Task Watchdog

Component Design Document

1 Description

The Task Watchdog component receives pets from components that execute in a periodic manner throughout the assembly. The receipt of a pet indicates that the component is running well and is what is referred to as a watchdog. If it detects that the component has stopped executing for some configurable time, called a limit, it will either ignore the fault, throw a warning event, or throw a fault and possibly stop servicing a downstream watchdog (usually a hardware watchdog) based on the component's configuration. The configuration is dependent on the input list generated from a yaml model for this component. The input requires a connector name, a limit for the number of ticks without a pet, the criticality of the component, and the action to take if the limit is exceeded which is one of the three described before.
In addition, the criticality of the task is also defined in the yaml model which determines if the watchdog component stops petting the downstream watchdog.

2 Requirements

The requirements for the Task Watchdog component are specified below.

1. The component will use an initial list of pet connections that have pet connectors to the component to monitor the pet for each respective component.
2. The component will produce faults as specified by the user when a pet has not been received for a specified number of ticks.
3. The component will service a downstream pet connector while all critical tasks are sending pets to the task watchdog component normally.
4. The component will be able to enable and disable watchdog by command for each petter.
5. The component will be able to change the limit value of ticks since the last pet by command for each petter.

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** - 1
- **Number of Commands** - 4
- **Number of Parameters** - *None*
- **Number of Events** - 10
- **Number of Faults** - 1
- **Number of Data Products** - 2
- **Number of Data Dependencies** - *None*
- **Number of Packets** - *None*

3.2 Diagram

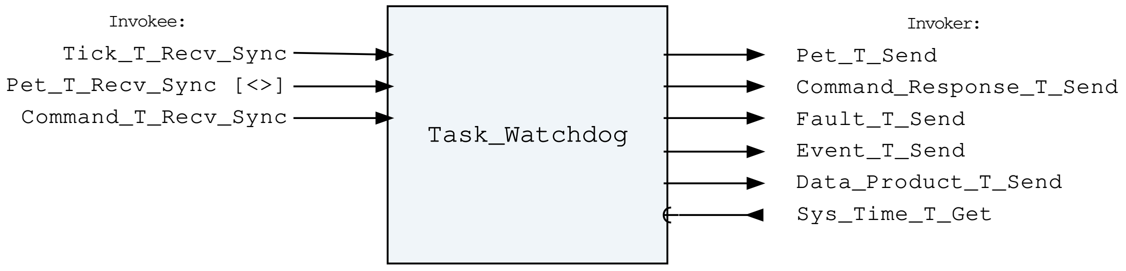

Figure 1: Task Watchdog component diagram.

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: Task Watchdog Invokee Connectors

Name	**Kind**	**Type**	**Return_Type**	**Count**
`Tick_T_Recv_Sync`	`recv_sync`	`Tick.T`	-	1
`Pet_T_Recv_Sync`	`recv_sync`	`Pet.T`	-	<>
`Command_T_Recv_Sync`	`recv_sync`	`Command.T`	-	1

Connector Descriptions:

- **`Tick_T_Recv_Sync`** - The schedule invokee connector.
- **`Pet_T_Recv_Sync`** - The arrayed pet receive connector. Upstream components call this connector to let the Task Watchdog know they are running OK.
- **`Command_T_Recv_Sync`** - The command receive connector

3.3.2 Invoker Connectors

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

Table 2: Task Watchdog Invoker Connectors

Name	Kind	Type	Return_Type	Count
Pet_T_Send	send	Pet.T	-	1
Command_Response_ T_Send	send	Command_Response. T	-	1
Fault_T_Send	send	Fault.T	-	1
Event_T_Send	send	Event.T	-	1
Data_Product_T_ Send	send	Data_Product.T	-	1
Sys_Time_T_Get	get	-	Sys_Time.T	1

Connector Descriptions:

- **Pet_T_Send** - The pet send connector. This is used to service a downstream watchdog component, usually a component which services a hardware-based watchdog.
- **Command_Response_T_Send** - This connector is used to register the components commands with the command router component.
- **Fault_T_Send** - Faults are sent on this connector.
- **Event_T_Send** - The post mortem log can be dumped using events.
- **Data_Product_T_Send** - Data products for limit values and states.
- **Sys_Time_T_Get** - The system time is retrieved via this connector.

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

Table 3: Task Watchdog Base Initialization Parameters

Name	Type
Pet_T_Recv_Sync_Count	Connector_Count_Type

Parameter Descriptions:

- **Pet_T_Recv_Sync_Count** - The size of the Pet_T_Recv_Sync invokee connector array.

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 4: Task Watchdog 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`

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.

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 5: Task Watchdog Implementation Initialization Parameters

Name	Type	Default Value
`Task_Watchdog_Entry_Init_List`	`Task_Watchdog_Types.Task_Watchdog_Init_List`	*None provided*

Parameter Descriptions:

- **`Task_Watchdog_Entry_Init_List`** - The list of components that have a watchdog to pet that need to be tracked by the task watchdog.

3.6 Commands

These are the commands for the Task Watchdog component.

Table 6: Task Watchdog Commands

Local ID	Command Name	Argument Type
0	`Enable_Watchdog_Pet_Checks`	-
1	`Disable_Watchdog_Pet_Checks`	-
2	`Set_Watchdog_Limit`	`Watchdog_Limit_Cmd.T`
3	`Set_Watchdog_Action`	`Watchdog_Action_Cmd.T`

Command Descriptions:

- **`Enable_Watchdog_Pet_Checks`** - Command to enable the watchdog component to check all connected components for incoming pets.
- **`Disable_Watchdog_Pet_Checks`** - Command to disable the watchdog component to check all connected components for incoming pets.

- **Set_Watchdog_Limit** - Set the limit value for the watchdog given an index and the new index value.
- **Set_Watchdog_Action** - Sets the action of a petter given the index of that petter and the updated action. Note that actions cannot be promoted to fault if they were not provided a fault id.

3.7 Parameters

The Task Watchdog component has no parameters.

3.8 Events

Below is a list of the events for the Task Watchdog component.

Table 7: Task Watchdog Events

Local ID	Event Name	Parameter Type
0	Watchdog_Pet_Checks_Enabled	-
1	Watchdog_Pet_Checks_Disabled	-
2	Watchdog_Limit_Set	Watchdog_Limit_Cmd.T
3	Watchdog_Action_Set	Watchdog_Action_Cmd.T
4	Watchdog_Limit_Change_Index_Out_Of_Range	Packed_Connector_Index.T
5	Watchdog_Action_Change_Index_Out_Of_Range	Packed_Connector_Index.T
6	Watchdog_Action_Change_Invalid_Transition_To_Fault	Packed_Connector_Index.T
7	Component_Exceeded_Pet_Limit	Packed_Connector_Index.T
8	Critical_Task_Not_Petting	Packed_Connector_Index.T
9	Invalid_Command_Received	Invalid_Command_Info.T

Event Descriptions:

- **Watchdog_Pet_Checks_Enabled** - Indicates a command was received to enable the checks on upstream pets.
- **Watchdog_Pet_Checks_Disabled** - Indicates a command was received to disable the checks on upstream pets.
- **Watchdog_Limit_Set** - An event to indicate that the limit was changed by command for a particular index.
- **Watchdog_Action_Set** - An event to indicate that the action was changed by command for a particular index.

- **`Watchdog_Limit_Change_Index_Out_Of_Range`** - Event indicating there was an error for the index range in the set limit command.
- **`Watchdog_Action_Change_Index_Out_Of_Range`** - Event indicating there was an error for the index range in the set action command.
- **`Watchdog_Action_Change_Invalid_Transition_To_Fault`** - Event indicating there was an error trying to set the action to fault. The petter did not have a fault declared in the model so the action cannot be set to fault.
- **`Component_Exceeded_Pet_Limit`** - Event to indicate a pet connector has not received a pet within the set limits for that component.
- **`Critical_Task_Not_Petting`** - Event to indicate that one or more of our critical tasks have not indicated a pet in the maximum limit of ticks. The hardware watchdog will not be pet in this case.
- **`Invalid_Command_Received`** - A command was received with invalid parameters.

3.9 Data Products

Data products for the Task Watchdog component.

Table 8: Task Watchdog Data Products

Local ID	Data Product Name	Type
0x0000 (0)	Watchdog_Component_Petter_State	Packed_Watchdog_ Component_State.T
0x0001 (1)	Pet_Connector_Action_States	Packed_U32.T

Data Product Descriptions:

- **`Watchdog_Component_Petter_State`** - Data product that tracks the global state to enable or disable all checks on the upstream watchdog pets.
- **`Pet_Connector_Action_States`** - 2-bit of state for each pet connector indicating the current action that will be taken if there is an error. Note that Packed_U32.T is just a placeholder type for this data product. The actual type of this data product will be autocoded and at assembly model ingest time.

3.10 Data Dependencies

The Task Watchdog component has no data dependencies.

3.11 Packets

The Task Watchdog component has no packets.

3.12 Faults

Faults for the Task Watchdog component

Table 9: Task Watchdog Fault

Local ID	Fault Name	Parameter Type
0x0001 (1)	Dummy_Fault	Packed_U16.T

Fault Descriptions:

- **`Dummy_Fault`** - Dummy fault which will be deleted and filled back in from the watchdog list

4 Unit Tests

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

4.1 *Task_ Watchdog_ Tests* Test Suite

This is a unit test suite for the Task Watchdog.

Test Descriptions:

- **`Test_Received_Pet`** - This unit test is here to make sure arrayed connectors work as expected.
- **`Test_Watchdog_Petter_Check_Command`** - This unit test tests the commanding to change the state of the component for checking upstream watchdog pets.
- **`Test_Watchdog_Action_Command`** - This unit test tests the commanding to change the action given the index of the petter in which to change.
- **`Test_Watchdog_Limit_Command`** - This unit test tests the commanding to change the limit for a particular index to a new specified value.
- **`Test_Invalid_Command`** - This unit test makes sure that an invalid command is handled gracefully.

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 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.

Data_Product.T:

Generic data product packet for holding arbitrary data types

Table 13: 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 14: 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 15: 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 16: 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

Fault.T:

Generic fault packet for holding arbitrary faults.

Table 17: Fault Packed Record : 152 bits *(maximum)*

Name	Type	Range	Size (Bits)	Start Bit	End Bit	Variable Length
Header	Fault_Header.T	-	88	0	87	-
Param_Buffer	Fault_Types. Parameter_ Buffer_Type	-	64	88	151	Header.Param_ Buffer_Length

Field Descriptions:

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

Fault_Header.T:

Generic fault header.

Table 18: Fault_Header Packed Record : 88 bits

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

Field Descriptions:

- **Time** - The timestamp for the fault.
- **Id** - The fault 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.

Packed_Connector_Index.T:

Single component record for holding packed connector index.

Table 20: Packed_Connector_Index Packed Record : 16 bits

Name	Type	Range	Size (Bits)	Start Bit	End Bit

Index	Connector_Types. Connector_Index_ Type	1 to 65535	16	0	15

Field Descriptions:

- **`Index`** - The 16-bit connector index.

Packed_U16.T:

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

Table 21: 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.

Packed_U32.T:

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

Table 22: 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.

Packed_Watchdog_Component_State.T:

State value packed for the task watchdog data products

Table 23: Packed_Watchdog_Component_State Packed Record : 8 bits

Name	Type	Range	Size (Bits)	Start Bit	End Bit
State	Task_Watchdog_ Enums.Watchdog_ Enabled_State.E	0 => Disabled 1 => Enabled	8	0	7

Field Descriptions:

- **`State`** - The state of the watchdog component to check all upstream petters.

Pet.T:

The pet datatype is used for servicing a watchdog. Included in this type is a count.

Table 24: Pet Packed Record : 32 bits

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

Field Descriptions:
- **Count** - The cycle number of the pet.

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 25: 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 26: 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.

Watchdog_Action_Cmd.T:

This record contains information for changing the limit of a specific watchdog connector.

Table 27: Watchdog_Action_Cmd Packed Record : 24 bits

Name	Type	Range	Size (Bits)	Start Bit	End Bit

Index	Connector_Types.Connector_Index_Type	1 to 65535	16	0	15
New_Action	Task_Watchdog_Enums.Watchdog_Action_State.E	0 => Disabled 1 => Warn 2 => Error_Fault	8	16	23

Field Descriptions:

- **`Index`** - The index of the connector that the command wants to change the limit of
- **`New_Action`** - The new value of the action for the specific associated connector

Watchdog_Limit_Cmd.T:

This record contains information for changing the limit of a specific watchdog connector.

Table 28: Watchdog_Limit_Cmd Packed Record : 32 bits

Name	Type	Range	Size (Bits)	Start Bit	End Bit
Index	Connector_Types.Connector_Index_Type	1 to 65535	16	0	15
New_Limit	Task_Watchdog_Types.Missed_Pet_Limit_Type	1 to 65534	16	16	31

Field Descriptions:

- **`Index`** - The index of the connector that the command wants to change the limit of
- **`New_Limit`** - The new value of the limit for the specific associated connector

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 29: 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.

Task_Watchdog_Enums.Watchdog_Enabled_State.E:

The state for if each watchdog is enabled or disabled for checking

Table 30: Watchdog_Enabled_State Literals:

Name	Value	Description
Disabled	0	
Enabled	1	

Task_Watchdog_Enums.Watchdog_Action_State.E:

The state for if each watchdog for which action it should take.

Table 31: Watchdog_Action_State Literals:

Name	Value	Description
Disabled	0	
Warn	1	
Error_Fault	2	