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.

```
---
# 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:

```
1  ------------------------------------------------------------------------------
   ↪  --------------
2  -- This file was autogenerated from /vagrant/adamant/src/components/ccsds_produ
   ↪  ct_extractor/test/test_assembly/test_products.extracted_products.yaml on
   ↪  2022-04-01 19:36.
3  --
4  -- Copyright: The University of Colorado, Laboratory for Atmospheric and Space
   ↪  Physics (LASP)
5  ------------------------------------------------------------------------------
   ↪  --------------
6
7  -- Standard includes:
8  with Product_Extractor_Types; use Product_Extractor_Types;
9  with Data_Product;
10 with Data_Product_Types; use Data_Product_Types;
11 with Ccsds_Space_Packet;
12 with Invalid_Product_Data;
13 with Sys_Time;
14
15 package Test_Products is
16
17   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;
18
19   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;
20
21   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;
22
23   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;
24
25   -- 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_produ
    ↪  ct_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;
```

```
16  with Packed_Byte.Validation;
17  with Packed_U32;
18  with Packed_U32.Validation;
19  with Packed_Natural;
20  with Packed_Natural.Validation;
21
22  package body Test_Products is
23
24    -- The implementation for each extract and validate of each extracted data
      ↪  product
25    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
26      Extraction_Status : Extract_Status;
27      Ignore : Sys_Time.T renames Timestamp;
28    begin
29      pragma Assert (Pkt.Header.Apid = 100);
30
31      -- Initialize out parameters:
32      Invalid_Data_Product := (
33        Id => Data_Product_Types.Data_Product_Id'First,
34        Errant_Field_Number => 0,
35        Errant_Field => (others => 0)
36      );
37
38      -- Use the generic extraction function using the autocoded values from the
        ↪  YAML file to get the information needed from the packet
39      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);
40
41      -- Make sure the extraction was successfull, at this point the only failure
        ↪  should be the length
42      case Extraction_Status is
43        when Length_Overflow => return Length_Error;
44        when Success => null;
45      end case;
46
47      -- Now make sure the data product is valid for the type that we are
        ↪  extracting using the validation from the packed type generation
48      declare
49        Ef : Unsigned_32;
50        Overlay : Packed_U16.T with Import, Convention => Ada, Address =>
          ↪  Dp.Buffer'Address;
51        Validation : constant Boolean := Packed_U16.Validation.Valid (R =>
          ↪  Overlay, Errant_Field => Ef);
52      begin
53        case Validation is
54          when True =>
55            return Success;
56          when False =>
57            -- When there is a validation error, fill in a data structure with
              ↪  the relevent information for the component to use to send an
              ↪  event.
58            declare
59              P_Type : Basic_Types.Poly_Type := (others => 0);
60            begin
61              -- Copy extracted value into poly type
```

```
62              Byte_Array_Util.Safe_Right_Copy (P_Type, Pkt.Data (10 .. 10 +
                ↪  Packed_U16.Size_In_Bytes - 1));
63              Invalid_Data_Product.Id := (0 + Id_Base);
64              Invalid_Data_Product.Errant_Field_Number := Ef;
65              Invalid_Data_Product.Errant_Field := P_Type;
66           end;
67           return Invalid_Data;
68        end case;
69     end;
70  end Extract_And_Validate_Test_Product_1;
71
72  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
73     Extraction_Status : Extract_Status;
74  begin
75     pragma Assert (Pkt.Header.Apid = 100);
76
77     -- Initialize out parameters:
78     Invalid_Data_Product := (
79        Id => Data_Product_Types.Data_Product_Id'First,
80        Errant_Field_Number => 0,
81        Errant_Field => (others => 0)
82     );
83
84     -- Use the generic extraction function using the autocoded values from the
       ↪  YAML file to get the information needed from the packet
85     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);
86
87     -- Make sure the extraction was successfull, at this point the only failure
       ↪  should be the length
88     case Extraction_Status is
89        when Length_Overflow => return Length_Error;
90        when Success => null;
91     end case;
92
93     -- Now make sure the data product is valid for the type that we are
       ↪  extracting using the validation from the packed type generation
94     declare
95        Ef : Unsigned_32;
96        Overlay : Packed_Byte.T with Import, Convention => Ada, Address =>
          ↪  Dp.Buffer'Address;
97        Validation : constant Boolean := Packed_Byte.Validation.Valid (R =>
          ↪  Overlay, Errant_Field => Ef);
98     begin
99        case Validation is
100          when True =>
101             return Success;
102          when False =>
103             -- When there is a validation error, fill in a data structure with
                ↪  the relevent information for the component to use to send an
                ↪  event.
104             declare
105                P_Type : Basic_Types.Poly_Type := (others => 0);
106             begin
107                -- Copy extracted value into poly type
108                Byte_Array_Util.Safe_Right_Copy (P_Type, Pkt.Data (16 .. 16 +
                   ↪  Packed_Byte.Size_In_Bytes - 1));
```

```
109             Invalid_Data_Product.Id := (1 + Id_Base);
110             Invalid_Data_Product.Errant_Field_Number := Ef;
111             Invalid_Data_Product.Errant_Field := P_Type;
112          end;
113          return Invalid_Data;
114       end case;
115    end;
116 end Extract_And_Validate_Test_Product_2;
117
118 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
119    Extraction_Status : Extract_Status;
120 begin
121    pragma Assert (Pkt.Header.Apid = 200);
122
123    -- Initialize out parameters:
124    Invalid_Data_Product := (
125       Id => Data_Product_Types.Data_Product_Id'First,
126       Errant_Field_Number => 0,
127       Errant_Field => (others => 0)
128    );
129
130    -- Use the generic extraction function using the autocoded values from the
    ↪  YAML file to get the information needed from the packet
131    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);
132
133    -- Make sure the extraction was successfull, at this point the only failure
    ↪  should be the length
134    case Extraction_Status is
135       when Length_Overflow => return Length_Error;
136       when Success => null;
137    end case;
138
139    -- Now make sure the data product is valid for the type that we are
    ↪  extracting using the validation from the packed type generation
140    declare
141       Ef : Unsigned_32;
142       Overlay : Packed_U32.T with Import, Convention => Ada, Address =>
       ↪  Dp.Buffer'Address;
143       Validation : constant Boolean := Packed_U32.Validation.Valid (R =>
       ↪  Overlay, Errant_Field => Ef);
144    begin
145       case Validation is
146          when True =>
147             return Success;
148          when False =>
149             -- When there is a validation error, fill in a data structure with
             ↪  the relevent information for the component to use to send an
             ↪  event.
150             declare
151                P_Type : Basic_Types.Poly_Type := (others => 0);
152             begin
153                -- Copy extracted value into poly type
154                Byte_Array_Util.Safe_Right_Copy (P_Type, Pkt.Data (8 .. 8 +
                ↪  Packed_U32.Size_In_Bytes - 1));
155                Invalid_Data_Product.Id := (2 + Id_Base);
156                Invalid_Data_Product.Errant_Field_Number := Ef;
```

```
157             Invalid_Data_Product.Errant_Field := P_Type;
158          end;
159          return Invalid_Data;
160       end case;
161    end;
162 end Extract_And_Validate_Test_Product_3;
163
164 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
165    Extraction_Status : Extract_Status;
166    Ignore : Sys_Time.T renames Timestamp;
167 begin
168    pragma Assert (Pkt.Header.Apid = 200);
169
170    -- Initialize out parameters:
171    Invalid_Data_Product := (
172       Id => Data_Product_Types.Data_Product_Id'First,
173       Errant_Field_Number => 0,
174       Errant_Field => (others => 0)
175    );
176
177    -- Use the generic extraction function using the autocoded values from the
    ↪  YAML file to get the information needed from the packet
178    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);
179
180    -- Make sure the extraction was successfull, at this point the only failure
    ↪  should be the length
181    case Extraction_Status is
182       when Length_Overflow => return Length_Error;
183       when Success => null;
184    end case;
185
186    -- Now make sure the data product is valid for the type that we are
    ↪  extracting using the validation from the packed type generation
187    declare
188       Ef : Unsigned_32;
189       Overlay : Packed_Natural.T with Import, Convention => Ada, Address =>
       ↪  Dp.Buffer'Address;
190       Validation : constant Boolean := Packed_Natural.Validation.Valid (R =>
       ↪  Overlay, Errant_Field => Ef);
191    begin
192       case Validation is
193          when True =>
194             return Success;
195          when False =>
196             -- When there is a validation error, fill in a data structure with
             ↪  the relevent information for the component to use to send an
             ↪  event.
197             declare
198                P_Type : Basic_Types.Poly_Type := (others => 0);
199             begin
200                -- Copy extracted value into poly type
201                Byte_Array_Util.Safe_Right_Copy (P_Type, Pkt.Data (20 .. 20 +
                ↪  Packed_Natural.Size_In_Bytes - 1));
202                Invalid_Data_Product.Id := (3 + Id_Base);
```

```
203             Invalid_Data_Product.Errant_Field_Number := Ef;
204             Invalid_Data_Product.Errant_Field := P_Type;
205          end;
206          return Invalid_Data;
207       end case;
208    end;
209 end Extract_And_Validate_Test_Product_4;
210
211 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
212    Extraction_Status : Extract_Status;
213    Ignore : Sys_Time.T renames Timestamp;
214 begin
215    pragma Assert (Pkt.Header.Apid = 300);
216
217    -- Initialize out parameters:
218    Invalid_Data_Product := (
219       Id => Data_Product_Types.Data_Product_Id'First,
220       Errant_Field_Number => 0,
221       Errant_Field => (others => 0)
222    );
223
224    -- Use the generic extraction function using the autocoded values from the
    ↪  YAML file to get the information needed from the packet
225    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);
226
227    -- Make sure the extraction was successfull, at this point the only failure
    ↪  should be the length
228    case Extraction_Status is
229       when Length_Overflow => return Length_Error;
230       when Success => null;
231    end case;
232
233    -- Now make sure the data product is valid for the type that we are
    ↪  extracting using the validation from the packed type generation
234    declare
235       Ef : Unsigned_32;
236       pragma Warnings (Off, "overlay changes scalar storage order");
237       Overlay : Packed_Natural.T_Le with Import, Convention => Ada, Address =>
       ↪  Dp.Buffer'Address;
238       pragma Warnings (On, "overlay changes scalar storage order");
239       Validation : constant Boolean := Packed_Natural.Validation.Valid (R =>
       ↪  Overlay, Errant_Field => Ef);
240    begin
241       case Validation is
242          when True =>
243             return Success;
244          when False =>
245             -- When there is a validation error, fill in a data structure with
             ↪  the relevent information for the component to use to send an
             ↪  event.
246             declare
247                P_Type : Basic_Types.Poly_Type := (others => 0);
248             begin
249                -- Copy extracted value into poly type
```

```
250             Byte_Array_Util.Safe_Right_Copy (P_Type, Pkt.Data (15 .. 15 +
                ↪  Packed_Natural.Size_In_Bytes - 1));
251             Invalid_Data_Product.Id := (4 + Id_Base);
252             Invalid_Data_Product.Errant_Field_Number := Ef;
253             Invalid_Data_Product.Errant_Field := P_Type;
254          end;
255          return Invalid_Data;
256       end case;
257    end;
258  end Extract_And_Validate_Test_Product_5;
259
260 end Test_Products;
```