Task Watchdog Generator

Autocoder User Guide

1 Description

The purpose of this generator is to provide the user a way to define the components that the system will use to perform watchdog checks on. The connector used with this is a Pet connector. This connection will be monitored and if there is not a response from the component within a specified amount of ticks, which is a limit that is defined in the model file by the user, then an action is taken (again specified by the user) to either do nothing, issue a warning event, or issue an event and fault. The current state of each petter is dynamically created into a data product to be sent anytime they are updated. There is also a data product produced for each limit of the petters defined. In addition to the data products, faults are also dynamically created from the petters where a fault id is defined. This indicates a fault should be created which will allow state transitions into the faulted state, even if not defined that way at the start.

The user can also defined a criticality of the petter. If the petter is not disabled, when a pet is not received from that connection exceeding the limit, then the critical flag is tripped which will stop the downstream petting of the task watchdog component.

Note the example shown in this documentation is used in the unit test of this component so that the reader of this document can see it being used in context. Please refer to the unit test code for more details on how this generator can be used.

2 Schema

The following pykwalify schema is used to validate the input YAML model. Model files must be named in the form *assembly_name.task_watchdog_list.yaml*. The *assembly_name* is the assembly which the component connections should come from, and the rest of the model file name must remain as shown. Generally this file is created in the same directory or near to the assembly model file. The schema is commented to show what each of the available YAML keys are and what they accomplish. Even without knowing the specifics of pykwalify schemas, you should be able to gleam some knowledge from the file below.

```
---
# This schema describes the yaml format for the task watchdog list of tasks
↪  that have a watchdog that needs to be pet.
type: map
mapping:
  description:
    type: str
    required: False

  # List of tasks for the task watchdog component.
  petters:
    seq:
      - type: map
        mapping:
          # The optional name of the connection petter used to name faults and
          ↪  data products related to the petter.
```

```
          name:
            type: str
            required: False
          # Name of the component instance connector that contains a watchdog
          ↪  to pet
          connector_name:
            type: str
            required: True
          # Description of the component watchdog
          description:
            type: str
            required: False
          # The number of ticks that go by without a pet before action needs to
          ↪  be taken.
          # Must be at least a value of 1 to 2^16 - 2
          limit:
            type: int
            range:
              min: 1
              max: 65534
            required: True
          # Determines what action to take if the watchdog fails
          action:
            type: str
            enum: ['disabled', 'warn', 'error_fault']
            required: False
          # If the action has the potential for a fault, then an id must be
          ↪  provided
          fault_id:
            type: int
            required: False
          # Determines if the petting of the watchdog stops if it shows
          ↪  incorrect behavior
          critical:
            type: bool
            required: True
    # A task watchdog must have at least one task to pet
    range:
      min: 1
    required: True
```

3 Example Input

The following shows an example input YAML model that is used in the unit testing for the task watchdog component. The example attempts to use all the variations of the optional inputs so that the user can see how they can use the options to generate their own desired input. Generally this file is created in the same directory or near to the assembly model file. This example adheres to the schema shown in the previous section, and is commented to give clarification.

```
---
petters:
  - name: Second_Component_First_Connection
    connector_name: Component_B.Pet_1_T_Send
    description: A test pet connector from a component that has two connections
    ↪  (1 of 2)
    limit: 3
    action: error_fault
```

```
    fault_id: 5
    critical: True
  - connector_name: Component_B.Pet_2_T_Send
    description: A test pet connector from a component that has two connections
    ↪  (2 of 2)
    limit: 1
    action: disabled
    fault_id: 2
    critical: False
  - name: First_Component
    connector_name: Component_A.Pet_T_Send
    description: A test connector for the task watchdog component
    limit: 2
    action: warn
    critical: False
```

As can be seen, specifying a petter consist of at least a connector name, a limit, an action, and the criticality. There is also the capability to include a name which will be used as the name of the generated data products and faults. If no name is specified, the generator will use a autogenerated name using the connector name.

4 Example Output

The example input shown in the previous section produces the following Ada output. The `Downsample_List` variable should be passed into the Task Watchdog component's `Init` procedure during assembly initialization.

The main job of the generator in this case was to verify the input YAML for validity and then to translate the data to an Ada data structure for use by the component. The generator will also create data products and faults as was mentioned before in this document

```
------------------------------------------------------------------------------
-- This file was autogenerated from /vagrant/adamant/src/components/task_watchdog/test/test_assembly/test_assembly.task_watchdog_list.yaml on 2022-04-01 19:38.
--
-- Copyright: The University of Colorado, Laboratory for Atmospheric and Space Physics (LASP)
------------------------------------------------------------------------------

-- Standard includes:
with Task_Watchdog_Types; use Task_Watchdog_Types;
with Task_Watchdog_Enums; use Task_Watchdog_Enums;

package Test_Assembly_Task_Watchdog_List is
  -- The initial list for the task watchdog component
  Task_Watchdog_Entry_Init_List : aliased Task_Watchdog_Init_List := (
    -- A test connector for the task watchdog component
    -- Component_A.Component_A.Pet_T_Send
    -- User defined name: First_Component
    1 => (
      Max_Missed_Pet_Limit => 2,
      Critical => False,
      Action => Watchdog_Action_State.Warn,
      Action_Id => 0,
      Petter_Has_Fault => False
```

```
    ),
    -- A test pet connector from a component that has two connections (1 of 2)
    -- Component_B.Component_B.Pet_1_T_Send
    -- User defined name: Second_Component_First_Connection
    2 => (
      Max_Missed_Pet_Limit => 3,
      Critical => True,
      Action => Watchdog_Action_State.Error_Fault,
      Action_Id => 5,
      Petter_Has_Fault => True
    ),
    -- A test pet connector from a component that has two connections (2 of 2)
    -- Component_B.Component_B.Pet_2_T_Send
    3 => (
      Max_Missed_Pet_Limit => 1,
      Critical => False,
      Action => Watchdog_Action_State.Disabled,
      Action_Id => 2,
      Petter_Has_Fault => True
    )
  );

end Test_Assembly_Task_Watchdog_List;
```