2023 DESIGN AND VERIFICATION™ DVCON CONFERENCE AND EXHIBITION UNITED STATES

SAN JOSE, CA, USA
FEBRUARY 27-MARCH 2, 2023

Avoiding Configuration Madness The Easy Way

Rich Edelman

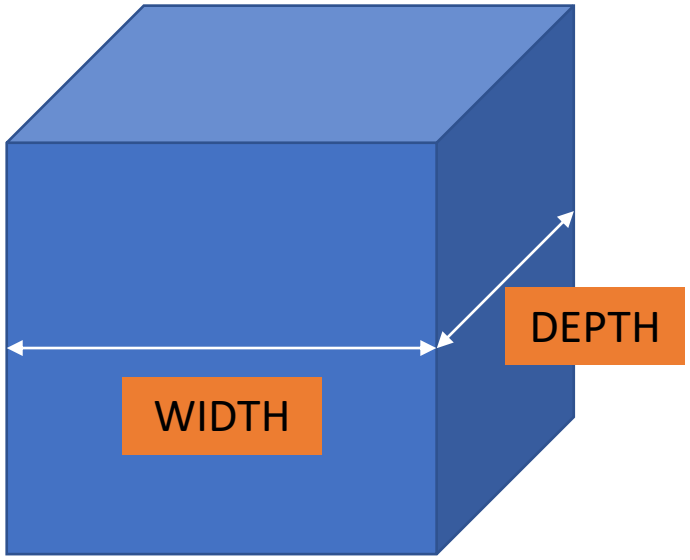
Siemens EDA

SIEMENS

accellera SYSTEMS INITIATIVE

What is Configuration?

- Knobs
- Controls
- Parameters
- Configure a test



Why is it madness?

- Typos in path names
- Wildcard misplacement
- Full paths
- Relative paths
- Debug
  - Really hard
  - Happens once, twice (a few times) and then never again. Not a daily task

Why are we here?

- UVM Config works “fine”
  - Small number of items
  - No wildcards (or few)
  - No apply_config_settings()
  - Don’t depend on type matching (hard to debug)
  - Don’t depend on precedence
    - At build_phase, higher level has higher precedence, same level is last setting wins
    - After build_phase, last setting wins
- But shouldn’t it be simpler?

What's the current good news / bad news?

- Original goals of UVM Config
  - Typed matching
  - Precedence
  - field_automation_macros
- Works great for a small number of sets and gets
  - Aside from being very sensitive.
    - Typos
    - Wildcards
    - Type matching

What should we do? (Hint: write code)

- The code is large
  - uvm_resource* and uvm_config* ➔ 4-6 files, about 3,000 lines
- The code is clever (& interesting) – time spent
- The code is layered
- The code is quite “integrated”
  - apply_config_settings() / **field automation**

Field Automation Macros

- Set a bunch
- MAGIC in super.build_phase()

```
class A extends uvm_component;
  int WIDTH;
  int DEPTH;
  bit[31:0]foo;
  `uvm_component_utils_begin(A)
    `uvm_field_int(WIDTH, UVM_ALL_ON)
    `uvm_field_int(DEPTH, UVM_ALL_ON)
    `uvm_field_int(foo,   UVM_ALL_ON)
  `uvm_component_utils_end

  function void build_phase(uvm_phase phase);
    super.build_phase(phase);
  endfunction
endclass
```

```
module top();
  int foo;
  initial begin
    uvm_config_db#(int)::set(null, "uvm_test_top.X1.A1", "WIDTH", 12);
    uvm_config_db#(int)::set(null, "uvm_test_top.X1.A1", "DEPTH", 16);
    uvm_config_db#(int)::set(null, "uvm_test_top.X1.A2", "WIDTH", 24);
    uvm_config_db#(int)::set(null, "uvm_test_top.X1.A2", "DEPTH", 32);
    run_test("test");
  end
endmodule
```

Field Automation Macros 2

- Type matching
  - Get for ‘int’ type works.
  - Get for ‘bit[31:0]’ fails. Surprise!

```
class A extends uvm_component;
  int WIDTH;
  int DEPTH;
  bit[31:0]foo;
  `uvm_component_utils_begin(A)
    `uvm_field_int(WIDTH, UVM_ALL_ON)
    `uvm_field_int(DEPTH, UVM_ALL_ON)
    `uvm_field_int(foo,   UVM_ALL_ON)
  `uvm_component_utils_end

  function void build_phase(uvm_phase phase);
    super.build_phase(phase);
  endfunction
endclass
```

Ok. **foo** is 8

Fail. **foo** is not found

```
uvm_config_db#(int)       ::set(null, "uvm_test_top.X1.A2", "foo", 8);
uvm_config_db#(int)       ::get(null, "uvm_test_top.X1.A2", "foo", foo);

uvm_config_db#(bit[31:0])::get(null, "uvm_test_top.X1.A2", "foo", foo);
```

```
module top();
  int foo;
  initial begin
    uvm_config_db#(int)::set(null, "uvm_test_top.X1.A1", "WIDTH", 12);
    uvm_config_db#(int)::set(null, "uvm_test_top.X1.A1", "DEPTH", 16);
    uvm_config_db#(int)::set(null, "uvm_test_top.X1.A2", "WIDTH", 24);
    uvm_config_db#(int)::set(null, "uvm_test_top.X1.A2", "DEPTH", 32);
    run_test("test");
  end
endmodule
```

Solution 1 – Don’t use it

- UVM Config DB is a global lookup table by name
  - Replace it with a GLOBAL
- Put control knobs in some global variable(s)
  - Integers
  - Classes
  - Structs

- But now every component must know those variable names

```
int global_i;

class C;
  int i;

  function new();
    i = global_i;
  endfunction
endclass

module top();
  initial begin
    global_i = 5;
  end
endmodule
```

Solution 2 – Use it. But just once.

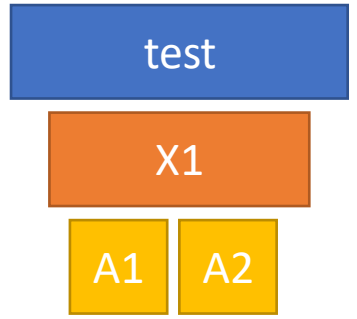
- Limited usefulness.
- You only get to set ONE thing ONCE.
  - Do 1 set
  - Do 1 get
- Use a configuration class to make it more appealing.

```
module top();
  initial begin
    uvm_config_db#(my_tb_config_t)::set(null, "uvm_test_top", "CONFIG", tb_config);
    run_test("test");
  end
endmodule
```

Solution 3 – Use it. But just a **few** times.

- You get the idea.

Solution 4 – Use it. Structure your configs




```
class my_A_config;
  int WIDTH;
  int DEPTH;
endclass

class my_B_config;
  int WIDTH;
  int DEPTH;
endclass

class my_X_config;
  my_A_config A1_config;
  my_A_config A2_config;
endclass
```

Hierarchy of Configurations

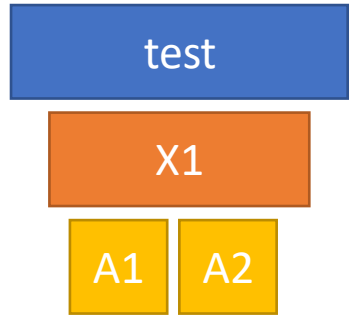
```
module top();
  my_X_config X_config;

  initial begin
    X_config = new();
    X_config.A1_config = new();
    X_config.A2_config = new();
    X_config.A1_config.WIDTH = 12;
    X_config.A1_config.DEPTH = 16;
    X_config.A2_config.WIDTH = 24;
    X_config.A2_config.DEPTH = 32;

    uvm_config_db#(my_X_config)::set(null, "*", "config", X_config);

    run_test("test");
  end
endmodule
```

Construct the configs

Fill in the values

Use the config_db once

Solution 4 – Use it. Structure your configs




```
class test extends uvm_test;
  X X1;
  my_X_config X_config;

   function void build_phase(uvm_phase phase);
     X1 = new("X1", this);
     uvm_config_db#(my_X_config)::get(null, "uvm_test_top", "config", X_config));
     X1.X_config = X_config;
  endfunction
endclass
```

Fetch the config in test

Assign the config (test➔ X1)

```
class X extends uvm_component;
  A A1, A2;
  my_X_config X_config;

  function void build_phase(uvm_phase phase);
    A1 = new("A1", this);
    A2 = new("A2", this);
    A1.A_config = X_config.A1_config;
    A2.A_config = X_config.A2_config;
  endfunction
endclass
```

Assign the config (X1➔ A1 & A2)

```
class A extends uvm_component;
  int WIDTH;
  int DEPTH;
  my_A_config A_config;

  function void connect_phase(uvm_phase phase);
    WIDTH = A_config.WIDTH;
    DEPTH = A_config.DEPTH;
  endfunction
endclass
```

Use the config'ed values

Solution 5 – Use something different

- Goals
  - Same set() and get() API
  - All configs extend from a common base class
  - No precedence algorithm
    - two settings is an error? warning?
  - Lookup algorithm does NOT change based on phase
  - Simple and transparent
  - Fast
  - Small (340 lines)

config_item ➔ Base class

```
class my_config extends config_item;
  function new(string name = "my_config");
    super.new(name);
  endfunction

  int value;

  virtual function string convert2string();
    ...
  endfunction
endclass
```

The configurable values – the knobs

Performance?

- 10,000 set operations
- 10,000 get operations

With Wildcards	Elapsed Time (sec)	Calls to uvm_re_match()
UVM Config	900	99,990,000
New Config	4	265,963

Without Wildcards	Elapsed Time (sec)	Calls to uvm_re_match()
UVM Config	360	99,990,000
New Config	3	3

```
config_db#(T)::get(null, name, "ID", value)
value = table_of_fields[field_name][instance_name][0];
```

NEW Config

```
uvm_config_db#(T)::get(null, name, "ID", value)
rq = lookup_regex_names(inst_name, field_name, uvm_resource#(T)::get_type()))
r = uvm_resource#(T)::get_highest_precedence(rq)
    rb = rp.get_highest_precedence(tq)
    $cast(rsrc, rb)
    return rsrc
value = r.read(cntxt)
```

UVM Config

The config_item base class

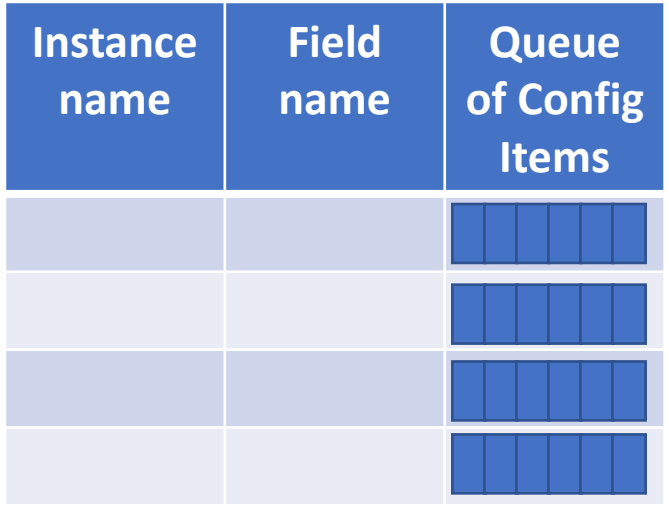
- All configs are derived from a single base class
- A configuration “extends” the base class and adds values/knobs as needed
- Each config has a handle to a list of accessors.
  - Each access is recorded
    - Kind: Get or Set
    - File and line number where the get / set was called

```
typedef struct {
   string kind; // GET or SET
   string file_name;
   int line_number;
 } accessor_t;
```

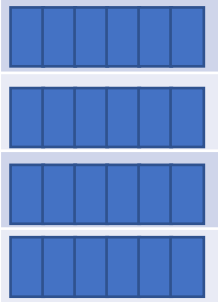
```
class config_item;
  accessor_t accessor_q[$];
  string name;
  virtual function string get_name();
    return name;
  endfunction
  function new(string name = "config_item");
    this.name = name;
  endfunction
  virtual function string convert2string();
    return $sformatf("%p", accessor_q);
  endfunction
endclass
```

The config_db

- The database is a associative array of config_items, indexed by instance path and field name (property name).
- Furthermore, there could be multiple values for the same (instance path, property name)
  - Isn’t this an error?
  - So, there is a list at [instance name, field name]



The config_db

Instance name	Field name	Queue of Config Items
		

```
typedef config_item table_of_fields_t[string][string][$];
```

```
class config_db extends uvm_object;
  `uvm_object_utils(config_db)
  table_of_fields_t table_of_fields;

  function void set(uvm_component scope, string instance_name, string field_name, config_item c,
     string file_name, int line_number);
  endfunction

  function config_item get(uvm_component scope, string instance_name, string field_name,
     string file_name, int line_number);
  endfunction

  function void print();
  endfunction
endclass
```

Wildcards!

- Can’t look that up in an associative array

Config Database + Container

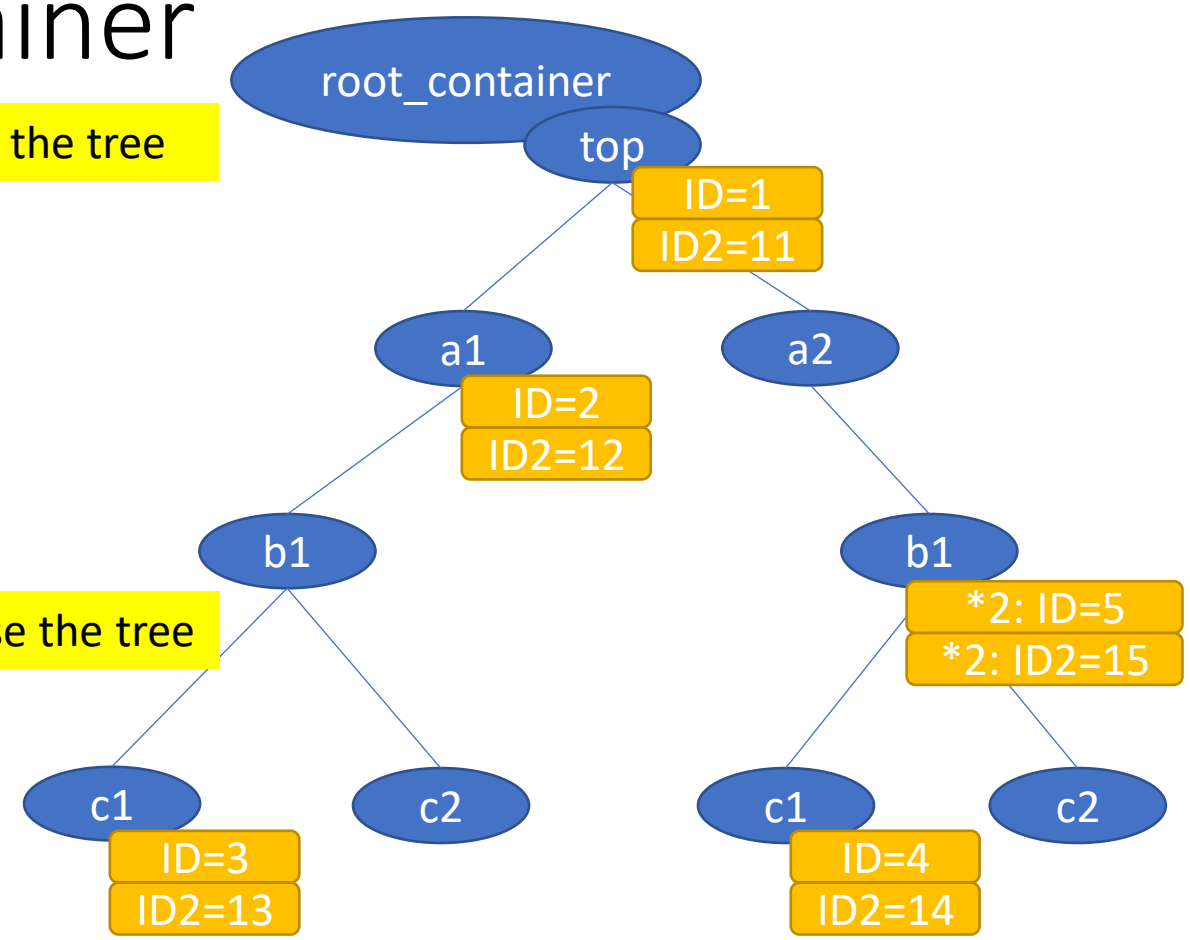
Instance name	Field name	Queue of Config Items

Make the tree

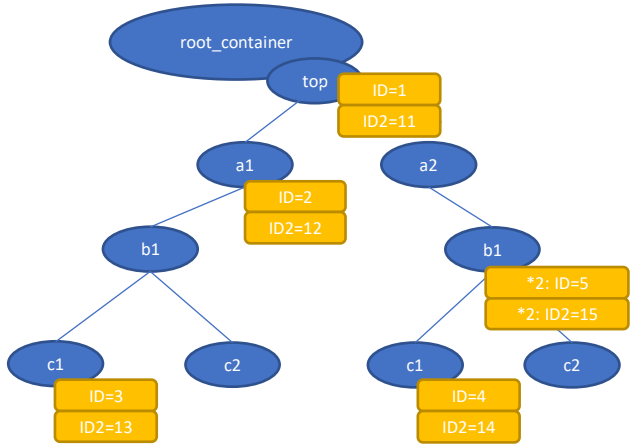
```
set("top", "ID", 1);
set("top", "ID2", 11);

set("top.a1", "ID", 2);
set("top.a1", "ID2", 12);

set("top.a1.b1.c1", "ID", 3);
set("top.a1.b1.c1", "ID2", 13);

set("top.a2.b1.c1", "ID", 4);
set("top.a2.b1.c1", "ID2", 14);

set("top.a2.b1.*2", "ID", 5);
set("top.a2.b1.*2", "ID2", 15);
```

Traverse the tree

```
get("top", "ID");
get("top.a1", "ID"););
get("top.a1.b1", "ID"););
get("top.a1.b1.c1", "ID"););
get("top.a1.b1.c2", "ID"););

get("top.a2", "ID"););
get("top.a2.b1", "ID"););
get("top.a2.b1.c1", "ID"););
get("top.a2.b1.c2", "ID"););

get("top.a2.b1.abcxyz2", "ID"););
```



Container



- The container is-an instance data structure
- Each node has a table indexed by field name

Field name	Queue of Config Items

```
typedef config_item     config_item_q_t[$];                // A queue of config items.
typedef config_item_q_t table_of_field_configs_t[string]; // An array of lists of config items,
                                                          //   index by field name
```

Instance kind of data

```
class container;
  string name; // instance name at this level
  container list_of_children[string];
  container parent;

  table_of_field_configs_t configs_for_a_field;
```

Table of fields

The set() routine

```
function void set(uvm_component scope, string instance_name,  string field_name,
                  config_item c, string file_name, int line_number);
  container container_h;

  table_of_fields[field_name][instance_name].push_front(c);

  container_h = root_container.make_container(instance_name);

  container_h.configs_for_a_field[field_name].push_front(c);

  add_accessor("SET", c, file_name, line_number);

endfunction
```

The get() routine

```
function config_item get(uvm_component scope, string instance_name, string field_name,
                                              input string file_name, int line_number);
  if ((table_of_fields.exists(field_name) == 0) ||
      (table_of_fields[field_name].exists(instance_name) == 0)) begin
    c = null;
  end
  else if (table_of_fields[field_name][instance_name].size() > 0)  begin
    c = table_of_fields[field_name][instance_name][0];
  end
  if (c == null) begin
    container container_ h = root_container.find_container(instance_name);
    if (container_h)
      c = container_h.configs_for_a_field[field_name][0];
  end
  add_accessor("GET", c, file_name, line_number);
endfunction
```

Recording accesses

- The accessor provides the callers' file and line number

```
db.set(null, "top.a.b.*",             "SPEED", my_speed_config, `__FILE__, `__LINE__)
db.get(null, "top.a.b.c.d.monitor1", "SPEED",                   `__FILE__, `__LINE__)
```

```
`define CONFIG_SET(db, scope, instance_name, field_name, value) db.set(scope, instance_name, field_name, value, `__FILE__, `__LINE__)
`define CONFIG_GET(db, scope, instance_name, field_name, value) db.get(scope, instance_name, field_name, value, `__FILE__, `__LINE__)
```

```
my_speed_config_t my_speed_config;

`CONFIG_SET(db, null, "top.a.b.*", "SPEED", my_speed_config)
```

```
my_speed_config_t speedconfig;

$cast(speedconfig, `CONFIG_GET(db, null, "top.a.b.c.d.monitor1", "SPEED"))
```

Printing the database

Instance name	Field name	Queue of Config Items

```
function void print();
  foreach (table_of_fields[field_name,instance_name]) begin
    config_item_q_t config_item_q;
    config_item_q = table_of_fields[field_name][instance_name];
    for (int i = 0; i < config_item_q.size(); i++) begin
      config_item ci;
      ci = config_item_q[i];
      $display("%s", ci.convert2string());
      for (int j = 0; j < ci.accessor_q.size(); j++) begin
        accessor_t accessor;
        accessor = ci.accessor_q[j];
        $display(...);
      end
    end
  end
endfunction
```

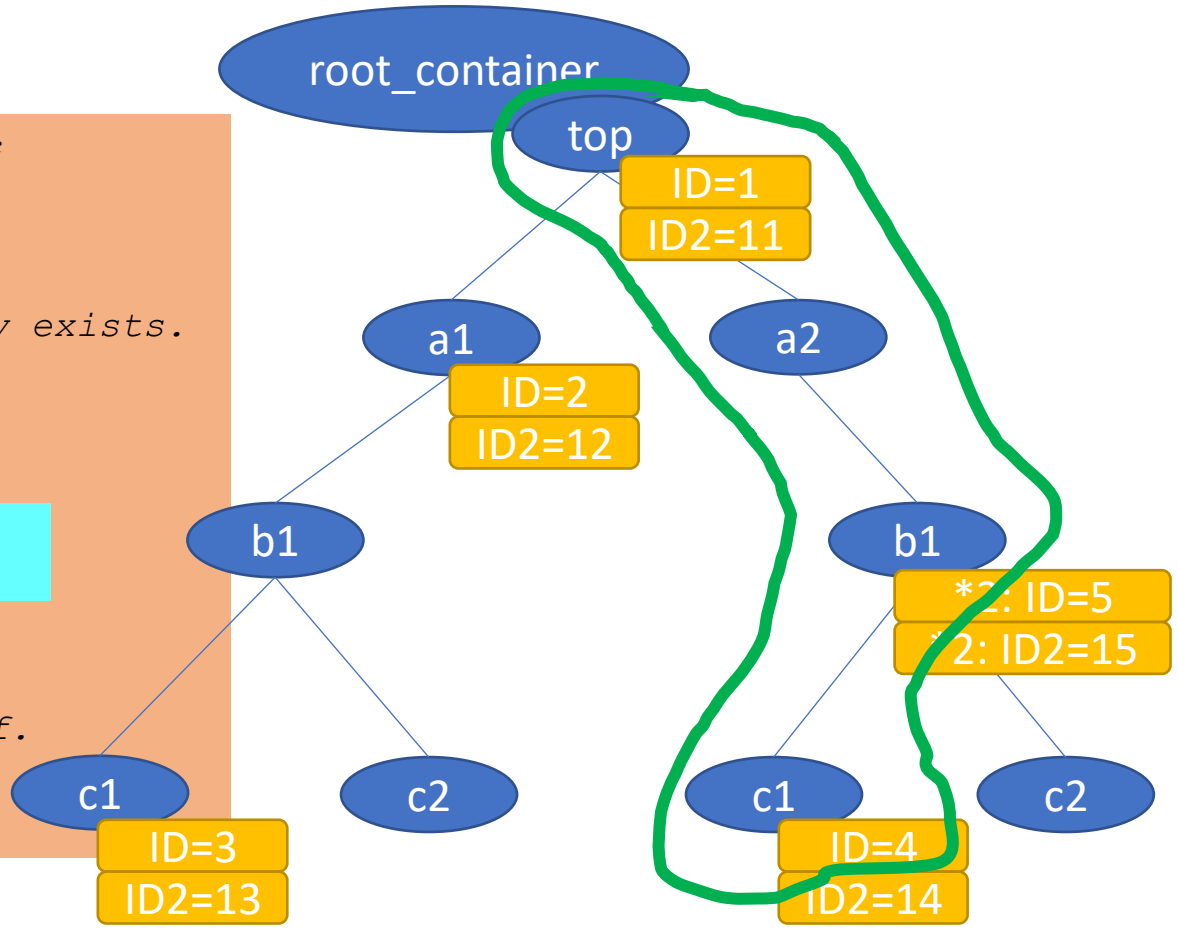
```
class config_item;
  accessor_t accessor_q[$];
  string name;
  ...
  virtual function string convert2string();
    return $sformatf("%p", accessor_q);
  endfunction
endclass
```

Make_Container()

```
set("top.a2.b1.c1", "ID", 4);
```

```
function  container make_container(string full_path_name);
  string_q q;
  container c;
  q = path_name_to_q(full_path_name);
  if (list_of_tops.exists(q[0])) begin // This top already exists.
    c = list_of_tops[q[0]];
  end
  else begin // This top doesn't exist.
    c = add(q[0]);
    list_of_tops[q[0]] = c;
  end
  for (int i = 1; i < q.size(); i++)
    c = c.add(q[i]);
  // Return the BOTTOM container - where we'll stuff stuff.
  return c;
endfunction
```

Tree builder



Find_Container() – in Words

- So boring

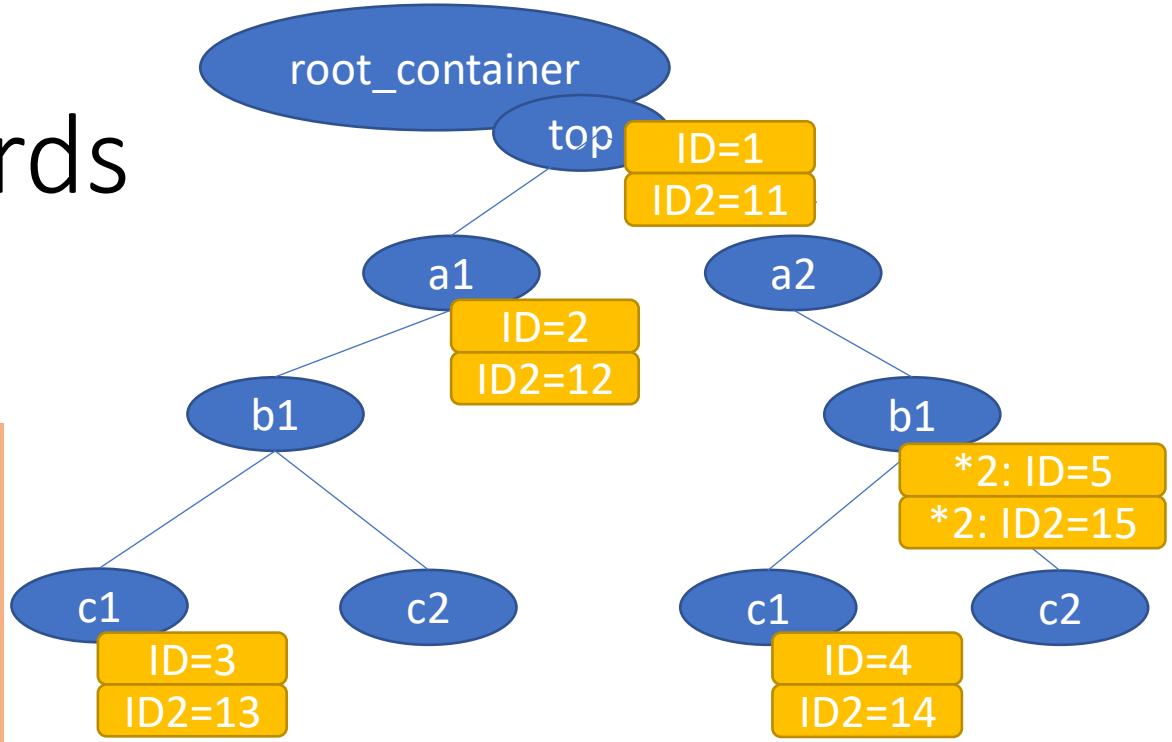
If the initial, simple associative array lookup fails, then the container tree is traversed, matching the parts of the instance path. By trying to match each element in the hierarchical path, the search list is shortened.

First find_container() tries to find a container match in the tree. Down at the bottom. It uses regular expression
matching, and parses the instance name into elements. It continues down the tree until there is no match or until the
entire instance name has been matched. If there is no match, it simply means this get() will fail – it will not find a
setting, since no set() created this instance name.

If a container is found, then it checks to see if a field (property) exists. If it does not exist, then it climbs up the
tree. This climbing the tree represents inheritance.

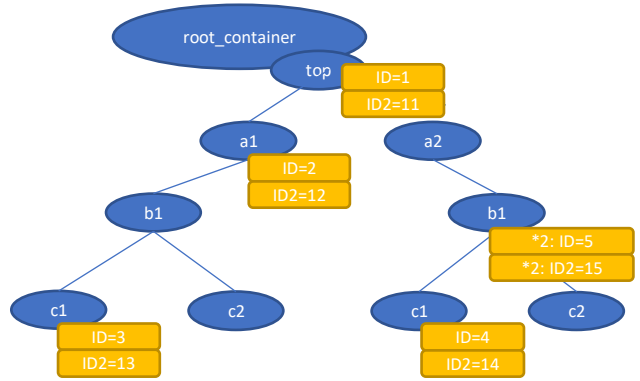
If a lower-level instance name does not have a field set, then climb up the tree to find a setting. The climbing the tree allows finding the higher-level settings or higher wildcards.

The settings that control an entire sub-tree.

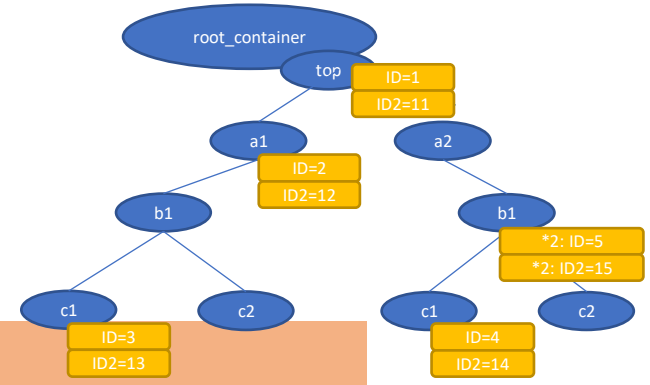


Find_Container() 1

```
typedef string string_q[$]; // A path name gets turned into a queue of names

function  container find_container(string full_path_instance_name);
  bit done;
  string_q q;
  container new_c, container_h;

  q = path_name_to_q(full_path_instance_name);
  new_c = null;
  if (list_of_tops.exists(q[0])) begin
    for (int i = 1; i < q.size(); i++) begin
      done = 0;
      if (i==1) container_h = list_of_tops[q[0]];
      else      container_h = new_c;

      // 1. Lookup the name.
      new_c = container_h.get(q[i]);
```

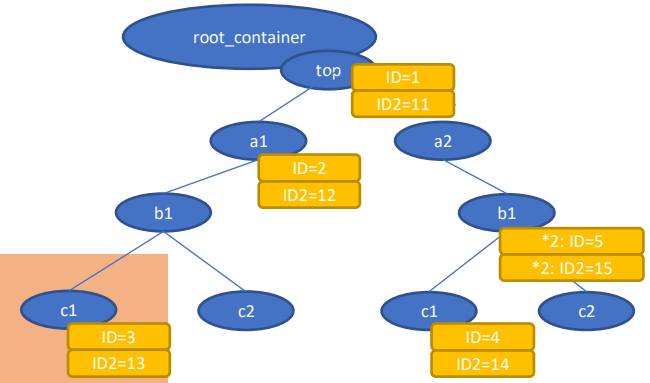


Find_Container() 2




```
// 2. Try to regex match ANY of the containers here
      if (new_c == null) begin
        // The NAME isn't found. Do any of the container children regex match?
        foreach (container_h.list_of_children[s]) begin
          number_of_regex_tries++;
          if (uvm_re_match(uvm_glob_to_re(s), q[i]) == 0) begin
            // Matched. Need to keep looping. We matched a child. Traverse it. Not done yet.
            new_c = container_h.list_of_children[s];
            // Does this new container itself have children? If not there's no hope...
            if (new_c.list_of_children.size() > 0) // If > 0, then there are children
              break;
            else
              new_c = null; // There are no children. Keep looping here.
          end
        end
      end
```

Find_Container() 3




```
      // 3. Try the FULL string
      if (new_c == null) begin
        string fstring = "";
        // OK. We went through the for loop, but no child matched.
        // Can we match the whole string against any of the "children"?
        for (int j = i; j < q.size(); j++)
          if (fstring == "") fstring = q[j];
          else               fstring = { fstring, ".", q[j]};
        foreach (container_h.list_of_children[s]) begin
          number_of_regex_tries++;
          if (uvm_re_match(uvm_glob_to_re(s), fstring) == 0) begin
            // Matched. Done.
            new_c = container_h.list_of_children[s]; // This is the one.
            done = 1;
            break;
          end
        end
      end
      if (done) break;
    end
  end
  return new_c;
endfunction
```

Examples – Test Programs

SET

```
for (int n_t = 0; n_t < N_t; n_t++)
  for (int n_e = 0; n_e < N_e; n_e++)
    for (int n_a = 0; n_a < N_a; n_a++)
      for (int n_m = 0; n_m < N_m; n_m++) begin
        //name = $sformatf("uvm_test_top.t_%0d.*_%0d.*_%0d.*_%0d", n_t, n_e, n_a, n_m);
          name = $sformatf("uvm_test_top.t_%0d.e_%0d.a_%0d.m_%0d", n_t, n_e, n_a, n_m);
          c = new($sformatf("my_special_config_item%0d", gid));
          c.value = gid++;
          number_of_set_calls++;
          $display("uvm_config_db#(my_special_config_item)::set(null, %s, ID, %0d);",
            name, c.value);
          if (NEW) begin
            config_db_h.`CONFIG_SET(null, name, "ID", c);
            config_db_h.`CONFIG_SET(null, name, "ID2", c);
          end
          else begin
            uvm_config_db#(my_special_config_item)::set(null, name, "ID", c);
            uvm_config_db#(my_special_config_item)::set(null, name, "ID2", c);
          end
        end
```

GET

```
rid = 0;
wid = 0;
for (int n_t = 0; n_t < N_t; n_t++)
  for (int n_e = 0; n_e < N_e; n_e++)
    for (int n_a = 0; n_a < N_a; n_a++)
      for (int n_m = 0; n_m < N_m; n_m++) begin
        name = $sformatf("uvm_test_top.t_%0d.e_%0d.a_%0d.m_%0d", n_t, n_e, n_a, n_m);
        $display("uvm_config_db#(my_special_config_item)::get(null, %s, ID);", name);
        number_of_get_calls++;
        if (NEW) begin
          $display("DEBUG INFO: Number of regex tries: %0d",
            config_pkg::number_of_regex_tries);
          $cast(read_c, config_db_h.get(null, name, "ID", `__FILE__, `__LINE__));
        end
        else begin
          $display("DEBUG INFO: Number of regex tries: %0d", uvm_re_match_stats());
          uvm_config_db#(my_special_config_item)::get(null, name, "ID", read_c);
        end
        if (read_c == null) begin
          $display("Error: ID not found for name");
        end
        else begin
          rid = read_c.value;
          if (rid != wid) $display("Error: %0d != %0d (%s)", rid, wid, name);
          else            $display("Match: %0d == %0d (%s)", rid, wid, name);
        end
        wid++;
      end
```

Loose Ends – see the appendix

- What About ‘apply_config_settings’? (See the Appendix “dire warning”
  - No attempt was made to integrate to apply_config_settings.
  - It’s a way to splatter values. And field_automation_macros. A bad idea.
- What UVM version does this work with?
  - uvm-1.1d, uvm-1.2, uvm-ieee ➔ It’s UVM independent
- Why not just use uvm_resource_db?
  - It’s part of the problem
- Didn’t uvm-1.2 and uvm-ieee improve things?
  - The code base looks relatively the same. Exercise for the reader

Most of the experiments were UVM-IEEE

Conclusion

- It’s OK to use the UVM Configuration Database. Plenty of people do.
  - Just be careful – it’s like playing with knives. Be careful
- It’s OK to use a different technique
  - Skip it
  - Just one.
  - A few.
  - Use once, hierarchical config
  - Write your own solution

It’s OK to “config” once or twice

It’s OK to write code
It’s OK to update/fix code

324 lines
Fast. Even with regex

Just found this. Surprise! Calling uvm_config_db#(T)::get(...) is not randomly stable
Be careful. But is has been in there since uvm-1.1d…

Questions

- Email rich.edelman@siemens.com for questions

accellera
SYSTEMS INITIATIVE