UVM Object Print- print-all var, do_print-selected var,

`sprint-all var`, that returns the formatted contents in a string format

`convert2string-all var,` that will return a string instead of printing the contents in a predefined format.

this function returns a string and hence it has to be within a `` `uvm_info `` or similar report macro.

A `uvm_object` is the base class from which all other UVM classes for data and components are derived. So it is logical for this class to have a common set of functions and features that can be availed by all its derived classes.

Some of the common functions usually required is the ability to print its contents, copy contents from one object to another, and possibly compare two objects. UVM has many automation macros that get expanded into full code during compilation and implements these functions for all classes with ease.

Example

The purpose of this example is to show how UVM automation macros can be used to print variable contents easily.

A class called Object is derived from `uvm_object` thereby inheriting all functions within the parent class. A few variables of different data types are declared and an attempt is made to print the content of these variables after randomization.

There are different variations of `` `uvm_field_* `` macros for each data type and the variables have to be registered with the macro corresponding to its data type. `UVM_DEFAULT` indicates that the given variable should be used for all the default UVM automation macros implemented by the object which are copy, print, compare and record.

```
typedef enum {FALSE, TRUE} e_bool;
class Object extends uvm_object;

        rand e_bool                                m_bool;
  rand bit[3:0]                            m_mode;
  rand byte                                        m_data[4];
  rand shortint                            m_queue[$];
  string                                                       m_name;

  constraint c_queue { m_queue.size() == 3; }

  function new(string name = "Object");
    super.new(name);
    m_name = name;
  endfunction

   // Each variable has to be registered with a macro corresponding to its
data
  // type. For example, "int" types use `uvm_field int, "enum" types use
  // `uvm_field_enum, and "string" use `uvm_field_string
  `uvm_object_utils_begin(Object)
        `uvm_field_enum(e_bool, m_bool, UVM_DEFAULT)
                                          `uvm_field_int            (m_mode,
UVM_DEFAULT)
        `uvm_field_sarray_int(m_data,           UVM_DEFAULT)
        `uvm_field_queue_int(m_queue,           UVM_DEFAULT)
                                                   `uvm_field_string(m_name,
UVM_DEFAULT)
  `uvm_object_utils_end
endclass
```

To test the above code, we'll simply use a base test class that will print contents of the object.

```
class base_test extends uvm_test;
  `uvm_component_utils(base_test)
  function new(string name = "base_test", uvm_component parent=null);
    super.new(name, parent);
  endfunction

  // In the build phase, create an object, randomize it and print
  // its contents
  function void build_phase(uvm_phase phase);
    Object obj = Object::type_id::create("obj");
    obj.randomize();
    obj.print();
  endfunction
endclass

module tb;
        initial begin
                run_test("base_test");
        end
endmodule
```

By default the UVM printer prints content of any object in a table format in which it specifies the name of the variable, data type of the variable, its size and the value. In the simulation output shown below, it can be seen that the object obj was randomized to the given values. It's quite interesting to note that even arrays (both static and queues) are printed with content of all their indices.

Simulation Log

```
ncsim> run
UVM_INFO  /playground_lib/uvm-1.2/src/base/uvm_root.svh(392)  @  0:  reporter
[UVM/RELNOTES]
----------------------------------------------------------------
UVM-1.2
(C) 2007-2014 Mentor Graphics Corporation
(C) 2007-2014 Cadence Design Systems, Inc.
(C) 2006-2014 Synopsys, Inc.
(C) 2011-2013 Cypress Semiconductor Corp.
(C) 2013-2014 NVIDIA Corporation
----------------------------------------------------------------

UVM_INFO @ 0: reporter [RNTST] Running test base_test...
-------------------------------------
Name       Type           Size  Value
-------------------------------------
obj        Object         -     @1899
  m_bool   e_bool         32    TRUE
  m_mode   integral       4     'hd
  m_data   sa(integral)   4     -
    [0]    integral       8     'h6c
    [1]    integral       8     'hf4
    [2]    integral       8     'he
    [3]    integral       8     'h58
  m_queue  da(integral)   3     -
    [0]    integral       16    'h3cb6
    [1]    integral       16    'h9ae9
    [2]    integral       16    'hd31d
  m_name   string         3     obj
-------------------------------------
UVM_INFO  /playground_lib/uvm-1.2/src/base/uvm_report_server.svh(847)  @  0:
reporter [UVM/REPORT/SERVER]
--- UVM Report Summary ---
```

Using do_print()

Using automation macros are not recommended these days because it introduces quite a lot of additional code and reduces simulator performance. Instead it is recommended to use the `do_*` callback hooks to implement the requirement. For example, to print the contents, the user can implement the function called `do_print` inside the derived object and not use the automation macro at all.

The `do_print` function is called by the `print` function by default and hence whatever is defined in `do_print` will be printed out. In this case we'll simply print three selected variables although you can print all variables as required.

```
class Object extends uvm_object;

  rand e_bool                             m_bool;
  rand bit[3:0]                           m_mode;
  rand byte                               m_data[4];
  rand shortint                           m_queue[$];
  string                                          m_name;

  constraint c_queue { m_queue.size() == 3; }

  function new(string name = "Object");
    super.new(name);
    m_name = name;
  endfunction

  // Use "do_print" instead of the automation macro
  `uvm_object_utils(Object)

   // This function simply uses the printer functions to print variables
based on their
   // data types. For example, "int" variables are printed using function
"print_field_int"
  virtual function void do_print(uvm_printer printer);
    super.do_print(printer);
    printer.print_string("m_bool", m_bool.name());
    printer.print_field_int("m_mode", m_mode, $bits(m_mode), UVM_HEX);
    printer.print_string("m_name", m_name);
  endfunction
endclass
```

This results in a very similar output like the one by `` `uvm_object_utils_* ``. Note that it prints only three variables because only three variables were implemented inside the `do_print` function.

Simulation Log

```
ncsim> run
UVM_INFO /playground_lib/uvm-1.2/src/base/uvm_root.svh(392) @ 0: reporter
[UVM/RELNOTES]
----------------------------------------------------------------
UVM-1.2
(C) 2007-2014 Mentor Graphics Corporation
```

```
(C) 2007-2014 Cadence Design Systems, Inc.
(C) 2006-2014 Synopsys, Inc.
(C) 2011-2013 Cypress Semiconductor Corp.
(C) 2013-2014 NVIDIA Corporation
----------------------------------------------------------------
UVM_INFO @ 0: reporter [RNTST] Running test base_test...
-------------------------------
Name      Type      Size  Value
-------------------------------
obj       Object    -     @1898
  m_bool  string    4     TRUE
  m_mode  integral  4     'hd
  m_name  string    3     obj
-------------------------------
UVM_INFO  /playground_lib/uvm-1.2/src/base/uvm_report_server.svh(847)  @  0:
reporter [UVM/REPORT/SERVER]
--- UVM Report Summary ---
```

Using sprint

UVM objects have another function called `sprint` that returns the formatted contents in a string format. It is a substitute of the `print` function and can be used as shown in the example below. Assume the class declaration is the same as the first example shown above without the use of UVM automation macros.

```
class base_test extends uvm_test;
  `uvm_component_utils(base_test)
  function new(string name = "base_test", uvm_component parent=null);
    super.new(name, parent);
  endfunction

  function void build_phase(uvm_phase phase);
    Object obj = Object::type_id::create("obj");
    obj.randomize();

    // Instead of calling print() function, let us call "sprint"
       `uvm_info(get_type_name(), $sformatf("Contents:  %s",  obj.sprint()),
UVM_LOW)
  endfunction
endclass

module tb;
        initial begin
                run_test("base_test");
        end
endmodule
```

As you can see, it produced the same table format as a string but prefixed by the custom text called "Contents:".

Simulation Log

```
ncsim> run
UVM_INFO /playground_lib/uvm-1.2/src/base/uvm_root.svh(392) @ 0: reporter
[UVM/RELNOTES]
----------------------------------------------------------------
UVM-1.2
(C) 2007-2014 Mentor Graphics Corporation
(C) 2007-2014 Cadence Design Systems, Inc.
(C) 2006-2014 Synopsys, Inc.
(C) 2011-2013 Cypress Semiconductor Corp.
(C) 2013-2014 NVIDIA Corporation
----------------------------------------------------------------

UVM_INFO @ 0: reporter [RNTST] Running test base_test...
UVM_INFO testbench.sv(98) @ 0: uvm_test_top [base_test] Contents:
-------------------------------
Name      Type      Size  Value
-------------------------------
obj       Object    -     @1902
  m_bool  string    4     TRUE
  m_mode  integral  4     'he
  m_name  string    3     obj
-------------------------------

UVM_INFO /playground_lib/uvm-1.2/src/base/uvm_report_server.svh(847) @ 0:
reporter [UVM/REPORT/SERVER]
--- UVM Report Summary ---
```

Using convert2string

There is another function available in `uvm_object` called `convert2string` that will return a string instead of printing the contents in a predefined format. This allows you to define the output into a format you like. For example, we can simply print contents into a single line as shown.

```
class Object extends uvm_object;

  rand e_bool                          m_bool;
  rand bit[3:0]                           m_mode;
  rand byte                               m_data[4];
  rand shortint                           m_queue[$];
  string                                          m_name;

  constraint c_queue { m_queue.size() == 3; }

  function new(string name = "Object");
    super.new(name);
    m_name = name;
  endfunction

  // Use "do_print" instead of the automation macro
  `uvm_object_utils(Object)

  virtual function string convert2string();
    string contents = "";
    $sformat(contents, "%s m_name=%s", contents, m_name);
    $sformat(contents, "%s m_bool=%s", contents, m_bool.name());
    $sformat(contents, "%s m_mode=0x%0h", contents, m_mode);
    foreach(m_data[i]) begin
      $sformat(contents, "%s m_data[%0d]=0x%0h", contents, i, m_data[i]);
    end
    return contents;
  endfunction
```

In the test class we will have to call `convert2string` instead of `print` or `sprint`. But remember that this function returns a string and hence it has to be within a `` `uvm_info `` or similar report macro.

```
class base_test extends uvm_test;
  `uvm_component_utils(base_test)
  function new(string name = "base_test", uvm_component parent=null);
    super.new(name, parent);
  endfunction

  function void build_phase(uvm_phase phase);
    Object obj = Object::type_id::create("obj");
    obj.randomize();
             `uvm_info(get_type_name(),   $sformatf("convert2string:   %s",
obj.convert2string()), UVM_LOW)
  endfunction
endclass

module tb;
```

```
	initial begin
		run_test("base_test");
	end
endmodule
```

Simulation Log

```
ncsim> run
UVM_INFO /playground_lib/uvm-1.2/src/base/uvm_root.svh(392) @ 0: reporter
[UVM/RELNOTES]
----------------------------------------------------------------
UVM-1.2
(C) 2007-2014 Mentor Graphics Corporation
(C) 2007-2014 Cadence Design Systems, Inc.
(C) 2006-2014 Synopsys, Inc.
(C) 2011-2013 Cypress Semiconductor Corp.
(C) 2013-2014 NVIDIA Corporation
----------------------------------------------------------------

UVM_INFO @ 0: reporter [RNTST] Running test base_test...
UVM_INFO testbench.sv(99) @ 0: uvm_test_top [base_test] convert2string:
    m_name=obj    m_bool=TRUE    m_mode=0xe    m_data[0]=0xf4    m_data[1]=0xe
m_data[2]=0x58 m_data[3]=0xbd
UVM_INFO /playground_lib/uvm-1.2/src/base/uvm_report_server.svh(847) @ 0:
reporter [UVM/REPORT/SERVER]
--- UVM Report Summary ---
```