UVM Framework Code Generator API Reference

Revision 3.6g_0

UVMF Code Generator API Reference Table of Contents

1 Introduction to the UVM Framework (UVMF) Code Generators

1.1 Overview

The UVM Framework provides code generators for creating interfaces, environments, and test benches. The generators provide a set of API's for characterizing content of the interface, environment, and test bench.

The interface generator creates the initial code for a UVM based protocol agent package and BFM's. The interface simulates as generated and is ready for the user to add protocol specific signal activity and seqeuences.

The environment generator creates the initial code for a UVM based chip or block level environment. Analysis components can be characterized, components instantiated, and all UVM connections between components can be done with the environment generator. The environment simulates as generated and is ready for the user to complete prediction functions.

The test bench generator creates the initial code for a UVM based test bench. The test bench generator creates the top level module, UVM test, virtual sequence, parameters package, and makefile for compilation and simulation. The generated top level virtual sequence automatically contains a random sequence for each listed UVMF agent. The test bench simulates as generated and is ready for the user to instantiate the DUT and begin creating test scenarios.

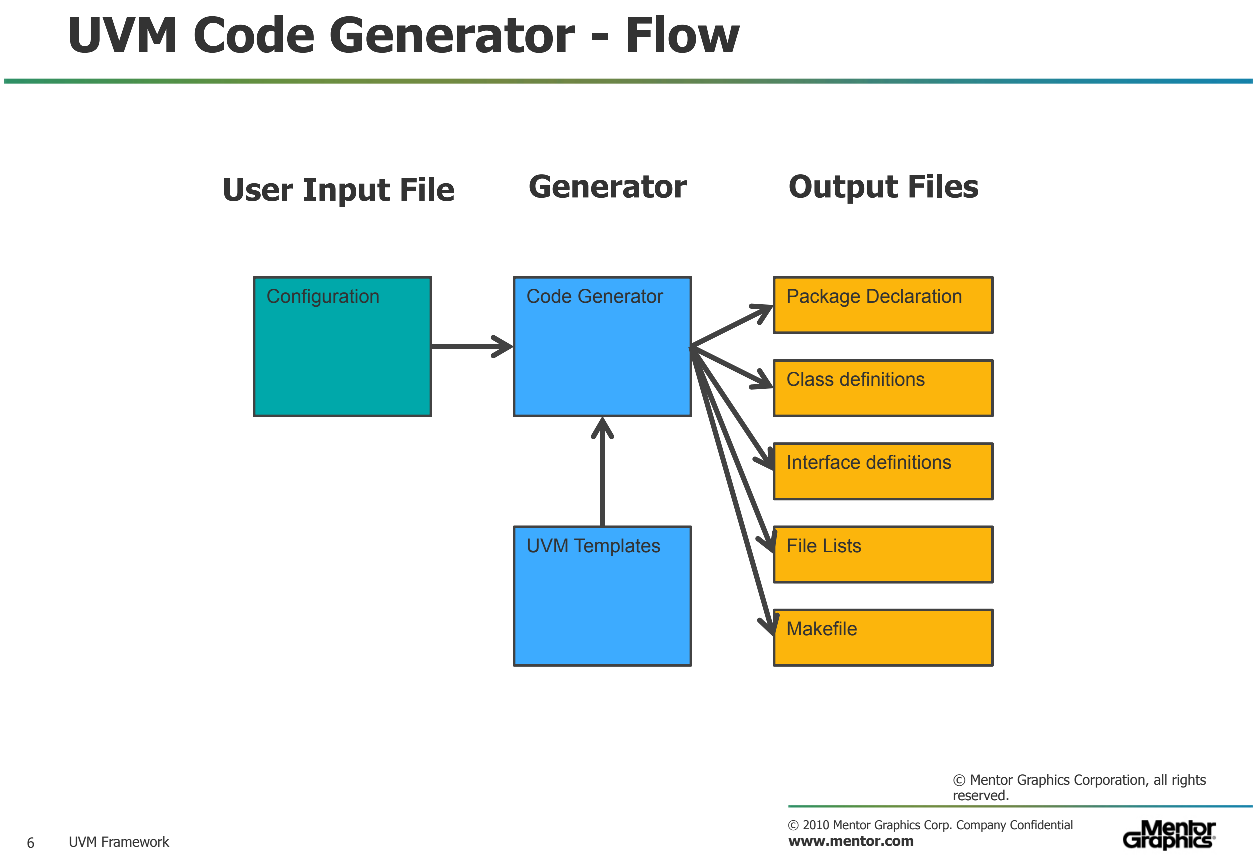
There are a number of example user input files for interfaces, environments, and test benches. The table below describes the examples found in templates/python/examples.

Interface Examples	**Description**
mem_if_config.py	User input file for generating an interface package named mem_pkg. This interface is used in block_a, block_b, block_c, and chip environments and test benches
pkt_if_config.py	User input file for generating an interface package named pkt_pkg. This interface is used in block_a, block_b, block_c, and chip environments and test benches
dma_if_config.py	User input file for generating an interface named dma_pkg. This interface is a responder interface and defines response data.
Environment Examples	**Description**
block_a_env_config.py	User input file for generating an environment that has no parametization. This environment is also used in chip_env.
block_b_env_config.py	User input file for generating an environment that has parametization. This environment is also used in chip_env.
block_c_env_config.py	User input file for generating an environment that has a QVIP configurator generated sub environment that contains

	standard protocols.
chip_env_config.py	User input file for generating a chip level environment that instantiates sub environments.
Test Bench Examples	
block_a_ben_config.py	User input file for generating a test bench to run the block_a environment.
block_b_ben_config.py	User input file for generating a test bench to run the block_b environment.
block_c_ben_config.py	User input file for generating a test bench to run the block_c environment.
chip_ben_config.py	User input file for generating a test bench to run the chip environment.

1.2 Generation flow

The diagram below shows the flow utilized by the UVMF generators. The user creates a text file that uses the provided API's for characterizing the interface, environment, or test bench. This configuration file creates an object that contains all of the information provided by the user in the arguments of the API's. The UVMF code generator, uvmf_gen.py, uses information in the object and the Jinja2 templatizing package to generate source files. Jinja2 is a free package that the user installs in order to use the UVMF code generator. Files generated include all classes, packages, BFM's, and makefiles required for an operational test bench that simulates as generated.



UVM Code Generator - Flow

User Input File

Generator

Output Files

Configuration

Code Generator

UVM Templates

Package Declaration

Class definitions

Interface definitions

File Lists

Makefile

6 UVM Framework

2 Major sections of user input

2.1.1 Initialize

The initialize section of user input contains three elements: python shell execution, import of UVMF code generator, and generation of object for holding all user information that characterizes the desired interface, environment, or test bench. The python shell execution and generator import is shown below and must be included in all user input files.

```
#! /usr/bin/env python

import uvmf_gen
```

The initialize API used for generating the object which holds all user information is generator specific and given below for the interface, environment, and test bench generators.

2.1.2 Body

The body section of user input contains UVMF provided API's used to characterize the content of the interface, environment, or test bench. Any number and combination of API's can be used to characterize the desired output. All of the API's are described in the "body API's" section of interface, environment, and test bench code generators below.

2.1.3 Finalize

The finalize section of user input contains the API that triggers file generation. This API is shown below and must be included in all user input files.

```
create()
```

3 Interface Code Generator

3.1 Interface initialize API

The interface initialize API creates a python object that contains all of the interface body API's listed below. These interface body API's are used to characterize information about the interface protocol. This information is used to create the following content:

Classes: Transaction, interface level sequence base, random sequence, coverage, driver, monitor, agent, agent configuration, UVM reg adapter, UVM reg predictor.

Package: Protocol package including all classes listed above.

BFM's: Driver and monitor.

Compilation flow: File list and Makefile

3.1.1 InterfaceClass()

This initialization API is used to create an object that stores all information about the interface. An example use is shown below. The protocol name is mem. The object created is named intf. All body API's and variables are members of this object and therefore must be preceeded by the name of the object and separated by a period.

```
intf = uvmf_gen.InterfaceClass('mem')
```

3.1.2 Interface variables

Name	**Value(s)**	**Description**
veloceReady	True|False	Interface generated is Veloce Ready|Friendly.
inFactReady	True|False	Interface generated is inFact ready. An inFact sequence and project direcories are created for use by inFact.
catapultReady	True|False	Enables HLS related requirements.
clock	'signalName'	Name of primary clock. Additional clocks must be added manually.
reset	'resetName'	Name of primary reset. Additional resets must be added manually. If interface has no

		reset use ‘dummy’ and remove associated code from interface.
resetAssertionLevel	True|False	Assertion level for this protocol.

3.2 Interface body APIs

These API’s are member functions of the interface object created using the InterfaceClass initialization API. These API’s may be used in any number and any order. They must preceede the create() API.

3.2.1 addParamDef()

Name	addParamDef
Description	This API adds a parameter to the interface classes and BFM's.

Usage	`addParamDef(`
	`'`*`parameterName`*`',`
	`'`*`parameterType`*`',`
	`'`*`parameterDefalutValue`*`'`
	`)`

Arguments	
`'`*`parameterName`*`'`	**Description:** Name of the parameter. **Value:** Any valid SV parameter name.
`'`*`parameterType`*`'`	**Description:** Type of the parameter. **Value:** Any valid SV parameter type.
`'`*`parameterDefalutValue`*`'`	**Description:** The default value for this parameter. **Value:** Any valid SV value for the parameter type.

3.2.2 addImport()

Name	addImport
Description	Adds an import of the selected package to the interface package.

Usage	`addImport(`
	`'`*`packageName`*`'`
	`)`

Arguments	
`'`*`packageName`*`'`	**Description:** The name of the package to be imported. **Value:** Any valid SV package. The location and compilation of this package must be added to the generated interface.

3.2.3 addHdlTypedef()

Name	addHdlTypedef
Description	This API adds adds a typedef that is visible to UVM content and BFM's.

Usage	`addHdlTypedef(`
	`'`*`typedefName`*`',`
	`'`*`typedefDefinition`*`'`
	`)`

Arguments	
`'`*`typedefName`*`'`	**Description:** The name of the typedef. **Value:** Any valid SV typedef name.
`'`*`typedefDefinition`*`'`	**Description:** Type definition of the typedef. **Value:** Any valid SV typedef definition.

3.2.4 addHvlTypedef()

Name	addHvlTypedef
Description	This API adds adds a typedef that is only visible to UVM content. BFM’s do not have visibility to this typedef.

Usage	`addHvlTypedef(`
	`'`*`typedefName`*`',`
	`'`*`typedefDefinition`*`'`
	`)`

Arguments	
`'`*`typedefName`*`'`	**Description:** The name of the typedef. **Value:** Any valid SV typedef name.
`'`*`typedefDefinition`*`'`	**Description:** Type definition of the typedef. **Value:** Any valid SV typedef definition.

3.2.5 addPort()

Name	addPort
Description	This API adds a signal to the interface package. Use this API for each signal in the protocol with the exception of the clock and reset defined in the interface variables.

Usage	`addPort(`
	`'signalName',`
	`'signalWidth',`
	`'signalDirection'`
	`)`

Arguments	
`'signalName'`	**Description:** Name of the signal. **Value:** Any valid SV signal name.
`'signalWidth'`	**Description:** Width of the signal. **Value:** Any valid SV signal width value. This value can be one of the parameters defined using addParamDef API.
`'signalDirection'`	**Description:** The direction of the signal from the perspective of the test bench as an initiator. **Value:** input|output|inout.

3.2.6 addConfigVar()

Name	addConfigVar
Description	This API adds a variable to the configuration class. This variable is automatically added to the convert2string function. A coverpoint for this variable is added to the generated covergroup.

Usage	`addConfigVar(`
	`'`*`variableName`*`',`
	`'`*`variableType`*`',`
	`isrand``=True|False`
	`)`

Arguments	
`'`*`variableName`*`'`	**Description:** Name of the variable. **Value:** Any valid SV variable name.
`'`*`variableType`*`'`	**Description:** Type of the variable. **Value:** Any valid SV variable type.
`isrand`	**Description:** Determines if the variable is randomizable. **Value:** True|False

3.2.7 addConfigVarConstraint()

Name	addConfigVarConstraint
Description	This API adds a constraint to the configuration class.

Usage	`addConfigVarConstraint(`
	`'`*`constraintName`*`',`
	`'`*`constraintBody`*`'`
	`)`

Arguments	
`'`*`constraintName`*`'`	**Description:** Name of the constraint. **Value:** Any valid SV constraint name.
`'`*`constraintBody`*`'`	**Description:** Body of the constraint. **Value:** Any valid SV constraint body.

3.2.8 addTransVar()

Name	addTransVar
Description	This API adds a variable to the transaction class. This variable is automatically added to the convert2string function. A coverpoint for this variable is added to the generated coverage component. This variable is automatically added to transaction viewing in the waveform viewer.

Usage	`addTransVar(`
	`'variableName',`
	`'variableType',`
	`isrand=True|False,`
	`isCompare=True|False`
	`)`

Arguments	
`'variableName'`	**Description:** Name of the variable. **Value:** Any valid SV variable name.
`'variableType'`	**Description:** Type of the variable. **Value:** Any valid SV variable type.
`isrand`	**Description:** Determines if the variable is randomizable. **Value:** True|False
`isCompare`	**Description:** Determines if the variable is included in the do_compare function. **Value:** True|False

3.2.9 addTransVarConstraint()

Name	addTransVarConstraint
Description	This API adds a constraint to the transaction class.

Usage	`addTransVarConstraint(`
	`'`*`constraintName`*`',`
	`'`*`constraintBody`*`'`
	`)`

Arguments	
`'`*`constraintName`*`'`	**Description:** Name of the constraint. **Value:** Any valid SV constraint name.
`'`*`constraintBody`*`'`	**Description:** Body of the constraint. **Value:** Any valid SV constraint body.

3.2.10 specifyResponseOperation()

Name	specifyResponseOperation
Description	This API specifies the condition which determines if responseData identified by the specifyResponseData API is returned to the BFM.

Usage	`specifyResponseOperation(`
	`'signalCondition'`
	`)`

Arguments	
`'signalCondition'`	**Description:** The condition which determines if response data is returned to the BFM. **Value:** Any valid SV expression that results in a boolean.

3.2.11 specifyResponseData()

Name	specifyResponseData
Description	This API specifies the data required by the BFM to complete a transfer initiated by a master.

Usage	`specifyResponseData(`
	`['returnedData']`
	`)`

Arguments	
`'`*`returnedData`*`'`	**Description:** This argument is a list of variables returned to the BFM in order to complete a transfer initiated by a master. These values must be listed as variables using the addTransVar API. **Value:** Any variable added to the transaction using the addTransVar API.

3.2.12 addDPIFile()

Name	addDPIFile
Description	This API adds a file to the list of C files to be compiled.

Usage	`addDPIFile(`
	`'fileName')`

Arguments	
`'fileName'`	**Description:** Name of C file to be compiled and linked. **Value:** Any valid file name including its extension.

3.2.13 addDPICompArgs()

Name	addDPICompArgs
Description	This API adds compilation flags to be used when compiling the C files identified using addDPIFile API.

Usage	`addDPICompArgs(`
	`'`*`compArgs`*`')`

Arguments	
`'`*`compArgs`*`'`	**Description:** Compile arguments to be used when compiling the C files identified using addDPIFile API. **Value:** Any valid compilation flag for the c compiler.

3.2.14 addDPILinkArgs()

Name	addDPILinkArgs
Description	This API adds flags to be used when linking the C files identified using addDPIFile API.

Usage	`addDPIFile(`
	`'`*`linkArgs`*`')`

Arguments	
`'`*`linkArgs`*`'`	**Description:** Arguments to be used when linking the C files identified using addDPIFile API. **Value:** Any valid flag for the c linker.

3.2.15 addDPISOName()

Name	addDPISOName
Description	This API sets the name of the shared object created by linking all compiled objects.

Usage	`addDPISOName(`
	`'`*`SOName`*`')`

Arguments	
`'`*`SOName`*`'`	**Description:** Name of the shared object created by linking all compiled objects. **Value:** Any valid SO name.

3.2.16 addDPIImport()

Name	addDPIImport
Description	This API adds a C function to be imported into SV using DPI.

Usage	`addDPIImport(`
	`'returnType',`
	`'functionName',`
	`'functionArguments',`
	`{'SVVariableName':'SVVariableType',`
	` 'SVVariableName':'SVVariableType'}`
	`)`

Arguments	
`'returnType'`	**Description:**Return type of C function. **Value:** Any valid C return type when using DPI.
`'functionName'`	**Description:** Name of C function to be imported into SV using DPI **Value:** Any valid C function name.
`'functionArguments'`	**Description:** All arguments of C function. **Value:** Any valid list of C function arguments when using DPI.

`{'SVVariableName':'SVVariableType',`
` 'SVVariableName':'SVVariableType'}`
	Description: The list of systemVerilog arguments that are equivalent types to the list of C arguments.
	Value: Any valid SV variable name and type when using DPI.

3.3 Interface finalize API

3.3.1 Create()

The finalize section of user input contains the API that triggers file generation. This API is shown below and must be included in all user input files.

```
create()
```

4 Environment Code Generator

4.1 Environment initialize API

The environment initialize API creates a python object that contains all of the environment body API's listed below. These environment body API's are used to characterize information about the environment. This information is used to create the following content:

Classes: Environment, environment configuration, predictors, coverage collection components, environment level sequence base.

Package: Environment package.

Compilation flow: File list and Makefile

4.1.1 EnvironmentClass()

This initialization API is used to create an object that stores all information about the environment. An example use is shown below. The environment name is block_b. The object created is named env. All body API's and variables are members of this object and therefore must be preceeded by the name of the object and separated by a period.

```
env = uvmf_gen.EnvironmentClass('block_b')
```

4.1.2 Environment variables

Name	**Value(s)**	**Description**
catapultReady	True|False	Enables HLS related requirements.

4.2 Environment body APIs

These API's are member functions of the environment object created using the EnvironmentClass initialization API. These API's may be used in any number and any order. They must preceede the create() API.

4.2.1 addParamDef()

Name	addParamDef
Description	This API adds a parameter to the environment classes.

Usage	`addParamDef(`
	`'parameterName',`
	`'parameterType',`
	`'parameterDefalutValue'`
	`)`

Arguments	
`'parameterName'`	**Description:** Name of the parameter. **Value:** Any valid SV parameter name.
`'parameterType'`	**Description:** Type of the parameter. **Value:** Any valid SV parameter type.
`'parameterDefalutValue'`	**Description:** The default value for this parameter. **Value:** Any valid SV value for the parameter type.

4.2.2 addTypedef()

Name	addTypedef
Description	This API adds adds a typedef to the environment package.

Usage	`addHvlTypedef(`
	`'typedefName',`
	`'typedefDefinition'`
	`)`

Arguments	
`'typedefName'`	**Description:** The name of the typedef. **Value:** Any valid SV typedef name.
`'typedefDefinition'`	**Description:** Type definition of the typedef. **Value:** Any valid SV typedef definition.

4.2.3 addImport()

Name	addImport
Description	Adds an import of the selected package to the environment package.

Usage	`addImport(`
	`'`*`packageName`*`'`
	`)`

Arguments	
`'`*`packageName`*`'`	**Description:** The name of the package to be imported. **Value:** Any valid SV package. The location and compilation of this package must be added to the generated environment.

4.2.4 addConfigVar()

Name	addConfigVar
Description	This API adds a variable to the configuration class for this environment. This variable is also automatically added as a coverpoint within a covergroup in the configuration class.

Usage	`addConfigVar(`
	`'`*`variableName`*`',`
	`'`*`variableType`*`',`
	`isrand``=True|False`
	`)`

Arguments	
`'`*`variableName`*`'`	**Description:** Name of the variable. **Value:** Any valid SV variable name.
`'`*`variableType`*`'`	**Description:** Type of the variable. **Value:** Any valid SV variable type.
`isrand`	**Description:** Determines if the variable is randomizable. **Value:** True|False

4.2.5 addConfigVarConstraint()

Name	addConfigVarConstraint
Description	This API adds a constraint to the configuration class.

Usage	`addConfigVarConstraint(`
	`'`*`constraintName`*`',`
	`'`*`constraintBody`*`'`
	`)`

Arguments	
`'`*`constraintName`*`'`	**Description:** Name of the constraint. **Value:** Any valid SV constraint name.
`'`*`constraintBody`*`'`	**Description:** Body of the constraint. **Value:** Any valid SV constraint body.

4.2.6 addAgent()

Name	addAgent
Description	This API adds an agent to the environment.

Usage	`addAgent(`
	`'instanceName',`
	`'protocol',`
	`'clock',`
	`'reset',`
	`{'protocolParam1':'protocolParam1Override',`
	` 'protocolParam2':'protocolParam2Override'}`
	`)`

Arguments	
`'instanceName'`	**Description:** Instance name of this agent. **Value:** Any valid SV instance name.
`'protocol'`	**Description:** Name of the protocol package. Example: mem_pkg would use the value mem **Value:** Any valid SV package name.
`'clock'`	**Description:** The name of the primary clock for this protocol. **Value:** Any valid SV signal name.
`'reset'`	**Description:** The name of the primary reset for this protocol. **Value:** Any valid SV signal name.
`'protocolParamN'`	**Description:** Name of the interface package parameter to be overridden. **Value:** Must match the interface parameter name to be overridden.
`protocolParamNOverride'`	**Description:** The value used to override the parameters default value. **Value:** Any valid SV value for overriding the identified parameter. Example: ‘5’ or ‘*testLevelParameterName*’

4.2.7 addSubEnv()

Name	addSubEnv
Description	This API adds a UVMF environment to this environment.

Usage	`addSubEnv(`
	`'subEnvironmentInstanceName',`
	`'subEnvironmentPackageName',`
	`numberOfAgents,`
	`{`
	`'subEnvironmentParameter1':'parameter1Override',`
	`'subEnvironmentParameter2':'parameter2Override'`
	`})`

Arguments	
`'subEnvironmentInstanceName'`	**Description:** Instance name for this environment. **Value:** Any valid SV instance name.
`'subEnvironmentPackageName'`	**Description:** Package name for this environment. **Value:** The name of the environment package that contains this environment. Example: for package named block_b_env_pkg use the value block_b
`numberOfAgents`	**Description:** Number of agents in this environment and any environments encapsulated by this environment. **Value:** Total number of agents.
`'subEnvironmentParameterN'`	**Description:** Parameter of this sub environment. **Value:** The parameter of this environment class.
`'parameterNOverride'`	**Description:** The value to override the specified environment parameter. **Value:** Any valid SV value for overridding the parameter.

4.2.8 addQvipSubEnv()

Name	addQvipSubEnv
Description	This API adds a UVMF sub environment generated by the QVIP Configurator. This API is generated by the QVIP Configurator and contains all selected standard protocols. This API should not be written manually.

Usage	`addQvipSubEnv(`
	`'`*`environmentInstanceName`*`',`
	`'`*`environmentPackageName`*`',`
	`['`*`agent1Name`*`', '`*`agent2Name`*`']`
	`)`

Arguments	
`'`*`environmentInstanceName`*`'`	**Description:** The instance name for this environment **Value:** Any valid SV instance name.
`'`*`environmentPackageName`*`'`	**Description:** The pakage name generated by the QVIP configurator. **Value:** Defined by the QVIP configurator.
`'`*`agentNName`*`'`	**Description:** Name of QVIP agent within environment. **Value:** Any valid SV instance name.

4.2.9 defineAnalysisComponent()

Name	defineAnalysisComponent
Description	This API is used to define an analysis component class. This can be used to create predictors, coverage components, etc. It creates a component that is an extension of uvm_component. Any number of analysis_exports and analysis_ports can be added to this component. This API creates the class definition and adds the class to the environment package.

Usage	`defineAnalysisComponent(`
	`keyword``,`
	`'`*`className`*`',`
	`{`
	` '`*`analysisExport1Name`*`':'`*`transactionType1`*`',`
	` '`*`analysisExport2Name`*`':'`*`transactionType2`*`'`
	`},`
	`{`
	`'`*`analysisPort1Name`*`':'`*`transactionType3`*`',`
	`'`*`analysisPort2Name`*`':'`*`transactionType4`*`'`
	`}`
	`)`

Arguments	
`keyword`	**Description:** Keyword used to select a template from the environment generator templates. **Value:** predictor
`'`*`className`*`'`	**Description:** Name given to the class generated by this API. **Value:** Any valid SV class name.
`'`*`analysisExport1Name`*`'`	**Description:** Name of the analysis_export to be created. **Value:** Any valid SV instance name.
`'`*`analysisPort1Name`*`'`	**Description:** Name of the analysis_port to be created. **Value:** Any valid SV instance name.
`'`*`transactionType1`*`'`	**Description:** Type of the transaction to be received by the analysis_export or broadcasted from the analysis_port. The type parameterization must match that of the component connected to this analysis_export or analysis_port. **Value:** Transaction type matching component connecting to each analysis_export or analysis_port.

4.2.10 addAnalysisComponent()

Name	addAnalysisComponent
Description	This API instantiates an analysis component.

Usage	`addAnalysisComponent(`
	`'instanceName',`
	`'classType'`
	`)`

Arguments	
'instanceName'	**Description:** Instance name for the component. **Value:** Any valid SV instance name.
'classType'	**Description:** Class name of the component to be instantiated. **Value:** The class name of the component to be instantiated.

4.2.11 addUvmfScoreboard()

Name	addUvmfScoreboard
Description	This API instantiates a UVMF scoreboard

Usage	`addUvmfScoreboard(`
	`'`*`instanceName`*`',`
	`'`*`scoareboardType`*`',`
	`'transactionType'`
	`)`

Arguments	
`'`*`instanceName`*`'`	**Description:** Instance name given to the scoreboard. **Value:** Any valid SV instance name.
`'`*`scoareboardType`*`'`	**Description:** Type of scoreboard to be instantiated. **Value:** Any of the UVMF scoreboard types: uvmf_in_order_scoreboard, uvmf_out_of_order_scoreboard, uvmf_in_order_scoreboard_array, uvmf_in_order_race_scoreboard. Custom scoreboards extended from uvmf_scoreboard_base can also be used.
`'transactionType'`	**Description:** Type of the transaction to be received by the scoreboard. The type parameterization must match that of the component connected to this scoreboard. **Value:** Transaction type matching component connecting to this scoreboard.

4.2.12 addConnection()

Name	addConnection
Description	This API connects two UVM components

Usage	`addConnection(`
	`'componentAInstanceName',`
	`'componentAConnectionPoint',`
	`'componentBInstanceName',`
	`'componentBConnectionPoint'`
	`)`

Arguments	
'componentAInstanceName'	**Description:** Instance name of the component to be connected. **Value:** Instance name of the component to be connected.
'componentAConnectionPoint'	**Description:** Connection point of component to be connected. **Value:** Any valid UVM connection port or export.
'componentBInstanceName'	**Description:** Instance name of the component to be connected. **Value:** Instance name of the component to be connected.
'componentBConnectionPoint'	**Description:** Connection point of component to be connected. **Value:** Any valid UVM connection port or export.

4.2.13 addQvipConnection()

Name	addQvipConnection
Description	This API connects a UVM component to the analysis_port within a QVIP agent.
Usage	addQvipConnection('*componentAInstanceName*', '*componentAConnectionPoint*', '*componentBInstanceName*', '*componentBConnectionPoint*')
Arguments	
'*componentAInstanceName*'	**Description:** Instance name of the component to be connected. **Value:** Instance name of the component to be connected preceeded by the instance name of the environment containing the QVIP agent. Example: qvip_env_pcie_rc where the QVIP Configurator generated environment is named qvip_env and the agent contained in the environment is pcie_rc.
'*componentAConnectionPoint*'	**Description:** Connection point of component to be connected. **Value:** Any valid UVM connection port or export.
'*componentBInstanceName*'	**Description:** Instance name of the component to be connected. **Value:** Instance name of the component to be connected.
'*componentBConnectionPoint*'	**Description:** Connection point of component to be connected. **Value:** Any valid UVM connection port or export.

4.2.14 addUVMCflags()

Name	adddUVMCflags
Description	Defines compilation flags to be used in compiling files identified using addUVMCfiles API. Use this API once to define all flags used.

Usage	`addUVMCflags(`
	`'flags'`
	`)`

Arguments	
`'flags'`	**Description:** All flags to be used in compiling files identified using addUVMCfiles. **Value:** Any valid compilation flags

4.2.15 addUVMClinkArgs()

Name	addUVMClinkArgs
Description	Defines UVMC link arguments. Use this API once to define all arguments used.

Usage	`addUVMClinkArgs(`
	`'`*`linkArguments`*`'`
	`)`

Arguments	
`'`*`linkArguments`*`'`	**Description:** All link arguments to be used with UVMC. **Value:** Any valid link arguments

4.2.16 addUVMCfile()

Name	addUVMCfiles
Description	This API is used to add a file to the list of files to be compiled for use with UVMC. Use this API once for each file to be compiled. Each use of this API adds the *fileName* to a list of files.

Usage	`addUVMCfiles(`
	`'`*`fileName`*`'`
	`)`

Arguments	
`'`*`fileName`*`'`	**Description:** file name of C file to be compiled for use with UVMC. **Value:** Any valid file name and path.

4.2.17 addAnalysisPort()

Name	addAnalysisPort
Description	This API is used to add a single instance of uvm_analysis_port to the environment. It will be parameterized to the specified type of sequence item and connected to the specified TLM port. *Note: No checking for valid type or existing port is done*

Usage	`addAnalysisPort(`
	`'portName','transactionType','connection'`
	`)`

Arguments	
`'portName'`	**Description:** Name of the analysis port to be instantiated **Value:** Any valid class handle name
`'transactionType'`	**Description:** Parameterization to use for the analysis port, should be a valid sequence item type matching the parameterization of the connected port. **Value:** Any valid sequence item type name. *Note, no checking is done by the script to ensure validity.*
`'connection'`	**Description:** Reference to another analysis port within the environment that will drive this analysis port. Can use hierarchical references to underlying agents if needed. Will be used as the argument to this port's '`connect()`" call. **Value**: Any valid TLM port within the environment. *Note, no checking is done by the script to ensure validity.*

4.2.18

4.2.19 addAnalysisExport()

Name	addAnalysisExport
Description	This API is used to add a single instance of uvm_analysis_export to the environment. It will be parameterized to the specified type of sequence item and connected to the specified TLM port. *Note: No checking for valid type or existing port is done*

Usage	`addAnalysisExport(`
	`'portName','transactionType','connection'`
	`)`

Arguments	
`'portName'`	**Description:** Name of the analysis export to be instantiated **Value:** Any valid class handle name
`'transactionType'`	**Description:** Parameterization to use for the analysis export, should be a valid sequence item type matching the parameterization of the connected port. **Value:** Any valid sequence item type name. *Note, no checking is done by the script to ensure validity.*
`'connection'`	**Description:** Reference to another port within the environment that will be driven by this export. Can use hierarchical references to underlying agents if needed. Will be used as the argument to this port's '`connect()`" call. **Value**: Any valid TLM port within the environment. *Note, no checking is done by the script to ensure validity.*

4.2.20 addDPIFile()

Name	addDPIFile
Description	This API adds a file to the list of C files to be compiled.

Usage	`addDPIFile(`
	`'`*`fileName`*`')`

Arguments	
`'`*`fileName`*`'`	**Description:** Name of C file to be compiled and linked. **Value:** Any valid file name including its extension.

4.2.21 addDPICompArgs()

Name	addDPICompArgs
Description	This API adds compilation flags to be used when compiling the C files identified using addDPIFile API.

Usage	`addDPICompArgs(`
	`'compArgs')`

Arguments	
'compArgs'	**Description:** Compile arguments to be used when compiling the C files identified using addDPIFile API. **Value:** Any valid compilation flag for the c compiler.

4.2.22 addDPILinkArgs()

Name	addDPILinkArgs
Description	This API adds flags to be used when linking the C files identified using addDPIFile API.

Usage	`addDPIFile(`
	`'linkArgs')`

Arguments	
`'linkArgs'`	**Description:** Arguments to be used when linking the C files identified using addDPIFile API. **Value:** Any valid flag for the c linker.

4.2.23 addDPISOName()

Name	addDPISOName
Description	This API sets the name of the shared object created by linking all compiled objects.

Usage	`addDPISOName(`
	`'SOName')`

Arguments	
`'SOName'`	**Description:** Name of the shared object created by linking all compiled objects. **Value:** Any valid SO name.

4.2.24 addDPIImport()

Name	addDPIImport
Description	This API adds a C function to be imported into SV using DPI.

Usage	`addDPIImport(`
	`'returnType',`
	`'functionName',`
	`'functionArguments',`
	`{'SVVariableName':'SVVariableType',`
	` 'SVVariableName':'SVVariableType'}`
	`)`

Arguments	
`'returnType'`	**Description:**Return type of C function. **Value:** Any valid C return type when using DPI.
`'functionName'`	**Description:** Name of C function to be imported into SV using DPI **Value:** Any valid C function name.
`'functionArguments'`	**Description:** All arguments of C function. **Value:** Any valid list of C function arguments when using DPI.

`{'SVVariableName':'SVVariableType',`
` 'SVVariableName':'SVVariableType'}`
	Description: The list of systemVerilog arguments that are equivalent types to the list of C arguments.
	Value: Any valid SV variable name and type when using DPI.

4.2.25 addRegisterModel()

Name	addRegisterModel
Description	This API integrates a UVM based register model into the environment. A thin register model is created to enable simulation of environment as generated. The thin register model should be owerwritten by the actual register model.

Usage	`addRegisterModel(`
	`'sequencer',`
	`'transactionType',`
	`'regAdapterType',`
	`'busMap',`
	`'`*`useAdapter`*`',`
	` '`*`useExplicitPrediction`*`'`
	`)`

Arguments	
`'sequencer'`	**Description:** Path to sequencer to be used with register model through bus map. **Value:** Any valid uvm sequencer handle.
`'transactionType'`	**Description:** The class type of transaction used by the sequencer connected to the register model through the bus map. **Value:** The transaction type used by the sequencer.
`'regAdapterType'`	**Description:** The class type of the uvm register adapter to be used with the register model. **Value:** Register adapter class type.
`'busMap'`	**Description:** The bus map to be used with the register model. **Value:** The bus map to be used with the register model.
`'`*`useAdapter`*`'`	**Description:** Determines if the environment will use a uvm based register adapter. **Value:** True or False
`'`*`useExplicitPrediction`*`'`	**Description:** Determines if the environment will use a uvm based register predictor for explicit prediction. **Value:** True or False

4.3 Environment finalize API

4.3.1 Create()

The finalize section of user input contains the API that triggers file generation. This API is shown below and must be included in all user input files.

```
create()
```

5 Test Bench Code Generator

5.1 Test bench initialize API

The test bench initialize API creates a python object that contains all of the test bench body API's listed below. These test bench body API's are used to characterize information about the test bench. This information is used to create the following content:

Classes: top level test, top level virtual sequence.

Packages: top level test package, top level sequence package. Top level parameters package.

Modules: hdl_top, hvl_top.

Compilation flow: File list and Makefile

5.1.1 BenchClass()

This initialization API is used to create an object that stores all information about the test bench. An example use is shown below. The test bench name is block_b_ben. The environment package name is block_b. Environment parameters can be overridden using the format shown below. The object created is named ben. All body API's and variables are members of this object and therefore must be preceeded by the name of the object and separated by a period.

```
ben = uvmf_gen.BenchClass('block_b_ben','block_b',
     {'envParameter1': 'envParameter1Override',
      'envParameter2': 'envParameter2Override'})
```

5.1.2 Test bench variables

Name	**Value(s)**	**Description**
veloceReady	True|False	Interface generated is Veloce Ready|Friendly.
inFactReady	True|False	Test bench generated is inFact ready. Makefile contains variables, switches, and arguments to run inFact.
catapultReady	True|False	Enables HLS related requirements.
clockHalfPeriod	'timeValue'	Time duration of half period. Example: '6ns', or '6'
clockPhaseOffset	'timeValue'	Time duration before first clock edge. Exaple: '25ns' or '25'
resetAssertionLevel	True|False	Assertion level of reset signal driven by test

		bench.
resetDuration	‘timeValue’	Time duration reset is asserted at start of simulation. Example: ‘100ns’, or ‘100’

5.2 Test bench body APIs

These API’s are member functions of the test bench object created using the BenchClass initialization API. These API’s may be used in any number and any order. They must preceede the create() API.

5.2.1 addParamDef()

Name	addParamDef
Description	This API adds a parameter to the test level parameter package.

Usage	`addParamDef(`
	`'`*`parameterName`*`',`
	`'`*`parameterType`*`',`
	`'`*`parameterDefalutValue`*`'`
	`)`

Arguments	
`'`*`parameterName`*`'`	**Description:** Name of the parameter. **Value:** Any valid SV parameter name.
`'`*`parameterType`*`'`	**Description:** Type of the parameter. **Value:** Any valid SV parameter type.
`'`*`parameterDefalutValue`*`'`	**Description:** The default value for this parameter. **Value:** Any valid SV value for the parameter type.

5.2.2 addImport()

Name	addImport
Description	Adds an import of the selected package in the test package, sequence package and top level module.

Usage	`addImport(`
	`'packageName'`
	`)`

Arguments	
'packageName'	**Description:** The name of the package to be imported. **Value:** Any valid SV package. The location and compilation of this package must be added to the generated test bench.

5.2.3 addTopLevel()

Name	addTopLevel
Description	This API is used to add a top level module to the vsim command line. Use this API once for each top level to be included. Each use of this API adds the *moduleName* to a list of modules.

Usage	`addUVMCfiles(`
	`'`*`moduleName`*`'`
	`)`

Arguments	
`'`*`moduleName`*`'`	**Description:** module name to be added as a top level module. **Value:** Any valid module name.

5.2.4 addBfm()

Name	addBfm
Description	This API adds a monitor BFM to the top level module, hdl_top. If the *activity* argument is ACTIVE then a driver BFM is also added to hdl_top. The test bench clock and reset are connected to these BFM's. If the *activity* argument is ACTIVE then the top level virtual sequence will automatically start a random sequence on this BFM.

Usage	`addBfm(`
	`'`*`instanceName`*`',`
	`'`*`protocol`*`',`
	`'`*`clock`*`',`
	`'`*`reset`*`',`
	`'`*`activity`*`',`
	`{'`*`protocolParam1`*`':'`*`protocolParam1Override`*`',`
	` '`*`protocolParam2`*`':'`*`protocolParam2Override`*`'}`
	`)`

Arguments	
`'`*`instanceName`*`'`	**Description:** Instance name of this BFM. **Value:** Any valid SV instance name.
`'`*`protocol`*`'`	**Description:** Name of the protocol package. Example: mem_pkg would use the value mem **Value:** Any valid SV package name.
`'`*`clock`*`'`	**Description:** The name of the primary clock given in the user input file for this protocol. **Value:** Must match value given to clock variable in user input file for this protocol.
`'`*`reset`*`'`	**Description:** The name of the primary reset given in the user input file for this protocol. **Value:** Must match value given to reset variable in user input file for this protocol.
`'`*`activity`*`'`	**Description:** Determines the ACTIVE/PASSIVE state for this BFM. **Value:** ACTIVE|PASSIVE
`'`*`protocolParamN`*`'`	**Description:** Name of the interface package parameter to be overridden. **Value:** Must match the interface parameter name to be overridden.
`protocolParamNOverride``'`	**Description:** The value used to override the parameters default value. **Value:** Any valid SV value for overriding the identified parameter. Example: '5' or '*testLevelParameterName*'

5.2.5 addQvipBfm()

Name	addQvipBfm
Description	Add a QVIP interface BFM. This API is generated by the QVIP Configurator. It is located in the package as SV comments.

Usage	`addQvipBfm(`
	`'`*`instanceName`*`',`
	`'`*`envPackage`*`',`
	`'`*`activity`*`'`
	`)`

Arguments	
`'`*`instanceName`*`'`	**Description:** Instance name of this BFM. **Value:** Name given to instance in QVIP Configurator.
`'envPackage'`	**Description:** Name of the environment package generated using the QVIP Configurator. **Value:** Value given test bench in QVIP Configurator.
`'`*`activity`*`'`	**Description:** Determines the ACTIVE/PASSIVE state of the BFM. **Value:** ACTIVE|PASSIVE

5.3 Test bench finalize API

5.3.1 Create()

The finalize section of user input contains the API that triggers file generation. This API is shown below and must be included in all user input files.

```
create()
```