The configuration database in the UVM is a highly versatile feature that allows the passing of objects and data to various components in the testbench. However, despite its versatility, the configuration database (uvm_config_db) can be a source of great confusion to those verification and design engineers who are trying to learn UVM. The goal of this paper is to demystify the confusion to those verification and design engineers who are trying to learn UVM. The next method that needs to be explored is the get() function which is used to retrieve items from the database. It is important to note that objects are not removed from the database when you call get(). The actual variable is passed in as an inout formal function argument and so is performed as a copy-in-copy-out operation.

### Argument Description

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>uvm_component</td>
<td>The context is the hierarchical starting point of where the database entry is.</td>
</tr>
<tr>
<td>string inst_name</td>
<td>The instance name is the hierarchical path that limits accessibility of the database entry.</td>
</tr>
<tr>
<td>string field_name</td>
<td>The field name is the label used as a lookup for the database entry.</td>
</tr>
<tr>
<td>T value</td>
<td>The value to be stored in the database of the parameterized type. By default the type is int.</td>
</tr>
</tbody>
</table>

The classic example of uvm_config_db usage is with sharing a virtual interface. A SystemVerilog interface is instantiated at the top level and now needs to be added to the uvm_config_db using the set() function.

```verilog
class uvm_config_db::set
________________________

static function void set(uvm_config db, string inst_name, string field_name, T value)
________________________

uvm_config_db(set, virtual int, inst_name, field_name, value)
```

In the following diagram, three different items have been added to the uvm_config_db: a virtual interface, an integer value, and a configuration object. Also, there is a generic calls to the set() and get() functions. To retrieve the integer value the label would be "retry_count" and the value stored in this entry would be assigned to the rty_cnt property in the object that is calling the get() function.

### Debug

The biggest source of bugs is due to the fact that many of the arguments to resource and configuration database methods are of type string. This means that typos in the actual arguments cannot be detected at compile time, but must wait until a test is actually run. Fortunately, there are debugging facilities available to help find the source of these problems. Two run-time options are available which can be used to turn on tracing of every write and read access to the databases.

```
+UVM_TESTNAME=my_test
+UVM_CONFIG_DB_TRACE
```

In this paper we demystify the use of the UVM’s resource and configuration databases. These are powerful facilities that are available to testbench writers that help with the configuration of the testbench itself as well as provide a repository for parameters that represent values required by different parts of the environment.

All of the code examples in this paper were from “Getting Started with UVM: A Beginner’s Guide” by Vanessa Cooper and published by Verilab Publishing. Copies of the code will be made available on request to the authors.