# Product Packets Generator

*Autocoder User Guide*

## 1 Description

The purpose of this generator is to provide a user friendly way of creating packets formed from a list of data products. The generator takes a YAML model file as input which specifies the packets to produce, the data products to put in each packet, the period that the packet will be emitted at, and whether the packet is enabled or disabled on startup. From this information, the generator autocodes an Ada specification file which contains a data structure that should be passed to the Product Packetizer component upon initialization.

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 *optional_name.assembly_name.product_packets.yaml* where *optional_name* is the specific name of this set of packets and is only necessary if there is more than one Product Packetizer component instance in an assembly. The *assembly_name* is the assembly which these product packets will be used in, 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.

```yaml
---
# This schema describes the yaml format for a data product packet suite.
type: map
mapping:
  # Description of the packet suite.
  description:
    type: str
    required: False
  # Many "with" dependencies are automatically deduced and included by
  # the generator. If you want to manually add a "with" statement, you
  # can list the names of the packages here.
  with:
    seq:
      - type: str
    required: False
  # List of packets to include in the suite.
  packets:
    seq:
      - type: map
        mapping:
          # Name of the packet.
          name:
```

```yaml
          type: str
          required: True
        # Description of the packet.
        description:
          type: str
          required: False
        # Identifier for the packet (in CCSDS this would be the APID).
        id:
          type: int
          required: True
        # Is the packet enabled or disabled upon initialization. By default
        # packets are enabled if this is not specified. This field can be set to:
        #   True - Packet is enabled and sent periodically
        #   False - Packet is disabled
        #   On_Change - Packet is only sent when data products have changed since last emission
        enabled:
          type: any
          required: False
        # The period (in ticks) in which to build the packet. This is the value set upon
        # initialization.
        period:
          type: str
          required: True
        # The offset (in ticks) at which to stagger the construction of this packet. An offset of
        # 5 will cause the packet to be built according to its period, but 5 ticks later than expected.
        # Note that the offset should be less than the period otherwise it will be mod'ed by the period
        # so that it is less than the period. For example, if the period is 3 and the offset is set to
        # 5, the actual offset used will be 2.
        #
        # This field can be used to stagger packet creation, allowing the user to evenly distribute
        # the work that this component does, so as to not cause cycle slips when many packets need to
        # be built on the same tick.
        offset:
          type: str
          required: False
        # If set to true then the packet is timestamped with the time found on the incoming Tick.T
        # instead of the current time as fetched via the time connector. By default, if not specified
        # this value is set to False.
        use_tick_timestamp:
          type: bool
          required: False
        # List of data products to include in packet
        data_products:
          seq:
            - type: map
              mapping:
                # The name of the data product. The name should be in the format
                # Component_Name.Data_Product_Name. The name is a required field unless
```

```yaml
                  # pad_bytes is specified.
                  name:
                    type: str
                    required: False
                  # Produce an event if the data product is ever not available when fetched. By default
                  # this is false.
                  event_on_missing:
                    type: bool
                    required: False
                  # Use this data product's timestamp as the packet timestamp. This may only be set true for
                  # a single data product per packet. By default this value is false.
                  use_timestamp:
                    type: bool
                    required: False
                  # Include this data product's timestamp just before its value in the actual packet.
                  # By default this value is false.
                  include_timestamp:
                    type: bool
                    required: False
                  # When the packet's enabled field is set to On_Change, this field determines whether
                  # this data product's timestamp should be checked to trigger packet emission. If true,
                  # the packet will be sent if this data product has been updated since the last packet
                  # emission. If false, this data product's timestamp is ignored for on_change checks.
                  # By default this value is true if not specified.
                  used_for_on_change:
                    type: bool
                    required: False
                  # Pad bytes can be used to insert a n-number of bytes of unused data into a packet. This is
                  # also useful to add in spacing for data products that do not exist yet, but are expected
                  # in the packet. Pad bytes can only be specified if no other fields are specified.
                  pad_bytes:
                    type: int
                    required: False
            range:
              min: 1
            required: True
    # A packet suite must have at least one packet.
    range:
      min: 1
    required: True
```

## 3 Example Input

The following is an example product packet input yaml file. Model files must be named in the form *optional_name.assembly_name.product_packets.yaml* where *optional_name* is the specific name of the product packets and is only necessary if there is more than one Product Packetizer component instance in an assembly. The *assemble_name* is the assembly which these packets will be used in, 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. This example adheres to the schema shown in the previous section, and is commented to give clarification.

```yaml
---
description: This is an example set of packets.
# starting id...
# assuming 1 hz tick
packets:
  - name: Packet_1 # must be unique, enforce by autocoder
    description: This is packet 1.
    id: 7
    data_products:
      - name: Test_Component_1_Instance.Data_Product_A
        use_timestamp: False
        include_timestamp: True
        event_on_missing: True
      - name: Test_Component_2_Instance.Data_Product_C
        event_on_missing: False
        use_timestamp: False
        include_timestamp: False
    period: "3" # create every 3 ticks
    enabled: True
  - name: Packet_2
    id: 9
    data_products:
      - name: Test_Component_2_Instance.Data_Product_D
      - name: Test_Component_1_Instance.Data_Product_B
        use_timestamp: True
    period: "1" # create every tick
    offset: "0"
    enabled: False
  - name: Packet_3 # must be unique, enforce by autocoder
    description: This is packet 1.
    id: 8
    use_tick_timestamp: False
    data_products:
      - name: Test_Component_1_Instance.Data_Product_A
        use_timestamp: False
        include_timestamp: True
        event_on_missing: True
      - name: Test_Component_2_Instance.Data_Product_C
        event_on_missing: False
        use_timestamp: False
        include_timestamp: False
    period: "3" # create every 3 ticks
    offset: "5" # This should act like an offset of 2, but we are testing that
    ↪  feature here.
    enabled: False
  - name: Packet_4
    description: This packet tests padding
    id: 12
    use_tick_timestamp: True
    data_products:
      - pad_bytes: 5
      - name: Test_Component_1_Instance.Data_Product_A
        use_timestamp: False
        include_timestamp: False
        event_on_missing: False
      - pad_bytes: 3
    period: "1" # create every tick
    offset: "0"
    enabled: False
  - name: Packet_5
    id: 15
    data_products:
      - name: Product_Packetizer_Instance.Packet_4_Period
      - name: Product_Packetizer_Instance.Packet_5_Period
      - name: Product_Packetizer_Instance.Packet_3_Period
    period: "2"
    enabled: False
```

## 4 Example Output

The example input shown in the previous section produces the following Ada output. The `Packet_List` variable should be passed into the Product Packetizer component's discriminant during assembly initialization.

The main job of the generator in this case was to verify the input YAML packets for validity and then to translate the data to an Ada data structure for use by the component.

```ada
-- Standard includes:
with Product_Packet_Types; use Product_Packet_Types;
with Packet_Types;
with Sys_Time.Arithmetic;

-- This is an example set of packets.
package Test_Assembly_Product_Packets_Test_Packets is

   -- Packet_1:
   -- This is packet 1.

   -- Packet_1 data product items:
   -- Total packet buffer size: 160 bits
   Packet_1_Items : aliased Packet_Items_Type := [
      -- Item entry for Test_Component_1_Instance.Data_Product_A:
      1 => (Data_Product_Id => 1, Use_Timestamp => False, Include_Timestamp =>
      ↪  True, Event_On_Missing => True, Used_For_On_Change => True,
      ↪  Packet_Period_Item => False, Size => 4),
      -- Item entry for Test_Component_2_Instance.Data_Product_C:
      2 => (Data_Product_Id => 3, Use_Timestamp => False, Include_Timestamp =>
      ↪  False, Event_On_Missing => False, Used_For_On_Change => True,
      ↪  Packet_Period_Item => False, Size => 10)
   ];

   -- Packet_1 packet description:
   Packet_1_Description : Packet_Description_Type := (
      Id => 7,
      Items => Packet_1_Items'Access,
      Period => 3,
      Offset => 0,
      Enabled => Product_Packet_Types.Enabled,
      Use_Tick_Timestamp => False,
      Count => Packet_Types.Sequence_Count_Mod_Type'First,
      Send_Now => False,
      Last_Emission_Time => Sys_Time.Arithmetic.Sys_Time_Zero
   );

   -- Packet_2:

   -- Packet_2 data product items:
   -- Total packet buffer size: 96 bits
   Packet_2_Items : aliased Packet_Items_Type := [
      -- Item entry for Test_Component_2_Instance.Data_Product_D:
      1 => (Data_Product_Id => 4, Use_Timestamp => False, Include_Timestamp => False, Event_On_Missing => False, Used_For_On_Change => True, Packet_Period_Item => False, Size => 2),
      -- Item entry for Test_Component_1_Instance.Data_Product_B:
      2 => (Data_Product_Id => 2, Use_Timestamp => True, Include_Timestamp => False, Event_On_Missing => False, Used_For_On_Change => True, Packet_Period_Item => False, Size => 10)
   ];

   -- Packet_2 packet description:
   Packet_2_Description : Packet_Description_Type := (
      Id => 9,
      Items => Packet_2_Items'Access,
      Period => 1,
      Offset => 0,
      Enabled => Product_Packet_Types.Disabled,
      Use_Tick_Timestamp => False,
      Count => Packet_Types.Sequence_Count_Mod_Type'First,
      Send_Now => False,
      Last_Emission_Time => Sys_Time.Arithmetic.Sys_Time_Zero
   );

   -- Packet_3:
   -- This is packet 1.

   -- Packet_3 data product items:
   -- Total packet buffer size: 160 bits
   Packet_3_Items : aliased Packet_Items_Type := [
      -- Item entry for Test_Component_1_Instance.Data_Product_A:
      1 => (Data_Product_Id => 1, Use_Timestamp => False, Include_Timestamp => True, Event_On_Missing => True, Used_For_On_Change => True, Packet_Period_Item => False, Size => 4),
      -- Item entry for Test_Component_2_Instance.Data_Product_C:
      2 => (Data_Product_Id => 3, Use_Timestamp => False, Include_Timestamp => False, Event_On_Missing => False, Used_For_On_Change => True, Packet_Period_Item => False, Size => 10)
   ];

   -- Packet_3 packet description:
   Packet_3_Description : Packet_Description_Type := (
      Id => 8,
      Items => Packet_3_Items'Access,
      Period => 3,
      Offset => 5,
      Enabled => Product_Packet_Types.Disabled,
      Use_Tick_Timestamp => False,
      Count => Packet_Types.Sequence_Count_Mod_Type'First,
      Send_Now => False,
      Last_Emission_Time => Sys_Time.Arithmetic.Sys_Time_Zero
   );

   -- Packet_4:
   -- This packet tests padding

   -- Packet_4 data product items:
   -- Total packet buffer size: 96 bits
   Packet_4_Items : aliased Packet_Items_Type := [
      -- Item entry for :
      1 => (Data_Product_Id => 0, Use_Timestamp => False, Include_Timestamp => False, Event_On_Missing => False, Used_For_On_Change => True, Packet_Period_Item => False, Size => 5),
      -- Item entry for Test_Component_1_Instance.Data_Product_A:
      2 => (Data_Product_Id => 1, Use_Timestamp => False, Include_Timestamp =>
       ↪ False, Event_On_Missing => False, Used_For_On_Change => True,
       ↪ Packet_Period_Item => False, Size => 4),
      -- Item entry for :
      3 => (Data_Product_Id => 0, Use_Timestamp => False, Include_Timestamp =>
       ↪ False, Event_On_Missing => False, Used_For_On_Change => True,
       ↪ Packet_Period_Item => False, Size => 3)
   ];

   -- Packet_4 packet description:
   Packet_4_Description : Packet_Description_Type := (
      Id => 12,
      Items => Packet_4_Items'Access,
      Period => 1,
      Offset => 0,
      Enabled => Product_Packet_Types.Disabled,
      Use_Tick_Timestamp => True,
      Count => Packet_Types.Sequence_Count_Mod_Type'First,
      Send_Now => False,
      Last_Emission_Time => Sys_Time.Arithmetic.Sys_Time_Zero
   );

   -- Packet_5:

   -- Packet_5 data product items:
   -- Total packet buffer size: 96 bits
   Packet_5_Items : aliased Packet_Items_Type := [
      -- Item entry for Product_Packetizer_Instance.Packet_4_Period:
      1 => (Data_Product_Id => 4, Use_Timestamp => False, Include_Timestamp =>
       ↪ False, Event_On_Missing => False, Used_For_On_Change => True,
       ↪ Packet_Period_Item => True, Size => 4),
      -- Item entry for Product_Packetizer_Instance.Packet_5_Period:
      2 => (Data_Product_Id => 5, Use_Timestamp => False, Include_Timestamp =>
       ↪ False, Event_On_Missing => False, Used_For_On_Change => True,
       ↪ Packet_Period_Item => True, Size => 4),
      -- Item entry for Product_Packetizer_Instance.Packet_3_Period:
      3 => (Data_Product_Id => 3, Use_Timestamp => False, Include_Timestamp =>
       ↪ False, Event_On_Missing => False, Used_For_On_Change => True,
       ↪ Packet_Period_Item => True, Size => 4)
   ];

   -- Packet_5 packet description:
   Packet_5_Description : Packet_Description_Type := (
      Id => 15,
      Items => Packet_5_Items'Access,
      Period => 2,
      Offset => 0,
      Enabled => Product_Packet_Types.Disabled,
      Use_Tick_Timestamp => False,
      Count => Packet_Types.Sequence_Count_Mod_Type'First,
      Send_Now => False,
      Last_Emission_Time => Sys_Time.Arithmetic.Sys_Time_Zero
   );

   -- List of packets for the packetizer to build:
   Packet_List : aliased Packet_Description_List_Type := [
      1 => Packet_1_Description,
      2 => Packet_2_Description,
      3 => Packet_3_Description,
      4 => Packet_4_Description,
      5 => Packet_5_Description
   ];

end Test_Assembly_Product_Packets_Test_Packets;
```

## 5 On-Change Packet Example

The Product Packetizer also supports “on-change” packets that emit only when one or more tracked data products have fresh timestamps relative to the last emission. The `enabled` enumeral value `On_Change` puts a packet into this mode, while the optional `used_for_on_change` flag attached to each item allows you to include data products that do not participate in the change detection. Any data product with `used_for_on_change = True` (the default) will cause the packet to fire once its timestamp advances; data products tagged with `False` are still copied into the packet but will not on their own schedule a send. Note that the configured packet period still matters—the component evaluates on-change packets on those interval boundaries, but suppresses the transmission if no tracked data product reports a timestamp newer than the last emission.

The dedicated on-change unit tests under `src/components/product_packetizer/test_on_change` exercise this behavior in isolation. The YAML below matches the model compiled in those tests and is intentionally minimal so it can double as documentation for configuring the feature.

### 5.1 On-Change YAML Model

```
---
description: This is an example set of packets.
# starting id...
# assuming 1 hz tick
packets:
  - name: Packet_1 # must be unique, enforce by autocoder
    description: Baseline periodic packet used by the on-change tests for
    ↪ reference behavior.
    id: 7
    data_products:
      - name: Test_Component_1_Instance.Data_Product_A
        use_timestamp: False
        include_timestamp: True
        event_on_missing: True
      - name: Test_Component_2_Instance.Data_Product_C
        event_on_missing: False
        use_timestamp: False
        include_timestamp: False
    period: "3" # create every 3 ticks
    enabled: False
  - name: Packet_6
    description: Minimal packet dedicated to validating the On_Change feature.
    id: 16
    data_products:
      - name: Test_Component_2_Instance.Data_Product_D
        used_for_on_change: True # When True, this data product's timestamp
        ↪ drives the on-change decision.
      - name: Test_Component_1_Instance.Data_Product_B
        used_for_on_change: False # When False, the data is included but does
        ↪ not trigger on-change sends.
    period: "1"
    enabled: On_Change # Packets configured this way emit only when a tracked
    ↪ data product timestamp advances.
```

## 5.2 Generated Output

Just like the periodic example, the generator produces an Ada structure that is supplied to the Product Packetizer instance. The output derived from the on-change YAML is shown below.

```ada
-- Standard includes:
with Product_Packet_Types; use Product_Packet_Types;
with Packet_Types;
with Sys_Time.Arithmetic;

-- This is an example set of packets.
package Test_Assembly_Product_Packets_Test_Packets is

   -- Packet_1:
   -- Baseline periodic packet used by the on-change tests for reference behavior.

   -- Packet_1 data product items:
   -- Total packet buffer size: 160 bits
   Packet_1_Items : aliased Packet_Items_Type := [
      -- Item entry for Test_Component_1_Instance.Data_Product_A:
      1 => (Data_Product_Id => 1, Use_Timestamp => False, Include_Timestamp => True, Event_On_Missing => True, Used_For_On_Change => True, Packet_Period_Item => False, Size => 4),
      -- Item entry for Test_Component_2_Instance.Data_Product_C:
      2 => (Data_Product_Id => 3, Use_Timestamp => False, Include_Timestamp => False, Event_On_Missing => False, Used_For_On_Change => True, Packet_Period_Item => False, Size => 10)
   ];

   -- Packet_1 packet description:
   Packet_1_Description : Packet_Description_Type := (
      Id => 7,
      Items => Packet_1_Items'Access,
      Period => 3,
      Offset => 0,
      Enabled => Product_Packet_Types.Disabled,
      Use_Tick_Timestamp => False,
      Count => Packet_Types.Sequence_Count_Mod_Type'First,
      Send_Now => False,
      Last_Emission_Time => Sys_Time.Arithmetic.Sys_Time_Zero
   );

   -- Packet_6:
   -- Minimal packet dedicated to validating the On_Change feature.

   -- Packet_6 data product items:
   -- Total packet buffer size: 96 bits
   Packet_6_Items : aliased Packet_Items_Type := [
      -- Item entry for Test_Component_2_Instance.Data_Product_D:
      1 => (Data_Product_Id => 4, Use_Timestamp => False, Include_Timestamp => False, Event_On_Missing => False, Used_For_On_Change => True, Packet_Period_Item => False, Size => 2),
      -- Item entry for Test_Component_1_Instance.Data_Product_B:
      2 => (Data_Product_Id => 2, Use_Timestamp => False, Include_Timestamp => False, Event_On_Missing => False, Used_For_On_Change => False, Packet_Period_Item => False, Size => 10)
   ];

   -- Packet_6 packet description:
   Packet_6_Description : Packet_Description_Type := (
      Id => 16,
      Items => Packet_6_Items'Access,
      Period => 1,
      Offset => 0,
      Enabled => Product_Packet_Types.On_Change,
      Use_Tick_Timestamp => False,
      Count => Packet_Types.Sequence_Count_Mod_Type'First,
      Send_Now => False,
      Last_Emission_Time => Sys_Time.Arithmetic.Sys_Time_Zero
   );

   -- List of packets for the packetizer to build:
   Packet_List : aliased Packet_Description_List_Type := [
      1 => Packet_1_Description,
      2 => Packet_6_Description
   ];

end Test_Assembly_Product_Packets_Test_Packets;
```

## 6 Special Items

The Product Packetizer allows you to specify “special" items to include in a packet that reflect internal data of the Product Packetizer component itself. Currently, the only supported “special" items are packet periods of the packets produced by the Product Packetizer. Packet 5, specified above, includes these items by specifying a data product within the Product Packetizer, ie. `Product_Packetizer_Instance.Packet_4_Period`. The Product Packetizer doesn’t actually have any data products, so this nomenclature instead denotes a special item. In this case, we want to include the current packet period value (a 4 byte unsigned integer) for Packet 4 into the packet. A period can be specified for any packet included in the YAML model using this pattern. Error checking at the modeling level will prevent you from specifying a packet period for a packet that does not exist.