Parameter Table Generator

Autocoder User Guide

1 Description

The purpose of this generator is to provide a user friendly way of defining the layout of a parameter table that will be used within the Parameters component. The generator takes a YAML model file as input which specifies the component instance parameters to include in the table, in the order specified. From this information, the generator autocodes an Ada specification file which contains a data structure that should be passed to the Parameters component upon initialization. This data structure provides the necessary information to the Parameters component to properly decode a parameter table upload and push new parameter values to the correct downstream component where those parameters will be used.

Note the examples shown in this documentation are used in the unit test suites 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 parameter table.
type: map
mapping:
  # Description of the parameter table.
  description:
    type: str
    required: False
  # The name of the component that this parameter table is being constructed for. This component
  # must exist in the assembly specified by this file's name otherwise an error will be thrown.
  # The component's name must be specified so that the generator can verify that any routed
  # parameters in the table actually exist in the assembly.
  parameters_instance_name:
    type: str
    required: True
  # List of parameters to include in the suite.
  parameters:
    seq:
      # Each parameter entry can be either:
      # 1. A single parameter string in the format "Component_Instance_Name"."Parameter_Name" or
      #    simply "Component_Instance_Name" to use all of the component's parameters.
```

```
      # 2. A list of parameter strings (grouped parameters) where all parameters
      ↪  in the list
      #    share the same entry in the parameter table. All parameters in a
      ↪  group must be the
      #    same type.
      # Note: Type validation is handled in the Python model since pykwalify
      ↪  doesn't support union types well.
      - type: any
    # A parameter table must have at least one parameter.
    range:
      min: 1
    required: True
```

3 Example Input

The following is an example parameter table input yaml file. Model files must be named in the form *optional_name.assembly_name.parameter_table.yaml* where *optional_name* is the specific name of the parameter table and is only necessary if there is more than one Parameters component instance in an assembly. 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.

```
---
# Optional - Description of parameter table
description: This is an example parameter table layout.
# Required - The name of the Parameters component instance that
# this table is being constructed for.
parameters_instance_name: Parameters_Instance
# Required - A list of parameters to include in the table in order
parameters:
  # You may specify a parameter name in the form:
  #   "component_Instance_Name"."Parameter_Name"
  - component_A.Parameter_I32
  - component_C.The_Tick
  - component_A.Parameter_U16
  # Or add ALL the parameters for a component on order
  # by simply specifying the "Component_Instance_Name"
  - component_B
```

As can be seen, specifying the parameter table layout simply consists of listing the parameters to include. You can either list just the component instance name, in which case all the parameters for that component will be included in the table in the order specified within the component's parameter model, or you can list individual component parameters, which provides fine grain control of which parameters are included and in what order.

4 Example Output

The example input shown in the previous section produces the following Ada output. The `parameter_Table_Entries` variable should be passed into the Parameters 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.

```
-- Standard includes:
with Parameters_Component_Types; use Parameters_Component_Types;

-- This is an example parameter table layout.
package Test_Parameter_Table is

   -- The size of the parameter table (in bytes). This includes the size of the parameter
   -- table header (Parameter_Table_Header.T) plus the size of all the parameter table data.
   Parameter_Table_Size_In_Bytes : constant Natural := 30;

   -- A list of the parameter table entries for use by the Parameters_Instance component.
   Parameter_Table_Entries : aliased Parameter_Table_Entry_List := [
      -- Parameter Component_A.Parameter_I32, size of 4 byte(s), Entry_ID 0.
      0 => (
         Id => 2,
         Entry_Id => 0,
         Component_Id => 1,
         Start_Index => 0,
         End_Index => 3
      ),
      -- Parameter Component_C.The_Tick, size of 12 byte(s), Entry_ID 1.
      1 => (
         Id => 5,
         Entry_Id => 1,
         Component_Id => 3,
         Start_Index => 4,
         End_Index => 15
      ),
      -- Parameter Component_A.Parameter_U16, size of 2 byte(s), Entry_ID 2.
      2 => (
         Id => 1,
         Entry_Id => 2,
         Component_Id => 1,
         Start_Index => 16,
         End_Index => 17
      ),
      -- Parameter Component_B.Parameter_U16, size of 2 byte(s), Entry_ID 3.
      3 => (
         Id => 3,
         Entry_Id => 3,
         Component_Id => 2,
         Start_Index => 18,
         End_Index => 19
      ),
      -- Parameter Component_B.Parameter_I32, size of 4 byte(s), Entry_ID 4.
      4 => (
         Id => 4,
         Entry_Id => 4,
         Component_Id => 2,
         Start_Index => 20,
         End_Index => 23
      )
   ];

end Test_Parameter_Table;
```

5 Grouped Parameters

The parameter table generator supports supports a feature called *grouped parameters*, which allows multiple parameters from different component instances to share the same memory location (entry) in the parameter table. This is useful when you want multiple components to use the exact same parameter value, essentially creating a union of parameters that must be kept synchronized. Of course, all parameters in a group must have the same type (and size), as they will be sharing the same memory.

When parameters are grouped together, they share the same `Entry_Id` in the parameter table, but retain unique `Parameter_Id` values. The key behaviors are:

- **Update Operations:** When a grouped entry is updated (via command or memory region upload), the new value is staged and updated to *all* parameters in the group simultaneously.
- **Fetch Operations:** When fetching parameters to create a dump, the Parameters component fetches from all parameters in the group. The *first* fetched value is used in the output. If subsequent fetches return different values, a `Parameter_Fetch_Value_Mismatch` info event is emitted to alert the user, but the first value is always used. This condition should never occur in a properly designed system.

5.1 Example Input

To specify grouped parameters in the YAML model, simply provide a list (array) of parameters instead of a single parameter name. The following example groups parameters from different component instances:

```yaml
---
description: Test parameter table with grouped parameters.
parameters_instance_name: Parameters_Instance
parameters:
  # Grouped parameters - same entry in table for both Component_A and
  ↪  Component_B
  - [component_A.Parameter_I32, component_B.Parameter_I32]
  - [component_A.Parameter_U16, component_B.Parameter_U16]
  - component_C.The_Tick
```

In this example:

- `component_A.Parameter_I32` and `component_B.Parameter_I32` are grouped together (they share Entry_ID 0)
- `component_A.Parameter_U16` and `component_B.Parameter_U16` are grouped together (they share Entry_ID 1)
- `component_C.The_Tick` is a standalone parameter (Entry_ID 2)

5.2 Example Output

The grouped parameter example shown above produces the following Ada output. Notice how multiple parameters share the same `Entry_Id` and the same `Start_Index` and `End_Index` values, indicating they occupy the same memory location:

```ada
-- Standard includes:
with Parameters_Component_Types; use Parameters_Component_Types;

-- Test parameter table with grouped parameters.
package Test_Grouped_Params is

   -- The size of the parameter table (in bytes). This includes the size of the
   ↪  parameter
```

```
   -- table header (Parameter_Table_Header.T) plus the size of all the parameter
   ↪  table data.
   Parameter_Table_Size_In_Bytes : constant Natural := 24;

   -- A list of the parameter table entries for use by the Parameters_Instance
   ↪  component.
   Parameter_Table_Entries : aliased Parameter_Table_Entry_List := [
      -- Parameter Component_A.Parameter_I32, size of 4 byte(s), Entry_ID 0.
      0 => (
         Id => 2,
         Entry_Id => 0,
         Component_Id => 1,
         Start_Index => 0,
         End_Index => 3
      ),
      -- Parameter Component_B.Parameter_I32, size of 4 byte(s), Entry_ID 0.
      1 => (
         Id => 4,
         Entry_Id => 0,
         Component_Id => 2,
         Start_Index => 0,
         End_Index => 3
      ),
      -- Parameter Component_A.Parameter_U16, size of 2 byte(s), Entry_ID 1.
      2 => (
         Id => 1,
         Entry_Id => 1,
         Component_Id => 1,
         Start_Index => 4,
         End_Index => 5
      ),
      -- Parameter Component_B.Parameter_U16, size of 2 byte(s), Entry_ID 1.
      3 => (
         Id => 3,
         Entry_Id => 1,
         Component_Id => 2,
         Start_Index => 4,
         End_Index => 5
      ),
      -- Parameter Component_C.The_Tick, size of 12 byte(s), Entry_ID 2.
      4 => (
         Id => 5,
         Entry_Id => 2,
         Component_Id => 3,
         Start_Index => 6,
         End_Index => 17
      )
   ];

end Test_Grouped_Params;
```

Important notes about the generated output:

- Parameters with `Id => 2` (Component_A.Parameter_I32) and `Id => 4` (Component_B.Parameter_I32) both have `Entry_Id => 0` and occupy bytes 0-3
- Parameters with `Id => 1` (Component_A.Parameter_U16) and `Id => 3` (Component_B.Parameter_U16) both have `Entry_Id => 1` and occupy bytes 4-5
- The parameter table contains 5 entries total, even though only 3 distinct memory locations (Entry_IDs 0, 1, and 2) are used

- The total parameter table size is only 18 bytes of data (plus header), despite managing 5 parameters, because grouped parameters share memory