UVM Object Copy/Clone - copy-all var, do_copy-selected var, clone-same as copy bt difference being that a clone will return an object with the copied contents(i.e.`$cast(obj2, obj1.clone());`object name (obj1) will be not overwritten,)

Note in the following example that `uvm_object_utils_*` macros are not used and hence the `print()` method we used above will not work as is. Instead another method called `convert2string` is implemented for both classes

---

`uvm_object` has many common functions like print, copy and compare that are available to all its child classes and can be used out of the box if UVM automation macros are used inside the class definition. In a previous article, `print`, `do_print` and use of automation macros to print were discussed.

Using automation macros

A class called Packet is defined with a single variable and registered using UVM automation macros between `uvm_object_utils_begin` and `uvm_object_utils_end`. A variable used with UVM_DEFAULT setting means that this variable will be included in all automated methods like copy, print, etc unless specifically mentioned.

An object of Packet is instantiated in another class called Object along with a bunch of other variables of different data types. Similar to the Packet class, all variables in Object are registered with UVM automation macros with the corresponding macro type. For example, `string` variables require `` `uvm_field_string ``.

```
typedef enum {FALSE, TRUE} e_bool;

class Packet extends uvm_object;
  rand bit[15:0]        m_addr;

  // Automation macros
  `uvm_object_utils_begin(Packet)
        `uvm_field_int(m_addr, UVM_DEFAULT)
  `uvm_object_utils_end

  function new(string name = "Packet");
    super.new(name);
  endfunction
endclass

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;
  rand Packet                         m_pkt;

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

  function new(string name = "Object");
    super.new(name);
    m_name = name;
    m_pkt = Packet::type_id::create("m_pkt");
    m_pkt.randomize();
  endfunction

  `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_field_object(m_pkt,                UVM_DEFAULT)
  `uvm_object_utils_end
endclass
```

Let us create a test class and create two objects of type Object, randomize both and display them first. Then contents of obj1 is copied into obj2 using the `copy` function. Implementation of this copy function is taken care of by the 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 obj1 = Object::type_id::create("obj1");
    Object obj2 = Object::type_id::create("obj2");
    obj1.randomize();
    obj1.print();
    obj2.randomize();
        obj2.print();

    obj2.copy(obj1);
    `uvm_info("TEST", "After copy", UVM_LOW)
```

```
      obj2.print();
    endfunction
  endclass

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

The first table shows the values of obj1 after randomization, the second table that of obj2 after randomization and the third one shows contents of obj2 after the copy method is called.

Simulation Log

```
ncsim> run
UVM_INFO @ 0: reporter [RNTST] Running test base_test...
--------------------------------------
Name         Type           Size  Value
--------------------------------------
obj1         Object         -     @1903
  m_bool     e_bool         32    TRUE
  m_mode     integral       4     'he
  m_data     sa(integral)   4     -
    [0]      integral       8     'hf4
    [1]      integral       8     'he
    [2]      integral       8     'h58
    [3]      integral       8     'hbd
  m_queue    da(integral)   3     -
    [0]      integral       16    'h9ae9
    [1]      integral       16    'hd31d
    [2]      integral       16    'ha96c
  m_name     string         4     obj1
  m_pkt      Packet         -     @1906
    m_addr   integral       16    'h3cb6
--------------------------------------
--------------------------------------
Name         Type           Size  Value
--------------------------------------
obj2         Object         -     @1908
  m_bool     e_bool         32    FALSE
  m_mode     integral       4     'he
  m_data     sa(integral)   4     -
    [0]      integral       8     'h60
    [1]      integral       8     'h24
    [2]     integral      8     'h27
    [3]     integral      8     'hb5
  m_queue   da(integral)  3     -
    [0]     integral      16    'he17f
    [1]     integral      16    'h98e6
    [2]     integral      16    'h5a41
  m_name    string        4     obj2
  m_pkt     Packet        -     @1910
    m_addr  integral      16    'h64c1
---------------------------------------
UVM_INFO testbench.sv(60) @ 0: uvm_test_top [TEST] After copy
---------------------------------------
Name        Type          Size  Value
---------------------------------------
obj2        Object        -     @1908
  m_bool    e_bool        32    TRUE
  m_mode    integral      4     'he
  m_data    sa(integral)  4     -
    [0]     integral      8     'hf4
    [1]     integral      8     'he
    [2]     integral      8     'h58
    [3]     integral      8     'hbd
  m_queue   da(integral)  3     -
    [0]     integral      16    'h9ae9
    [1]     integral      16    'hd31d
    [2]     integral      16    'ha96c
  m_name    string        4     obj1
  m_pkt     Packet        -     @1909
    m_addr  integral      16    'h3cb6
---------------------------------------
UVM_INFO /playground_lib/uvm-1.2/src/base/uvm_report_server.svh(847) @ 0: reporter
[UVM/REPORT/SERVER]
--- UVM Report Summary ---
```

Automation macros introduce a lot of additional code and is not generally recommended

Using do_copy

Another way is for the user to implement `do_copy` method inside the child class and assign the value from each variable of the class to be copied into the current one. Note in the following example that `uvm_object_utils_*` macros are not used and hence the `print()` method we used above will not work as is. Instead another method called `convert2string` is implemented for both classes so that it returns the contents in a string format when called.

Just like the way `print` calls `do_print` method, a `copy` calls the `do_copy` method and implementation of the function is all that is required to be done.

```
typedef enum {FALSE, TRUE} e_bool;

class Packet extends uvm_object;
  rand bit[15:0]        m_addr;

  // Function is used to return contents of this class in a
  // string format
  virtual function string convert2string();
    string contents;
    contents = $sformatf("m_addr=0x%0h", m_addr);
  endfunction

  `uvm_object_utils(Packet)

  // Implementation of "do_copy". A generic uvm_object called "rhs"
  // is received and type casted into Packet called "_pkt". Then
  // m_addr is copied from _pkt to the variable in current class
  virtual function void do_copy(uvm_object rhs);
    Packet _pkt;
    super.do_copy(rhs);
    $cast(_pkt, rhs);
        m_addr = _pkt.m_addr;
    `uvm_info(get_name(), "In Packet::do_copy()", UVM_LOW)
  endfunction

  function new(string name = "Packet");
    super.new(name);
  endfunction
endclass

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;
  rand Packet                       m_pkt;

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

  function new(string name = "Object");
```

```
      super.new(name);
      m_name = name;
      m_pkt = Packet::type_id::create("m_pkt");
      m_pkt.randomize();
    endfunction

    // Function used to return contents of this class in a
    // string format
    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

    `uvm_object_utils(Object)

    // "rhs" does not contain m_bool, m_mode, etc since its a parent
    // handle. So cast into child data type and access using child handle
    // Copy each field from the casted handle into local variables
    virtual function void do_copy(uvm_object rhs);
      Object _obj;
      super.do_copy(rhs);
      $cast(_obj, rhs);
      m_bool      = _obj.m_bool;
      m_mode      = _obj.m_mode;
      m_data      = _obj.m_data;
      m_queue = _obj.m_queue;
      m_name      = _obj.m_name;
      m_pkt.copy(_obj.m_pkt);
      `uvm_info(get_name(), "In Object::do_copy()", UVM_LOW)
    endfunction
  endclass
```

Like in the earlier example, two objects are created and contents of one is copied into another. Because `do_print` method is not implemented and automation macros are not used, `convert2string` will be used to print contents of each class.

```
  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 obj1 = Object::type_id::create("obj1");
    Object obj2 = Object::type_id::create("obj2");
    obj1.randomize();
    `uvm_info("TEST", $sformatf("Obj1.print: %s", obj1.convert2string()), UVM_LOW)
    obj2.randomize();
    `uvm_info("TEST", $sformatf("Obj2.print: %s", obj2.convert2string()), UVM_LOW)

    obj2.copy(obj1);
    `uvm_info("TEST", "After copy", UVM_LOW)
    `uvm_info("TEST", $sformatf("Obj2.print: %s", obj2.convert2string()), UVM_LOW)
  endfunction
endclass

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

Simulation Log

```
ncsim> run
UVM_INFO @ 0: reporter [RNTST] Running test base_test...
UVM_INFO testbench.sv(93) @ 0: uvm_test_top [TEST] Obj1.print:  m_name=obj1
m_bool=TRUE m_mode=0xe m_data[0]=0xf4 m_data[1]=0xe m_data[2]=0x58 m_data[3]=0xbd
UVM_INFO testbench.sv(95) @ 0: uvm_test_top [TEST] Obj2.print:  m_name=obj2
m_bool=FALSE m_mode=0xe m_data[0]=0x60 m_data[1]=0x24 m_data[2]=0x27 m_data[3]=0xb5
UVM_INFO testbench.sv(26) @ 0: reporter [m_pkt] In Packet::do_copy()
UVM_INFO testbench.sv(79) @ 0: reporter [obj2] In Object::do_copy()
UVM_INFO testbench.sv(98) @ 0: uvm_test_top [TEST] After copy
UVM_INFO testbench.sv(99) @ 0: uvm_test_top [TEST] Obj2.print:  m_name=obj1
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 ---
```

Using clone method

`clone` method works exactly the same as a `copy` method, the difference being that a clone will return an object with the copied contents. So this saves some trouble of creating the second object before copy.

```
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);
  	// Create obj1, but only declare handle for obj2
    Object obj2;
    Object obj1 = Object::type_id::create("obj1");
    obj1.randomize();
    `uvm_info("TEST", $sformatf("Obj1.print: %s", obj1.convert2string()), UVM_LOW)

    // Use $cast to clone obj1 into obj2
    $cast(obj2, obj1.clone());
    `uvm_info("TEST", "After clone", UVM_LOW)
    `uvm_info("TEST", $sformatf("Obj2.print: %s", obj2.convert2string()), UVM_LOW)
  endfunction
endclass

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

Simulation Log

```
ncsim> run
UVM_INFO @ 0: reporter [RNTST] Running test base_test...
UVM_INFO testbench.sv(105) @ 0: uvm_test_top [TEST] Obj1.print:  m_name=obj1
m_bool=TRUE m_mode=0xe m_data[0]=0xf4 m_data[1]=0xe m_data[2]=0x58 m_data[3]=0xbd
UVM_INFO testbench.sv(20) @ 0: reporter [m_pkt] In Packet::do_copy()
UVM_INFO testbench.sv(86) @ 0: reporter [obj1] In Object::do_copy()
UVM_INFO testbench.sv(108) @ 0: uvm_test_top [TEST] After clone
UVM_INFO testbench.sv(109) @ 0: uvm_test_top [TEST] Obj2.print:  m_name=obj1
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]
```