Product Name	Type
`0x0000 (0)`	`P_Output`	`Packed_F32.T`
`0x0001 (1)`	`I_Output`	`Packed_F32.T`
`0x0002 (2)`	`D_Output`	`Packed_F32.T`
`0x0003 (3)`	`Ff_Output`	`Packed_F32.T`
`0x0004 (4)`	`Pid_Error`	`Packed_F32.T`
`0x0005 (5)`	`Pid_Error_Mean`	`Packed_F32.T`
`0x0006 (6)`	`Pid_Error_Variance`	`Packed_F32.T`
`0x0007 (7)`	`Pid_Error_Max`	`Packed_F32.T`

Data Product Descriptions:

- **`P_Output`** - The output proportional value of the last control cycle used to help determine how to get to the desired location.
- **`I_Output`** - The output integrator value of the last control cycle used to help smoothly get to the desired location as well as determine overshoot and settling time.
- **`D_Output`** - The output derivative value of the last control cycle which determines how fast the controller reaches its desired location.

- **`Ff_Output`** - The output of the last feed forward value used in the controller to overcome sources of friction.
- **`Pid_Error`** - The output of the last control cycle error calculated by the controller.
- **`Pid_Error_Mean`** - The mean value of error seen in the controller over a desired, and set data length.
- **`Pid_Error_Variance`** - The variance of the error seen in the controller over a desired, and set data length.
- **`Pid_Error_Max`** - The max error seen in the controller over a desired, and set data length.

3.9 Packets

Data products for the pid controller component.

Table 9: Pid Controller Packets

Local ID	Packet Name	Type
`0x0000 (0)`	`Pid_Controller_Diagnostic_Packet`	*Undefined*

Packet Descriptions:

- **`Pid_Controller_Diagnostic_Packet`** - The diagnostic packet that is issued based on the number of samples set by command. Samples are taken at the control rate. Includes error, reference, and current.

4 Unit Tests

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

4.1 *Pid_ Controller_ Tests* Test Suite

This is a unit test suite for the Pid Controller component

Test Descriptions:

- **`Test_Diagnostic_Packet`** - This unit test exercises starting the diagnostic packet after being commanded.
- **`Test_Update_Data_Products`** - This unit test exercises updating the Data products appropriately.
- **`Test_Database_Update_Period`** - This unit test exercises the data product update period command.
- **`Test_Pid_Controller`** - This test is a basic test to make sure that the controller
- **`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.
- **`Test_Start_Diagnostics_Command`** - This unit test exercises updating a the diagnostic samples by command.
- **`Test_Set_Controller_Statistic_Duration_Command`** - This unit test exercises updating the length of the array used to calculate statistics and thus the duration of the statistic

period.

- **`Test_Moving_Average_Unused`** - This test makes sure that if the moving_average object is unused, that no statistics come out and nothing breaks.

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 10: 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 11: 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 12: 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.

Control_Input.T:

Generic control input.

Table 13: Control_Input Packed Record : 168 bits

Name	Type	Range	Size (Bits)	Start Bit	End Bit
Time	Sys_Time.T	-	64	0	63
Measured_ Value	Short_Float	-3.40282e+38 to 3.40282e+38	32	64	95
Commanded_ Value	Short_Float	-3.40282e+38 to 3.40282e+38	32	96	127
Feed_ Forward_ Value	Short_Float	-3.40282e+38 to 3.40282e+38	32	128	159
First_ Iteration	Boolean	0 => False 1 => True	8	160	167

Field Descriptions:

- **`Time`** - Time tag saved when the data was gathered.
- **`Measured_Value`** - The current measured value of the control.

- **Commanded_Value** - The current commanded value of the control.
- **Feed_Forward_Value** - The current feed forward value for the control.
- **First_Iteration** - This variable should be set to True if this is the first iteration of a new control run. When set to true, the controller will reset its internal state, setting any accumulated derivative/integral control terms to zero. This should be done whenever the caller has switched between control modes.

Control_Output.T:

Generic control output.

Table 14: Control_Output Packed Record : 128 bits

Name	Type	Range	Size (Bits)	Start Bit	End Bit
Time	Sys_Time.T	-	64	0	63
Output_ Value	Short_Float	-3.40282e+38 to 3.40282e+38	32	64	95
Error	Short_Float	-3.40282e+38 to 3.40282e+38	32	96	127

Field Descriptions:

- **Time** - Time tag saved when the data was gathered.
- **Output_Value** - The control output value.
- **Error** - The current control error.

Data_Product.T:

Generic data product packet for holding arbitrary data types

Table 15: 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 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

Event.T:

Generic event packet for holding arbitrary events

Table 17: 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 18: 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 19: 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 20: 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_F32.T:

Single component record for holding packed 32-bit floating point number.

Table 21: Packed_F32 Packed Record : 32 bits

Name	Type	Range	Size (Bits)	Start Bit	End Bit
Value	Short_Float	-3.40282e+38 to 3.40282e+38	32	0	31

Field Descriptions:

- **Value** - The 32-bit floating point number.

Packed_Natural_Duration.T:

Single component record for holding packed Natural value that represents a duration.

Table 22: Packed_Natural_Duration Packed Record : 32 bits

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

Field Descriptions:

- **Duration** - The 32-bit Natural that represents a duration.

Packed_Positive.T:

Single component record for holding packed Positive value.

Table 23: Packed_Positive Packed Record : 32 bits

Name	Type	Range	Size (Bits)	Start Bit	End Bit
Value	Positive	1 to 2147483647	32	0	31

Field Descriptions:

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

Packed_U16.T:

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

Table 24: 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.

Packet.T:

Generic packet for holding arbitrary data

Table 25: 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 26: 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

Parameter.T:

Generic parameter packet for holding a generic parameter

Table 27: 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 that contains the parameter type

Parameter_Header.T:

Generic parameter header for holding arbitrary parameters

Table 28: 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 29: Parameter_Update Packed Record : 312 bits *(maximum)*

Name	**Type**	**Range**	**Size (Bits)**	**Start Bit**	**End Bit**	**Variable Length**
Table_Id	Parameter_ Types. Parameter_ Table_Id	0 to 65535	16	0	15	-
Operation	Parameter_ Enums. Parameter_ Operation_ Type.E	0 => Stage 1 => Update 2 => Fetch 3 => Validate	8	16	23	-
Status	Parameter_ Enums. Parameter_ Update_ Status.E	0 => Success 1 => Id_Error 2 => Validation_Error 3 => Length_Error	8	24	31	-
Param	Parameter. T	-	280	32	311	-

Field Descriptions:

- **Table_Id** - The ID for the table that contains this parameter
- **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 30: 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.

Command_Enums.Command_Response_Status.E:

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

Table 31: 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 32: 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.
Validate	3	Validate all the parameters.

Parameter_Enums.Parameter_Update_Status.E:

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

Table 33: 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.