

# VMM Performance Analyzer User Guide

---

VMM Performance Analyzer Version 1.1  
December 2009



# Copyright Notice and Proprietary Information

Copyright © 2009 Synopsys, Inc. All rights reserved. This software and documentation contain confidential and proprietary information that is the property of Synopsys, Inc. The software and documentation are furnished under a license agreement and may be used or copied only in accordance with the terms of the license agreement. No part of the software and documentation may be reproduced, transmitted, or translated, in any form or by any means, electronic, mechanical, manual, optical, or otherwise, without prior written permission of Synopsys, Inc., or as expressly provided by the license agreement.

## Right to Copy Documentation

The license agreement with Synopsys permits licensee to make copies of the documentation for its internal use only. Each copy shall include all copyrights, trademarks, service marks, and proprietary rights notices, if any. Licensee must assign sequential numbers to all copies. These copies shall contain the following legend on the cover page:

This document is duplicated with the permission of Synopsys, Inc., for the exclusive use of \_\_\_\_\_ and its employees. This is copy number \_\_\_\_\_.”

## Destination Control Statement

All technical data contained in this publication is subject to the export control laws of the United States of America. Disclosure to nationals of other countries contrary to United States law is prohibited. It is the reader's responsibility to determine the applicable regulations and to comply with them.

## Disclaimer

SYNOPSIS, INC., AND ITS LICENSORS MAKE NO WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, WITH REGARD TO THIS MATERIAL, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE.

## Registered Trademarks (®)

Synopsys, AMPS, Cadabra, CATS, CRITIC, CSim, Design Compiler, DesignPower, DesignWare, EPIC, Formality, HSIM, HSPICE, iN-Phase, in-Sync, Leda, MAST, ModelTools, NanoSim, OpenVera, PathMill, Photolynx, Physical Compiler, PrimeTime, SiVL, SNUG, SolvNet, System Compiler, TetraMAX, VCS, Vera, and YIELDirector are registered trademarks of Synopsys, Inc.

## Trademarks (™)

AFGen, Apollo, Astro, Astro-Rail, Astro-Xtalk, Aurora, AvanWaves, Columbia, Columbia-CE, Cosmos, CosmosEnterprise, CosmosLE, CosmosScope, CosmosSE, DC Expert, DC Professional, DC Ultra, Design Analyzer, Design Vision, DesignerHDL, Direct Silicon Access, Discovery, Encore, Galaxy, HANEX, HDL Compiler, Hercules, Hierarchical Optimization Technology, HSIMplus, HSPICE-Link, iN-Tandem, i-Virtual Stepper, Jupiter, Jupiter-DP, JupiterXT, JupiterXT-ASIC, Liberty, Libra-Passport, Library Compiler, Magellan, Mars, Mars-Xtalk, Milkyway, ModelSource, Module Compiler, Planet, Planet-PL, Polaris, Power Compiler, Raphael, Raphael-NES, Saturn, Scirocco, Scirocco-i, Star-RCXT, Star-SimXT, Taurus, TSUPREM-4, VCS Express, VCSi, VHDL Compiler, VirSim, and VMC are trademarks of Synopsys, Inc.

## Service Marks (SM)

MAP-in, SVP Café, and TAP-in are service marks of Synopsys, Inc.

SystemC is a trademark of the Open SystemC Initiative and is used under license.  
ARM and AMBA are registered trademarks of ARM Limited.  
Saber is a registered trademark of SabreMark Limited Partnership and is used under license.  
All other product or company names may be trademarks of their respective owners.

# Contents

---

## 1 Introduction

## 2 Analyzing Performance of a Resource

- Defining Performance ..... 3
  - Collected Data ..... 4
- Data Collection ..... 5
  - Tenures ..... 8
- SQL Schema ..... 9
  - Runs Table ..... 11
  - Data Table List Table ..... 12
  - vmm\_perf\_analyzer Data Table ..... 12
- Linking SQL with VCS ..... 13
  - SQLite ..... 14
  - ASCII Database ..... 15
- User Data ..... 15
  - User-Defined Data Table ..... 15
  - User-Defined Tenure Data ..... 16

## A Performance Analysis Support Classes

- Class Summary ..... 20
- vmm\_perf\_analyzer ..... 21
  - Summary ..... 21

vmm_perf_analyzer::new()	22
vmm_perf_analyzer::log	24
vmm_perf_analyzer::now()	26
vmm_perf_analyzer::start_tenure()	28
vmm_perf_analyzer::suspend_tenure()	30
vmm_perf_analyzer::resume_tenure()	32
vmm_perf_analyzer::end_tenure()	34
vmm_perf_analyzer::abort_tenure()	36
vmm_perf_analyzer::add_tenure()	38
vmm_perf_analyzer::psdisplay()	40
vmm_perf_analyzer::report()	42
vmm_perf_analyzer::get_db()	44
vmm_perf_analyzer::save_db()	46
vmm_perf_analyzer::save_db_txt()	48
vmm_perf_analyzer::reset()	50
vmm_perf_analyzer_callbacks::analyze_tenure()	52
vmm_perf_analyzer::append_callback()	54
vmm_perf_analyzer::prepend_callback()	56
vmm_perf_analyzer::unregister_callback()	58
vmm_perf_tenure	60
Summary	60
vmm_perf_tenure::new()	61
vmm_perf_tenure::get_tenure_id()	63
vmm_perf_tenure::get_initiator_id()	65
vmm_perf_tenure::get_target_id()	67
vmm_perf_tenure::get_tr()	69
vmm_perf_tenure::psdisplay()	71

## **B SQL Database Support Classes**

Class Summary	74
vmm_sql_db	75
Summary	75
vmm_sql_db::log	76
vmm_sql_db::status()	77
vmm_sql_db::get_dbname()	78

vmm_sql_db::get_info()	79
vmm_sql_db::statement()	81
vmm_sql_db::get_table_names()	82
vmm_sql_db::create_table()	84
vmm_sql_db::get_table()	86
vmm_sql_db::commit()	88
vmm_sql_db::close()	90
vmm_sql_db_ascii	92
Summary	92
vmm_sql_db_ascii::new()	93
vmm_sql_db_sqlite	94
Summary	94
vmm_sql_db_sqlite::new()	95
vmm_sql_db_mysql	96
Summary	96
vmm_sql_db_mysql::new()	97
vmm_sql_table	99
Summary	99
vmm_sql_table::get_tblname()	100
vmm_sql_table::get_db()	102
vmm_sql_table::insert()	104



# 1

## Introduction

---

In many designs, it is important to analyze and report on the performance of a shared resource utilization. The shared resource may be a bus, a memory, a DMA channel or any other design element that is used by more than one initiator over a period of time.

As a generic VMM package, the Performance Analyzer (PAN) is not based on nor requires specific shared resources, transactions or hardware structures. It can be used to collect statistical coverage metrics relating to the utilization of a specific shared resource.

Performance is analyzed based on user-defined atomic resource utilization called *tenures*. A tenure refers to any activity on a shared resource with a well-defined starting and ending point. A tenure is uniquely identified by an automatically-assigned identifier. A tenure may also identify a specific initiator or a specific target or both an initiator and a target.

For example, a tenure on a SoC bus could represent a write transaction from a specific master to a specific slave.

Tenures may overlap if the shared resource can handle more than one operation concurrently. Usually, tenures are mutually exclusive but can be suspended and later resumed to allow a higher-priority tenure to access the shared resource.

Performance data is stored in an SQL database, one database per simulation run. Performance metrics can then be subsequently generated from the SQL database. Performance metrics can be reported for a single simulation only or they can be generated from a database merged from multiple databases from other simulations. Please refer to [“SQL Schema” on page 9](#) for a detailed description of the SQL schema used to store performance data.

# 2

## Analyzing Performance of a Resource

---

This chapter describes how to measure the performance of a resource. The examples shown in this chapter are taken from the example that can be found in `${VMM_HOME}/sv/perf/examples/tl_bus`. The classes used to implement the measurement are specified in [Appendix A, "Performance Analysis Support Classes"](#).

---

### Defining Performance

The first thing you need to know is exactly what you are analyzing. You can use the VMM Performance Analysis package to measure and analyze many different performance aspects of a design. At its core, it is a simple data collection engine that gives you the ability to perform post-simulation statistical analysis and plotting of the collected data. It is your decision what this data represents and how it conveys an analysis of a performance aspect of the design.

Performance is measured for a specific resource (e.g. the entire design, a specific arbiter, a memory or a bus). Individual transactions, requests or transformations are performed on that resource. These individual transactions, requests or transformations are called "tenures". The performance for the resource may be how it is utilized by the combination of all the tenures performed on it. The performance may also be how fast the resource is able to perform the combination of all the tenures.

For example, measuring the performance of an on-chip bus would involve measuring how long it takes for each bus access. A tenure would thus correspond to a READ or WRITE cycle on that bus. When measuring the performance of an arbiter, a tenure would correspond to a request-grant operation.

Data is collected for each individual tenure that is performed on the resource whose performance is being measured. Performance is analyzed in a post-processing step, using the collected data to compute a specific metric (such as an average) or plot a specific graph (such as latency vs arrive rate).

---

## **Collected Data**

For each tenure, the Performance Analysis package collects the following data:

- Start and end time
- Who initiated it and who was the target
- How long it was active (end time minus start time minus time spent suspended)
- If it was aborted or completed

For an on-chip bus, the start and end time would correspond to the time the READ or WRITE was started and eventually completed, respectively. The initiator identity would be an integer identifying the bus master. The target identity would be an integer identifying the targeted slave.

For an arbiter, the start time would correspond to the time when the request line is asserted. The end time would correspond to the time when the corresponding grant line is asserted or the request is removed. The latter might be marked as an aborted tenure. The initiator identity would be an integer identifying the requestor. There would not be any specific target identified as the target is the arbiter in all cases.

---

## Data Collection

Data collection is the first step when analyzing the performance of a resource. Data is collected for each resource in a separate instance of the `vmm_perf_analyzer` class. It is a good idea to locate these instances in the extension of the `vmm_env` class implementing your verification environment. These instances should be allocated in the *build* phase.

For example, two instances of the `vmm_perf_analyzer` class would be required to analyze the performance of the on-chip bus and arbiter in the same design.

### *Example 2-1 Per-Resource Analyzer*

```
class tb_env extends vmm_env;
    ...
    vmm_sql_db_sqlite db;
    vmm_perf_analyzer bus_perf;
    vmm_perf_analyzer arb_perf;
```

```

...
virtual function void build();
    super.build();

    this.db = new("perf_data.db");
    this.bus_perf = new("Bus", this.db);
    this.arb_perf = new("Arb", this.db);
endfunction: build
...
endclass: tb_env

```

Upon completion of the simulation, any buffered data must be written to the database using the [vmm\\_perf\\_analyzer::save\\_db\(\)](#) method. Optionally, basic performance statistics can be displayed using the [vmm\\_perf\\_analyzer::report\(\)](#) method.

### *Example 2-2 Completing Data Collection*

```

class tb_env extends vmm_env;
...
virtual task report();
    this.bus_perf.save_db();
    this.arb_perf.save_db();

    this.bus_perf.report();
    this.arb_perf.report();
    super.report();
endtask: build
...
endclass: tb_env

```

Because the collection of performance data may impact the runtime performance of the simulation, it is recommended that you make the data collection optional. A compile-time or runtime command line should be provided to explicitly turn on performance collection. If the switch is not specified, the performance data collection infrastructure should not be created.

### *Example 2-3 Compile-Time Optional Performance Analysis*

```
class tb_env extends vmm_env;
...
virtual function void build();
    super.build();
`ifdef PERF_ANALYSIS
    this.db = new("perf_data.db");
    this.bus_perf = new("Bus", this.db);
    this.arb_perf = new("Arb", this.db);
`endif
endfunction: build
...
endclass: tb_env
```

### *Example 2-4 Runtime Optional Performance Analysis*

```
class tb_env extends vmm_env;
    bit do_perf = 0;
...
function new();
...
    do_perf = $test$plusargs("perf_analysis");
endfunction: new
...
virtual function void build();
    super.build();
    if (this.do_perf) begin
        this.db = new("perf_data.db");
        this.bus_perf = new("Bus", this.db);
        this.arb_perf = new("Arb", this.db);
    end
endfunction: build
...
endclass: tb_env
```

---

## Tenures

Operations that consume the resource must be mapped into individual tenures. There must be one instance of the `vmm_perf_tenure` class for each operation that is performed on the bus. Tenures are associated with the instance of the `vmm_perf_analyzer` class that corresponds to the resource operated on.

The start and end times of a tenure is identified by calling the `vmm_perf_analyzer::start_tenure()` and `vmm_perf_analyzer::end_tenure()` methods respectively.

How and where a tenure is created and updated depends on the capabilities of the verification environment and of the transactors it contains. Much like functional coverage sampling, they can be created and updated by monitoring channels, notifications or callbacks.

For example, a tenure on an on-chip bus can be tracked by grabbing the bus transaction descriptor at the exit of the transaction generator then waiting for the `vmm_data::STARTED` and `vmm_data::ENDED` indications.

### *Example 2-5 Creating and Managing a Tenure*

```
class bus_tr_tenure extends bus_tr_atomic_gen_callbacks;
    ...
    virtual task post_inst_gen(bus_tr_atomic_gen gen,
                              bus_tr          tr,
                              ref bit         drop);

    fork
        begin
            vmm_perf_tenure tenure = new(...);
            tr.notify.wait_for(vmm_data::STARTED);
            env.bus_perf.start_tenure(tenure);
        end
    endfork;
```

```
        tr.notify.wait_for(vmm_data::ENDED);
        env.bus_perf.end_tenure(tenure);
    end
    join_none
    endtask: post_inst_gen
endclass: bus_tr_tenure
```

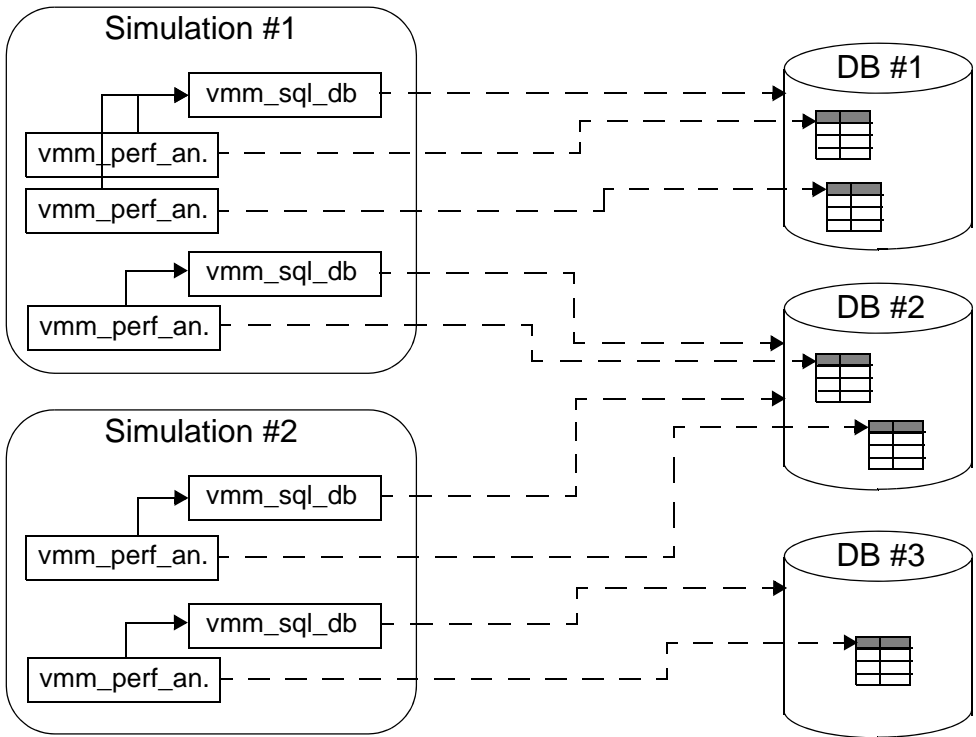
---

## SQL Schema

An instance of the [vmm\\_perf\\_analyzer](#) class will store all performance data collected during a single simulation in a single data table.

When instantiating a performance analyzer object, you choose which database the data table will be created in by specifying an instance of the [vmm\\_sql\\_db](#) class. The data table will be created in the database associated with the access class specified at the creation of the performance analyzer object. As illustrated in [Figure 2-1](#), a single simulation can thus create multiple tables (one per [vmm\\_perf\\_analyzer](#) instance) in one or more databases (one per [vmm\\_sql\\_db](#) instance).

Figure 2-1 Simulations and Databases



Data tables are named after the name of the [vmm\\_perf\\_analyzer](#) class instance with which they are associated. Care must be taken that data tables be uniquely named within the same database. If a different database is used for each data table, care must be taken that the databases be uniquely named across different simulation running concurrently on a CPU farm.

Merging operations will likely be easier if a single database is used for all of the data tables for a single simulation run.

---

## Runs Table

The SQL access layer creates in each database a table named "vmm\_runs" that contains one row for each simulation run whose data is included in the database. There is only one runs table in a database. The runs table is the top-level table that is used to navigate the performance data stored in a database. [Table 2-1](#) describes the schema used for the runs table.

*Table 2-1 Runs Table Schema*

<b>Name</b>	<b>Type</b>	<b>Description</b>
hostname	VARCHAR(255)	Hostname where simulation was run.
utime	INTEGER	Unix timestamp of the system real clock time when the first DB was opened, specified as the number of seconds elapsed since 00:00:00 on January 1, 1970 (UTC)
systime	VARCHAR(30)	The local system real clock time when the first DB was opened formatted as "YYYYMMDD HH::MM::SS".
tzone	SMALLINT	Decimal representation of the HHMM (e.g. -730 for -7:30) offset between local time and UTC [-1200..1200].
seed	INTEGER	Initial seed used.
tables	VARCHAR(255)	Name of the data table list table

When merging database data from different runs, the content of the individual runs tables should be concatenated into a single runs table.

---

## Data Table List Table

The SQL access layer maintains a table in each database that enumerates the names of all the data tables created during a specific run in that database. The name of the data table list table is implementation-specific and can be obtained from the runs table entry corresponding to the run. [Table 2-2](#) describes the schema used for the data table list table.

*Table 2-2 Data Table List Table Schema*

Name	Type	Description
tblname	VARCHAR(255)	Data table name.
datakind	TINYINT	0x00=Performance Data, other values are user-defined.

When merging database data from different runs, the merged set of data table list tables is composed of all the data table list tables in each database.

---

## vmm\_perf\_analyzer Data Table

Each instance of the [vmm\\_perf\\_analyzer](#) class creates a data table containing a record of the performance data for each terminated tenure. The name of the data table is specified by the user when

[vmm\\_perf\\_analyzer::new\(\)](#) is called and can be obtained from the data list table. [Table 2-3](#) describes the schema used for the data table.

*Table 2-3 Performance Analyzer Data Table Schema*

Name	Type	Description
tenureID	INTEGER	Tenure identifier.
initiatorID	INTEGER	Tenure initiator identifier. -1 if none.
targetID	INTEGER	Tenure target identifier. -1 if none.
start	BIGINT	Absolute time of tenure start.
end	BIGINT	Absolute time of tenure completion.
active	BIGINT	Amount of time tenure was active.
aborted	TINYINT	0: Tenure completed normally. Was aborted otherwise.
user	user	Additional user-defined columns, as defined in <a href="#">vmm_sql_db::create_table()</a>

When merging database data from different runs, the merged set of data tables is composed of all the data tables in each database.

---

## Linking SQL with VCS

There are many different implementations of SQL-compatible databases. The VMM SQL interface currently supports SQLite. Instruction for installing and using the appropriate SQL implementation are provided in this section.

If neither of these SQL implementation is readily available, a simple-minded ASCII database implementation is available without requiring any SQL code in your VCS simulation. See [vmm\\_sql\\_db\\_ascii](#) for more details.

---

## SQLite

If you use the [vmm\\_sql\\_db\\_sqlite](#) class, it is necessary to include the required SQLite code in your VCS simulation.

This version of the VMM Performance Analysis package has been developed and tested using SQLite 3.3.16 and should also work on slightly older versions. A binary or source distribution can be obtained at:

```
http://www.sqlite.org/download.html
```

Follow the instructions included in the distribution to compile and install SQLite. Ensure that the options used to build SQLite match the configuration of the VCS installation you will be using. For example, the `-m32` option will be required if a 32bit VCS installation is used on a 64-bit machine.

The environment variable `SQLITE3_HOME` should be set to the absolute path of the SQLite installation. This will make it easier to relocate an environment to a different network where the location of the SQLite installation may be different

```
% setenv SQLITE3_HOME /path/to/sqlite3
```

The following files must then be available:

```
${SQLITE3_HOME}/include/sqlite3.h  
${SQLITE3_HOME}/lib/libsqlite3.so
```

The path `${SQLITE3_HOME}/lib` must be added to your `LD_LIBRARY_PATH` environment variable:

```
setenv LD_LIBRARY_PATH ${SQLITE3_HOME}/  
lib:${LD_LIBRARY_PATH}
```

When invoking VCS, you must specify the following command-line options:

```
% vcs ... $VMM_HOME/shared/src/vmm_sqlite_interface.c \  
-CFLAGS -I${SQLITE3_HOME}/include
```

---

## ASCII Database

If you use the [vmm\\_sql\\_db\\_ascii](#) class, it is necessary to include the required implementation support code in your VCS simulation.

When invoking VCS, you must specify the following command-line options:

```
% vcs ... $VMM_HOME/shared/src/vmm_sqltxt_interface.c
```

---

## User Data

If required, it is possible to add user-defined data to the SQL database. Data that is valid for the entire simulation can be added as a user-defined table. Data that is valid for a specific tenure can be added to the performance data table.

---

### User-Defined Data Table

A user-defined table can be created in a SQL database to store information that pertains to the entire simulation. For example, DUT configuration information could be stored in a user-defined table.

A user-defined table is created using the `vmm_sql_db::create_table()` method. The user-defined table will be added to the “Data Table List Table” . The data type and name of each column must be defined as a string of comma-separated names and types.

### *Example 2-6 Creating a User-Defined Data Table*

```
vmm_sql_db_sqlite db = new(...);  
vmm_sql_table cfg_data = db.create_table("cfg",  
    "num_masters INTEGER, num_slaves INTEGER");
```

Data can be added to the table by using the `vmm_sql_table::insert()` method. This method can be called repeatedly to add new rows in the user-defined data table. A table containing configuration information would typically contain only one row. The data is added as a string of comma-separated formatted values corresponding to each column defined at the time of the table creation.

### *Example 2-7 Adding Data to a User-Defined Data Table*

```
cfg_tbl.insert($psprintf("%0d, %0d",  
    this.cfg.bus.n_masters,  
    this.cfg.bus.n_slaves));
```

When creating a table, it is a good idea to ensure that its name will be unique when merged with other databases.

---

## **User-Defined Tenure Data**

User-defined data can be added to the performance data for each tenure. This user-defined data is then stored in the SQL database and resides in the same table row as the other performance data for that tenure. For example, a clock cycle count timestamp information could be stored with each tenure.

User-defined tenure data must be defined when the instance of the [vmm\\_perf\\_analyzer](#) class is created. The data type and name of each additional performance data column must be defined as a string of comma-separated names and types.

### *Example 2-8 Adding User-Defined Performance Data*

```
class tb_env extends vmm_env;
...
vmm_sql_db_sqlite db;
vmm_perf_analyzer bus_perf;
vmm_perf_analyzer arb_perf;
...
virtual function void build();
    super.build();

    this.db = new("perf_data.db");
    this.bus_perf = new("Bus", this.db);
    this.arb_perf = new("Arb", this.db, , , "stamp
BIGINT");
    endfunction: build
...
endclass: tb_env
```

Additional tenure data must be specified when the [vmm\\_perf\\_analyzer::end\\_tenure\(\)](#) method is called. The data is specified as a string of comma-separated formatted values corresponding to each additional column defined at the time of the performance analyzer creation.

### *Example 2-9 Specifying User-Defined Performance Data*

```
this.env.arb_perf.end_tenure(tenure, $sprintf("%0d",
cycle_count));
```



# A

## Performance Analysis Support Classes

---

This appendix provides detailed information about the classes that are used to implement and support performance analysis.

The OpenVera and SystemVerilog classes have identical functionality and features. Therefore, this appendix documents this information together. The heading used to introduce a method uses the SystemVerilog name. The OpenVera name will be identical except for the few cases where a `_t` suffix is appended to indicate that it may be a blocking method.

Usage examples are usually specified in a single language. However, that should not deter users of the other language as they would be almost identical. This appendix provides more, different examples instead of nearly identical examples in each language.

The classes are documented in alphabetical order. The methods in each class are documented in a logical order, where methods that accomplish similar results are documented sequentially. A summary

of all available methods with cross references to the page where their detailed documentation can be found is provided at the beginning of each class specification.

---

## Class Summary

- [vmm\\_perf\\_analyzer](#) ..... page 21
- [vmm\\_perf\\_tenure](#) ..... page 60

## vmm\_perf\_analyzer

This class is a performance analyzer for a single shared resource and a single simulation. There must be an instance of this class for each shared resource whose performance is to be independently analyzed. For example, in a SoC design with two busses, two instances should be used if the performance of each bus is to be independently analyzed.

For guidelines on using this class, see [Chapter 2, "Analyzing Performance of a Resource"](#).

---

### Summary

•	<code>vmm_perf_analyzer::new()</code> .....	page 22
•	<code>vmm_perf_analyzer::log</code> .....	page 24
•	<code>vmm_perf_analyzer::now()</code> .....	page 26
•	<code>vmm_perf_analyzer::start_tenure()</code> .....	page 28
•	<code>vmm_perf_analyzer::suspend_tenure()</code> .....	page 30
•	<code>vmm_perf_analyzer::resume_tenure()</code> .....	page 32
•	<code>vmm_perf_analyzer::end_tenure()</code> .....	page 34
•	<code>vmm_perf_analyzer::abort_tenure()</code> .....	page 36
•	<code>vmm_perf_analyzer::add_tenure()</code> .....	page 38
•	<code>vmm_perf_analyzer::psdisplay()</code> .....	page 40
•	<code>vmm_perf_analyzer::report()</code> .....	page 42
•	<code>vmm_perf_analyzer::save_db()</code> .....	page 46
•	<code>vmm_perf_analyzer::save_db_txt()</code> .....	page 48
•	<code>vmm_perf_analyzer::reset()</code> .....	page 50
•	<code>vmm_perf_analyzer_callbacks::analyze_tenure()</code> .....	page 52
•	<code>vmm_perf_analyzer::append_callback()</code> .....	page 54
•	<code>vmm_perf_analyzer::prepend_callback()</code> .....	page 56
•	<code>vmm_perf_analyzer::unregister_callback()</code> .....	page 58

## vmm\_perf\_analyzer::new()

Create a new instance of a performance analyzer.

### SystemVerilog

```
function new(    string          name,
               vmm_sql_db    db,
               int unsigned max_n_initiators    = 0,
               int unsigned max_n_targets      = 0,
               int unsigned max_n_concurrent   = 1,
               string          user_schema = "")
```

### OpenVera

```
task new(    string          name,
            vmm_sql_db db,
            integer max_n_initiators    = 0,
            integer max_n_targets      = 0,
            integer max_n_concurrent   = 1,
            string  user_schema = "")
```

### Description

Create an instance of a performance analyzer for a specific shared resource with the specified name. The specified name will be used as the name of the table that will contain the performance data collected by this instance. It must be unique within a simulation but similar across different simulation for the same resource. The specified name is also used as the instance name of the message interface found in the [vmm\\_perf\\_analyzer::log](#) class property.

A maximum number of tenure initiators, tenure targets and concurrent tenures can also be specified. By default, only one tenure can exist from an unlimited number of initiators on an unlimited number of targets. If a maximum number of initiators or targets is

specified, an error will be issued if tenures associated with this performance analyzer identify more initiators or targets than the analyzer supports. This is an absolute maximum number of initiators or targets, not the maximum value for the initiator or target identifiers.

Specifying 0 as the maximum number of initiators, targets or concurrently active tenures specifies no maximum number.

Additional user-defined data columns can be added to the table schema. They are appended to the pre-defined columns (see [“SQL Schema” on page 2-9](#)) and are specified as additional comma-separated creation definitions (see [vmm\\_sql\\_db::create\\_table\(\)](#)). If additional user-defined data columns are specified, data for these columns must be specified when calling [vmm\\_perf\\_analyzer::end\\_tenure\(\)](#).

## Examples

### *Example A-1*

```
class my_env extends vmm_env;
  . . .
  vmm_perf_analyzer ad_bus_perf;
  vmm_perf_analyzer arbitor_perf;
  . . .
  virtual function void build();
    . . .
    this.ad_bus_perf = new("DATA/ADDRESS_BUS", db);
    this.arbitor_perf = new("ARBITOR", db, 0, 0,
      this.masters.size(), "");
    . . .
  endfunction
  . . .
endclass
```

## vmm\_perf\_analyzer::log

Message service interface for this analyzer.

### SystemVerilog

```
vmm_log log;
```

### OpenVera

```
rvm_log log;
```

### Description

Message service interface used to issue all messages from this instance of the performance analyzer.

The name of the message service interface is "Performance Analyzer" and the instance name is the name of the performance analyzer specified in the [vmm\\_perf\\_analyzer::new\(\)](#) constructor.

### Examples

#### *Example A-2*

```
class my_env extends vmm_env;
    . . .
    vmm_perf_analyzer ad_bus_perf;
    . . .
    virtual function void build();
        . . .
        this.ad_bus_perf = new("DATA/ADDRESS_BUS", db);
        `vmm_note(ad_bus_perf.log, "BUS Performance Analyser New
            Done");
        . . .
    endfunction
    . . .
```

```
endclass
```

## vmm\_perf\_analyzer::now()

Get current time for the resource.

### SystemVerilog

```
virtual function time now();
```

### OpenVera

```
virtual function bit[63:0] now();
```

### Description

Return the current absolute time used to measure resource usage. Can be simulation time (the default), or any other user-defined time measure, such a clock cycle count.

By default, returns the current simulation time.

### Examples

#### *Example A-3*

```
class my_env extends vmm_env;
    . . .
    vmm_perf_analyzer ad_bus_perf;
    . . .
    virtual function void build();
        . . .
        this.ad_bus_perf = new("DATA/ADDRESS_BUS", db);
        . . .
    endfunction
    . . .
    virtual task report();
        string str;
        $swrite(str, "Report generated at Time : %t",
```

```
ad_bus_perf.now());  
    `vmm_note(log, str);  
    . . .  
    endtask: report  
endclass
```

## vmm\_perf\_analyzer::start\_tenure()

Specify the start of a tenure.

### SystemVerilog

```
function void start_tenure(vmm_perf_tenure tenure);
```

### OpenVera

```
task start_tenure(vmm_perf_tenure tenure);
```

### Description

Indicates that the specified tenure is starting on the shared resource.

If the number of started or resumed but incomplete tenures exceeds the specified maximum number of concurrent tenures, an error is reported.

If the started tenure identifies a new initiator or a new target and exceeds the specified maximum number of initiator or target respectively, an error is reported.

### Examples

#### *Example A-4*

```
class bus_tr_tenure extends bus_tr_atomic_gen_callbacks;
    . . .
    virtual task post_inst_gen(bus_tr_atomic_gen gen,
                              bus_tr           tr,
                              ref bit          drop);
    fork
        begin
            vmm_perf_tenure tenure = new(gen.stream_id,
```

```

tr.addr[31:16],
tr);
tr.notify.wait_for(vmm_data::STARTED);
this.env.ad_bus_perf.start_tenure(tenure);
end
. . .
join_none

endclass

class my_env extends vmm_env;
. . .
vmm_perf_analyzer ad_bus_perf;
. . .
virtual function void build();
    bus_tr_tenure bus_tenure_inst = new(this);
    . . .
    this.ad_bus_perf = new("DATA/ADDRESS_BUS", db);
    . . .
endfunction
. . .
endclass

```

## vmm\_perf\_analyzer::suspend\_tenure()

Specify the suspension of a tenure.

### SystemVerilog

```
function void suspend_tenure(vmm_perf_tenure tenure);
```

### OpenVera

```
task suspend_tenure(vmm_perf_tenure tenure);
```

### Description

Indicates that the specified tenure has been suspended. A suspended tenure no longer utilizes the shared resource but is not yet completed. Suspending a tenure will not shorten its start-to-end duration measurement. Suspending a tenure will reduce the utilization of the shared resource. A suspended tenure must eventually be resumed to be completed.

Other tenures may be started while a tenure is suspended without affecting the suspended tenure. A suspended tenure does not count toward the maximum number of concurrently active tenures on the shared resource.

If the specified tenure is not currently active on the shared resource, an error is reported.

### Examples

A *write* transaction that is interrupted by a *retry* status may be specified as suspended and will resume once it is retried.

A DMA channel access that is interrupted by a higher priority request may also be specified as suspended and will resume once access is regained.

### *Example A-5*

```
class bus_tr_tenure extends bus_tr_atomic_gen_callbacks;
    . . .
    virtual task post_inst_gen(bus_tr_atomic_gen gen,
                               bus_tr           tr,
                               ref bit           drop);
    fork
        begin
            vmm_perf_tenure tenure = new(gen.stream_id,
                                         tr.addr[31:16],
                                         tr);
            tr.notify.wait_for(vmm_data::STARTED);
            this.env.ad_bus_perf.start_tenure(tenure);
            . . .
            if (tr.status == bus_tr::RETRY)
                this.env.ad_bus_perf.suspend_tenure(tenure);
            . . .
        end
    . . .
    join_none
endclass
```

## **vmm\_perf\_analyzer::resume\_tenure()**

Specify that a suspended tenure is resuming.

### **SystemVerilog**

```
function void resume_tenure(vmm_perf_tenure tenure);
```

### **OpenVera**

```
task resume_tenure(vmm_perf_tenure tenure);
```

### **Description**

Indicates that the specified previously-suspended tenure has been resumed. A resumed tenure utilizes the shared resource.

If the number of started or resumed but incomplete tenures exceeds the specified maximum number of concurrent tenures, an error is reported.

If the specified tenure is not currently suspended on the shared resource, an error is reported.

### **Examples**

A *write* transaction, suspended by a *retry* status, may be specified as resumed when it is retried.

A DMA channel access, suspended by a higher priority request, may also be specified as resumed once access is regained.

#### *Example A-6*

```
class bus_tr_tenure extends bus_tr_atomic_gen_callbacks;
```

```

    . . .
virtual task post_inst_gen(bus_tr_atomic_gen gen,
                          bus_tr           tr,
                          ref bit         drop);
    fork
        begin
            vmm_perf_tenure tenure = new(gen.stream_id,
                                         tr.addr[31:16],
                                         tr);
            tr.notify.wait_for(vmm_data::STARTED);
            this.env.ad_bus_perf.start_tenure(tenure);
            . . .
            if (tr.status == bus_tr::RETRY)
                this.env.ad_bus_perf.suspend_tenure(tenure);
            . . .
            if (. . .)
                this.env.ad_bus_perf.resume_tenure(tenure);
        end
    . . .
    join_none

endclass

```

## **vmm\_perf\_analyzer::end\_tenure()**

Specify the end of a tenure.

### **SystemVerilog**

```
function void end_tenure(vmm_perf_tenure tenure,  
    string more_data = "");
```

### **OpenVera**

```
task end_tenure(vmm_perf_tenure tenure,  
    string more_data = "");
```

### **Description**

Indicates that the specified tenure has been completed. A completed tenure no longer utilizes the shared resource and cannot be suspended nor resumed.

A completed tenure no longer counts toward the maximum number of concurrently active tenures on the shared resource.

If the specified tenure is not currently active on the shared resource, an error is reported.

If additional user-defined columns in the SQL table associated with this performance analyzer instance were specified (see [vmm\\_perf\\_analyzer::new\(\)](#)), a string of suitable comma-separated data value images must be provided.

## Examples

### *Example A-7*

```
class bus_tr_tenure extends bus_tr_atomic_gen_callbacks;
    . . .
    virtual task post_inst_gen(bus_tr_atomic_gen gen,
                               bus_tr           tr,
                               ref bit           drop);
    fork
        begin
            vmm_perf_tenure tenure = new(gen.stream_id,
                                         tr.addr[31:16],
                                         tr);
            tr.notify.wait_for(vmm_data::STARTED);
            this.env.ad_bus_perf.start_tenure(tenure);
            . . .
            tr.notify.wait_for(vmm_data::ENDED);
            this.env.ad_bus_perf.end_tenure(tenure);
            . . .
        end
    . . .
    join_none
endclass
```

## **vmm\_perf\_analyzer::abort\_tenure()**

Specify the premature end of a tenure.

### **SystemVerilog**

```
function void abort_tenure(vmm_perf_tenure tenure,  
    string more_data = "");
```

### **OpenVera**

```
task abort_tenure(vmm_perf_tenure tenure,  
    string more_data = "");
```

### **Description**

Indicates that the specified tenure has been aborted. An aborted tenure represents a wasted use of a shared resource and is tracked separately from completed tenures. An aborted tenure no longer utilizes the shared resource and cannot be suspended nor resumed.

An aborted tenure no longer counts toward the maximum number of concurrently active tenures on the shared resource.

If additional user-defined columns in the SQL table associated with this performance analyzer instance were specified (see [vmm\\_perf\\_analyzer::new\(\)](#)), a string of suitable comma-separated data value images must be provided.

If the specified tenure is not currently active on the shared resource, an error is reported.

## Examples

### *Example A-8*

```
class bus_tr_tenure extends bus_tr_atomic_gen_callbacks;
    . . .
    virtual task post_inst_gen(bus_tr_atomic_gen gen,
                              bus_tr          tr,
                              ref bit         drop);
    fork
        begin
            vmm_perf_tenure tenure = new(gen.stream_id,
                                         tr.addr[31:16],
                                         tr);
            tr.notify.wait_for(vmm_data::STARTED);
            this.env.ad_bus_perf.start_tenure(tenure);
            . . .
            if (tr.status == bus_tr::IS_OK)
                this.env.ad_bus_perf.abort_tenure(tenure);
            . . .
        end
    . . .
    join_none
endclass
```

## vmm\_perf\_analyzer::add\_tenure()

Add a fully-specified tenure.

### SystemVerilog

```
function bit add_tenure(int initiator_id = -1,
    int      target_id    = -1,
    time     start_time,
    time     end_time,
    vmm_data tr           = null,
    time     active_time = 0,
    bit      aborted     = 0,
    string   more_data   = "");
```

### OpenVera

```
function bit add_tenure(integer initiator_id = -1,
    integer    target_id    = -1,
    bit [63:0] start_time,
    bit [63:0] end_time,
    vmm_data   tr           = null,
    bit [63:0] active_time = 0,
    bit        aborted     = 0,
    string     more_data   = "");
```

### Description

Add a fully-specified tenure to the performance analysis database. This method is functionally-equivalent to calling [vmm\\_perf\\_analyzer::start\\_tenure\(\)](#) followed by [vmm\\_perf\\_analyzer::end\\_tenure\(\)](#) after some time. It can be used when the performance data for a tenure is procedurally computed and not recorded over time.

Adding a tenure does not count toward the maximum number of concurrently active tenures on the shared resource.

If the specified tenure is not valid, an error is reported and FALSE is returned.

If additional user-defined columns in the SQL table associated with this performance analyzer instance were specified (see [vmm\\_perf\\_analyzer::new\(\)](#)), a string of suitable comma-separated data value images must be provided.

## Examples

### Example A-9

```
class bus_tr_tenure extends bus_tr_atomic_gen_callbacks;
    . . .
    virtual task post_inst_gen(bus_tr_atomic_gen gen,
                               bus_tr           tr,
                               ref bit           drop);

        fork
            begin
                vmm_perf_tenure tenure = new(gen.stream_id, tr);
                tr.addr[31:16],
                tr.notify.wait_for(vmm_data::STARTED);
                this.env.ad_bus_perf.start_tenure(tenure);
                . . .
            end
        begin
            this.env.ad_bus_perf.add_tenure(
                1, 1, str_t, end_t, tr, 100, 0, "new tenure");
        end
    . . .
    join_none

endclass

class my_env extends vmm_env;
    vmm_perf_analyzer ad_bus_perf;
    . . .
endclass
```

## **vmm\_perf\_analyzer::psdisplay()**

Create a description of the status of the performance analyzer.

### **SystemVerilog**

```
virtual function string psdisplay(string prefix = "");
```

### **OpenVera**

```
virtual function string psdisplay(string prefix = "");
```

### **Description**

Create a human-readable description of the status of the performance analyzer.

### **Examples**

#### *Example A-10*

```
class my_env extends vmm_env;
    . . .
    vmm_perf_analyzer ad_bus_perf;
    . . .
    virtual function void build();
        . . .
        this.ad_bus_perf = new("DATA/ADDRESS_BUS", db);
        . . .
    endfunction
    . . .
    virtual task report();
        if (this.cfg.perf_on) begin
            `vmm_note(this.log, ad_bus_perf.psdisplay("DATA/
                ADDRESS BUS Performance :"));
            . . .
            super.report();
        end
    end
```

```
    endtask: report  
endclass
```

## **vmm\_perf\_analyzer::report()**

Report simple performance metrics.

### **SystemVerilog**

```
function void report(    string name          = "",
                        bit   brief           = 1);
```

### **OpenVera**

```
task report(    string name          = "",
               bit   brief           = 1);
```

### **Description**

Create a human-readable report of simple performance metrics (minimum, maximum and average tenure durations and percentage of utilization) for the shared resource associated with this instance of the performance analyzer, for this simulation.

If a filename is specified, the report is written to the specified file instead of the standard output.

If an initiator and/or target identifier is specified, metrics are provided for that initiator and/or target.

The report includes all performance data collected so far. Subsequent reports will contain this data and any data subsequently collected.

## Examples

### *Example A-11*

```
class my_env extends vmm_env;
    . . .
    vmm_perf_analyzer ad_bus_perf;
    . . .
    virtual function void build();
        . . .
        this.ad_bus_perf = new("DATA/ADDRESS_BUS", db);
        . . .
    endfunction
    . . .
    virtual task report();
        if (this.cfg.perf_on) begin
            this.ad_bus_perf.report("BusPerformance.txt", 1);
            this.ad_bus_perf.report();
        end
        super.report();
    endtask: report
endclass
```

## vmm\_perf\_analyzer::get\_db()

Get SQL database interface.

### SystemVerilog

```
function vmm_sql_db get_db();
```

### OpenVera

```
function vmm_sql_db get_db();
```

### Description

Return the SQL database interface associated with this performance analyzer instance.

### Examples

#### *Example A-12*

```
program test;

    class tb_env extends vmm_env;
        vmm_sql_db db, db1;
        vmm_perf_analyzer ad_bus_perf;
        . . .
        virtual function void build();
            super.build();
            vmm_sql_db_ascii db = new("10basic_sqltxt.sql");
            this.db = db;
            . . .
            this.ad_bus_perf = new("Bus", db);
            . . .
        endfunction
        . . .
    endclass
```

```
initial
begin
    tb_env env = new;
    env.run();
    env.db1 = env.ad_bus_perf.get_db();
end
endprogram
```

## **vmm\_perf\_analyzer::save\_db()**

Commit current performance analysis metrics.

### **SystemVerilog**

```
function void save_db();
```

### **OpenVera**

```
task save_db();
```

### **Description**

Save the current performance analysis for the shared resource associated with this instance of the performance analyzer, for this simulation. The SQL table only includes data about completed tenures. Any active tenures are not included and a warning is issued if there are active tenures.

Performance analysis metrics can be committed to a table multiple times during a simulation. Saving to a table does not reset the performance analysis data. Subsequent rows in the table will include subsequent performance data.

### **Examples**

#### *Example A-13*

```
class my_env extends vmm_env;
    . . .
    vmm_perf_analyzer ad_bus_perf;
    . . .
    virtual function void build();
        . . .
        this.ad_bus_perf = new("DATA/ADDRESS_BUS", db);
```

```
        . . .
endfunction
. . .
virtual task report();
    if (this.cfg.perf_on) begin
        this.ad_bus_perf.save_db();
        . . .
    end
    super.report();
endtask: report

endclass
```

## **vmm\_perf\_analyzer::save\_db\_txt()**

Save current performance analysis metrics.

### **SystemVerilog**

```
function void save_db_txt(string name);
```

### **OpenVera**

```
task save_db_txt(strign name);
```

### **Description**

Save the current performance analysis for the shared resource associated with this instance of the performance analyzer, for this simulation to the specified file in a human-readable ASCII file. The table in the ASCII file only includes data about completed tenures. Any active tenures are not included and a warning is issued if there are active or suspended tenures.

Performance analysis metrics can be saved multiple times during a simulation. Saving does not reset the performance analysis data. Subsequent file saves will overwrite any existing file and include subsequent performance data.

### **Examples**

#### *Example A-14*

```
class my_env extends vmm_env;
    . . .
    vmm_perf_analyzer ad_bus_perf;
    . . .
    virtual function void build();
    . . .
```

```
        this.ad_bus_perf = new("DATA/ADDRESS_BUS", db);
        . . .
endfunction
. . .
virtual task report();
    if (this.cfg.perf_on) begin
        this.ad_bus_perf.save_db_txt("bus_perf_sql_db.txt");
        . . .
    end
endtask
endclass
```

## **vmm\_perf\_analyzer::reset()**

Reset current performance analysis metrics.

### **SystemVerilog**

```
function void reset(reset_e rst_typ = SOFT);
```

### **OpenVera**

```
task reset(reset_e rst_typ = SOFT);
```

### **Description**

Reset the performance data collected up to this point for the shared resource associated with this instance of the performance analyzer. This clears all accumulated performance analysis data for this simulation and flushes the table in the SQL database.

If the reset is specified as HARD, all registered callbacks are unregistered.

### **Examples**

#### *Example A-15*

```
class tb_env extends vmm_env;
    vmm_sql_db db, db1;
    vmm_perf_analyzer ad_bus_perf;
    . . .
    virtual function void build();
        super.build();
        vmm_sql_db_ascii db = new("10basic_sqltxt.sql");
        this.db = db;
        . . .
        this.ad_bus_perf = new("Bus", db);
        . . .
```

```
endfunction
. . .
task reset();
    . . .
    ad_bus_perf.reset();
endtask
endclass
```

## vmm\_perf\_analyzer\_callbacks::analyze\_tenure()

Tenure analysis callback method.

### SystemVerilog

```
function void analyze_tenure(vmm_perf_analyzer analyzer,
    vmm_perf_tenure tenure,
    ref time          start_time,
    ref time          end_time,
    ref time          active_time,
    ref bit           aborted,
    ref string        more_data,
    ref bit           filtered);
```

### OpenVera

```
task analyze_tenure(vmm_perf_analyzer analyzer,
    vmm_perf_tenure tenure,
    ref bit [63:0] start_time,
    ref bit [63:0] end_time,
    ref bit [63:0] active_time,
    ref bit       aborted,
    ref string    more_data,
    ref bit       filtered);
```

### Description

Callback invoked after a tenure is ended by calling [vmm\\_perf\\_analyzer::end\\_tenure\(\)](#), aborted by calling [vmm\\_perf\\_analyzer::abort\\_tenure\(\)](#) or explicitly added using [vmm\\_perf\\_analyzer::add\\_tenure\(\)](#). The performance data associated with the tenure may be modified.

If the start, end or active times are modified to be invalid or inconsistent, an error message is issued and the tenured is filtered out.

If the filtered argument is TRUE after all of the registered callbacks have been called, the tenure will not be added to the database.

## Examples

### Example A-16

```
class my_perf_callbacks extends
vmm_perf_analyzer_callbacks;

    function void analyze_tenure(vmm_perf_analyzer analyzer,
                                vmm_perf_tenure tenure,
                                ref time start_time,
                                ref time end_time,
                                ref time active_time,
                                ref bit aborted,
                                ref string more_data,
                                ref bit filtered);
        . . .
    endfunction

endclass

class my_env extends vmm_env;
    . . .
    vmm_perf_analyzer ad_bus_perf;
    my_perf_callbacks my_perf_cb;
    . . .
    virtual function void build();
        this.my_perf_cb = new;
        this.ad_bus_perf = new("DATA/ADDRESS_BUS", db);
        ad_bus_perf.append_callback(my_perf_cb);
        . . .
    endfunction
    . . .
endclass
```

## vmm\_perf\_analyzer::append\_callback()

Appends a callback extension instance.

### SystemVerilog

```
function void append_callback(  
    vmm_perf_analyzer_callbacks cbs)
```

### OpenVera

```
task append_callback(vmm_perf_analyzer_callbacks cbs)
```

### Description

Appends the specified callback extension instance to the registered callbacks for this performance analyzer instance. Callbacks are invoked in the order of registration.

### Examples

#### *Example A-17*

```
class my_perf_callbacks extends  
vmm_perf_analyzer_callbacks;  
    . . .  
endclass  
  
class tb_env extends vmm_env;  
    vmm_perf_analyzer arb_perf;  
    my_perf_callbacks my_perf_cb;  
    . . .  
    virtual function void build();  
        this.my_perf_cb = new;  
        this.arb_perf = new("ARBITOR", db, 0, 0,  
            this.masters.size(), "");  
        . . .
```

```
        this.arb_perf.append_callback(my_perf_cb);  
        . . .  
    endfunction  
endclass
```

## vmm\_perf\_analyzer::prepend\_callback()

Prepends a callback extension instance.

### SystemVerilog

```
function void prepend_callback(  
    vmm_perf_analyzer_callbacks cbs)
```

### OpenVera

```
task prepend_callback(vmm_perf_analyzer_callbacks cbs)
```

### Description

Prepends the specified callback extension instance to the registered callbacks for this performance analyzer instance. Callbacks are invoked in the reverse order of registration.

### Examples

#### *Example A-18*

```
class my_perf_callbacks extends  
vmm_perf_analyzer_callbacks;  
    . . .  
endclass  
  
class tb_env extends vmm_env;  
    vmm_perf_analyzer arb_perf;  
    my_perf_callbacks my_perf_cb;  
    . . .  
    virtual function void build();  
        this.my_perf_cb = new;  
        this.arb_perf = new("ARBITOR", db, 0, 0,  
            this.masters.size(), "");
```

```
        . . .
        this.arb_perf.prepend_callback(my_perf_cb);
        . . .
    endfunction
endclass
```

## vmm\_perf\_analyzer::unregister\_callback()

Removes a callback extension instance.

### SystemVerilog

```
function void unregister_callback(  
    vmm_perf_analyzer_callbacks cbs)
```

### OpenVera

```
task unregister_callback(vmm_perf_analyzer_callbacks cbs)
```

### Description

Removes the specified callback extension instance from the registered callbacks for this performance analyzer instance. A warning message is issued if the callback instance has not been previously registered.

### Examples

#### *Example A-19*

```
class my_perf_callbacks extends  
vmm_perf_analyzer_callbacks;  
    . . .  
endclass  
  
class tb_env extends vmm_env;  
    vmm_perf_analyzer arb_perf;  
    my_perf_callbacks my_perf_cb;  
    . . .  
    virtual function void build();  
        this.my_perf_cb = new;  
        this.arb_perf = new("ARBITOR", db, 0, 0,  
this.masters.size(), "");
```

```
        . . .  
        this.arb_perf.unregister_callback(my_perf_cb);  
        . . .  
    endfunction  
endclass
```

## vmm\_perf\_tenure

This class is a shared resource tenure descriptor. A tenure descriptor instance describes a unique shared resource access.

For guidelines on using this class, see [Chapter 2, "Analyzing Performance of a Resource"](#).

---

### Summary

- `vmm_perf_tenure::new()` ..... page 61
- `vmm_perf_tenure::get_tenure_id()` ..... page 63
- `vmm_perf_tenure::get_initiator_id()` ..... page 65
- `vmm_perf_tenure::get_target_id()` ..... page 67
- `vmm_perf_tenure::get_tr()` ..... page 69
- `vmm_perf_tenure::psdisplay()` ..... page 71

## **vmm\_perf\_tenure::new()**

Create a new instance of a tenure descriptor

### **SystemVerilog**

```
function new(    int unsigned initiator_id = 0,  
               int unsigned target_id    = 0,  
               vmm_data    tr           = null);
```

### **OpenVera**

```
task new(    integer initiator_id = 0,  
           integer target_id    = 0,  
           rvm_data tr          = null);
```

### **Description**

Create an instance of a tenure descriptor. The identity of the initiator or target of the tenure may be optionally specified. A transaction descriptor describing the transaction corresponding to the utilization of the shared resource may also be associated with the tenure descriptor to ease traceability and debugging.

The mapping of initiator or target identifier to actual hardware structures (such as a bus master or slave) or software thread (such as an interrupt service routine) is left to the user.

A performance analysis for a specific resource may limit the maximum number of unique initiators or targets. This limits the absolute number of different initiator or slave identifiers, not their identifier values. This is designed to catch usage errors in pre-defined hardware structures. For example, a specific SoC has a

pre-determined number of bus masters or bus slaves. It is thus impossible to have tenures with more unique initiator or target identifiers than bus master or slaves, respectively.

## Examples

### *Example A-20*

```
class bus_tr_tenure extends bus_tr_atomic_gen_callbacks;
    . . .
    virtual task post_inst_gen(bus_tr_atomic_gen gen,
                               bus_tr           tr,
                               ref bit           drop);
        . . .
        fork
        begin
            vmm_perf_tenure tenure = new(gen.stream_id, -1, tr);
            . . .
        end
        . . .
        join_none
        . . .
    endtask
endclass
```

## vmm\_perf\_tenure::get\_tenure\_id()

Get the unique identifier of a tenure descriptor

### SystemVerilog

```
function int unsigned get_tenure_id();
```

### OpenVera

```
function integer get_tenure_id();
```

### Description

Return the unique tenure identifier of this tenure descriptor automatically assigned upon construction.

### Examples

#### *Example A-21*

```
class bus_tr_tenure extends bus_tr_atomic_gen_callbacks;
    . . .
    virtual task post_inst_gen(bus_tr_atomic_gen gen,
                               bus_tr            tr,
                               ref bit            drop);
        . . .
        fork
        begin
            vmm_perf_tenure tenure = new(gen.stream_id, -1, tr);
            `vmm_note(log, $psprintf("Tenure Id::
                                     %0d",tenure.get_tenure_id()));
        . . .
        end
        . . .
        join_none
        . . .
    endtask
```

endclass

## vmm\_perf\_tenure::get\_initiator\_id()

Get the identifier of the tenure initiator

### SystemVerilog

```
function int unsigned get_initiator_id();
```

### OpenVera

```
function integer get_initiator_id();
```

### Description

Return the identifier of the initiator of this tenure descriptor specified upon construction.

### Examples

#### *Example A-22*

```
class bus_tr_tenure extends bus_tr_atomic_gen_callbacks;
    . . .
    virtual task post_inst_gen(bus_tr_atomic_gen gen,
                               bus_tr           tr,
                               ref bit          drop);
        . . .
        fork
        begin
            vmm_perf_tenure tenure = new(gen.stream_id, -1, tr);
            `vmm_note(log, $psprintf("Initiator Id::
                                     %0d", tenure.get_initiator_id()));
        . . .
        end
        . . .
        join_none
        . . .
    endtask
```

endclass

## vmm\_perf\_tenure::get\_target\_id()

Get the identifier of the tenure target

### SystemVerilog

```
function int unsigned get_target_id();
```

### OpenVera

```
function integer get_target_id();
```

### Description

Return the identifier of the target of this tenure descriptor specified upon construction.

### Examples

#### *Example A-23*

```
class bus_tr_tenure extends bus_tr_atomic_gen_callbacks;
    . . .
    virtual task post_inst_gen(bus_tr_atomic_gen gen,
                               bus_tr            tr,
                               ref bit           drop);
        . . .
        fork
        begin
            vmm_perf_tenure tenure = new(gen.stream_id, -1, tr);
            `vmm_note(log, $psprintf("Target Id::
                                     %0d",tenure.get_target_id()));
        . . .
        end
        . . .
        join_none
        . . .
    endtask
```

endclass

## vmm\_perf\_tenure::get\_tr()

Get the transaction descriptor of the tenure

### SystemVerilog

```
function vmm_data get_tr();
```

### OpenVera

```
function rvm_data get_tr();
```

### Description

Return the transactor descriptor of the transaction associated with this tenure descriptor, as specified upon construction.

### Examples

#### *Example A-24*

```
class bus_tr_tenure extends bus_tr_atomic_gen_callbacks;
    . . .
    virtual task post_inst_gen(bus_tr_atomic_gen gen,
                              bus_tr          tr,
                              ref bit         drop);
        vmm_data tenure_xaction;
        . . .
        fork
        begin
            vmm_perf_tenure tenure = new(gen.stream_id, -1, tr);
            tenure_xaction = tenure.get_tr();
            `vmm_note(log, $psprintf("Tenure Transaction details::
                %s ",tenure_xaction.psdisplay))
            . . .
        end
        . . .
    join_none
```

```
    . . .  
    endtask  
endclass
```

## **vmm\_perf\_tenure::psdisplay()**

Create a description of the tenure.

### **SystemVerilog**

```
virtual function string psdisplay(string prefix = "");
```

### **OpenVera**

```
virtual function string psdisplay(string prefix = "");
```

### **Description**

Create a human-readable description of the tenure descriptor.

### **Examples**

#### *Example A-25*

```
class bus_tr_tenure extends bus_tr_atomic_gen_callbacks;
    . . .
    virtual task post_inst_gen(bus_tr_atomic_gen gen,
                               bus_tr          tr,
                               ref bit          drop);
        . . .
        fork
        begin
            vmm_perf_tenure tenure = new(gen.stream_id, -1, tr);
            `vmm_note(log, $psprintf("Tenure descriptor details::
                %s ",tenure.psdisplay))
            . . .
        end
        . . .
        join_none
        . . .
    endtask
```

```
endclass
```

# B

## SQL Database Support Classes

---

This appendix provides detailed information about the classes that are used to implement and support a generic interface to one or more SQL databases.

The OpenVera and SystemVerilog classes have identical functionality and features. Therefore, this appendix documents this information together. The heading used to introduce a method uses the SystemVerilog name. The OpenVera name will be identical except for the few cases where a `_t` suffix is appended to indicate that it may be a blocking method.

Usage examples are usually specified in a single language. However, that should not deter users of the other languages as they would be almost identical. This appendix provides more, different examples instead of nearly identical examples in each language.

The classes are documented in alphabetical order. The methods in each class are documented in a logical order, where methods that accomplish similar results are documented sequentially. A summary of all available methods with cross references to the page where their detailed documentation exists is provided at the beginning of each class specification.

---

## Class Summary

- [vmm\\_sql\\_db](#) ..... page 75
- [vmm\\_sql\\_db\\_ascii](#) ..... page 92
- [vmm\\_sql\\_db\\_sqlite](#) ..... page 94
- [vmm\\_sql\\_db\\_mysql](#) ..... page 96
- [vmm\\_sql\\_table](#) ..... page 99

## vmm\_sql\_db

This class is a generic interface to a SQL database. Each instance of this class corresponds to a single database.

Instances of this class may not be created directly as it is a virtual class. Rather, instances of classes extended from this class, such as [vmm\\_sql\\_db\\_sqlite](#) and [vmm\\_sql\\_db\\_mysql](#), must be instantiated.

---

### Summary

- [vmm\\_sql\\_db::log](#) ..... page 76
- [vmm\\_sql\\_db::status\(\)](#) ..... page 77
- [vmm\\_sql\\_db::get\\_dbname\(\)](#) ..... page 78
- [vmm\\_sql\\_db::statement\(\)](#) ..... page 81
- [vmm\\_sql\\_db::get\\_table\\_names\(\)](#) ..... page 82
- [vmm\\_sql\\_db::create\\_table\(\)](#) ..... page 84
- [vmm\\_sql\\_db::get\\_table\(\)](#) ..... page 86
- [vmm\\_sql\\_db::commit\(\)](#) ..... page 88
- [vmm\\_sql\\_db::close\(\)](#) ..... page 90

## **vmm\_sql\_db::log**

Message service interface for SQL operation.

### **SystemVerilog**

```
vmm_log log;
```

### **OpenVera**

```
rvm_log log;
```

### **Description**

Message interface used to issue messages from SQL operations. Includes debug as well as error messages.

The name of the message interface is "SQLdb" and the instance name is the name of the database associated with the SQL database access class instance.

### **Examples**

#### *Example B-1*

```
class my_env extends vmm_env;
    ...
    vmm_sql_db db;
    ...
    vmm_sql_db_ascii db = new("sql_data.sql");
    ...
    `vmm_note(this.db.log,"vmm_sql_db:log displayed ");
    ...
endclass
```

## **vmm\_sql\_db::status()**

Return the status of the last SQL operation.

### **SystemVerilog**

```
function int status()
```

### **OpenVera**

```
function integer status()
```

### **Description**

Returns 0 if the last SQL operation was successful.

### **Examples**

#### *Example B-2*

```
class my_env extends vmm_env;
  ...
  vmm_sql_db db;
  ...
  vmm_sql_db_ascii db = new("sql_data.sql");
  ...
  if(this.db.status() == 0)
    `vmm_note(this.db.log,$psprintf({"vmm_sql_db:status()
      called:", "The last SQL operation was succesful."});
  ...
endclass
```

## **vmm\_sql\_db::get\_dbname()**

Name of the database.

### **SystemVerilog**

```
function string get_dbname()
```

### **OpenVera**

```
function string get_dbname()
```

### **Description**

Returns the name of the database associated with this instance of the SQL access interface class.

### **Examples**

#### *Example B-3*

```
class my_env extends vmm_env;
    ...
    vmm_sql_db db;
    ...
    vmm_sql_db_ascii db = new("sql_data.sql");
    ...

    `vmm_note(this.db.log,$psprintf("vmm_sql_db:get_db_name()
        called", " Name of the database. = %s",
        this.db.get_dbname()));
    ...
endclass
```

## vmm\_sql\_db::get\_info()

Get database configuration information.

### SystemVerilog

```
function string get_info(string format)
```

### OpenVera

```
function string get_info(string format)
```

### Description

Return the requested database configuration information formatted as specified.

The format specification can contain characters and placeholders that are replaced by simulation-specific information, as outlined in [Table B-1](#). The system time used for the placeholder value is the value of the clock time of the host machine when the first instance of the `vmm_sql_db` class is created in the simulation. This ensures that all names generated during a simulation has the same system time value.

*Table B-1 Name Placeholders*

Placeholder	Replaced with
%D	Current day of the month of the system time formatted as 01-31
%h	Hostname.
%H	Current 24-hour of the system time formatted as 00-23
%m	Current minutes of the system time formatted as 00-59
%M	Current month of the system time formatted as 01-12
%p	Top-level program or module name.

*Table B-1 Name Placeholders*

<b>Placeholder</b>	<b>Replaced with</b>
%s	Simulation seed.
%S	Current seconds of the system time formatted as 00-59
%t	Current system time formatted as YYYYMMDDHHmmSS.
%Y	Current calendar of the system time formatted as YYYY
%%	A '%' character.
\${var[?=val]}	The content of the environment variable named "var". If undefined, can optionally be replaced by a specified default value "val". If the variable name starts with a '+', the name is interpreted as a user-specified command-line option instead of an environment variable.
\$\$	A '\$' character.

## **Examples**

### *Example B-4*

```
$write("%s\n", sql.get_info("Time is %t"));
```

## **vmm\_sql\_db::statement()**

Execute a SQL statement.

### **SystemVerilog**

```
function int statement(string sql_stmt)
```

### **OpenVera**

```
function integer statement(string sql_stmt)
```

### **Description**

Execute the specified SQL statement. Returns non-zero if an error occurs.

Currently, only non-query statements (e.g. "SELECT") can be executed as there is no mechanism for retrieving the results from a database query.

### **Examples**

#### *Example B-5*

```
class my_env extends vmm_env;
    ...
    vmm_sql_db db;
    ...
    vmm_sql_db_ascii db = new("sql_data.sql");
    ...
    if(this.db.statement("SQL STATEMENT") == 0)
        `vmm_note(this.db.log,$psprintf("No error in execution
            of SQL statement"));
    ...
endclass
```

## vmm\_sql\_db::get\_table\_names()

Get a list of existing table names.

### SystemVerilog

```
function int get_table_names(output string tablenames[],
    input string regexp = ".")
```

### OpenVera

```
function integer get_table_names(var string tablenames[],
    string regexp = ".")
```

### Description

Fill the dynamic array with the names of all the tables found in the SQL database that match the specified regular expression. Returns the number of table names that matched the specified regular expression.

The regular expression can contain placeholders that are replaced by simulation-specific information, as outlined in [Table B-1](#). The system time used for the placeholder value is the value of the clock time of the host machine when the first instance of the [vmm\\_sql\\_db](#) class is created in the simulation. This ensures that all names generated during a simulation have the same system time value.

### Examples

#### *Example B-6*

```
class my_env extends vmm_env;
    string tbl_names [];
    ...
    vmm_sql_db db;
```

```
...
vmm_sql_db_ascii db = new("sql_data.sql");
...
vmm_sql_table cfg_tbl = db.create_table("cfg", "num_mstrs
    INTEGER,num_slvs INTEGER");
...
`vmm_note(this.db.log,$psprintf(
    "vmm_sql_db:get_table_names() called existing tables
    = %0d", this.db.get_table_names(tbl_names,"cfg")));
...
endclass
```

## vmm\_sql\_db::create\_table()

Create a table and access interface.

### SystemVerilog

```
function vmm_sql_table create_table(string tablename,  
    string scheme,  
    byte  datakind = 255)
```

### OpenVera

```
function vmm_sql_table create_table(string tablename,  
    string  scheme,  
    bit [7:0] datakind = 255)
```

### Description

Create a table and associated access interface for the specified table. Returns NULL if the table already exists. To obtain an access interface to an existing table, use [vmm\\_sql\\_db::get\\_table\(\)](#).

The name of the table can contain placeholders that are replaced by simulation-specific information, as outlined in [Table B-1](#).

The schema is specified as the definition portion of the table creation SQL statement (the portion between parenthesis, not including the parenthesis themselves):

```
TABLE CREATE IF NOT EXISTS tbl_name (create_definition)
```

### Examples

#### *Example B-7*

```
db.create_table("tr_trace",
```

```
"time INTEGER, data1 INTEGER, data2 INTEGER, location TEXT");
```

## vmm\_sql\_db::get\_table()

Get a table access interface to an existing table.

### SystemVerilog

```
function vmm_sql_table get_table(string tablename)
```

### OpenVera

```
function vmm_sql_table get_table(string tablename)
```

### Description

Create a table access interface for the specified table. If the table does not exist, NULL is returned. A table can be created using the [vmm\\_sql\\_db::create\\_table\(\)](#) method.

### Examples

#### *Example B-8*

```
class my_env extends vmm_env;
    ...
    vmm_sql_db db;
    ...
    vmm_sql_db_ascii db = new("sql_data.sql");
    ...
    vmm_sql_table cfg_tbl = db.create_table("cfg", "num_mstrs
        INTEGER,num_slvs INTEGER");
    vmm_sql_table temp;
    ...
    temp = db.get_table("cfg");
    if(temp == null)
        `vmm_warning(this.log,"Table does not exists");
    ...
endclass
```

```
endclass
```

## vmm\_sql\_db::commit()

Commit data to the database.

### SystemVerilog

```
function void commit()
```

### OpenVera

```
task commit()
```

### Description

Flush all and any pending or buffered data to the physical database storage.

### Examples

#### *Example B-9*

```
class my_env extends vmm_env;
  ...
  vmm_sql_db db;
  ...
  vmm_sql_db_ascii db = new("sql_data.sql");
  ...
  begin
    vmm_sql_table cfg_tbl =
      db.create_table("cfg", "num_mstrs INTEGER, num_slvs
        INTEGER");
    ...
    cfg_tbl.insert($psprintf("%0d,%0d",
      this.cfg.bus.n_masters, this.cfg.bus.n_slaves));
    ...
    db.commit();
    ...
  end
```

```
...  
endclass
```

## **vmm\_sql\_db::close()**

Close the database.

## **SystemVerilog**

```
function void close()
```

## **OpenVera**

```
task close()
```

## **Description**

Commit all pending data then close the database and release all memory used by the SQL access interface instance.

The database can no longer be used after it has been closed. A new SQL access interface instance must be created on it.

## **Examples**

### *Example B-10*

```
class my_env extends vmm_env;
    ...
    vmm_sql_db db;
    ...
    vmm_sql_db_ascii db = new("sql_data.sql");
    ...
    begin
        vmm_sql_table cfg_tbl =
            db.create_table("cfg","num_mstrs INTEGER, num_slvs
                INTEGER");
        ...
        cfg_tbl.insert($psprintf("%0d, %0d",
            this.cfg.bus.n_masters,this.cfg.bus.n_slaves));
    end
endclass
```

```
        ...  
    end  
    ...  
    db.close();  
    ...  
endclass
```

## vmm\_sql\_db\_ascii

This class is a pseudo-implementation of the SQL database access class that uses a simple ASCII file as database. Instances of this class are used to access ASCII files.

The content of each database file is the script of SQL statements that were used to create it. The SQL script can then be played back on an actual SQL command interpreter or interpreted using a PERL script to reconstruct the content of individual tables.

It is an extension of the [vmm\\_sql\\_db](#) class.

---

### Summary

- [vmm\\_sql\\_db\\_ascii::new\(\)](#) ..... page 93
- [vmm\\_sql\\_db::log](#) ..... page 76
- [vmm\\_sql\\_db::status\(\)](#) ..... page 77
- [vmm\\_sql\\_db::get\\_dbname\(\)](#) ..... page 78
- [vmm\\_sql\\_db::statement\(\)](#) ..... page 81
- [vmm\\_sql\\_db::close\(\)](#) ..... page 90

## **vmm\_sql\_db\_ascii::new()**

Create a new ASCII file database access interface.

### **SystemVerilog**

```
function new(string dbname,  
            bit append = "0")
```

### **OpenVera**

```
task new(string dbname,  
        bit append = 0)
```

### **Description**

Create an SQL access interface class to the specified ASCII file database. If the database does not exist, it is created. If "append" is specified as FALSE, the database is overwritten.

The name of the database can contain placeholders that are replaced by simulation-specific information, as outlined in [Table B-1](#).

### **Examples**

#### *Example B-11*

```
class my_env extends vmm_env;  
    ...  
    vmm_sql_db db;  
    ...  
    vmm_sql_db_ascii db = new("sql_data.sql");  
    ...  
endclass
```

## vmm\_sql\_db\_sqlite

This class is an SQLite implementation of the SQL database access class. Instances of this class are used to access SQLite databases.

It is an extension of the [vmm\\_sql\\_db](#) class.

---

### Summary

- [vmm\\_sql\\_db\\_sqlite::new\(\)](#) ..... page 95
- [vmm\\_sql\\_db::log](#) ..... page 76
- [vmm\\_sql\\_db::status\(\)](#) ..... page 77
- [vmm\\_sql\\_db::get\\_dbname\(\)](#) ..... page 78
- [vmm\\_sql\\_db::statement\(\)](#) ..... page 81
- [vmm\\_sql\\_db::close\(\)](#) ..... page 90

## **vmm\_sql\_db\_sqlite::new()**

Create a new sqlite database access interface.

### **SystemVerilog**

```
function new(string dbname)
```

### **OpenVera**

```
task new(string dbname)
```

### **Description**

Create an SQL access interface class to the specified sqlite database. If the database does not exist, it is created.

The name of the database can contain placeholders that are replaced by simulation-specific information, as outlined in [Table B-1](#).

### **Examples**

#### *Example B-12*

```
class my_env extends vmm_env;
  ...
  vmm_sql_db db;
  ...
  vmm_sql_db_sqlite db = new("sql_data.db");
  ...
endclass
```

## vmm\_sql\_db\_mysql

This class is a MySQL implementation of the SQL database access class. Instances of this class are used to access SQLite databases.

It is an extension of the [vmm\\_sql\\_db](#) class.

---

### Summary

- [vmm\\_sql\\_db\\_mysql::new\(\)](#) ..... page 97
- [vmm\\_sql\\_db::log](#) ..... page 76
- [vmm\\_sql\\_db::status\(\)](#) ..... page 77
- [vmm\\_sql\\_db::get\\_dbname\(\)](#) ..... page 78
- [vmm\\_sql\\_db::statement\(\)](#) ..... page 81
- [vmm\\_sql\\_db::close\(\)](#) ..... page 90

## vmm\_sql\_db\_mysql::new()

Create a new MySQL database access interface.

### SystemVerilog

```
function new(    string hostname,
               int    portnum,
               string username,
               string password,
               string dbname)
```

### OpenVera

```
task new(    string hostname,
            int    portnum,
            string username,
            string password,
            string dbname)
```

### Description

Create an SQL access interface class to the specified MySQL database on the specified MySQL server. If the database does not exist, it is created.

The name of the database can contain placeholders that are replaced by simulation-specific information, as outlined in [Table B-1](#).

### Examples

#### *Example B-13*

```
class tb_env extends vmm_env;
    vmm_sql_db  db;
    . . .
    virtual function void build();
```

```

super.build();
if (this.cfg.perf_on) begin
    vmm_sql_db_ascii db = new("10basic_sqltxt.sql");
    this.db = db;
    . . .
begin
    vmm_sql_table cfg_tbl = db.create_table("cfg",
        "num_mstrs INTEGER, num_slvs INTEGER");
    `vmm_note(log, cfg_tbl.get_tblname());
    . . .
end
    . . .
end
    . . .
endfunction
    . . .
endclass

```

## vmm\_sql\_table

This class is a generic interface to a table in an SQL database. Each instance of this class corresponds to a single table.

Instances of this class may not be created directly. They are created using the [vmm\\_sql\\_db::get\\_table\(\)](#).

---

### Summary

- [vmm\\_sql\\_table::get\\_tblname\(\)](#) ..... page 100
- [vmm\\_sql\\_table::get\\_db\(\)](#) ..... page 102
- [vmm\\_sql\\_table::insert\(\)](#) ..... page 104

## vmm\_sql\_table::get\_tblname()

Name of the table.

### SystemVerilog

```
function string get_tblname()
```

### OpenVera

```
function string get_tblname()
```

### Description

Returns the name of the table associated with this instance of the access interface class.

### Examples

#### *Example B-14*

```
class tb_env extends vmm_env;
    vmm_sql_db  db, db1;
    . . .
    virtual function void build();
        super.build();
        if (this.cfg.perf_on) begin
            vmm_sql_db_ascii db = new("10basic_sqltxt.sql");
            this.db = db;
            . . .
        begin
            vmm_sql_table cfg_tbl = db.create_table("cfg",
                "num_mstrs INTEGER, num_slvs INTEGER");
            cfg_tbl.insert($psprintf("%0d, %0d",
                this.cfg.bus.n_masters,
                this.cfg.bus.n_slaves));

            db.commit();
            db1 = cfg_tbl.get_db();
        end
    endclass
```

```
        . . .  
        end  
    . . .  
    end  
    . . .  
endfunction  
. . .  
endclass
```

## vmm\_sql\_table::get\_db()

Get the SQL database that contains the table

### SystemVerilog

```
function vmm_sql_db get_db()
```

### OpenVera

```
function vmm_sql_db get_db()
```

### Description

Returns the access interface to the database that contains this table.

### Examples

#### *Example B-15*

```
class tb_env extends vmm_env;
    vmm_sql_db db;

    . . .
    virtual function void build();
        super.build();
        if (this.cfg.perf_on) begin
            vmm_sql_db_ascii db = new("10basic_sqltxt.sql");
            this.db = db;

            . . .
            begin
                vmm_sql_table cfg_tbl = db.create_table("cfg",
                    "num_mstrs INTEGER, num_slvs INTEGER");
                cfg_tbl.insert($psprintf("%0d, %0d",
                    this.cfg.bus.n_masters,
                    this.cfg.bus.n_slaves));

                . . .
            end
        end
end
```

```
        . . .  
    end  
    . . .  
endfunction  
    . . .  
endclass
```

## **vmm\_sql\_table::insert()**

Insert a new row in the SQL table.

### **SystemVerilog**

```
function int insert(string data)
```

### **OpenVera**

```
function integer insert(string data)
```

### **Description**

Insert a new row in the table. The row data is a comma-separated list of values as defined by the table schema (see [vmm\\_sql\\_db::create\\_table\(\)](#)).

Returns 0 if the new row is successfully inserted in the table.