Product Extractor Generator

Autocoder User Guide

1 Description

The purpose of this generator is to provide a user the ability to extract data from packet and create data products out of them. The generator takes a YAML model file as an input which is used by the component to determine which packets to extract data products from as well as the offset and type in the packet the component needs to extract. This is performed through an Ada specification and implementation that is autocoded to perform the extraction and verification of the data product. In the case that extracting a data product is invalid for the specified type, then an invalid data structure is returned with the value of the invalid product.

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. 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 product extractor list.
type: map
mapping:
  # Description of the data products from all packets to be extracted.
  description:
    type: str
    required: False
  # List of data products to include in the suite.
  data_products:
    seq:
      - type: map
        mapping:
          # Name of the product.
          name:
            type: str
            required: True
          # Description of the data_products item.
          description:
            type: str
            required: False
          # Type of the product.
          type:
            type: str
            required: True
          # Apid of the product in the associated packet.
          apid:
            type: int
            required: True
          # Offset of the product in the associated packet.
          offset:
            type: int
            required: True
          # Time format of the packet.
          time:
            type: str
            enum: ['current_time', 'packet_time']
            required: True
    # A data product extractor must have at least one product to extract.
    range:
      min: 1
    required: True
```

3 Example Input

The following is an example extracted product input yaml file. Model files must be named in the form *assembly_name.extracted_products.yaml* where the specific name of the extracted products is not allowed. The *assembly_name* is the assembly which this table will be used, 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.

```
---
data_products:
  - name: Test_Product_1
    type: Packed_U16.T
    apid: 100
    offset: 10
    time: packet_time
  - name: Test_Product_2
    type: Packed_Byte.T
    apid: 100
    offset: 16
    time: current_time
  - name: Test_Product_3
    type: Packed_U32.T
    apid: 200
    offset: 8
    time: current_time
  - name: Test_Product_4
    type: Packed_Natural.T
    apid: 200
    offset: 20
    time: packet_time
  - name: Test_Product_5
    description: Test product for the little endian version of a packed type.
    type: Packed_Natural.T_Le
    apid: 300
    offset: 15
    time: packet_time
```

The specified data products consist of everything that the user would normally be required to include when defining a data product for a component, with the addition of the offset in the packet and corresponding APID. The type is also required which must be in the form of a packed type. The last required field is the time type which is either the current time of extraction from the packet or the time contained in the packet.

4 Example Output

The example input shown in the previous section produces the following Ada output. The `Data_Product_Extraction` variable should be passed into the CCSDS Product Extractor component's `Init` procedure during assembly initialization.

The main job of the generator here was to verify the input YAML for validity and then to translate the data to an Ada data structure and Ada extraction and verification function for each product for use by the component. The generator also dynamically creates the data products for the component based on the YAML input.

Ads file:

```ada
---------------------------------------------------------------------------------------------
-- This file was autogenerated from /vagrant/adamant/src/components/ccsds_product_extractor/test/test_assembly/test_products.extracted_products.yaml on 2022-04-01 19:36.
--
-- Copyright: The University of Colorado, Laboratory for Atmospheric and Space Physics (LASP)
---------------------------------------------------------------------------------------------

-- Standard includes:
with Product_Extractor_Types; use Product_Extractor_Types;
with Data_Product;
with Data_Product_Types; use Data_Product_Types;
with Ccsds_Space_Packet;
with Invalid_Product_Data;
with Sys_Time;

package Test_Products is

   function Extract_And_Validate_Test_Product_1 (Pkt : in Ccsds_Space_Packet.T; Id_Base : in Data_Product_Types.Data_Product_Id; Timestamp : in Sys_Time.T; Dp : out Data_Product.T; Invalid_Data_Product : out Invalid_Product_Data.T) return Product_Status;

   function Extract_And_Validate_Test_Product_2 (Pkt : in Ccsds_Space_Packet.T; Id_Base : in Data_Product_Types.Data_Product_Id; Timestamp : in Sys_Time.T; Dp : out Data_Product.T; Invalid_Data_Product : out Invalid_Product_Data.T) return Product_Status;

   function Extract_And_Validate_Test_Product_3 (Pkt : in Ccsds_Space_Packet.T; Id_Base : in Data_Product_Types.Data_Product_Id; Timestamp : in Sys_Time.T; Dp : out Data_Product.T; Invalid_Data_Product : out Invalid_Product_Data.T) return Product_Status;

   function Extract_And_Validate_Test_Product_4 (Pkt : in Ccsds_Space_Packet.T; Id_Base : in Data_Product_Types.Data_Product_Id; Timestamp : in Sys_Time.T; Dp : out Data_Product.T; Invalid_Data_Product : out Invalid_Product_Data.T) return Product_Status;

   -- Test product for the little endian version of a packed type.
```

```
  function Extract_And_Validate_Test_Product_5 (Pkt : in Ccsds_Space_Packet.T; Id_Base : in Data_Product_Types.Data_Product_Id; Timestamp : in Sys_Time.T; Dp : out Data_Product.T; Invalid_Data_Product : out Invalid_Product_Data.T) return Product_Status;
  Extract_Products_100 : aliased Extractor_List := (
    0 => Extract_And_Validate_Test_Product_1'Access,
    1 => Extract_And_Validate_Test_Product_2'Access
  );
  Extract_Products_200 : aliased Extractor_List := (
    0 => Extract_And_Validate_Test_Product_3'Access,
    1 => Extract_And_Validate_Test_Product_4'Access
  );
  Extract_Products_300 : aliased Extractor_List := (
    0 => Extract_And_Validate_Test_Product_5'Access
  );

  -- Initial product extraction list containing the information to extract all the products requested by each apid
  Data_Product_Extraction_List : aliased Extracted_Product_List := (
    0 => (
      Apid => 100,
      Extract_List => Extract_Products_100'Access
    ),
    1 => (
      Apid => 200,
      Extract_List => Extract_Products_200'Access
    ),
    2 => (
      Apid => 300,
      Extract_List => Extract_Products_300'Access
    )
  );
  Data_Product_Extraction_List_Access : constant Extracted_Product_List_Access := Data_Product_Extraction_List'Access;

end Test_Products;
```

Adb file:

```
--------------------------------------------------------------------------------------------
-- This file was autogenerated from /vagrant/adamant/src/components/ccsds_product_extractor/test/test_assembly/test_products.extracted_products.yaml on 2022-04-01 19:36.
--
-- Copyright: The University of Colorado, Laboratory for Atmospheric and Space Physics (LASP)
--------------------------------------------------------------------------------------------

-- Standard includes:
with Basic_Types;
with Byte_Array_Util;
with Ccsds_Primary_Header; use Ccsds_Primary_Header;
with Interfaces; use Interfaces;
with Extract_Data_Product; use Extract_Data_Product;
with Packed_U16;
with Packed_U16.Validation;
with Packed_Byte;
with Packed_Byte.Validation;
with Packed_U32;
with Packed_U32.Validation;
with Packed_Natural;
with Packed_Natural.Validation;

package body Test_Products is

  -- The implementation for each extract and validate of each extracted data
  ↪   product
  function Extract_And_Validate_Test_Product_1 (Pkt : in Ccsds_Space_Packet.T;
  ↪   Id_Base : in Data_Product_Types.Data_Product_Id; Timestamp : in
  ↪   Sys_Time.T; Dp : out Data_Product.T; Invalid_Data_Product : out
  ↪   Invalid_Product_Data.T) return Product_Status is
    Extraction_Status : Extract_Status;
    Ignore : Sys_Time.T renames Timestamp;
  begin
    pragma Assert (Pkt.Header.Apid = 100);

    -- Initialize out parameters:
    Invalid_Data_Product := (
      Id => Data_Product_Types.Data_Product_Id'First,
      Errant_Field_Number => 0,
      Errant_Field => (others => 0)
    );

    -- Use the generic extraction function using the autocoded values from the
    ↪   YAML file to get the information needed from the packet
    Extraction_Status := Extract_Data_Product.Extract_Data_Product (Pkt => Pkt,
    ↪   Offset => 10, Length => Packed_U16.Size_In_Bytes, Id => (0 + Id_Base),
    ↪   Timestamp => Sys_Time.Serialization.From_Byte_Array (Pkt.Data
    ↪   (Pkt.Data'First .. Pkt.Data'First +
    ↪   Sys_Time.Serialization.Serialized_Length - 1)), Dp => Dp);

    -- Make sure the extraction was successfull, at this point the only failure
    ↪   should be the length
    case Extraction_Status is
      when Length_Overflow => return Length_Error;
      when Success => null;
    end case;

    -- Now make sure the data product is valid for the type that we are
    ↪   extracting using the validation from the packed type generation
    declare
      Ef : Unsigned_32;
      Overlay : Packed_U16.T with Import, Convention => Ada, Address =>
      ↪   Dp.Buffer'Address;
      Validation : constant Boolean := Packed_U16.Validation.Valid (R =>
      ↪   Overlay, Errant_Field => Ef);
    begin
      case Validation is
        when True =>
          return Success;
        when False =>
          -- When there is a validation error, fill in a data structure with
          ↪   the relevent information for the component to use to send an
          ↪   event.
          declare
            P_Type : Basic_Types.Poly_Type := (others => 0);
          begin
            -- Copy extracted value into poly type
            Byte_Array_Util.Safe_Right_Copy (P_Type, Pkt.Data (10 .. 10 + Packed_U16.Size_In_Bytes - 1));
            Invalid_Data_Product.Id := (0 + Id_Base);
            Invalid_Data_Product.Errant_Field_Number := Ef;
            Invalid_Data_Product.Errant_Field := P_Type;
          end;
          return Invalid_Data;
      end case;
    end;
  end Extract_And_Validate_Test_Product_1;

  function Extract_And_Validate_Test_Product_2 (Pkt : in Ccsds_Space_Packet.T; Id_Base : in Data_Product_Types.Data_Product_Id; Timestamp : in Sys_Time.T; Dp : out Data_Product.T; Invalid_Data_Product : out Invalid_Product_Data.T) return Product_Status is
    Extraction_Status : Extract_Status;
  begin
    pragma Assert (Pkt.Header.Apid = 100);

    -- Initialize out parameters:
    Invalid_Data_Product := (
      Id => Data_Product_Types.Data_Product_Id'First,
      Errant_Field_Number => 0,
      Errant_Field => (others => 0)
    );

    -- Use the generic extraction function using the autocoded values from the YAML file to get the information needed from the packet
    Extraction_Status := Extract_Data_Product.Extract_Data_Product (Pkt => Pkt, Offset => 16, Length => Packed_Byte.Size_In_Bytes, Id => (1 + Id_Base), Timestamp => Timestamp, Dp => Dp);

    -- Make sure the extraction was successfull, at this point the only failure should be the length
    case Extraction_Status is
      when Length_Overflow => return Length_Error;
      when Success => null;
    end case;

    -- Now make sure the data product is valid for the type that we are extracting using the validation from the packed type generation
    declare
      Ef : Unsigned_32;
      Overlay : Packed_Byte.T with Import, Convention => Ada, Address => Dp.Buffer'Address;
      Validation : constant Boolean := Packed_Byte.Validation.Valid (R => Overlay, Errant_Field => Ef);
    begin
      case Validation is
        when True =>
          return Success;
        when False =>
          -- When there is a validation error, fill in a data structure with the relevent information for the component to use to send an event.
          declare
            P_Type : Basic_Types.Poly_Type := (others => 0);
          begin
            -- Copy extracted value into poly type
            Byte_Array_Util.Safe_Right_Copy (P_Type, Pkt.Data (16 .. 16 + Packed_Byte.Size_In_Bytes - 1));
            Invalid_Data_Product.Id := (1 + Id_Base);
            Invalid_Data_Product.Errant_Field_Number := Ef;
            Invalid_Data_Product.Errant_Field := P_Type;
          end;
          return Invalid_Data;
      end case;
    end;
  end Extract_And_Validate_Test_Product_2;

  function Extract_And_Validate_Test_Product_3 (Pkt : in Ccsds_Space_Packet.T; Id_Base : in Data_Product_Types.Data_Product_Id; Timestamp : in Sys_Time.T; Dp : out Data_Product.T; Invalid_Data_Product : out Invalid_Product_Data.T) return Product_Status is
    Extraction_Status : Extract_Status;
  begin
    pragma Assert (Pkt.Header.Apid = 200);

    -- Initialize out parameters:
    Invalid_Data_Product := (
      Id => Data_Product_Types.Data_Product_Id'First,
      Errant_Field_Number => 0,
      Errant_Field => (others => 0)
    );

    -- Use the generic extraction function using the autocoded values from the YAML file to get the information needed from the packet
    Extraction_Status := Extract_Data_Product.Extract_Data_Product (Pkt => Pkt, Offset => 8, Length => Packed_U32.Size_In_Bytes, Id => (2 + Id_Base), Timestamp => Timestamp, Dp => Dp);

    -- Make sure the extraction was successfull, at this point the only failure should be the length
    case Extraction_Status is
      when Length_Overflow => return Length_Error;
      when Success => null;
    end case;

    -- Now make sure the data product is valid for the type that we are extracting using the validation from the packed type generation
    declare
      Ef : Unsigned_32;
      Overlay : Packed_U32.T with Import, Convention => Ada, Address => Dp.Buffer'Address;
      Validation : constant Boolean := Packed_U32.Validation.Valid (R => Overlay, Errant_Field => Ef);
    begin
      case Validation is
        when True =>
          return Success;
        when False =>
          -- When there is a validation error, fill in a data structure with the relevent information for the component to use to send an event.
          declare
            P_Type : Basic_Types.Poly_Type := (others => 0);
          begin
            -- Copy extracted value into poly type
            Byte_Array_Util.Safe_Right_Copy (P_Type, Pkt.Data (8 .. 8 + Packed_U32.Size_In_Bytes - 1));
            Invalid_Data_Product.Id := (2 + Id_Base);
            Invalid_Data_Product.Errant_Field_Number := Ef;
            Invalid_Data_Product.Errant_Field := P_Type;
          end;
          return Invalid_Data;
      end case;
    end;
  end Extract_And_Validate_Test_Product_3;

  function Extract_And_Validate_Test_Product_4 (Pkt : in Ccsds_Space_Packet.T; Id_Base : in Data_Product_Types.Data_Product_Id; Timestamp : in Sys_Time.T; Dp : out Data_Product.T; Invalid_Data_Product : out Invalid_Product_Data.T) return Product_Status is
    Extraction_Status : Extract_Status;
    Ignore : Sys_Time.T renames Timestamp;
  begin
    pragma Assert (Pkt.Header.Apid = 200);

    -- Initialize out parameters:
    Invalid_Data_Product := (
      Id => Data_Product_Types.Data_Product_Id'First,
      Errant_Field_Number => 0,
      Errant_Field => (others => 0)
    );

    -- Use the generic extraction function using the autocoded values from the YAML file to get the information needed from the packet
    Extraction_Status := Extract_Data_Product.Extract_Data_Product (Pkt => Pkt, Offset => 20, Length => Packed_Natural.Size_In_Bytes, Id => (3 + Id_Base), Timestamp => Sys_Time.Serialization.From_Byte_Array (Pkt.Data (Pkt.Data'First .. Pkt.Data'First + Sys_Time.Serialization.Serialized_Length - 1)), Dp => Dp);

    -- Make sure the extraction was successfull, at this point the only failure should be the length
    case Extraction_Status is
      when Length_Overflow => return Length_Error;
      when Success => null;
    end case;

    -- Now make sure the data product is valid for the type that we are extracting using the validation from the packed type generation
    declare
      Ef : Unsigned_32;
      Overlay : Packed_Natural.T with Import, Convention => Ada, Address => Dp.Buffer'Address;
      Validation : constant Boolean := Packed_Natural.Validation.Valid (R => Overlay, Errant_Field => Ef);
    begin
      case Validation is
        when True =>
          return Success;
        when False =>
          -- When there is a validation error, fill in a data structure with the relevent information for the component to use to send an event.
          declare
            P_Type : Basic_Types.Poly_Type := (others => 0);
          begin
            -- Copy extracted value into poly type
            Byte_Array_Util.Safe_Right_Copy (P_Type, Pkt.Data (20 .. 20 + Packed_Natural.Size_In_Bytes - 1));
            Invalid_Data_Product.Id := (3 + Id_Base);
```

```
            Invalid_Data_Product.Errant_Field_Number := Ef;
            Invalid_Data_Product.Errant_Field := P_Type;
          end;
          return Invalid_Data;
      end case;
    end;
  end Extract_And_Validate_Test_Product_4;

  function Extract_And_Validate_Test_Product_5 (Pkt : in Ccsds_Space_Packet.T; Id_Base : in Data_Product_Types.Data_Product_Id; Timestamp : in Sys_Time.T; Dp : out Data_Product.T; Invalid_Data_Product : out Invalid_Product_Data.T) return Product_Status is
    Extraction_Status : Extract_Status;
    Ignore : Sys_Time.T renames Timestamp;
  begin
    pragma Assert (Pkt.Header.Apid = 300);

    -- Initialize out parameters:
    Invalid_Data_Product := (
      Id => Data_Product_Types.Data_Product_Id'First,
      Errant_Field_Number => 0,
      Errant_Field => (others => 0)
    );

    -- Use the generic extraction function using the autocoded values from the YAML file to get the information needed from the packet
    Extraction_Status := Extract_Data_Product.Extract_Data_Product (Pkt => Pkt, Offset => 15, Length => Packed_Natural.Size_In_Bytes, Id => (4 + Id_Base), Timestamp => Sys_Time.Serialization.From_Byte_Array (Pkt.Data (Pkt.Data'First .. Pkt.Data'First + Sys_Time.Serialization.Serialized_Length - 1)), Dp => Dp);

    -- Make sure the extraction was successfull, at this point the only failure should be the length
    case Extraction_Status is
      when Length_Overflow => return Length_Error;
      when Success => null;
    end case;

    -- Now make sure the data product is valid for the type that we are extracting using the validation from the packed type generation
    declare
      Ef : Unsigned_32;
      pragma Warnings (Off, "overlay changes scalar storage order");
      Overlay : Packed_Natural.T_Le with Import, Convention => Ada, Address => Dp.Buffer'Address;
      pragma Warnings (On, "overlay changes scalar storage order");
      Validation : constant Boolean := Packed_Natural.Validation.Valid (R => Overlay, Errant_Field => Ef);
    begin
      case Validation is
        when True =>
          return Success;
        when False =>
          -- When there is a validation error, fill in a data structure with the relevent information for the component to use to send an event.
          declare
            P_Type : Basic_Types.Poly_Type := (others => 0);
          begin
            -- Copy extracted value into poly type
            Byte_Array_Util.Safe_Right_Copy (P_Type, Pkt.Data (15 .. 15 + Packed_Natural.Size_In_Bytes - 1));
            Invalid_Data_Product.Id := (4 + Id_Base);
            Invalid_Data_Product.Errant_Field_Number := Ef;
            Invalid_Data_Product.Errant_Field := P_Type;
          end;
          return Invalid_Data;
      end case;
    end;
  end Extract_And_Validate_Test_Product_5;

end Test_Products;
```