GenSys®
Customization Guide

Version O-2018.06, June 2018

SYNOPSYS®

Synopsys, Inc.
690 E. Middlefield Road
Mountain View, CA 94043
www.synopsys.com

Contents

Preface

This preface includes the following sections:

- About This User Guide
- Customer Support

About This User Guide

The *Gensys Customization Guide* describes details on customizing GenSys by using different APIs.

Audience

The Formality User Guide provides information about Formality concepts, procedures, file types, menu items, and methodologies with a hands-on tutorial to get you started with the tool.

Additionally, you need to understand the following concepts:

- Logic design and timing principles
- Logic simulation tools
- Linux operating system

Related Publications

For additional information about the GenSys tool, see the documentation on the Synopsys SolvNet® online support site at the following address:

https://solvnet.synopsys.com/DocsOnWeb

You might also want to see the documentation for the following related Synopsys products:

- Design Compiler®
- HDL Compiler™
- PrimeTime® Suite
- ESP

Related Publications

For additional information about GenSys, see the documentation on SolvNet at the following address:

https://solvnet.synopsys.com/DocsOnWeb

Release Notes

Information about new features, enhancements, changes, known limitations, and resolved Synopsys Technical Action Requests (STARs) is available in the *GenSys Release Notes* on the SolvNet site.

To see the *GenSys Release Notes*,

1. Go to the SolvNet Download Center located at the following address:

   https://solvnet.synopsys.com/DownloadCenter

2. Select GenSys, and then select a release in the list that appears.

Conventions

The following conventions are used in Synopsys documentation.

Convention	Description
`Courier`	Indicates syntax, such as `write_file`.
`Courier italic`	Indicates a user-defined value in syntax, such as `write_file` *`design_list`*.
`Courier bold`	Indicates user input—text you type verbatim—in examples, such as `prompt>` **`write_file top`**
[ ]	Denotes optional arguments in syntax, such as `write_file [-format` *`fmt`*`]`
...	Indicates that arguments can be repeated as many times as needed, such as *`pin1 pin2 ... pinN`*
|	Indicates a choice among alternatives, such as `low | medium | high`
Ctrl+C	Indicates a keyboard combination, such as holding down the Ctrl key and pressing C.
\	Indicates a continuation of a command line.
/	Indicates levels of directory structure.
Edit > Copy	Indicates a path to a menu command, such as opening the Edit menu and choosing Copy.

Customer Support

Customer support is available through SolvNet online customer support and through contacting the Synopsys Technical Support Center.

Reporting an Error

The Technical Publications team welcomes your feedback and suggestions on this publication. Provide specific feedback and, if possible, attach a snapshot. Send your feedback at spyglass_support@synopsys.com.

Accessing SolvNet

The SolvNet site includes a knowledge base of technical articles and answers to frequently asked questions about Synopsys tools. The SolvNet site also gives you access to a wide range of Synopsys online services including software downloads, documentation, and technical support.

To access the SolvNet site, go to the following address:

https://solvnet.synopsys.com

If prompted, enter your user name and password. If you do not have a Synopsys user name and password, follow the instructions to sign up for an account.

If you need help using the SolvNet site, click HELP in the top-right menu bar.

Contacting the Synopsys Technical Support Center

If you have problems, questions, or suggestions, you can contact the Synopsys Technical Support Center in the following ways:

- Open a support case to your local support center online by signing in to the SolvNet site at https://solvnet.synopsys.com, clicking Support, and then clicking “Open A Support Case.”
- Send an e-mail message to your local support center.
    - E-mail support_center@synopsys.com from within North America.
    - Find other local support center e-mail addresses at

      https://www.synopsys.com/support/global-support-centers.html
- Telephone your local support center.

- Call (800) 245-8005 from within North America.
- Find other local support center telephone numbers at https://www.synopsys.com/support/global-support-centers.html

1

Creating Custom Generators

This section describes the following:

- Creating Custom Generators Using GenSys Objects
- Creating Custom Generators Using Perl APIs
- Creating Custom Generator to Change Net Names in Generated RTL
- Creating Custom Generators Using IP-XACT TGI API Functions

Creating Custom Generators Using GenSys Objects

Creating custom generators involves:

1. Registering the Custom Generator
2. Creating the Custom Generator Function
3. Specifying the Custom Generator File
4. Running the Custom Generators

Registering the Custom Generator

Create a text file named genesis.pl that contains the following contents:

```
use
genesisapis;RegisterGenerator(      <genName>,      <genFuncName>);...
```

Where <*genName*> is the custom generator's name (displayed in the Generator Menu of the GenSys Main Window) and <*genFuncName*> is the name of the custom generator's Perl function.

For example:

```
RegisterGenerator(  “myGen1”,  “myGen1_func”,);
```

You can declare as many custom generators as required in one genesis.pl file.

The genesis.pl file should require the Perl file containing the custom generator's Perl function.

Creating the Custom Generator Function

The Custom Generator Function File

The custom generator function should be described in a Perl file that should contain the following call at the start:

```
use Genesis;
```

Then, you create the custom generator function and associated functions, if any.

The Perl file should end with:

```
1;
```

You can create as many custom generator functions as required in one Perl file.

Name of the Perl file containing a custom generator function is same as that declared in the genesis.pl file for that custom generator.

The Custom Generator Function

Each custom generator function should follow the following format:

```
sub <genFuncName>{  my $root = Genesis->new("root");
...}...
```

Here, <*genFuncName*> is the custom generator function name (same name as that declared in the genesis.pl file for this custom generator).

All custom generator functions must start from the comRoot object using the Perl new method with root as the argument value. Then, the function can traverse the design hierarchy under the comRoot object to other objects using the methods as described in Object-to-Methods Tree and GenSys Perl Methods.

The NULL-checking for the comRoot object is recommended.

You can create additional functions in different files and require these files in the main Perl file containing the custom generator function.

Specifying the Custom Generator File

Specify the directory containing your genesis.pl file using the -I command-line while invoking GenSys. Then, GenSys reads the genesis.pl file located in the specified directory and loads the custom generators declared in this file.

You can also create multiple genesis.pl files located in different directories and specify them using the -I command-line while invoking GenSys. Then, GenSys reads the files as they are specified and loads the custom generators declared in these files.

In addition to the above, you can also load custom generators by saving the genesis.pl file in the AddInGenerators folder in the current working directory of GenSys. You can also keep multiple genesis.pl files in sub folders of the AddInGenerators folder. All valid custom generators declared in these files are loaded when GenSys is launched.

Running the Custom Generators

All valid custom generators are added to the Generator Menu of the GenSys Main Window.

To run a custom generator, select the custom generator from the Generator Menu.

Creating Custom Generators Using Perl APIs

Creating custom generators involves:

1. Registering the Custom Generator
2. Creating the Custom Generator Function
3. Specifying the Custom Generator File
4. Running the Custom Generators
5. Evaluating Performance of Custom Generators
6. Modifying the Custom Generator Function Parameters

Registering the Custom Generator

Create a text file named genesis.pl that contains the following contents:

```
use
genesisapis;RegisterGenerator(      <genName>,      <genFuncName>);...
```

Where <*genName*> is the custom generator's name (displayed in the Generator Menu of the GenSys Main Window) and <*genFuncName*> is the name of the custom generator's Perl function.

For example:

```
RegisterGenerator(  “myGen1”,  “myGen1_func”);
```

You can register as many custom generators as required in one genesis.pl file.

The genesis.pl file should require the Perl file containing the custom generator's Perl function.

Creating the Custom Generator Function

The Custom Generator Function File

The custom generator function should be described in a Perl file that should contain the following call at the start:

```
use genesisapis;
```

Then, you create the custom generator function and associated functions, if any.

The Perl file should end with:

```
1;
```

You can create as many custom generator functions as required in one Perl file.

Name of the Perl file containing a custom generator function is same as that declared in the genesis.pl file for that custom generator.

The Custom Generator Function

Each custom generator function should follow the following format:

```
sub <genFuncName>{  my $com_root =
GetcomRoot();  if(!$com_root){    printf ("ERROR: comRoot is
NULL!!!\n");    exit 0;  }...}...
```

Here, *<genFuncName>* is the custom generator function name (same name as that declared in the genesis.pl file for this custom generator).

All custom generator functions must start from the comRoot object using the GetcomRoot API function. Then, the function can traverse the design hierarchy under the comRoot object to other objects as described in Design Hierarchy.

The NULL-checking for the comRoot object is recommended.

While designing the custom generator function, you can use the GenSys API functions.

Please refer the example generator files located in the $SPYGLASS_HOME/auxi/Genesis/generators directory while creating your custom generator functions.

You can create additional functions in different files and require these files in the main Perl file containing the custom generator function.

Specifying the Custom Generator File

Specify the directory containing your genesis.pl file using the -I command-line while invoking GenSys. Then, GenSys reads the genesis.pl file located in the specified directory and loads the custom generators declared in this file.

You can also create multiple genesis.pl files located in different directories and specify them using the -I command-line while invoking GenSys. Then, GenSys reads the files as they are specified and loads the custom generators declared in these files.

In addition to the above, you can also load custom generators by saving the genesis.pl file in the AddInGenerators folder in the current working directory of GenSys. You can also keep multiple genesis.pl files in sub folders of the AddInGenerators folder. All valid custom generators declared in these files are loaded when GenSys is launched.

Running the Custom Generators

All valid custom generators are added to the Generator Menu of the GenSys Main Window.

To run a custom generator, select the custom generator from the Generator Menu.

Evaluating Performance of Custom Generators

To evaluate the performance of Perl functions called from custom generators, GenSys allows you to generate trace details for these functions. Tracing is done for the functions registered as generators, validation functions, change functions, or color functions in the schema.

If GenSys is run using the --perflog command-line option, GenSys logs the details of execution of Perl functions in a log file, gensys.perflog. For each execution of a Perl function, the time taken to execute the function and the memory consumed for that run are logged. In addition, the total time taken for multiple runs of a Perl function in the current session is also logged.

You can also print your own custom messages in gensys.perflog using the Tcl command, print_to_perf_log.

GenSys also allows you to print the total time taken by a Perl function in the Log window in GenSys GUI. You can do this by using the Tcl command, dump_perl_stats. For more details on these Tcl commands, refer to the *Tcl Commands Reference Guide*.

Modifying the Custom Generator Function Parameters

In GenSys, the generators and other custom functions that are invoked from the Generator menu, can display a form or dialog (Custom Generated dialog), which allows you to see the list of parameters recognized by the function, along with their initial values; you can also modify any of those values before proceeding with the action.

GenSys provides a Perl API (available in the GenGui package) createGeneratorDialog that takes the parameter list as an array of hashes, along with the validation and execution callbacks. See the Example 1 section for details on how the parameter and its values can be modified.

Validation Callback Functions

These callback functions validate the parameter values on the form. The functions return a GO/NO-GO result, modify the parameter list to identify the invalid parameters, and provide diagnostic messages for each, as described later in this section.

Execution Callback Functions

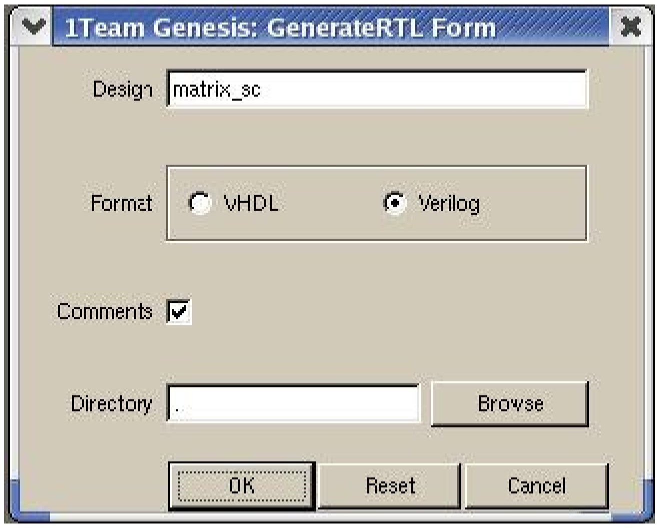
These callback functions are called if the validation succeeds. This callback is passed the name-value pairs in a hash, as described in the example below, and it returns information about generated files/directories or diagnostic messages, as described later in this section.

The Custom Generated Dialog

Apart form the parameters, a custom generated dialog contains three standard buttons. Each button performs a specific function, described below:

- **OK**: This button passes an array list of hashes as a reference to the validation callback. If the validation call is successful, it passes the name-value pairs of all the fields as a hash to the execution callback. If the validation call fails, the dialog updates itself according to the updated IsValid and Message field values.
- **Reset**: This button resets the values of the fields to the value provided in the InitValue field.
- **Cancel**: This button cancels the invocation of the dialog.

Following figure shows an example of the custom generated dialog for the custom generator RTLGenerator (See the Example 1 section for details on the RTLGenerator custom generator):


1Team Genesis: GenerateRTL Form

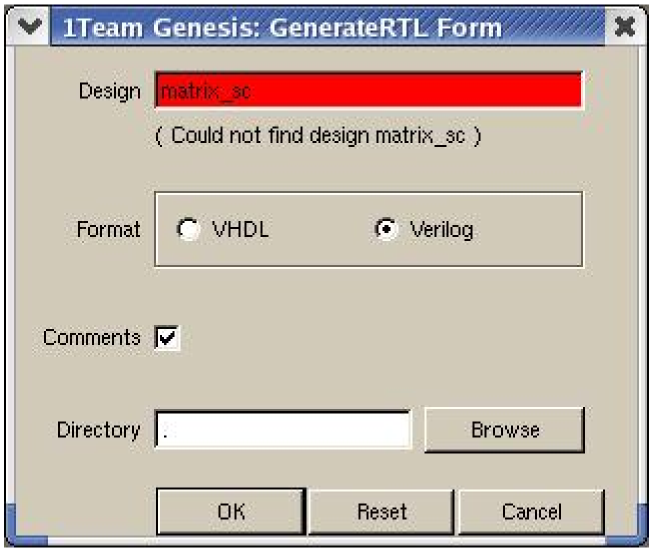
Design: matrix_sc

Format: VHDL [✓ Verilog]

Comments: ✓

Directory: .

The following figure indicates an invalid value entered for a parameter used in the custom generator RTLGenerator:



Design matrix_sc

(Could not find design matrix_sc)

Format ( ) VHDL (•) Verilog

Comments [✓]

Directory .

Example 1

As described in the example below, you can register the menu option RTLGenerator to the Generator menu. When the RTLGenerator menu option is selected, the generateRTL function is called.

```
menuSetItems('Generator', [    ['cascade',    -text => 'Custom Dialog
Generators',    -underline => 0,    -menuitems =>    [['action',
-text => 'RTLGenerator',     -underline => 0,     -sub =>
\&generateRTL,     ],],  ],]);
```

The createGeneratorDialog call in generateRTL constructs the dialog with the parameters Design, Format, Directory, and Comments and registers the validation and execution callbacks.

Following are the various keys of the hash that is used to pass the details of the parameters to be created.

- Name: Specifies name of the parameter.
- Description: Specifies description of the parameter.
- Value: Specifies value of the parameter.
- InitValue: Specifies value to which the parameter is set if the user clicks the Reset button.
- ViewType: Specifies input type as it should be displayed on the dialog. The valid values are String(default) | Enum | EditableEnum | Dir | File | MultiSelect | RadioButton | CheckBox.

- PossibleValues: Specifies ;(semi colon) separated values that the parameter can take and that are to be shown to the user to choose from. This key is valid only if ViewType is Enum | EditableEnum | File | MultiSelect | RadioButton .

  If ViewType is File the value of this parameter acts as the filter for type of files to be shown. For example, "*.tcl;*.cmd" . If ViewType is MultiSelect then Value is ; (semi colon) separated list of selected values.
- IsValid: Specifies whether the parameter field has an invalid value. Takes values 0 | 1(default)
- Message: Specifies diagnostic message for the parameter. This message is shown alongside the parameter in case the parameter has an invalid value(i.e IsValid is 0).

The following code generates a dialog with the parameters, Design, Format, Comments, and Directory.

```
sub generateRTL{
      createGeneratorDialog("GenerateRTL",
      [{
       Name => "Design",
       Description => "The name of the design",
       Value => "matrix_sc",
       InitValue => "matrix_sc",
       IsValid => "1",
       Message => "",
       ViewType => "String"
      },
      {
       Name => "Format",
       Description => "RTL netlist format",
       Value => "Verilog",
       PossibleValues => "VHDL;Verilog",
       InitValue => "Verilog",
       Message => "",
       IsValid => "1",
       ViewType => "RadioButton"
      },
      {
       Name => "Comments",
       Description => "Include comments in RTL ?",
       Value => "1",
       InitValue => "1",
       IsValid => "1",
       Message => "",
       ViewType => "CheckBox"
```

```
        },
        {
         Name => "Directory",
         Description => "Directory where files will be
                 dumped",
         Value => ".",
         InitValue => ".",
         IsValid => "1",
         Message => "",
         ViewType => "Dir"
        }],\&validate,\&executeGenerator);
}
```

The following code validates the parameter values on the dialog/form.

```
sub validate{       my $retStatus = 1;       my $arg_list = shift;
my @hashlist = @{$arg_list};        foreach
$item(@hashlist)       {           my $currFieldName =
$item->{'Name'};          my $currFieldValue =
$item->{'Value'};          if($currFieldName eq
"Design")          {              my $root->Genesis->new("root") or die
                            "Could not find root";            my
$des->designs($currFieldValue);            if($des){}            else
           {              $item->{'IsValid'} =
'0';             $item->{'Message'} = "Could not find design
$currFieldValue";              $retStatus = 0;              }       }
}    return $retStatus;}
```

Once the validation succeeds, the execution callback is called with the name-value pairs of the parameters in a hash. The name-value pairs can be accessed in the execution callback as shown below.

```
sub executeGenerator{    #access the name value pairs in the hash    my
$hashRef = shift;    my %hash = %{$hashRef};    while(my($name,$value) =
each(%hash)){    }}
```

The execution callback can return information about the generated files/directories or diagnostic messages in a hash as described below.

Following are the various keys of the hash that is returned by the execution callback.

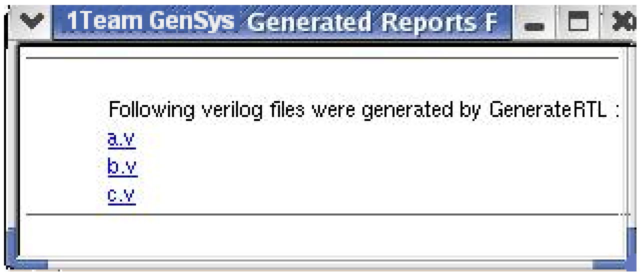
- ResType : Specifies what kind of information is the callback returning. Valid values are File | Dir | Message
- ResSubType : In case ResType is Message, this specifies the type of the message Info(default) | Error | Warning. If ResType is File, it describes the file type log | xml | verilog | vhdl | pdf | doc | report to specify the type of viewer to be used to display the file if the user wants to review it from the report.
- ResValue : Specifies a ; (semi colon) separated list of generated files/dirs in case ResType is File | Dir . If ResType is Message, this contains the message itself.

When the execution callback returns, based on ResType either a report of the generated files/dirs is shown or a message is displayed indicating that no files are generated.

For example, if ResType is File, a report of the generated files is displayed, as shown below:

```
       my $info = {          ResType => "File",          ResSubType =>
"verilog"          ResValue => "a.v;b.v;c.v"        }       return
$info;
```

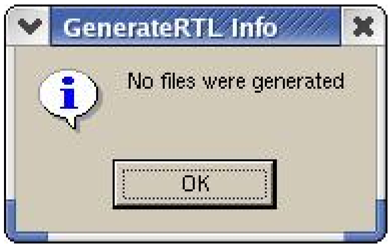
The following figure shows the report listing the files generated by RTL:



Following verilog files were generated by GenerateRTL :
a.v
b.v
c.v

```
In another example, if ResType is Message, a message is displayed, as
shown below:
my $info = {            ResType => "Message",            ResSubType =>
"Info"           ResValue => "No files were generated"
            }return $info;
```

The following figure shows the message that appears:



No files were generated

Creating Custom Generator to Change Net Names in Generated RTL

You can write your own custom generator to change net names in the generated RTL.

Example 2

Consider an example in which you want to generate net names based on the destination of a connection. In this case, you can write a custom generator in which:

- You write a function that accepts source terminal, and provide the required functionality to change the net name based on destination. This function should return the final net name.
- You pass this function to the genRTLNetNamingConvention Perl API. This API is used to traverse the database and generate a new net name.

The following is a sample generator file that implements the above functionality.

```
use Genesis;use genesisapis;use warnings;sub net_name_function{  ##
Function is passed the Source Terminal ##  my $srcTerm =
shift;  if(!$srcTerm) { return ""; }  ## try to find the source instance
name and source portname ##  my $srcPort =
Genesis::TerminalGetPort($srcTerm);  my $srcInst =
Genesis::TerminalGetParent($srcTerm);  if(!defined($srcInst)) { $srcInst
= ""; }  my $srcInstName =
Genesis::ComponentInstGetName($srcInst);  if((!defined($srcInstName)) or
($srcInstName eq "")) {    $srcInst =
Genesis::TerminalGetParent($srcInst);    $srcInstName =
Genesis::ComponentInstGetName($srcInst);  }  if(!defined($srcInstName)) {
$srcInstName = ""; }  if(!defined($srcInst)) { $srcInst = ""; }  ##
Iterate over all the fanouts of this driver net ##  my $fanoutList =
Genesis::RTLTerminalGetFanoutList($srcTerm);  if(!defined($fanoutList)) {
$fanoutList = 0; }  my $dstTerm = "";  my $dstInst =
"";  while($fanoutList && ($newDstTerm =
iter_getdata($fanoutList)))   {   iter_next($fanoutList);   ## find the
driven port and instance name ##   my $destPort =
Genesis::TerminalGetPort($newDstTerm);   my $portName =
Genesis::PortGetName($destPort);   my $newDstInst =
Genesis::TerminalGetParent($newDstTerm);   my $newDstInstName =
Genesis::ComponentInstGetName($newDstInst);   if(!defined($newDstInstName
)) { $newDstInstName = ""; }   if($newDstInstName eq "") {    $newDstInst
= Genesis::TerminalGetParent($newDstInst);    $newDstInstName =
Genesis::ComponentInstGetName($newDstInst);   }   if(!defined($newDstInst
Name)) { $newDstInstName = "";
next;}   if(!defined($portName)){next;}   if(!defined($newDstInst)) {
$newDstInst = 0; next;}   ## create a new  net name --> based on the
destination -->   ## assumption is that there is a    ## single driver
--> else, apply some logic here... ##   $dest_based_net_name =
$newDstInstName."_".$portName;   return $dest_based_net_name;  }  return
"";}genRTLNetNamingConvention("net_name_function");1;
```

Example 3

Consider an example in which you want to follow a separate naming convention for net names for IO and non-IO connections. In this case, you can create a Perl function, funcName, that accepts the following two arguments:

- First argument is the TerminalClass pointer (corresponding to an instance terminal driving a connection).
- Second argument is a default net name which is automatically generated by GenSys if the funcName function does not return anything.

Following is the example of implementing the specified Perl function to accomplish this task:

```
use Genesis;sub generate_net_name_new{  #printg "*********** inside
generate_net_name_new  #function ****************************";  my
$srcInstType = shift;  my $dstInstType = shift;  if(!defined
$srcInstType)  {    $srcInstType = "";  }  if(!defined
$dstInstType)  {    $dstInstType = "";  }  my $srcTerm = shift;  my
$dstTerm = shift;  my $devPin = shift;  my $ioPath = shift;  my $ioMode =
shift;  my $retVal = "";  if(($srcInstType eq "CORE") || ($dstInstType eq
"CORE"))  {    #MUX to core connectivities ; Get
$retVal  }  elsif(($srcInstType eq "PAD") || ($dstInstType eq
"PAD"))  {    #suitable IO operation to get $retVal    --> other
operations    return $retVal;  }sub io_net_naming_new{##############IO
based functionality starts##############my ($srcTerm,$netName) =
@_;if(!$srcTerm){  if(!$netName)  {    return "";  }}my $srcPort =
Genesis::TerminalGetPort($srcTerm);my $srcInst =
Genesis::TerminalGetParent($srcTerm);if(!defined($srcInst)){  $srcInst =
"";}my $srcInstName =
Genesis::ComponentInstGetName($srcInst);if((!defined($srcInstName)) or
($srcInstName eq "")){  $srcInst =
Genesis::TerminalGetParent($srcInst);  $srcInstName =
Genesis::ComponentInstGetName($srcInst);}if(!defined($srcInstName)){  $sr
cInstName = "";}if(!defined($srcInst)){  $srcInst = "";}my $deviceName =
"";my $ioPath = "";my $ioMode = "";my $srcInstType = "";my $dstInstType =
"";if((defined($srcInstName)) and ($srcInstName ne "")){  $devicePinName
=  Genesis::AbstractBaseGetAttrValue($srcInst,"DEVICE-PIN");  $deviceName
= getDevicePinName($devicePinName);  $ioPath =
Genesis::AbstractBaseGetAttrValue($srcInst,"IO-PATH");  $ioModeVal=
Genesis::AbstractBaseGetAttrValue($srcInst,"IO-MODE");  my $ioMode =
getIoModeString($ioModeVal);  $srcInstType =
Genesis::AbstractBaseGetAttrValue($srcInst,"IO-TYPE");  if(!defined($devi
ceName))  {    $deviceName = "";  }  if(!defined($ioPath))  {    $ioPath
= "";  }  if(!defined($ioMode))  {    $ioMode = "";
  }  if(!defined($srcInstType))  {    $srcInstType =
"";  }  if($srcInstType eq "")  {    my $coreAttrVal =
    Genesis::AbstractBaseGetAttrValue($srcInst,"CORE");    my
$selectCntrlAttrVal =
    Genesis::AbstractBaseGetAttrValue($srcInst,"SELECT_CONTROLLER");    i
f($coreAttrVal)    {      $srcInstType =
"CORE";    }    elsif($selectCntrlAttrVal)    {      $srcInstType =
"SELECT_CONTROLLER";    }  }}else{  $srcInstType = "TOP_DESIGN";}# Other
IO operations can be includedmy $retVal = "";$retVal =
generate_net_name_new($srcInstType, $dstInstType,$srcTerm, $dstTerm,
$deviceName, $ioPath, $ioMode);if((defined($retVal)) and ($retVal ne
"")){  #suitable IO operation to get $retVal}if($retVal eq $srcInstName)
{  $retVal = "";}
#################### IO based functionality ends #################
################ Non-IO based functionality  starts
##############if($retVal eq ""){  if($netName)  {    $returnedNetName =
uc($netName);    return $returnedNetName;  }}################ Non-IO
based functionality ends ################return $retVal;}
############### perl Subroutine that is registered ###############sub
funcName{  my ($a,$b) = @_;  $newnetname =
io_net_naming_new($a,$b);  return $newnetname; }
###########registering user perl function funcName through
genRTLNetNamingConvention#########sub SetNetName
{  genRTLNetNamingConvention("funcName");}
RegisterGenerator("SetNetName","SetNetName");1;
```

Creating Custom Generators Using IP-XACT TGI API Functions

Such generators are called Tight Generator Interface (TGI) generators.

TGI is a mechanism that allows a client generator to access the design or component description of a Design Environment (DE) present at a remote location.

The DE and the generator communicate with each other by sending messages using Simple Object Access Protocol (SOAP) standard specified in the Web Services Description Language (WSDL). SOAP provides a simple mechanism for sending messages in an XML format by using the Hyper Text Transfer Protocol (HTTP) or other transport protocols.

Creating TGI generators involves the following steps:

1. Registering the TGI Generator
2. Specifying the TGI Generator File
3. Running the TGI Generators

Registering the TGI Generator

Create a text file named genesis.pl that contains the following contents:

```
use genesisapis;RegisterTgiGenerator(      <genName>,      <url>);...
```

Where <*genName*> is the name of the TGI generator (displayed in the Generator Menu of the GenSys Main Window), and <*url*> refers to the location of the client generator, which can be HTTP path or file path information of the client.

Consider the following example to register a TGI generator:

```
RegisterTgiGenerator(  "mytgi_gen",  "http://trishul:53184/
"  $arg1,  $arg2);
```

The above command registers the TGI generator, mytgi_gen, in GenSys for the client generator available at the path, http://trishul:53184/. Here, $arg1 and $arg2 are the arguments of the generator, mytgi_gen.

You can register as many TGI generators as required in one genesis.pl file.

Writing a TGI Generator

GenSys supports a set of pre-defined TGI Perl API functions using which a client creates TGI generators. These API functions are listed below:

TGI API Function	Purpose
addAdHocExternalPortReference	Adds an external port reference to an existing adhoc connection
addAdHocInternalPortReference	Adds an internal port reference to an existing adhoc connection. An identical port reference must not already exist in the ad-hoc connection.
addComponentInstance	Adds a new component instance
addHierConnection	Adds a new hierarchical connection
addInterconnection	Adds a new interconnection between components
addMonitorInterconnection	Adds a new interconnection between a component and monitor. If there is already a monitorInterconnection for the given componentRef/componentInterfaceRef, the monitor connection is added to that element
end	Terminates the connection to the Design Environment
getAdHocConnectionExternalPortDetails	Returns a list for an external connection containing the portRef, left, and right attribute values
getAdHocConnectionInternalPortReferenceDetails	Returns a list for an internal connection containing the componentRef, portRef, left, and right attribute values
getBusDefinitionExtends	Returns the vendor, library, name, and version (VLNV) of the bus definition being extended
getBusDefinitionID	Returns the ID for the bus definition with the given VLNV
getBusDefinitionVLNV	Returns the Vendor Library Name Version of the bus definition

TGI API Function	Purpose
getComponentInstanceComponentID	Returns the ID for the component associated with the given instance (crossing from design to component file)
getComponentInstanceConfigurableElementIDs	Returns the configurable element IDs of the given component instance. The use of this function is not recommended.
getComponentInstanceName	Returns the instance name of the component
getComponentInstanceVLNV	Returns the vendor, library, name, and version of the component (from the design file)
getComponentGeneratorIDs	Returns the list of generator IDs of the component
getComponentPortIDs	Returns the list of component model port IDs
getComponentVLNV	Returns the vendor, library, name, and version of the component (from the component file)
getComponentViewIDs	Returns the list of model view IDs
getComponentInstanceID	Returns the component instance ID of the named component instance in the given design
getDesignAdHocConnectionIDs	Returns the list of adhoc connection element IDs
getDesignComponentInstanceIDs	Returns component instance IDs of the given design
getDesignHierConnectionIDs	Returns the list of hierarchical connection element IDs
getDesignID	Returns the ID of the current or top design
getDesignInterconnectionIDs	Returns the list of interconnection element IDs
getDesignMonitorInterconnectionIDs	Returns the list of monitorInterconnection element IDs
getDesignVLNV	Returns the vendor, library, name, and version of the design
getGeneratorApiType	Returns the API type of the generator
getGeneratorExecutable	Returns the executable name associated with the generator

TGI API Function	Purpose
getGeneratorGroups	Returns the list of group names of the generator
getGeneratorIsHidden	Returns the value of hidden attribute on the generator
getGeneratorPhase	Returns the phase number of the generator
getGeneratorPhaseScope	Returns the scope of the generator phase: local or global
getGeneratorScope	Returns the scope of the generator
getGeneratorTransportMethods	Returns the list of transport methods of the generator
getHierConnectionDetails	Returns the list containing the interface name, component reference, and interface reference
getInterconnectionActiveInterfaces	Returns the active interfaces as a list: componentID1 interfaceID1 componentID2 interfaceID2.
getMonitorInterconnectionInterfaces	Returns the active interface and monitor interfaces as a list in componentID interfaceID format. The active interface comes first in the list.
getName	Returns the name of the specified element
getPortDefaultValue	Returns the default value of the port
getPortDirection	Returns the direction of the port
getPortRange	Returns the list of the left and right range of the port
getVendorExtensions	Returns the complete XML text of the vendor extension element including the spirit:vendorExtension tag, as a well formed XML document
init	This is an API initialization function. It must be called before any other API call
removeAdHocExternalPortReference	Removes the external port reference from an existing adhoc connection
removeAdHocInternalPortReference	Removes the internal port from existing adhoc connection. The adhoc connection is removed when the last port reference is removed.

TGI API Function	Purpose
removeComponentInstance	Removes the specified component instance
removeHierConnection	Removes the existing hierarchical connection
removeInterconnection	Removes the interconnection between components
removeMonitorInterconnection	Removes the interconnection between a component and monitor. When the last monitor reference is removed, the entire monitorInterconnection element will be removed.
replaceComponentInstance	Replaces the specified component with the new component provided
setPortDefaultValue	Sets the default value of the given port
setPortRange	Sets the left/right range for the given port
getBusDefinitionDirectConnection	Indicates whether or not the bus definition supports direct connections
getBusDefinitionMaxMasters	Maximum # of masters supported by this bus definition
getBusDefinitionMaxSlaves	Maximum # of slaves supported by this bus definition.
getBusDefinitionSystemGroupNames	List of system group names for this bus definition
getBridgeIsOpaque	Value of the opaque attribute
getBridgeMasterID	The slave interface master interface reference ID
getBusInterfaceBitSteering	BitStreering description of the bus interface: on or off
getBusInterfaceBitsInLAU	The number bits in the least addressable unit. If none exists, the default 8 bits is returned
getBusInterfaceConnectionRequired	Connection required for this bus interface
getBusInterfaceEndianness	The endianess of the bus interface, big or little. The default is little
getBusInterfaceGroupName	Group name of a system, mirroredSystem, or monitor bus interface

TGI API Function	Purpose
getBusInterfaceMasterAddressSpaceID	ID of the master addressSpace
getBusInterfaceMasterBaseAddress	Base address of the master addressSpace
getBusInterfaceMirroredSlaveRange	The address range of the mirrored slave interface
getBusInterfaceMirroredSlaveRemapAddressIDs	List of remap address IDs of the mirrored slave interface
getBusInterfaceMonitorInterfaceMode	Indicates the mode of interface being monitored, slave, master, system, mirrorslave, mirrormaster or mirrorslave
getBusInterfaceSlaveBridgeIDs	List of slave bridge IDs
getBusInterfaceSlaveFileSetGroupIDs	List of fileSetGroup IDs
getBusInterfaceSlaveMemoryMapID	ID of the memoryMap referenced from a slave interface
getRemapAddressRemapStateID	Remap state ID of the given remap address element
getRemapAddressValue	Remap address of the given remap address element
setBusInterfaceBitSteering	Set bus interface bit steering value
setBusInterfaceMasterBaseAddress	Set base address of the master bus interface
setBusInterfaceMirroredSlaveRange	Set address range for the associated interface
setRemapAddressValue	Set remap address value for the associated interface
getChannelBusInterfaceIDs	List of busInterface IDs in this channel
getComponentBusInterfaceIDs	List of interface IDs.
getComponentChannelIDs	A list of channel IDs
getComponentChoiceIDs	List of choices IDs
getComponentCpuIDs	List of cpu IDs of the component

TGI API Function	Purpose
getComponentElementType	Returns the name of the XML element associated with the component (currently only 'component'). This call is being provided to cover a future scenario where there can be different types of component elements instantiated in a design (e.g. macroComponent elements)
getComponentFileSetIDs	List of file set IDs
getComponentOtherClockDriverIDs	List of clock driver IDs of the component
getComponentRemapStateIDs	A list of remap state IDs
getComponentWhiteboxElementIDs	List of whitebox element IDs of the component
getCpuAddressSpaceIDs	List of address space reference IDs of the cpu
getAdHocConnectionExternalPortReferenceIDs	List of external ad-hoc port reference element IDs. Hyper edge support not provided in this API.
getAdHocConnectionInternalPortReferenceIDs	List of internal ad-hoc port reference element IDs. Hyper edge support not provided in this API.
getGeneratorIsHidden	Value of hidden attribute on the generator
getInterfaceBusTypeVLNV	List of VLNV of the bus definition
getInterfaceMode	Mode of the interface: master, slave, system, mirroredMaster, mirroredSlave, mirroredSystem or monitor
getInterfacePortMapIDs	List of interface port map IDs
getLogicalPhysicalMapIDs	List of the logical and physical port map IDs
getPortMapRange	List of left and right range of the port map
getDescription	Return the description of the specified element
getVendorExtensions	Returns the complete XML text of the vendor extension element including the spirit:vendorExtension tag, as a well formed XML document

TGI API Function	Purpose
setVendorExtensions	Set vendor extensions. This call is only supported for elements within a spirit:design
getClockDriverName	Name of the clock driver
getClockDriverPeriod	Clock period of the given clock
getClockDriverPulseDuration	Clock period of the given clock
getClockDriverPulseOffset	Clock pulse offset of the given clock
getClockDriverPulseValue	Clock pulse value of the given clock
getClockDriverSource	Source name of the clock driver
getPortClockDriverID	Element ID of clock driver element, if present
getPortConstraintSetIDs	List of constraint sets IDs of the port
getPortSingleShotDriverID	Element ID of single shot driver element, if present
getPortSingleShotPulseDuration	Clock period of the port
getPortSingleShotPulseOffset	Clock pulse offset of the port
getPortSingleShotPulseValue	Clock pulse value of the port
getPortStyle	Returns 'wire' or 'transactional' to indicate the port style. (Implemented as per IPXACT 1.2 schema)
setClockDriverPeriod	Set period of the given clock port
setClockDriverPulseDuration	Set pulse duration of the given clock port
setClockDriverPulseOffset	Set pulse offset value of the given clock port
setClockDriverPulseValue	Set pulse value of the given clock port
setPortSingleShotPulseDuration	Set pulse duration of given single shot port
setPortSingleShotPulseOffset	Set pulse offset of given single shot port
setPortSingleShotPulseValue	Set pulse value of given single shot port

Specifying the TGI Generator File

Specify the directory containing your genesis.pl file using the -I command-line while invoking GenSys. Then, GenSys reads the genesis.pl file located in the specified directory and loads the TGI generators declared in this file.

You can also create multiple genesis.pl files located in different directories and specify them using the -I command-line while invoking GenSys. Then, GenSys reads the files as they are specified and loads the custom generators declared in these files.

In addition to the above, you can also load TGI generators by saving the genesis.pl file in the AddInGenerators folder in the current working directory of GenSys. You can also keep multiple genesis.pl files in sub folders of the AddInGenerators folder. All valid custom generators declared in these files are loaded when GenSys is launched.

Running the TGI Generators

All valid TGI generators are added to the Generator Menu of the GenSys Main Window.

To run a TGI generator, select the TGI generator from the Generator Menu.

Alternatively, you can also run the TGI generator by using the run command-line option. Here, you need to specify the TGI generator name along with the run command-line option. For example, the following command runs the TGI generator, gen:

```
run gen
```

2

Creating Custom Reports

Reports provide useful information about the state of the design. They are a mechanism for querying information about the design and help extensively in Debug and Analysis. This section describes how to create custom reports using the GenSys reporting infrastructure and APIs. You can create reports in Perl, Tcl, or template and run them using the GenSys GUI menu or TCL command prompt.

Creating custom report involves:

1. Registering the Custom Report
2. Defining the Report Body (Report Functionality)
3. Registering the Custom Report

To view an example of creating a custom report, refer to Example for Creating a Custom Report.

Registering the Custom Report

To register a custom report in GenSys, you first need to create it. For example, create a custom report called myreports.pl. Next, open genesis.pl and create an entry, as shown in the following snippet, to the custom report:

```
use genesisapis;
my $this = __FILE__;
$this =~ s/\/genesis.pl//;;
require "$this/myreports.pl";
1;
```

In myreports.pl, connect to the custom report environment:

```
package sampleReport;
use gensysReport;
use Genesis;
use genesisapis;
```

You can register reports by calls from Perl, PTT or Tcl, independent of what language the report is written in.

- Perl (See details below):

  ```
  gensysReport::RegisterReport($name,$rfile,$category,$codeType,$executa
  ble,$menu,$trigger,$graph,$product,$prompt,$ra_options)
  ```

- PTT

  ```
  register_report($name,$rfile,$category,$codeType,$executable,$menu,$tr
  igger,$graph,$product,$prompt,$ra_options)
  ```

- Tcl

  ```
  register_report <name> <rfile> <category> <codeType> <executable>
  <menu> <trigger> <graph> <product> <prompt> <ra_options>
  ```

Register the report (s) by using the RegisterReport Perl API command.

```
registerreport <report-name>, <report-file-name>, <report-category>,
<code-type>,  <executable>, <menu-entry>, <trigger>, <product>,
<options>, <popup-dialog>, <prompt>, <range>, <graph>
```

The argument details of this Perl API are as follows:

Argument	Specifies
<report-name>	The name of the report
<report-file-name>	The name of the output report file that contains the report contents. If you want the file name as per the report data, for example <designname>_report.txt, you can customize the report file name later within the report execution function using the **SetReportFileName** API.
<report-category>	The category of the report.
<code-type>	Whether the report body is written in Perl, Tcl, or template. The options are perl, tcl, or template.
<executable>	The name of the Perl function (Tcl or template file name that contains the report functionality) of the custom report.
<menu-entry>	Specifies how the menu entry should appear. For example, <item>::<sub_item>::<sub_sub_item> By default, the report name appears in the menu entry.
<trigger>	The Tcl command name that you want to associate the report with. For example, to run every time an object is saved. Leave this argument empty if you do not want to run the report automatically. Multiple Tcl commands can be given as space separated list of values. The report would be run on successful execution of the commands. An example of the usage of trigger is as follows: my $trigger = 'save';
<product>	The product name under which the report is to be activated. For example, a report registers under the registers product would be deactivated if the registers license is not available.

Argument	Specifies
<options>	Options that are shared between Perl, Menu, Popup Dialog, Trigger, Tcl, and PTT. Set an option in one and use it in any. The following option management switches are provided: 1) -help: Lists autogenerated usage (from options name and message) 2) -currentoptions: Lists current options (from options name and value) 3) -resetoptions: Resets value to InitValue 4) -noexec: Suppresses execution. For example, you can use it for setting options for subsequent triggers. 5) -triggerOn (list): For example, 'save' - will execute on each save (Tip: use save -top) 6) -triggerOff (list): For example, 'load gload' - will stop execution of this report on opens through the Tcl or GUI browser.
<popup-dialog>	Whether options can be set through a Popup Dialog.
<prompt>	Whether a dialog is defined in options. If not displayed, the options can only be set through Tcl: my $prompt = 1;

<range>	Whether options can be checked for range. Currently, if Boolean or integer. Set option value in the registration as follows:
	Type => 'Integer',
	Min => 0,
	Max => 15,

<graph>	For future use

Defining the Report Body (Report Functionality)

Reports can be written in Perl, PTT or Tcl:

- Perl: Refer to Example of Creating the Report Body using Perl.

- Ptt: Refer to the <install dir>/SPYGLASS_HOME/auxi/Genesis/features/Reports/ExampleReports/templates directory. The example report is registered in Perl by pttReportRegn1.pl in the same folder for clarity.
- Tcl: Refer to <install dir>/SPYGLASS_HOME/auxi/Genesis/features/Reports/ExampleReports/tcl directory. The example report is registered in Perl by tclReportRegn1.pl in the same folder for clarity.

Example of Creating the Report Body using Perl

Though this is a Perl example, it can be in another file. As long as it can be read by the genesis.pl file. Perform the following steps:

1. Create the report function and setup.

```
sub myPackaging {
   my $root = Genesis->new("root");
   my $p_design = $root->active("design");
   $p_design->elaborate;
   my $design_name = $p_design->name;
   my $design_vnlv = $p_design->vnlv->version;
```

2. Collect user input from dialogs, if required.

```
$design_name = CreateDesignBrowser($design_name,'Please select one
module to run report on');
```

3. Set report file, if needed. This overrides the report file name specified in the RegisterReport() API).

```
setReportFileName($rfile,"Packaging_$design_name");
```

4. Write report header.

```
WriteReportTitle ("\nPackaging\n top: $design_name; version:
$design_vnlv ");
```

5. Mine data from design. You can use any published GenSys APIs. Write data to the report file. Use report APIs, such as WriteReportBody, WriteReportTableHeader and WriteReportTableRow, to write report contents. For a list of reporting APIs, refer to Custom Report APIs.
6. In Perl, you do not need to close the file because it automatically closes when the subroutine completes. However, for Tcl and PTT, you have to close it.
7. After all reports registered (and defined, if Perl), return 1

```
}
     1;
```

Note:
You need to restart GenSys after each set of changes to a Perl report definition. For reports written in Tcl or PTT, GenSys restart is only required if the report registration changes.

Running the Custom Report

To run the custom report, you can use either the GUI menu, Tcl command prompt, or Perl.

- GUI

  ```
  Report-> <report name>
  ```

- Tcl

  ```
  run {Reports-><report name>}
  ```

- Perl

  ```
  <package>::<report_name>
  ```

Example for Creating a Custom Report

In the following example, you create a custom report that lists out design ports information.

```
my $name ='Report3A';
my $rfile = $name.'.txt';
my $menu = 'Report3A: perl regn, perl report, sub is
sampleReport::myReport3A, options: no trigger. Prints out options';
my $codeType = 'perl';
my $executable = \&sampleReport::myReport3A;
my $trigger = '';
my $prompt = 1;
my @options =(
    {
Name => 'Integer1A',
Description => 'Enter an integer',
Value => '',
InitValue => '2',
ViewType => 'String',
Message => 'Enter an integer 1A',
Type => 'Integer',
Min => 0,
Max => 15,
},
{
Name => 'Boolean2A',
Description => 'Enter a Boolean',
Value => '',
```

```
InitValue => '0',
ViewType => 'String',
Message => 'Enter a Boolean 2A',
Type => 'Boolean',
},
{
Name => 'String3A',
Description => 'string 3A',
Value => '',
InitValue => '<type here>',
ViewType => 'String',
Message => 'Enter a string 3A',
},
    );

gensysReport::RegisterReport
($name,$rfile,$category,$codeType,$executable,$menu,$trigger, $graph,
$product,$prompt,\@options);

sub myReport3A {
my
($name,$rfile,$category,$codeType,$executable,$menu,$trigger,$instanceBro
wser, $graph,$hierarchy,$detail,$product,$prompt, $ra_options,$rh_hash) =
@_;
    my @options = @$ra_options;
my %hash = %$rh_hash;
print " myReport3A called\n";
gensysReport::WriteReportTitle("Report 3A title: $name");
my $opt = Dumper (@options);
gensysReport::WriteReportBody ($opt, 120);
$opt =  "direct: ".$options[1]{'Name'}." \n";
gensysReport::WriteReportBody ($opt, 120);
gensysReport::WriteReportBody ('Report 3A body2', 120);
};
```

For more examples, add the following to the GenSys startup script:

-I $GENESIS_HOME/auxi/Genesis/features/Reports/exampleReports

3

Using GenSys Perl Objects

The GenSys Perl API Objects is a thin layer on top of a C-API, which is directly connected to the GenSys internal object model. This type of setup provides a direct mechanism for writing very efficient writer or generator applications. However, the setup is certainly not efficient as measured by ease-of-use or ease of understanding.

A new object-oriented API has been developed as a layer on top of the base API and aims to address the problem of ease-of-use and understanding. The Perl Objects API provides a ease-of-use layer and does not remove access to the base-level C-API.

Working with the GenSys Perl Objects

This section provides information on how the GenSys Perl Objects work. The basic structure is as follows:

1. Get the object-model root (comRoot). The object-model root provides a handle from which you can iterate down to the object-model.

   ```
   $root = Genesis->new("root");
   ```

2. From the object-model root, use the relation methods to get to other objects or use attribute methods to get values. To get to other objects, use the relation methods as shown below:

   ```
   @complist = @{$root->components};
   ```

   Use attribute methods to get values as shown below:

   ```
   $name = $comp->name;
   ```

Some relation methods return a single object, for example, the relation between a component instance and the master component of that instance. While, some relation methods return multiple objects, for example, the relation between a design and component instances in that design. A relation that can return multiple objects has a plural name, for example, designs, components, interfaces. A relation that can return only one object has a singular name, for example, master. All attribute methods are singular.

However, there is one exception to the above rule. Where meaningful, you can use a plural relation to get a single object by specifying the name of that object as an argument to the method. For example,

```
$inst = $des->instances(“RAM01”);
```

However, the above approach cannot be used on objects that have no name, for example, ad-hoc and interface connections.

Method application can be nested, where meaningful as shown below:

```
$first->terminal->port->name
```

Object-to-Methods Tree

The Objects-to-Methods tree is described below where the top item is an object type and its leaves are the methods applicable to the object type with the method return type annotated as L for list, S for string, O for Objects, and I for Boolean or Integer:

`root`

- |-> L designs
  |-> L components
  |-> L interfacedefs
  |-> I delcomp
  |-> O tableschema
  |-> O columnschema
  |-> O active

`design`

- |-> S name
  |-> S shortname
  |-> S description
  |-> S formatteddescription
  |-> L instances
  |-> L interfaces
  |-> L ports
  |-> L attributes
  |-> L tables
  |-> L aconnects
  |-> L tconnects
  |-> L iconnects
  |-> S xmlfile
  |-> O datasource
  |-> L parameters
  |-> L partitions
  |-> I rename
  |-> L requestors
  |-> O vnlv
  |-> elaborate
  |-> unelaborate

|-> L filesets

|-> L views

`channel`

- |-> O name

  |-> I maxmasters

  |-> I axslaves

  |-> L mirroredmasters

  |-> L mirroredslaves

  |-> L attributes

  |-> O datasource

`mirroredmaster`

- |-> O name

`mirroredslave`

- |-> O name

  |-> I range

  |-> L remapstates

`remapstate`

- |-> O name

  |-> S baseaddress

`bridgememorymap`

- |-> O name

  |-> L bridgeremapstates

`bridgeremapstate`

- |-> O name

  |-> L masteroffsetpairs

`masteroffsetpair`

- |-> O name

  |-> S baseaddress

  |-> I bitoffset

`bridge`

- |-> S slaveinterfacename
  |-> O bridgememorymap
  |-> L bridgepaths
  |-> L attributes
  |-> O datasource

`bridgepath`

- |-> S masterinterfacename
  |-> I range
  |-> S type
  |-> S baseaddress
  |-> I bitoffset
  |-> S width
  |-> S lau

`component`

- |-> S name
  |-> S type
  |-> S shortname
  |-> S description
  |-> S formatteddescription
  |-> L instances
  |-> L interfaces
  |-> L ports
  |-> L attributes
  |-> L tables
  |-> S xmlfile
  |-> O datasource
  |-> I rename
  |-> I isdesign
  |-> O vnlv

|-> L parameters

|-> S partition

|-> L requestors

|-> L registerobjects

|-> L channels

|-> L bridges

|-> L bridgememorymaps

|-> L filesets

|-> L views

|-> L signaldefns

|-> L signalassigns

|-> L bitfielddefns

|-> L bitenumdefns

|-> L regfiledefns

|-> L addrmapdefns

|-> L propertydefns

|-> L memorymaps

|-> L registers

|-> L banks

bank

- |-> S name

  |-> I arraysize

  |-> S offset

  |-> I arrayoffset

  |-> L bankelements

  |-> S shadowdepth

  |-> S mode

  |-> S writecontrol

  |-> S readcontrol

  |-> S head

|-> S tail

`bankelement`

- ||-> S name
  |-> S type
  |-> O register
  |-> I arraysize
  |-> S offset
  |-> S memory
  |-> S bank

`signaldefns`

- |-> S signal
  |-> I signal_width
  |-> S sync
  |-> I cpuif_reset
  |-> I field_reset
  |-> S active
  |-> S RDLname
  |-> S description
  |-> L propertyassigns

`signalassign`

- |-> S dest_port
  |-> S dest_reg
  |-> S dest_bf
  |-> S dest_pin
  |-> S dest_port
  |-> S source_port
  |-> S source_bf
  |-> S source_pin
  |-> S source_value

`bitfielddefn`

- |-> S field
  |-> S hw
  |-> S sw
  |-> S sticky
  |-> S resetsignal
  |-> S encode
  |-> S precedence
  |-> S RDLname
  |-> S description
  |-> S clocksignal
  |-> I dontcompare
  |-> I donttest
  |-> S index
  |-> S counterproperty
  |-> S hwaccessproperty
  |-> S interruptproperty
  |-> S swaccessproperty

`bitenumdefn`

- ||-> S enum_name
  |-> S enumtokendefns
  |-> S RDLname

`enumtokendefn`

- |-> S mnemonic
  |-> S value
  |-> S RDLname
  |-> S description

`regfiledefn`

- ||-> S regfile
  |-> S allocation_operator

|-> S alignment

|-> I sharedextbus

|-> S sharedextbus

|-> S RDLname

|-> S description

|-> L propertyassigns

|-> I dontcompare

|-> I donttest

`addrmapdefn`

- |-> S map

  |-> S RDLname

  |-> S alignment

  |-> I sharedextbus

  |-> S endianess

  |-> S addressing

  |-> I rsvdset

  |-> I rsvdsetX

  |-> S orientation

  |-> S description

  |-> L propertyassigns

  |-> I dontcompare

  |-> I donttest

`propertydefn`

- |-> S property

  |-> S comp

  |-> S type

  |-> I default

  |-> S default_default

  |-> S RDLname

  |-> S description

`propertyassign`

- |-> S property
  |-> S value

`memorymap`

- |-> S name
  |-> S map
  |-> S alignment
  |-> I sharedextbus
  |-> S endianess
  |-> S addressing
  |-> I rsvdset
  |-> I rsvdsetX
  |-> S orientation
  |-> S RDLname
  |-> S description
  |-> I dontcompare
  |-> I donttest

`interfacedef`

- |-> S name
  |-> S shortname
  |-> S description
  |-> S formatteddescription
  |-> L lports
  |-> L attributes
  |-> L tables
  |-> S xmlfile
  |-> O datasource
  |-> I rename
  |-> O vnlv

`interface`

- |-> S name
  |-> S shortname
  |-> S description
  |-> S formatteddescription
  |-> L lports
  |-> L interfacedef
  |-> L attributes
  |-> S type
  |-> I ismirror
  |-> O datasource
  |-> S powerdomain
  |-> S voltage

`parmeter`

- |-> S name
  |-> S value
  |-> L children
  |-> S type
  |-> S description
  |-> I const
  |-> I rtl

`instance`

- |-> S name
  |-> S shortname
  |-> S description
  |-> S formatteddescription
  |-> O master
  |-> L ifinstances
  |-> L aconnects
  |-> L tconnects

|-> L iconnects

|-> L attributes

|-> L terminals

|-> L tables

|-> O datasource

|-> S powerdomain

|-> S voltage

|-> L parameters

|->   elaborate

|->   unelaborate

|-> S partition

`addressif`

- |-> S start

  |-> S end

  |-> S size

  |-> S description

  |-> S target_inst

  |-> O datasource

  |-> L attributes

  |-> S target_inst

  |-> S target_element

  |-> S target_element_type

  |-> L memory_path

  |-> S memory_path_name

`terminal`

- |-> S name

  |-> O port

`port`

- |-> S name

|-> S description

|-> S formatteddescription

|-> S lname

|-> S type

|-> S dir

|-> S default

|-> S lsb

|-> S msb

|-> S width

|-> I is_scalar

|-> L attributes

|-> O datasource

|-> S powerdomain

|-> S voltage

|-> S shortname

|-> S align

|-> S clockrate

|-> S is_open

|-> O controlclock

|-> O controlreset

|-> O hdl_type

|-> O hdl_type_lsb

|-> O hdl_type_msb

|-> O hdl_type_language

|-> O hdl_type_library

|-> O hdl_type_package

|-> S interfaceGroup

`lport`

- |-> S name

  |-> S shortname

|-> S description

|-> S formatteddescription

|-> S lname

|-> S type

|-> S dir

|-> S default

|-> S lsb

|-> S msb

|-> S width

|-> I registered

|-> L attributes

|-> I optional

|-> O datasource

|-> S actdir

`lconnect`

- |-> S shortname

  |-> S description

  |-> S formatteddescription

  |-> O isopen

  |-> O istemp

  |-> S backref

  |-> S command

  |-> S morebackref

  |-> S deltadelay

  |-> L attributes

  |-> O datasource

`aconnect`

- |-> O first

  |-> O second

  |-> S align

|-> S plane

|-> S backref

|-> S command

|-> I is_elab_generated

|-> S morebackref

|-> S deltadelay

|-> L attributes

|-> O datasource

|-> S shortname

|-> S description

|-> S formatteddescription

`tconnect`

- |-> S shortname

  |-> S description

  |-> S formatteddescription

  |-> O first

  |-> O value

  |-> O istemp

  |-> S backref

  |-> S command

  |-> I is_elab_generated

  |-> S morebackref

  |-> S deltadelay

`iconnect`

- |-> S shortname

  |-> S description

  |-> S formatteddescription

  |-> O first

  |-> O second

  |-> S align

|-> S plane

|-> S backref

|-> S command

|-> S morebackref

|-> S deltadelay

|-> L attributes

|-> O datasource

`aconnect_1`

- |-> S lsb

  |-> S msb

  |-> O instance

  |-> O terminals

  |-> O parent

  |-> O port

`aconnect_2`

- |-> S lsb

  |-> S msb

  |-> O instance

  |-> O terminals

  |-> O parent

  |-> O port

`iconnect_1`

- |-> O instance

  |-> O ifinstance

  |-> O parent

  |-> L splices

  |-> L tieoffs

`iconnect_2`

- |-> O instance

|-> O ifinstance

|-> O parent

|-> L splices

|-> L tieoffs

`ifinstance`

- |-> O interface

  |-> O instance

  |-> L terminals

  |-> L iconnects

  |-> I is_exported

  |-> S is_open

`register`

- |-> I arraysize

  |-> S access_type

  |-> S name

  |-> S shortname

  |-> S description

  |-> S formatteddescription

  |-> S width

  |-> S offset

  |-> I volatile

  |-> L attributes

  |-> L bitfields

  |-> O datasource

  |-> S reset

  |-> S resetmask

  |-> I reserved

  |-> I regwidth

  |-> I accesswidth

  |-> O order

|-> I external

|-> O errextbus

|-> I shared

|-> O defnname

|-> O RDLname

|-> L propertyassigns

|-> I dontcompare

|-> I donttest

`registergroup`

- |-> I arraysize

  |-> I arrayoffset

  |-> S name

  |-> L registers

  |-> L attributes

  |-> O datasource

  |-> I offset

  |-> S description

  |-> S shortname

  |-> S formatteddescription

`registerobject`

- |-> S name

  |-> S shortname

  |-> S description

  |-> S formatteddescription

  |-> S type

  |-> O register

  |-> O registergroup

  |-> L attributes

  |-> O datasource

`bitfield`

- |-> S name
  |-> S description
  |-> S shortname
  |-> S formatteddescription
  |-> S functiontype
  |-> S lsb
  |-> S msb
  |-> O reset
  |-> S access_type
  |-> S actualreset
  |-> I volatile
  |-> L attributes
  |-> O datasource
  |-> L bitenums
  |-> S rtype
  |-> S wtype
  |-> S vtype
  |-> I reserved
  |-> S resetmask
  |-> S sw
  |-> S hw
  |-> S RDLname
  |-> S precedence
  |-> S encode
  |-> S field
  |-> S sticky
  |-> S resetsignal
  |-> L propertyassigns
  |-> S clocksignal
  |-> I dontcompare

|-> I donttest

|-> S counterproperty

|-> S hwaccessproperty

|-> S interruptproperty

|-> S swaccessproperty

`bitenum`

- |-> S access_type

  |-> L attributes

  |-> O datasource

  |-> S name

  |-> I value

  |-> S description

  |-> S formatteddescription

  |-> I maxvalue

  |-> I minvalue

  |-> S RDLname

  |-> S shortname

`memoru`

- |-> I arraysize

  |-> S bankaligntype

  |-> S name

  |-> S description

  |-> S formatteddescription

  |-> S shortname

  |-> I volatile

  |-> S access_type

  |-> S offset

  |-> S endian

  |-> L attributes

  |-> O datasource

|-> I maxdatawidth

|-> I size

`attribute`

- |-> S name

  |-> S type

  |-> S value

  |-> S help

  |-> S category

`splice`

- |-> I lsb

  |-> I msb

  |-> S portname

`partition`

- |-> S name

  |-> S path

  |-> S language

  |-> L rtlfiles

`rtlfiles`

- |-> S name

`table`

- |-> **S** name

  |-> I tid

  |-> S ptablecolname

  |-> I ptablerownum

  |-> I istab

  |-> I nrows

  |-> I ncols

  |-> L columndata

  |-> L rowdata

  |-> L colnames

|-> O ptable

|-> O cell

|-> S evaluate

|-> I validate

|-> O subtable

|-> setrowreadonly

|-> refresh

|-> insertrowbefore

|-> insertrowafter

|-> updaterow

|-> insertatend

|-> setdata

|-> setreadonly

|-> deleterow

|-> deleteallrows

|-> O tableschema

`cell`

- |-> L enumchoices

  |-> O ptable

  |-> S type

  |-> S data

  |-> I intdata

  |-> I istable

  |-> O gettable

  |-> S ptablecolname

  |-> I ptablerownum

  |-> S expr

`datasource`

- |-> S person

  |-> S method

|-> S reason

|-> S datetime

|-> S type

|-> I frozen

`vnlv`

- |-> S version

  |-> S name

  |-> S library

  |-> S vendor

`tableschema`

- |-> L columnschemas

  |-> L hiddencolumns

  |-> S name

  |-> S hiername

  |-> I isevalcolpresent

  |-> I isevalallowed

  |-> I isvalidcolpresent

  |-> I isvalidateallowed

  |-> S evalrowfunction

  |-> S key

  |-> I ishidden

  |-> I isreadonly

  |-> I isbasetable

  |-> I isextensiontable

  |-> I isflattable

  |-> I istab

  |-> S rowtype

  |-> I line

  |-> S file

  |-> S help

|-> S version

|-> S columnschema

|-> S numrows

|-> I isautofill

|-> S personalityperlfunction

`columnschema`

- |-> I addvalidperlfunction

  |-> L dependcolumns

  |-> L keycolumns

  |-> L cellenums

  |-> O tableschema

  |-> O autofillschema

  |-> S name

  |-> S hiername

  |-> S help

  |-> I ishidden

  |-> I isreadonly

  |-> I isextension

  |-> I iseval

  |-> I iskey

  |-> I isvalidate

  |-> S celltype

  |-> S perlfunction

  |-> S changeperlfunction

  |-> S validateperlfunction

  |-> S file

  |-> I line

  |-> S personalityperlfunction

  |-> I setvalidperlfunctionstatus

`autofillschema`

- |-> S perlfunction

  |-> S default

`rtlcomponent`

- |-> **O** comcomponent

  |-> **L** rtlinstances

  |-> **L** rtlports

  |-> **S** name

`rtlinstance`

- |-> **O** masterrtlcomponent

  |-> **O** parentrtlcomponent

  |-> **S** name

`rtlport`

- |-> **S** name

`fileset`

- |-> **L** files

  |-> **L** groups

  |-> **L** attributes

  |-> **O** datasource

  |-> S description

  |-> **S** shortname

  |-> **S** formatteddescription

  |-> **S** dependency

`filesetref`

- |-> **L** files

  |-> **L** groups

  |-> **L** attributes

  |-> **O** datasource

  |-> S description

  |-> **S** shortname

|-> **S** formatteddescription

|-> **S** dependency

`view`

- |-> **O** design

  |-> **L** filesetrefs

  |-> **L** attributes

  |-> **O** datasource

GenSys Perl Methods

GenSys Perl Methods are used to access the objects in the GenSys database.

access_type

Returns the access type for the memory.

Applies to

memory, bitenum, register

Arguments

None

Returns

The memory access type as one of the following:

READ	WRITE
READ_WRITE	READ_CLEAR
EXECUTE	STICKY
RESERVED	READ_WRITE_EXEC
NOT_ACCESSIBLE	PROGRAM_MEMORY
READ_EXECUTE	WRITE_CLEAR

READ_WRITE_CLEAR	READ_WRITE_SET

The access type for a register can be one of the following:

READ	WRITE	READ_WRITE	OTHER

accesswidth

Returns the access width of the SystemRDL register object

Applies to

register

Arguments

None

Returns

Access width of the SystemRDL register object.

actdir

Returns the actual logical port direction

Applies to

lport

Arguments

None

Description

Returns the actual logical port direction. If the port direction is not set then this method returns undef.

Returns

Actual logical port direction.

---

active

Returns pointer to the active design/component/register/bitfield/instance/interfacedef/typetree/interface/bitenum/port/connection/ table/subtable/partitiontable/partitiontablehierarchy/activefile/activelang/rtlcomponent type object.

Applies to

root

Arguments

Type name of the object. Type name can have any of the values such as design, component, register, bitfield, instance, interfacedef, typetree, interface, bitenum, port, connection, table, subtable, partitiontable, partitiontablehierarchy, activefile, activelang, or rtlcomponent.

Returns

Active object of the given type

---

actualreset

Returns the actual reset value for bitfield

Applies to

bitfield

Arguments

None

Returns

Reset value for the bitfield.

aconnects

Returns the list of Adhoc connections under the component instance

Applies to

instance

Arguments

None

Returns

List of Adhoc connections under the component instance.

addressing

Returns the addressing information of a SystemRDL object

Applies to

memorymap, addrmapdefn

Arguments

None

Returns

Addressing information (compact, regalign, or fullalign) of a SystemRDL object.

addvalidperlfunction

Adds a validate perl function for a column schema.

Applies to

columnschema

Arguments

Name of validate perl function, status

Returns

1 if the validate perl function is added; otherwise returns 0.

---

align

Returns the align value of the connection, port.

Applies to

aconnect, iconnect, port

Arguments

None

Returns

Align value of the connection and the port.

---

alignment

Returns the alignment of a SystemRDL object

Applies to

memorymap, regfiledefn, addrmapdefn

Arguments

None

Returns

Alignment of a SystemRDL object

allocation_operator

Returns the allocation operator information of a regfile object of a component

Applies to

regfiledefn

Arguments

None

Returns

at, incr, or next_at

arrayoffset

Returns the array offset value for the Register Group.

Applies to

registergroup

Arguments

None

Returns

The array offset value for the Register Group. The array offset refers the space between two elements of a register group array.

arraysize

Returns the array size for the register/register group.

Applies to

memory, registergroup

Arguments

None

Returns

Array size of register/register group.

---

attributes

Returns the list of attributes or the named attribute under the object.

Applies to

design, component, interfacedef, interface, instance, interface, port, aconnect, iconnect, lconnect, bitenum, alias, addressif, channel, bridge, lport

Arguments

None or the attribute name

Returns

List of attributes for the object or the named attribute of the object.

---

autofillschema

Returns the Autofill Schema associated with a Column Schema.

Applies to

columnschema

Arguments

None

---

backref

Returns the backref string for the connection.

Applies to

aconnect, lconnect, iconnect, tconnect

Arguments

None

Returns

Backref string for the connection

---

bankaligntype

Returns the alignment of the memory bank.

Applies to

memory

Arguments

None

Returns

Alignment, such as PARALLEL, SERIAL, or UNDEF, of memory bank.

---

baseaddress

Returns the base address of the master interface associated with a master/base address pair and base address associated with a bridge path.

Applies to

remapstate and bridgepath

Arguments

None

Returns

Base address of the master interface associated with a master/base address pair and base address associated with a bridge path.

---

bitenums

Returns list of enum values or the named bitenum of bitfield.

Applies to

bitfields

Arguments

None or bitenum name.

Returns

List of bitenum values or the named bitenum under the bitfield.

---

bitfields

Returns the list of BitFields for the register.

Applies to

register

Arguments

None

Returns

List of BitFields for the specified register.

---

bitoffset

Returns the bitoffset of the master interface associated with a master/base address pair and bit offset associated with a bridge path.

Applies to

masteroffsetpair and bridgepath

Arguments

None

Returns

Bitoffset of the master interface associated with a master/base address pair and bit offset associated with a bridge path.

---

bridgememorymap

Returns the bridge memory-map pointer corresponding to a bus bridge.

Applies to

bridge.

Arguments

None

Returns

Bridge memory-map pointer corresponding to a bus bridge

---

bridgememorymaps

Returns the list of memory maps of bus bridges for a component

Applies to

component

Arguments

None

Returns

List of memory maps of bus bridges for a component.

---

bridges

Returns the list of bus bridges for a component

Applies to

component

Arguments

None

Returns

List of of bus bridges for a component.

---

bridgeremapstates

Returns the list of bridge remap states associated with a bridge memory-map.

Applies to

bridgememorymap

Arguments

None

Returns

List of bridge remap states associated with a bridge memory-map

---

bridgepaths

Returns the list of bridge paths corresponding to a bus bridge.

Applies to

bridge.

Arguments

None

Returns

List of bridge paths corresponding to a bus bridge.

---

category

Returns the attribute category.

Applies to

attribute

Arguments

None

Returns

Attribute category.

---

cell

Returns cell at given column and row number.

Applies to

table

Arguments

Column name and row number

Returns

Returns cell at given column and row number.

---

cellenums

Returns a list of enum values for column schema when the column type is ENUM_TYPE.

Applies to

columnschema

Arguments

None

---

celltype

Returns cell type of column schema.

Applies to

columnschema

Arguments

None

Returns

Cell type of column schema (specified as INT_TYPE, BOOL_TYPE, STRING_TYPE, ENUM_TYPE, or TABLE_TYPE.)

---

changeperlfunction

Returns the change Perl function associated with the column schema.

Applies to

columnschema

Arguments

None

---

channels

Returns the list of bus channels for the component

Applies to

component

Arguments

None

Returns

List of bus channels for the component.

---

children

Returns the list of child TypeTrees for the parameter.

Applies to

parameter

Arguments

None

Returns

List of child TypeTrees of the parameter.

---

clockrate

Returns the clockrate field of the port.

Applies to

port

Arguments

None

Return type

Returns the clockrate field data of the port.

---

clocksignal

Returns the value of a clock signal for the bitfield and bitfielddefn object.

Applies to

bitfield, bitfielddefn

Arguments

None

Return type

Returns the clock signal value of the bitfield and bitfielddefn object.

---

colnames

Returns list of column names in a table.

Applies to

table

Arguments

None

Return type

Returns list of column names in a table.

---

columndata

Returns list of cells in the given column name.

Applies to

table

Arguments

Column name

Return type

Returns list of cells in the given column name.

---

columnschema

Returns the column schema for the table schema.

Applies to

root, tableschema

Arguments

(For root) full hierarchical name of the column;

(For tableschema) column name and the table pointer

Returns

(For root) Returns the columnschema pointer, if found; Otherwise, returns NULL. (Here, the column schema is searched by the full hierarchical column name)

(For tableschema) Returns the columnschema pointer; Otherwise, returns NULL.

columnschemas

Returns a list of column schemas for the table schema.

Applies to

tableschema

Arguments

None

comcomponent

Returns the COM component/design pointer for the RTL component

Applies to

rtlcomponent

Arguments

None

Returns

Returns the COM component/design pointer for the RTL component. In case of partitioning, if RTL component does not have any corresponding COM component, its parent COM component is returned.

command

Returns the Tcl command executed to create the connection

Applies to

aconnect, lconnect, iconnect, tconnect

Arguments

None

Returns

The Tcl command, which was executed to create this connection

---

comp

Returns component information of a PropertyDefn object

Applies to

propertydefn

Arguments

None

Returns

field, reg, regfile, addrmap, or all

---

components

Returns the list of components or the named component under the root

Applies to

root

Arguments

None or component name

Returns

List of components or the named component under the root.

---

const

Returns the const field for a parameter.

Applies to

parameter

Arguments

None

---

controlclock

Returns the control clock port pointer of the port.

Applies to

port

Arguments

None

---

controlreset

Returns the control reset port pointer of the port.

Applies to

port

Arguments

None

---

counterproperty

Returns the value of the argument specified.

Applies to

bitfield, bitfielddefn

Arguments

Specify only one of the following arguments at a time.

counter, threshold, saturate, incrthreshold, incrsaturate, overflow, underflow, incrvalue, incr, incrwidth, decrvalue, decr, decrwidth, decrsaturate, decrthreshold

Returns

Returns the value of the argument.

---

cpuif_reset

Returns 1 or 0 for the SystemRDL signal object.

Applies to

signaldefn

Arguments

0 or 1

---

data

Returns string data in the cell.

Applies to

cell

Arguments

None

Returns

Returns string data in the cell.

---

datasource

Returns the data source of the object.

Applies to

design, component, interfacedef, interface, instance, interface, port, register, bitfield, memory, bitenum, addressif, channel, bridge, lport, aconnect, iconnect, lconnect, bitenum

Arguments

None

Returns

Returns the data source of the object.

---

datetime

Returns the change date and time for the data source.

Applies to

datasource

Arguments

None

Returns

Returns the change date and time for the data source.

---

default

Returns the default value of the port/logical port/interface port, autofill, or PropertyDefn schema.

Applies to

port, lport, autofillschema, propertydefn

Arguments

None

Returns

Default value of the port/logical port/interface port/PropertyDefn.

---

default_default

Returns the default value of the PropertyDefn object.

Applies to

propertydefn

Arguments

None

Returns

Default value of the PropertyDefn object.

---

defnname

Returns the defnname value of the SystemRDL register object

Applies to

register

Arguments

None

Returns

Defnname value of the SystemRDL register object.

---

delcomp

Deletes the component from the root.

Description

Deletes the component from the root based on the VNV information. It also deletes the instances of the component along with the connections associated with the instances.

Applies to

root

Arguments

Name, vendor, and version of the component.

Returns

1 if the component is deleted; otherwise returns 0.

---

deleteallrows

Delete all the rows in the table.

Applies to

table

Arguments

None

Returns

Nothing

---

deleterow

Deletes the specified row.

Applies to

table

Arguments

Row number *rownumber*

Returns

Nothing

---

deltadelay

Returns the deltadelay value for a connection.

Applies to

aconnect, lconnect, iconnect, tconnect

Arguments

None

Returns

String value that is the deltadelay value for a connection

---

dependcolumns

Returns a list of dependent columns names on the column schema.

Applies to

columnschema

Arguments

None

---

description

Returns the description of the specified object.

Applies to

design, component, interfacedef, interface, instance, interface, port, bienum, signaldefn, bitfielddefn, enumtokendefn, regfiledefn, addrmapdefn, propertydefn, memorymap, parameter

Arguments

None

Returns

Description for the specified object.

---

designs

Returns the list of designs or the named design under the root.

Applies to

root

Arguments

None or the design name

Returns

List of designs or the named design under the root.

---

dest_bf

Returns the signal destination field.

Applies to

signalassign

Arguments

None

Returns

Returns the signal destination field.

---

dest_pin

Returns the signal destination pin.

Applies to

signalassign

Arguments

None

Returns

Returns the signal destination pin.

---

dest_port

Returns the signal destination port.

Applies to

signalassign

Arguments

None

Returns

Returns the signal destination port.

---

dest_reg

Returns the signal destination register.

Applies to

signalassign

Arguments

None

Returns

Returns the signal destination register.

dir

Returns the direction of the port/logical port/interface port.

Applies to

port, lport

Arguments

None

Returns

One of the following values:

IN
OUT
INOUT

elaborate

Elaborates the instance parameters or design connections.

Applies to

instance, design

Arguments

None

Description

Elaborates the design connections for a design or elaborates the instance parameters for an instance. After elaboration, instance parameter values would be available.

Returns

None

---

encode

Returns the encode value of a bitifeld or bitfielddefn

Applies to

bitfields, bitfielddefn

Arguments

None

Returns

Encode value of a bitfield or bitfielddefn

---

end

Returns the end address of the Address Interface.

Applies to

addressif

Arguments

None

Returns

End Address of the Address Interface.

---

endian

Returns the endian type for the memory.

Applies to

memory

Arguments

None

Returns

BIG for Big Endian type memory or SMALL for Little Endian type memory.

---

endianess

Returns the endianess information of a SystemRDL object

Applies to

memorymap, addrmapdefn

Arguments

None

Returns

Endianess information (bigendian or littleendian) of a systemRDL object.

---

enumchoices

Returns list of items in the given cell of type ENUM.

Applies to

cell

Arguments

None

Returns

Returns list of items in the given cell of type ENUM.

---

enum_name

Returns enum name

Applies to

bitenumdefn

Arguments

None

Returns

Enum name of the bitenumdefn object

---

enumtokendefns

Returns enum tokens

Applies to

bitenumdefn

Arguments

None

Returns

Enum tokens of the bitenumdefn object

---

errextbus

Returns the errextbus value of the SystemRDL register object

Applies to

register

Arguments

None

Returns

errextbus value of the SystemRDL register object.

---

evalrowfunction

Returns the eval row function associated with table schema.

Applies to

tableschema

Arguments

None

---

evaluate

Returns a new table pointer.

Applies to

table

Arguments

None

Description

Evaluates a dynamic table *table* and returns a new table if the table *table* could be evaluated. Otherwise, returns the original table *table*.

---

expr

returns the live-formula expression string of the cell if the cell contains live-formula, else returns ""

Applies to

cell

Arguments

None

Returns

Live-formula expression string in the cell

---

external

Returns the external value of the SystemRDL register object

Applies to

register

Arguments

None

Returns

External value (0 or 1) of the SystemRDL register object.

field

Returns the field value of a bitifeld or bitfielddefn

Applies to

bitfields, bitfielddefn

Arguments

None

Returns

Field value of a bitfield or bitfielddefn

field_reset

Returns 1 or 0 for the SystemRDL signal object

Applies to

bitfields

Arguments

None

Returns

1 or 0

file

Returns the filename in which the object is defined.

Applies to

tableschema, columnschema

Arguments

None

---

first

Returns the first end of the connection.

Applies to

aconnect, iconnect, tconnect

Arguments

None

Returns

First end (aconnect_1, iconnect_1, or tconnect_1) of the connection.

Notes

The second method returns the second end of the connection.

---

formatteddescription

Returns the formated description string or HTML format string

Applies to

design, component, interfacedef, interface, designconfiguration, fileset, filesetref, view, filename, instance, port, lport, aconnect, iconnect, lconnect, tconnect, registerobject, register, bitfield, bitenum, memory, registergroup

Arguments

None

Returns

Formated description string or HTML format string

For example, consider the following text in a description:

Bold string

In this case, the string returned by the description method and the formatteddescription method is as follows:

Method	String Returned
description	"Bold string"
formatteddescription	"@b@Bold @/b@string"

frozen

Returns the Data Frozen value for data source.

Applies to

datasource

Arguments

None

Returns

Data frozen value for data source.

functiontype

Returns the function type of the bitfield.

Applies to

bitfield

Arguments

None

Returns

One of the following values:

ADDRESS	COMMAND	CONSTANT	CONTROL	COUNTER
DATA	MEM	RESERVED	STATUS	

gettable

Returns pointer to the subtable or NULL.

Applies to

cell

Arguments

None

Returns

Pointer to the subtable if the cell is a subtable. Otherwise, returns NULL.

groups

Returns the list of groups for the fileset

Applies to

fileset, filesetref

Arguments

None

Returns

Returns the list of groups for the fileset

hdl_type

Returns the HDL type of the port.

Applies to

port

Arguments

None

Returns

HDL type of the port.

hdl_type_lsb

Returns the LSB of the HDL type.

Applies to

port

Arguments

None

Returns

LSB of the HDL type.

hdl_type_msb

Returns the MSB of the HDL type.

Applies to

port

Arguments

None

Returns

MSB of the HDL type.

hdl_type_language

Returns the language of the HDL type.

Applies to

port

Arguments

None

Returns

Language of the HDL type.

hdl_type_library

Returns the library of the HDL type.

Applies to

port

Arguments

None

Returns

Library of the HDL type.

hdl_type_package

Returns the package of the HDL type.

Applies to

port

Arguments

None

Returns

Package of the HDL type.

help

Returns the help message for the object.

Applies to

attribute, tableschema, columnschema

Arguments

None

Returns

Help message associated with the object.

hiddencolumns

Returns a list of hidden columns for table schema.

Applies to

tableschema

Arguments

None

---

hiername

Returns the hierarchical name of the object.

Applies to

tableschema, columnschema

Arguments

None

---

hw

Returns hw value of a bitifeld or bitfielddefn

Applies to

bitfields, bitfielddefn

Arguments

None

Returns

Hw value of a bitfield or bitfielddefn

---

hwaccessproperty

Returns the value of the argument specified

Applies to

bitfield, bitfielddefn

Arguments

Specify only one of the following arguments at a time.

we, wel, anded, ored, xored, fieldwidth, hwclr, hwset, hwenable, hwmask

Returns

Returns the value of the argument specified.

---

iconnects

Returns the list of interface connections for the component instance.

Applies to

instance

Arguments

None

Returns

List of interface connections for the component instance.

---

ifinstance

Returns the interface instance for the interface connection.

Applies to

iconnect_1, iconnect_2

Arguments

None

Returns

Interface Instance of the interface connection.

---

ifinstances

Returns the list of interface instances under the component instance.

Applies to

instance

Arguments

None

Returns

List of interface instances for the component instance.

---

interface

Returns the interface master for the interface instance.

Applies to

ifinstance

Arguments

None

Returns

Interface master of the interface instance.

---

interfaceGroup

Returns interface group for the port.

Applies to

port

Arguments

None

Returns

Interface group for the port.

---

interruptproperty

Returns the value of the argument specified

Applies to

bitfield, bitfielddefn

Arguments

Specify only one of the following arguments at a time.

intr, enable, mask, haltenable, haltmask

Returns

Returns the value of the argument specified.

---

insertatend

Inserts a row in the end and updates that row with the given values.

Applies to

table

Arguments

Comma-separated list of column name-value pairs *colname*=>*value*

Returns

Nothing

---

insertrowafter

Inserts a blank row after the specified row.

Applies to

table

Arguments

Row number *rownumber*

Returns

Nothing

---

insertrowbefore

Inserts a blank row before the specified row.

Applies to

table

Arguments

Row number *rownumber*

Returns

Nothing

---

instance

Returns an instance

Applies to

aconnect_1, aconnect_2, iconnect_1

Arguments

None

Returns

An instance.

---

instances

Returns the list of instances or the named instance under the design/component.

Applies to

design, component

Arguments

None or the instance name

Returns

List of instances for the design/component or the named instance of the design/component.

---

intdata

Returns int data in the cell.

Applies to

cell

Arguments

None

Returns

Returns int data in the cell.

---

interfacedef

Returns the parent interface definition for the interface.

Applies to

interface

Arguments

None

Returns

Parent interface definition for the specified interface.

---

interfacedefs

Returns the list of interface definitions or the named interface definition under the root.

Applies to

root

Arguments

None or the interface definition name

Returns

List of interface definitions under the root or the named interface definition under the root.

---

interfaces

Returns the list of interfaces or the named interface under the design/component.

Applies to

design, component

Arguments

None or the interface name

Returns

List of interfaces for the design/component or the named interface of the design/component.

interruptproperty

Returns the value of interrupt

Applies to

bitfield, bitfielddefn

Arguments

Interrupt that you want to access from property table.

Returns

Value of the interrupt.

isautofill

Checks whether the table schema contains Autofill.

Applies to

tableschema

Arguments

integer value

Returns

1 if the table schema contains Autofill. Otherwise, returns 0.

---

isbasetable

Checks whether the table schema is a base table.

Applies to

tableschema

Arguments

None

Returns

1 if the table schema is a base table. Otherwise, returns 0

---

isdesign

Checks whether the component is a design.

Applies to

component

Arguments

None

Returns

1 if the component is a design. Otherwise, returns 0.

---

is_elab_generated

Checks whether the adhoc connection is generated during elaboration.

Applies to

aconnect, tconnect

Arguments

None

Returns

1 if the adhoc connection is generated during elaboration. Otherwise, returns 0.

---

iseval

Checks whether the column schema is evaluatable.

Applies to

columnschema

Arguments

None

Returns

1 if the column schema is specified as an evaluatable column. Otherwise, returns 0.

---

isevalallowed

Recursively checks if the table schema has any evaluatable columns.

Applies to

tableschema

Arguments

None

Returns

1 if evaluatable columns are present in the table schema hierarchy. Otherwise, returns 0.

---

isevalcolpresent

Checks whether the table schema has evaluatable columns.

Applies to

tableschema

Arguments

None

Returns

1 if the table schema has evaluatable columns. Otherwise, returns 0.

---

isextension

Checks whether the column schema is an extension.

Applies to

columnschema

Arguments

None

Returns

1 if the column schema is an extension. Otherwise, returns 0.

---

isextensiontable

Checks if the table schema is an extension table.

Applies to

tableschema

Arguments

None

Returns

1 if the table schema is an extension table. Otherwise, returns 0.

isflattable

Checks whether the table schema is a flat table.

Applies to

tableschema

Arguments

None

Returns

1 if the table schema is a flat table. Otherwise, returns 0.

is_exported

Checks whether the interface instance is the instance of an exported interface.

Applies to

ifinstance

Arguments

None

Returns

1 if the interface instance is an instance of an exported instance. Otherwise returns 0.

---

ishidden

Checks if the object is hidden.

Applies to

tableschema, columnschema

Arguments

None

Returns

1 if the table schema is hidden. Otherwise, returns 0.

---

iskey

Checks whether the column schema is key column.

Applies to

columnschema

Arguments

None

Returns

1 if the column schema is a key column. Otherwise, returns 0.

---

ismirror

Checks whether the Interface is a mirror image of the Interface definition.

Applies to

interface

Arguments

None

Returns

Returns a string (YES/NO/MONITOR) according to the mirror type of the interface.

---

isMultipleInstantiations

Checks if there are multiple instances of a design/component

Applies to

design, component

Arguments

design or component object

Returns

0 in case of single instantiation of a design/component

1 in case of multiple instantiations of a design/component

---

isopen

Checks whether the logical connection is an open connection.

Applies to

lconnect

Arguments

None

Returns

Returns 1 if the logical connection is an open connection. Otherwise, returns 0.

---

is_open

Returns the open field of the port.

Applies to

port, ifinstance

Arguments

None

Returns

Returns the open field data of the port.

---

isreadonly

Checks if the object is read-only.

Applies to

tableschema, columnschema

Arguments

None

Returns

Returns 1 if the table schema is read-only. Otherwise, returns 0.

---

is_scalar

Checks whether the port is a scalar port.

Applies to

port

Arguments

None

Returns

1 if the port is a scalar port. Otherwise returns 0.

---

istab

Checks whether the table schema is a tab.

Applies to

tableschema, table

Arguments

None

Returns

Returns 1 if the table schema has been specified as tab. Otherwise, returns 0.

---

istable

Checks whether the cell is a subtable of another table.

Applies to

cell

Arguments

None

Returns

Returns 1 if the cell is a subtable of another table. Otherwise, returns 0.

---

istemp

Checks whether the logical or tieoff connection is a temporary connection.

Applies to

lconnect, tconnect

Arguments

None

Returns

Returns 1 if the logical or tieoff connection is a temporary connection. Otherwise, returns 0.

---

isvalidate

Checks whether Validate Perl function is specified for column schema.

Applies to

columnschema

Arguments

None

Returns

1 if validate Perl function has been specified for column schema. Otherwise, returns 0.

---

isvalidcolpresent

Checks whether table schema has validation columns.

Applies to

tableschema

Arguments

None

Returns

1 if the table schema has validation columns. Otherwise, returns 0.

---

isvalidateallowed

Recursively checks whether validation columns are present in the table schema.

Applies to

tableschema

Arguments

None

Returns

1 if validation columns are present in the table schema. Otherwise, returns 0.

---

key

Returns the name of the key column for the table schema.

Applies to

tableschema

Arguments

None

keycolumns

Returns the key columns for dynamic columns associated with the column schema.

Applies to

columnschema

Arguments

None

lau

Returns the LAU associated with a bridge path.

Applies to

bridgepath.

Arguments

None

Returns

LAU associated with a bridge path.

library

Returns the library name of the versioned item.

Applies to

vnlv

Arguments

None

Returns

Library name of the versioned item.

---

lname

Returns the logical name for the port/logical port/interface port.

Applies to

port, lport, bitenum

Arguments

None

Returns

Logical name of the port/logical port/interface port.

---

line

Returns the line number at which the object is defined.

Applies to

tableschema, columnschema

Arguments

None

---

lports

Returns the list of logical ports of the interface definition or the interface

Applies to

interfacedef, interface

Arguments

None

Returns

List of logical ports of the interface definition or the interface.

---

lsb

Returns the LSB value of the port/connection.

Applies to

port, lport, aconnect_1, aconnect_2, splice

Arguments

None

Returns

LSB value of the port or connection.

---

map

Returns the map name of a SystemRDL object

Applies to

memorymap, addrmapdefn

Arguments

None

Returns

Map name of a SystemRDL object

master

Returns the master component for the component instance.

Applies to

instance

Arguments

None

Returns

Master component of the component instance.

masterinterfacename

Returns the list of bridge paths corresponding to a bus bridge.

Applies to

bridgepath

Arguments

None

Returns

List of bridge paths corresponding to a bus bridge.

masteroffsetpairs

Returns the list of master interface/base address pair associated with each bridge remap state.

Applies to

bridgeremapstate

Arguments

None

Returns

List of master interface/base address pair associated with each bridge remap state.

---

masterrtlcomponent

Returns master RTL component for the RTL instance.

Applies to

rtlinstance

Arguments

None

Returns

Master RTL component for the RTL instance.

---

maxdatawidth

Returns the maximum data width of memory.

Applies to

memory

Arguments

None

Returns

Maximum data width of memory.

maxmasters

Returns the maximum number of masters of a bus channel.

Applies to

channel

Arguments

None

Returns

Maximum number of masters of a bus channel

axslaves

Returns the maximum number of slaves of a bus channel.

Applies to

channel

Arguments

None

Returns

Maximum number of slaves of a bus channel

maxvalue

Returns maximum value of valuerange.

Applies to

bitenum

Arguments

None

Returns

Maximum value of valuerange.

---

method

Returns the change method for data source.

Applies to

datasource

Arguments

None

Returns

Change method for data source.

Maximum value of valuerange.

---

memory

Returns the MemoryBlock object.

Applies to

address

Arguments

None

Returns

MemoryBlock object.

memorymap

Returns the memory map of a component.

Applies to

address

Arguments

None

Returns

Memory map of a component.

memory_path

Returns memory path to the target node.

Applies to

addressif

Arguments

None

Returns

Memory path to the target node.

memory_path_name

Returns memory path name to the target node.

Applies to

addressif

Arguments

None

Returns

Memory path name to the target node.

---

minvalue

Returns minimum value of valuerange.

Applies to

bitenum

Arguments

None

Returns

Minimum value of valuerange.

---

mirroredmasters

Returns the list of mirrored master interfaces for the bus channel.

Applies to

channel

Arguments

None

Returns

List of mirrored master interfaces for the bus channel

mirroredslaves

Returns the list of mirrored slaves interfaces for the bus channel.

Applies to

bitenum

Arguments

None

Returns

List of mirrored slaves interfaces for the bus channel

mnemonic

Returns the mnemonic of an enum token

Applies to

enumtokendefn

Arguments

None

Returns

Mnemonic of an enum token

morebackref

Returns more backref information.

Applies to

aconnect, lconnect, iconnect, tconnect

Arguments

None

Returns

More backref information, if any, which is added during elaboration phase

msb

Returns the MSB value of the port/connection.

Applies to

port, lport, aconnect_1, aconnect_2, splice

Arguments

None

Returns

MSB value of the port or connection.

name

Returns the object name.

Applies to

design, component, interfacedef, interface, address, parameter, instance, terminal, port, lport, interface, register, memory, attribute, table, rtlfiles, vnlv, tableschema, columnschema, partition, rtlcomponent, rtlinstance, rtlport, channel, mirroredmaster, bridgeremapstate, remapstate, bridgememorymap, masteroffsetpair, mirroredslave, memorymap

Arguments

None

Returns

Name of the object.

ncols

Returns number of columns in the table.

Applies to

table

Arguments

None

Returns

Number of columns in the table.

nrows

Returns number of rows in the table.

Applies to

table

Arguments

None

Returns

Number of rows in the table.

numrows

Returns the number of rows of the table.

Description

Returns the number of rows of the table after evaluation. If the table schema contains an EVAL ROW function, the output of the function is evaluated and number of rows of the table is returned.

If the table schema does not contain an EVAL ROW function, -1 is returned.

Applies to

tableschema

Arguments

table pointer (for which the number of rows is to be evaluated)

---

offset

Returns the offset value for the register/register memory.

Applies to

register, memory

Arguments

None

Returns

Offset value of the register/register memory.

---

optional

Returns the Optional field data for the port, logical port.

Applies to

lport, port

Arguments

None

Returns

Optional field data.

order

Returns the order of the SystemRDL register object

Applies to

register

Arguments

None

Returns

Order (lsb0 or msb0) of the SystemRDL register object.

orientation

Returns the orientation information of a SystemRDL object

Applies to

memorymap, addrmapdefn

Arguments

None

Returns

orientation information (lsb0 or msb0) of a SystemRDL object.

parameters

Returns list of parameters or the named parameter.

Applies to

design, component, instance

Arguments

None or parameter name

Returns

List of all parameters or the named parameter under design/component/instance.

---

parent

Returns the parent interface for the interface connection.

Applies to

iconnect_1, iconnect_2

Arguments

None

Returns

Parent instance of the interface connection.

---

parentrtlcomponent

Returns parent RTL component for the RTL instance.

Applies to

rtlinstance

Arguments

None

Returns

Parent RTL component for the RTL instance.

perlfunction

Returns the Perl function associated with the object.

Applies to

columnschema, autofillschema

Arguments

None

person

Returns the change person name for the data source

Applies to

datasource

Arguments

None

Returns

Change person name for data source

personalityperlfunction

Gives the personality Perl function from the table/column schema object.

Applies to

tableschema, columnschema

Arguments

None

Returns

Returns the personality Perl function of the table/column schema object.

---

partition

Returns partition name for the component/component instance.

Applies to

component, instance

Arguments

None

Returns

Partition name for component/component instance.

---

partitions

Returns list of partition objects.

Applies to

design

Arguments

None

Returns

List of partition objects.

---

path

Returns the path of generated RTL files.

Applies to

partition

Arguments

None

Returns

Path of RTL files generated for the given partition object.

---

plane

Returns the plane name for the connection.

Applies to

iconnect, aconnect

Arguments

None

Returns

Plane name for the connection.

---

port

Returns the port for the terminal.

Applies to

terminal

Arguments

None

Returns

Port for the specified terminal.

---

portname

Returns the port name of the splice port.

Applies to

splice

Arguments

None

Returns

Port name of the splice port.

---

ports

Returns the list of ports or the named port under the design/component.

Applies to

design, component

Arguments

None or the port name.

Returns

List of ports for the object or the named port of the object.

---

powerdomain

Returns the power domain of the port, interface, or instance.

Applies to

port, interface, instance

Arguments

None

Returns

Power domain of the port.

---

precedence

Returns the precedence value of a bitifeld or bitfielddefn

Applies to

bitfields, bitfielddefn

Arguments

None

Returns

Precedence value, hw or sw, of a bitfield or bitfielddefn

---

property

Returns the property name of a SystemRDL object

Applies to

propertydefn, propertyassign

Arguments

None

Returns

Property name of a SystemRDL object

---

ptable

Returns parent table pointer.

Applies to

table, cell

Arguments

None

Returns

Pointer to the parent table.

---

ptablecolname

Returns the parent column name for the table cell or sub-table.

Applies to

table, cell

Arguments

None

Returns

Parent column name for the table cell or sub-table.

---

ptablerownum

Returns the parent row number for the table cell or sub-table.

Applies to

table, cell

Arguments

None

Returns

Parent row number for the table cell or sub-table.

---

range

Returns the range associated with a bridge path.

Applies to

bridgepath

Arguments

None

RDLname

Returns the RDL name of a SystemRDL object

Applies to

register, bitfield, bitenum, memorymap, signaldefn, bitfielddefn, enumtokendefn, regfiledefn, addrmapdefn, propertydefn, memorymap, bitenumdefn

Arguments

None

Returns

RDL name of a SystemRDL object.

reason

Returns the change reason for data source.

Applies to

datasource

Arguments

None

Returns

Change reason for the data source.

refresh

Refresh the table

Applies to

table

Arguments

None

Returns

Nothing

regfile

Returns the regfile object of a component

Applies to

regfiledefn

Arguments

None

Returns

Regfile object of a component

---

register

Returns the register for the alias.

Applies to

reserved

Arguments

None

Returns

Physical register for the alias.

---

registered

Checks whether the logical port is a registered logical port.

Applies to

lport

Arguments

None

Returns

1 if the logical port is a registered logical port. Otherwise returns 0.

registergroup

Returns pointer to Register Group or NULL.

Applies to

registerobject

Arguments

None

Returns

Pointer to Register Group, if Register Object is of type Register Group. Otherwise, returns *NULL*.

registerobjects

Returns list of registers/register groups.

Applies to

component

Arguments

None

Returns

List of registers/register groups.

regwidth

Returns the width of the SystemRDL register object.

Applies to

register

Arguments

None

Returns

Width of the SystemRDL register object.

---

registers

Returns the list of registers for the register block or the register group.

Applies to

group

Arguments

None

Returns

List of registers in the register block or the register group.

---

remapstates

Returns the list of remapstates associated with a mirrored slave.

Applies to

mirroredslave.

Arguments

None

Returns

List of remapstates associated with a mirrored slave

rename

Renames the object.

Applies to

design, component, interfacedef

Arguments

New name of the design, component, interfacedef

Returns

0 on success and 1 on failure.

reserved

Checks whether the bitfield is reserved or not

Applies to

bitfield

Arguments

None

Returns

Returns 1 if the bitfield is reserved. Otherwise returns 0.

requestors

Returns list of requestor names.

Applies to

component, design

Arguments

None

Returns

List of requestor interface names or instance names of requestor interface.

---

reset

Returns the reset value of the bitfield.

Applies to

bitfield

Arguments

None

Returns

Reset value of the bitfield.

---

resetmask

Returns the reset mask for the register/register group.

Applies to

bitfield

Arguments

None

Returns

Reset mask for the register/register group.

resetsignal

Returns the reset signal of a bitfield or bitfielddefn.

Applies to

bitfield, bitfielddefn

Arguments

None

Returns

Reset signal of the bitfield or bitfielddefn.

rowdata

Returns list of cell in the given row number.

Applies to

table

Arguments

Row number

Returns

Returns list of cell in the given row number.

rowtype

Returns type of rows for the table schema

Applies to

tableschema

Arguments

None

Returns

Returns type of rows for the table schema (specified as SINGLE or MULTIPLE.) Returns Eval where the schema specifies ROW_TYPE :: perl function() for the table.

---

rsvdset

Returns the rsvdset information of a SystemRDL object

Applies to

memorymap, addrmapdefn

Arguments

None

Returns

Rsvdset information (1 or 0) of a systemRDL object.

---

rsvdsetX

Returns the rsvdsetX information of a SystemRDL object

Applies to

memorymap, addrmapdefn

Arguments

None

Returns

RsvdsetX information (1 or 0) of a systemRDL object.

rtl

Returns the RTL field for a parameter.

Applies to

parameter

Arguments

None

rtlfiles

Returns the list of RTL files for the partition.

Applies to

partition

Arguments

None

Returns

List of RTL files for the partition object.

rtlinstances

Returns the list of RTL instances for the RTL component.

Applies to

rtlcomponent

Arguments

None

Returns

List of RTL instances for the RTL component.

---

rtlports

Returns list of RTL ports for the RTL component.

Applies to

rtlcomponent

Arguments

None

Returns

List of RTL ports for the RTL component

---

rtype

Returns the RType of the bitfield.

Applies to

bitfield

Arguments

None

Returns

RType of the bitfield.

---

language

Returns the language of the generated RTL files.

Applies to

partition

Arguments

None

Returns

Language of the RTL files generated for each unique partition set, such as Verilog, VHDL, SV, or SystemC.

second

Returns the second end of the connection.

Applies to

aconnect, iconnect

Arguments

None

Returns

Second end (aconnect_2 or iconnect_2) of the connection.

Notes

The first method returns the first end of the connection.

setdata

Updates the table cell with the given data.

Applies to

table

Arguments

Column name *colname*, Row number *rownumber*, Row data *rowdata*

Returns

Nothing

---

setreadonly

Sets the table as read-only.

Applies to

table

Arguments

Read-only flag *readonlyflag*

Description

Sets the table and all its subtable as read-only, when the *readonlyflag* is true. When the *readonlyflag* is false, the table and all its subtables will be write-enabled in the GUI.

Returns

Nothing

---

setrowreadonly

Sets a row of a table as read-only.

Applies to

table

Arguments

Row number *rownumber*, Read-only flag *readonlyflag*

Description

Sets a row *rownumber* of the table in GUI as read-only if *readonlyflag* is set to true. If the *readonlyflag* is set to false, the row is set as write-enabled.

All the subtables of the table corresponding to the row *rownumber* are also made read-only or write-enabled depending on the value of *readonlyflag*.

No row of the table other than *rownumber* is affected.

Returns

Nothing

---

setvalidperlfunctionstatus

Sets the run status of the validate perl function of a column schema.

Applies to

columnschema

Arguments

Name of validate perl function, status

Returns

1 if the status of validate perl function is reset; otherwise returns 0.

---

shared

Returns the shared value of the SystemRDL register object

Applies to

register

Arguments

None

Returns

Shared value (0 or 1) of the SystemRDL register object.

---

sharedextbus

Returns the sharedextbus information of a SystemRDL object

Applies to

memorymap, regfiledefn, addrmapdefn

Arguments

None

Returns

Sharedextbus information (0 or 1).

---

shortname

Returns the short name of the register block or register or port.

Applies to

register, port, design, component, interfacedef, interface, designconfiguration, fileset, filesetref, view, filename, instance, port, lport, aconnect, iconnect, lconnect, tconnect, registerobject, register, bitfield, bitenum, memory, registergroup

Arguments

None

Returns

Short name of the register block or the register or the port.

---

signal

Returns the SystemRDL signal name.

Applies to

signaldefn

Arguments

None

Returns

SystemRDL signal name

---

signal_width

Returns the width of SystemRDL signal.

Applies to

signaldefn

Arguments

None

Returns

Width of the SystemRDL signal

---

size

Returns the size of memory.

Applies to

memory

Arguments

None

Returns

Size of the memory.

---

slaveinterfacename

Returns the slave interface name associated with a bus bridge.

Applies to

bridge.

Arguments

None

Returns

Slave interface name associated with a bus bridge.

---

source_bf

Returns the signal source field.

Applies to

signalassign

Arguments

None

Returns

Returns the signal source field.

---

source_pin

Returns the signal source pin.

Applies to

signalassign

Arguments

None

Returns

Returns the signal source pin.

---

source_port

Returns the signal source port.

Applies to

signalassign

Arguments

None

Returns

Returns the signal source port.

---

source_reg

Returns the signal source register.

Applies to

signalassign

Arguments

None

Returns

Returns the signal source register.

---

source_value

Returns the signal source value.

Applies to

signalassign

Arguments

None

Returns

Returns the signal source value.

---

splices

Returns a list of splice ports.

Applies to

iconnect_1, iconnect_2

Arguments

None

Returns

List of splice ports of iconnect_1 and iconnect_2.

---

start

Returns the start address of the Address Interface.

Applies to

addressif

Arguments

None

Returns

Start Address.

---

sticky

Returns sticky information for a bitfield or bitfielddefn.

Applies to

bitfield, bitfielddefn

Arguments

None

Returns

Sticky information

---

subtable

Returns subtable pointer

Applies to

table

Arguments

Column name *colname*, Row number *rownumber*

Returns

Subtable pointer

sw

Returns sw value of a bitifeld or bitfielddefn

Applies to

bitfields, bitfielddefn

Arguments

None

Returns

Sw value of a bitfieldmor bitfielddefn

swaccessproperty

Returns the value of the argument specified

Applies to

bitfield, bitfielddefn

Arguments

Specify only one of the following arguments at a time.

rclr, rset, woset, woclr, swwe, swwel, swmod, swacc, singlepulse

Returns

Returns the value of the argument specified.

sync

Returns the value sync, async, or unconstrained for a SystemRDL signal.

Applies to

signaldefn

Arguments

None

Returns

sync, async, or unconstrained

target_element

Returns target element name of the Address Interface.

Applies to

addressif

Arguments

None

Returns

Target element name of the address interface object.

target_element_type

Returns target element type of the Address Interface.

Applies to

addressif

Arguments

None

Returns

Target element type of the address interface object.

---

target_inst

Returns target instance name of the Address Interface.

Applies to

addressif

Arguments

None

Returns

Target instance name.

Start Address.

---

tables

Returns the list of tables under the root or specified object.

Applies to

design, component, interfacedef, instance

Arguments

None

Returns

List of tables under the root.

tableschema

Returns table schema for the root, column schema, and table schema.

Applies to

root, columnschema, table

Arguments

None (for columnschema and table schema), full hierarchical tablename (for root)

Returns

(For root) Returns tableschema pointer; otherwise returns NULL.

(For columnschema) Returns the corresponding table schema if the columnschema TYPE is specified as TABLE_TYPE; otherwise, returns NULL.

(For table) Returns tableschema pointer.

tconnects

Returns the list of adhoc to tieoff connections on a design/instance

Applies to

design, instance

Arguments

None

Returns

List of adhoc to tieoff connections on a design or instance.

terminals

Returns the list of terminals for the component/interface instance.

Applies to

instance, ifinstance

Arguments

None

Returns

List of terminals of the component instance or interface instance.

---

tid

Returns the table ID.

Applies to

table

Arguments

None

Returns

ID of the table

---

tieoffs

Returns a list of splice tieoff values.

Applies to

iconnect_1, iconnect_2

Arguments

None

Returns

List of splice tieoff values of iconnect_1 and iconnect_2.

type

Returns the object type.

Applies to

component, port, lport, interface, attribute, datasource, parameter, cell, register, propertydefn

Arguments

None

Returns

The object type as one of the following:

Object	Return Type values
component	CPU, BUSLOGIC, MEMORY, PERIPHERAL, SUBSYSTEM, OTHERS
port, lport	CLK, RST, EVT, DATA, ADDR, CONTROL, TIEOFF, IO_INPUT, IO_OUTPUT, IO_PAD, IO_SELECT, IO_OEN, IO_PULLEN, FUNCTION_IN, TEST_IN, SCANIN, SCANOUT, DFT, CORE
interface	MASTER, SLAVE
attribute	ATTR_STRING, ATTR_INT, ATTR_BOOL, ATTR_ENUM, ATTR_HEX
datasource	STRING
parameter	INT, STRING, BOOL, ENUM, ARRAY, RECORD
cell	TABLE, INT, BOOL, STRING, ENUM, PERL
register	REGISTER, GROUP

Object	Return Type values
propertydefn	string, number, boolean, ref

unelaborate

Unelaborates the instance parameters or design connections.

Applies to

instance, design

Arguments

None

Description

Unelaborates the design connections for a design or unelaborates the instance parameters for an instance. After unelaboration, master parameter values would be available.

Returns

None

updaterow

Updates a row with the given values.

Applies to

table

Arguments

Row number *rownumber*, Comma-separated list of column name-value pairs *colname=>value*

Returns

Nothing

validate

Returns the number of invalid cells.

Applies to

table

Description

Validates that table and returns the number of invalid cells.

Arguments

None

validateperlfunction

Returns the validation Perl function associated with the column schema.

Applies to

columnschema

Arguments

None

value

Returns the value of the object.

Applies to

parameter, attribute, bitenum, tconnect, enumtokendefn, propertyassign

Arguments

None

Returns

ValueTree of the parameter or the value of the attribute or bitenum, or the tieoff value for the adhoc to tieoff connection or value of an enum token.

---

vendor

Returns the vendor name of the versioned item.

Applies to

vnlv

Arguments

None

Returns

Vendor name of the versioned item.

---

version

Returns the version.

Applies to

vnlv, tableschema

Arguments

None

Returns

Version of the versioned item.

---

vnlv

Returns the versioned name of the object.

Applies to

component, design, interfacedef

Arguments

None

Returns

Versioned name of the object.

---

vtype

Returns the VType of the bitfield.

Applies to

bitfield

Arguments

None

Returns

VType of the bitfield.

---

volatile

Checks whether the register/memory is volatile.

Applies to

register, memory

Arguments

None

Returns

1 if the register/memory is volatile. Otherwise returns 0.

---

voltage

Returns the voltage of the port, interface, or instance.

Applies to

Port, interface, instance

Arguments

None

Returns

Returns the voltage of port, interface, or instance.

---

width

Returns the width of the port/logical port/register/blocksize.

Applies to

port, lport, register

Arguments

None

Returns

Width of the port/logical port/register.

---

wtype

Returns the WType of the bitfield.

Applies to

bitfield

Arguments

None

Returns

WType of the bitfield.

---

xmlfile

Returns the XML file for the object.

Applies to

design, component, interfacedef

Arguments

None

Returns

Name of the XML file.

4

Performing Perl Validation and Evaluation

This section covers the following topics:

- Performing Perl Validation
- Performing Perl Evaluation

Performing Perl Validation

Perl validation is performed to validate data for any particular column in a table. The first step in performing validation is to define the validation schema, which is described in the next section.

Defining Validation Schema

To perform validation for any column, define the validation schema for that column in the schema file for the table. Specify a Perl function to validate the value in a column using the VALIDATE keyword. In addition, you can also specify arguments to be used in the Perl function using the VALIDATE_ARG (optional) keyword. Then, define the Perl function/ subroutine in a Perl file.

In addition to the arguments defined in the schema using the VALIDATE_ARG keyword, four more arguments are passed to the Perl function, by default. These four arguments are:

- Table pointer
- Column name
- Row number
- Information string

where, information string contains the value VALIDATE.

Consider an example. If you need to create a validation check for the default column of a port, such that the default value for a port is not less than 0 and is equal to *(MSB-LSB)+1*; include the VALIDATE and VALIDATE_ARG keywords for the default column in the schema of the Ports table, as given below:

```
VALIDATE :: defaultValueCheckVALIDATE_ARG  :: LSBVALIDATE_ARG  :: MSB
```

The defaultvalueCheck function will be called with three arguments. The first argument will be value of the current column; in this case it is default. The second argument will be value of column LSB and the third argument will be value of column MSB. Also, the first four default arguments will be ignored in this example.

Define the Perl routine defaultValueCheck for the validation check in the Perl file, as given below:

```
sub defaultValueCheck{  shift; shift; shift;shift;#ignore table/
columnName/row/                           #infoStr default args  my
@rvalues = 0;  my $default = GetString($_[0]);  my $lsb =
GetString($_[1]);  my $msb =  GetString($_[2]);  my $size = 1;   #prune
spaces from $default  $default =~ s/ //g;  if($lsb ne "" && $msb ne
"")  {     #size = 2**n-1 , where n = $msb-$lsb+1;     my $w = $msb -
$lsb + 1;     $size = (2 ** $w) - 1;  }  if( $default < 0
)  {     $rvalues[0]  = 0;     $rvalues[1] = "[ERROR]: Default value must
NOT     be negative!\n";  }  elsif( $default eq "")  {     $rvalues[0]  =
1;   #will not flag incase no     value  }  elsif(($msb  eq "" || $lsb eq
"") && ($default ne  "0" && $default ne "1"))  {     $rvalues[0] = 0;
#if no msb/lsb then default      value should be 0/1     $rvalues[1] =
"[ERROR]: Default value must be      0 or 1!\n";  }  elsif($default >
$size)  {     $rvalues[0]  = 0;   #default value must be      equal to
2**n-1  , n == width     $rvalues[1] = "[ERROR]: Default value must be
     equal to the width of the port!";  }  else  {     $rvalues[0]  =
1;  } return @rvalues;
```

Note:
The shift keyword is used here to ignore the first four default arguments — table, column name, row number, and information string.

The VALIDATE Perl subroutine returns an array of values. The first array element rvalues[0] returns the status 0 or 1 and the second array element $rvalues[1] returns the message that will appear if the data entered in the field in invalid.

The first element of the returned array contains value 1 if the data is valid and contains value 0 if the data is invalid and corresponding error message is stored in the second element.

Now, restart the GenSys application to make the validation check effective. When the validation check is performed, the cells that do not match the criteria are marked with red color and an error message appears when you move the mouse pointer over the field.

Performing Perl Evaluation

Perl evaluation enables you to easily find and work with data in a column or a defined range. When you define an evaluation schema for a column, a combo box appears in the cells of that column displaying the associated list of choices. These choices are displayed based on the values matching the criteria specified in the schema.

The evaluation schema is defined using EVAL_ARG and EVAL_FUNC functions.

By default, four arguments are passed to EVAL_FUNC, in addition to the arguments defined in the schema. These four arguments are:

- Table pointer
- Column name
- Row number

- Information string

The EVAL_FUNC function needs to return an array of string values that can be displayed in the combo-box of the field.

Specifying Simple Criteria

To specify a simple criteria, you can use the EVAL_ARG function in the schema.

For example, if you need that the control_clock column in the Interfaces table of a component should display all the ports in the component, you can create an evaluation schema specifying this criteria for the control_clock column.

For this, include the EVAL_ARG function for control_clock in the Interfaces table schema, as given below:

```
EVAL_ARG :: Ports.Name
```

If you want that the combo box should display all the port names that have the direction as IN, you can specify the EVAL_ARG function as given below:

```
EVAL_ARG :: “Port[direction=IN].Name”
```

Note that you need to include the quotation marks if more than one criteria is specified.

Another example is given below, where all the port names with direction IN and the MSB equal to 4 should be displayed in the drop-down list.

```
EVAL_ARG :: “Port[direction=IN][MSB=4].Name”
```

Specifying Complex Criteria

If you need to specify a complex criteria using || or && operators, you need to use the Perl sub-routine and the EVAL_FUNC function.

For example, if you want to display YES and NO values in the volatile column of Ports table, specify the EVAL_FUNC function for volatile column in the interface schema, as given below:

```
EVAL_FUNC :: getYesNoValues
```

Create a Perl sub-routine for the evaluation function as given below:

```
sub getYesNoValues{  shift; shift; shift;shift;   my @rlist =
0;  $rlist[0] = "YES";  $rlist[1] = "NO";  return @rlist;}
```

Note that the shift keyword is used to ignore the table, column name, row, infoStr default arguments.

Also, the return statement should return an array of string values.

Include the Perl sub-routine in the .pl file and specify it with the -I option while running GenSys. Other chapters of this book describe how to specify custom generators.

5

Registering and Evaluating Tcl Commands

GenSys enables you to call Perl sub-routines as Tcl commands, so that the Perl APIs can be made available from Tcl. The approach to do this is to register the Perl sub-routines as Tcl commands.

Registering Tcl Commands

Inside any Perl script, use the Tcl interpreter Perl object, $genesis_tcl_obj to create, evaluate, or delete the Tcl commands created from the Perl APIs. The Perl script should contain the use Tcl statement for using the Tcl.pm package. The Tcl.pm package enables registration of Perl APIs as Tcl commands.

A sample Perl script is given below.

```
use Tcl;use genesisapis;sub myfunc{    my($clientdata, $interp, $cmdname,
@args) = @_;    foreach(@args){        printg "You passed
$_\n";    }}$main::genesis_tcl_obj->CreateCommand("gen_cmd",\&myfunc);$ma
in::genesis_tcl_obj->Eval("gen_cmd done something");1;
```

Tcl Command Methods

CreateCommand

Usage

```
CreateCommand (CMDNAME, CMDPROC, CLIENTDATA)
```

Description

The CreateCommand method binds a new procedure named <*CMDNAME*> into the interpreter.

The <*CMDPROC*> argument is the Perl function reference called to register the new command <*CMDNAME*>. <*CMDPROC*> can be a subname, a subreference, or an anonymous sub.

The <*CLIENTDATA*> is an optional argument. It can be any Perl scalar, for example, a reference to some other data.

Examples

Some of the examples of creating Tcl command using the CreateCommand method are as follows:

1. Use the following code to create a command mycmd which will execute some function and will not return any value:

```
sub ex1 {   my ($clientdata,$interp,$command,@args) = @_;   ...Use first and second
arguments of Tcl
command   ...   MyRoutine($args[0],$args[1]);}$main::genesis_tcl_obj->CreateCommand("m
ycmd","utils::ex1");
```

1. Use the following code to create a command mycmd which will do something and return a single value (in Tcl):

```
sub ex1 {    my ($clientdata,$interp,$command,@args) = @_;    ...do something    ...
return $value;}$main::genesis_tcl_obj->CreateCommand("mycmd","utils::ex1");
```

2. Use the following code to create a command mycmd which will do something and return a list of values (in Tcl):

```
sub ex1 {    my ($clientdata,$interp,$command,@args) = @_;    ...do something    ...
return [$value1, $value2,
$value3];}$main::genesis_tcl_obj->CreateCommand("mycmd","utils::ex1");
```

DeleteCommand

Usage

`DeleteCommand (CMDNAME)`

Description

The DeleteCommand method deletes the registered Tcl command <*CMDNAME*> from the interpreter.

Eval

Usage

`Eval (CMDNAME+CMDARGS)`

Description

The Eval method evaluates the registered command <*CMDNAME*> with <*CMDARGS*>. The <*CMDARGS*> is passed to <*CMDPROC*> in the <*LIST*>, as specified in Invoking the Tcl Command section.

Examples

You can evaluate a Tcl command by using the Eval method from inside Perl function as given below:

```
$main::genesis_tcl_obj->Eval("add_instance -name PLL0 -master PLL");
```

You can evaluate a Tcl file from inside Perl function as given below:

```
$main::genesis_tcl_obj->Eval("run pll.tcl");
```

Use the specification as shown below:

```
$main::genesis_tcl_obj->Eval("run pll.tcl");
```

and do not use (EvalFile) specifications as shown below:

```
$main::genesis_tcl_obj->EvalFile("pll.tcl");
```

Invoking the Tcl Command

When <*CMDNAME*> is invoked in the Tcl Command window, the arguments passed to the Perl function <*CMDPROC*> are (CLIENTDATA, INTERP, LIST) where <*INTERP*> is a Perl object for the Tcl interpreter which is called; <*LIST*> is a Perl list of the arguments with which <*CMDNAME*> was called. In Tcl, the first element of the list is <*CMDNAME*> itself.

Example of Registering a Generator in Tcl

The following example shows how to register a report generator using Tcl:

```
RegisterGenerator("genClks", "genClks");sub genClks {    my $root =
Genesis->new(“root”);    #...}my %args = {‘-option1’ => ‘default1’, ‘-option2’ =>
‘default2’};$genesis_tcl_obj->CreateCommand("genClks", \&tclGenClks, \%args);# at this
point we have a tcl command genClks -option1 foo -option2# When executed, this command
would call tclGenClks with, among other things \%args as an argument
sub tclGenClks {    my ($args, $interp, $cmdname, @userargs) = @_;    # do some
processing, like arg checking.       genClks();  ## call with appropriate args..}
```

6

Using GenSys API Functions

This chapter describes the GenSys Perl API functions that can be used to create custom generator functions. See Creating Custom Generators Using Perl APIs for description of creating the custom generators using the GenSys Perl API functions.

Using the GenSys Perl API Functions

The GetcomRoot API function does not take any argument.

Most of the GenSys Perl API functions take only one argument, that is, the object on which the API function is being called.

Some of the GenSys Perl API functions take the targeted object followed by one or more other arguments as a comma-separated list.

The following example shows the usage of API functions that take different number of arguments:

```
sub myGenerator{  my $com_root = GetcomRoot();    my $designList =
comRootGetDesignList($com_root);  while(iter_getdata($designList))    $design =
iter_getdata($designList);    $compInst = DesignGetCompInstByName($design,
“INST1”);...
```

Note:

Some of the Perl API functions return pointers to the currently loaded design, interface, and component objects. You can use the closeall Tcl command to close these objects. All the internal pointers earlier returned by GenSys Perl APIs/OO methods or Tcl APIs become inaccessible after using the closeall command. Hence, such (stored) pointers should not be used after a call to this command.

Return Values of GenSys API Functions

The GenSys API functions have the following types of return values:

Returned Type	**Returned Value Type**
Single Object	Pointer to the returned object
List of objects	Pointer to the list of pointers to the objects matching the function condition
True or False	1 if the function condition is true. Otherwise returns 0
Number	Integer
Names	string

Design Hierarchy

You can access the GenSys design objects under the following hierarchies:

- Design Hierarchy Under the Design Object
- Design Hierarchy Under the Component Object
- Design Hierarchy Under the Interface Definition Object
- Design Hierarchy Under the BaseClass Object

Design Hierarchy Under the Design Object

The design hierarchy under the Design object is as follows:

```
comRoot
|-> Design
    |-> Connection
    |-> Component Instance
    |   |-> Elaborated View
    |   |-> Unelaborated View
    |   |-> Parameter
    |   |   |-> TypeTree
    |   |   |   |-> TypeInfo
    |   |   |   |-> ParamType
    |   |   |-> ValueTree
    |   |       |-> ValueTree
    |   |-> Interface Instance
    |   |   |-> Connection
    |   |   |-> Terminal
    |   |-> Address Interface
    |   |-> Connection
    |   |   |-> ConnectionTypeItem
    |   |       |-> Adhoc Connection
    |   |       |-> CommonConnection
    |   |       |-> Interface Connection
    |   |       |-> Logical Connection
    |   |       |-> TieOff Connection
    |   |-> Terminal
    |-> Interface
    |   |-> Logical Port
    |   |-> PortMap
    |->Port
```

Design Hierarchy Under the Component Object

The design hierarchy under the Component object is as follows:

```
comRoot|-> Component
    |-> Parameter
    |   |-> TypeTree
    |   |   |-> TypeInfo
    |   |   |-> ParamType
    |   |-> ValueTree
    |       |-> ValueTree
    |-> Port
    |-> Interface
    |   |-> Logical Port
    |   |-> PortMap
    |-> Address Interface
        |-> Memory Block
        |   |-> CommonConnection
        |   |-> Splice
        |       |   |-> Interface Connection
        |       |   |-> Register Data
        |       |-> Register Data
        |           |-> BitField
        |               |-> BitEnumValue
```

Design Hierarchy Under the Interface Definition Object

The design hierarchy under the Interface Definition object is as follows:

```
comRoot
|-> Interface Definition
    |-> Logical Port
```

Design Hierarchy Under the BaseClass Object

The design hierarchy under the BaseClass object is as follows:

```
BaseClass
|-> Attribute
|-> DataSourceClass
```

|-> VersionedNameClass

Iterator Processing API Functions

GenSys provides the following API functions to process the list of objects (*itr*) returned by other API functions:

FreeIter

Declaration

```
void FreeIter(void* itr);
```

Description

Releases the memory allocated for the iterator itr containing list of objects returned from C to Perl. User needs to call this function after traversing the list returned from C.

Returns

Nothing

FreeStrIter

Declaration

```
void FreeStrIter(void* itr);
```

Description

Releases the memory allocated for the iterator itr containing list of string objects returned from C to Perl. User needs to call this function after traversing the list returned from C.

Returns

Nothing

iter_getdata

Declaration

```
void* iter_getdata(void* itr);
```

Description

Returns the data at the current position of the iterator.

Returns

The data at the current position of the iterator *itr*.

Note:
The iter_getdata function does not advance the iterator.

iter_getstrdata

Declaration

```
const char* iter_getstrdata(void* itr);
```

Description

Returns the data at the current position of the iterator as string.

Returns

The data at the current position of the iterator *itr* as string.

Note:
The iter_getstrdata function does not advance the iterator.

iter_next

Declaration

```
void* iter_next(void* itr);
```

Description

Moves the iterator to the next element in the iterator.

Returns

Nothing

---

iter_prev

Declaration

```
void* iter_prev(void* itr);
```

Description

Moves the iterator to the previous element in the iterator.

Returns

Nothing

---

itr_rewind

Declaration

```
void itr_rewind(void* itr);
```

Description

Rewinds the iterator to the first element in the iterator.

Returns

Nothing

Object-Specific API Functions

The object-specific API functions are as follows:

Address Interface

The Address Interface object has the following design hierarchy:

comRoot -> Design -> Component Instance -> Address Interface

Use the following API functions to process the Address Interface objects:

AddressInterfaceGetEndAddress

Declaration

```
int AddressInterfaceGetEndAddress(void* obj);
```

Description

Returns the end address of address interface object obj.

Arguments

`obj`

Address interface object

Returns

End address of the address interface object

AddressInterfaceGetMemoryPath

Declaration

```
void* AddressInterfaceGetMemoryPath(void* obj)
```

Description

Returns the memory path to the target node.

Memory path is defined as a list of instance/interface names along with the memory path to the target.

Arguments

`obj`

Address interface object

Returns

Memory path to the target node

AddressInterfaceGetMemoryPathName

Declaration

```
Const char* AddressInterfaceGetMemoryPathName(void* obj)
```

Description

Returns memory path name to the target node.

Memory path name consists instance/ interface names separated by __ as shown in the following example:

```
instance1__interface1__instance2__interface2__...
```

In the above example, instance(i)/interface(i) are instance/interface names along with the memory path to the target node.

Arguments

`obj`

Address interface object

Returns

Memory path name to the target node

AddressInterfaceGetSize

Declaration

```
int AddressInterfaceGetSize(void* obj);
```

Description

Returns the size of address interface object obj.

Arguments

`obj`

Address interface object

Returns

Size of the address interface object

AddressInterfaceGetStartAddress

Declaration

```
int AddressInterfaceGetStartAddress(void* obj);
```

Description

Returns the start address of address interface object obj.

Arguments

`obj`

Address interface object

Returns

Start address of the address interface object

AddressInterfaceGetTargetElementName

Declaration

```
const char* AddressInterfaceGetTargetElementName(void* obj);
```

Description

Returns the register/interface/memory information for the target element of address interface object obj.

Arguments

`obj`

Address interface object

AddressInterfaceGetTargetElementType

Declaration

```
const char* AddressInterfaceGetTargetElementType(void* obj);
```

Description

Returns the type (REGISTER, MEMORY, INTERFACE, or RESERVED) of the target element of address interface object obj.

Arguments

`obj`

Address interface object

AddressInterfaceGetTargetInstName

Declaration

```
const char* AddressInterfaceGetTargetInstName(void* obj);
```

Description

Returns the target instance name of address interface object obj.

Arguments

`obj`

Address interface object

Returns

Target instance name of the address interface object

AddressMapDefn

The AddressMapDefn object has the following design hierarchy:

comRoot -> Component -> AddressMapDefn

Use the following API functions to process the AddressMapDefn objects:

AddressMapDefnGetAddressing

Declaration

```
char* AddressMapDefnGetAddressing(void* AddressMapDefn);
```

Description

Returns the addressing information (compact, regalign, or fullalign) for the AddressMapDefn object returned by the ComponentGetAddresssMapDefnByName or ComponentGetAddressMapDefnList function.

AddressMapDefnGetAlignment

Declaration

```
char* AddressMapDefnGetAlignment(void* AddressMapDefn);
```

Description

Returns the alignment information for the AddressMapDefn object returned by the ComponentGetAddresssMapDefnByName or ComponentGetAddressMapDefnList function.

AddressMapDefnGetDontCompare

Declaration

```
Int AddressMapDefnGetDontCompare(void* bf)
```

Description

Returns the DontCompare value for the AddressMapDefn object.

AddressMapDefnGetDontTest

Declaration
```
Int AddressMapDefnGetDontTest(void* bf)
```

Description

Returns the DontTest value for the AddressMapDefn object.

AddressMapDefnGetEndianess

Declaration
```
char* AddressMapDefnGetEndianess(void* AddressMapDefn);
```

Description

Returns the endianness information (bigendian or littleendian) for the AddressMapDefn object returned by the ComponentGetAddresssMapDefnByName or ComponentGetAddressMapDefnList function.

AddressMapDefnGetMap

Declaration
```
char* AddressMapDefnGetMap(void* AddressMapDefn);
```

Description

Returns the memory map name for the AddressMapDefn object returned by the ComponentGetAddresssMapDefnByName or ComponentGetAddressMapDefnList function.

AddressMapDefnGetOrientation

Declaration
```
char* AddressMapDefnGetOrientation(void* AddressMapDefn);
```

Description

Returns the orientation information (lsb0 or msb0) for the AddressMapDefn object returned by the ComponentGetAddresssMapDefnByName or ComponentGetAddressMapDefnList function.

AddressMapDefnGetPropertyAssignByName

Declaration

```
void* AddressMapDefnGetPropertyAssignByName(void* AddressMapDefn, char
name)
```

Description

Returns named property of the AddressMapDefn object returned by the ComponentGetAddresssMapDefnByName or ComponentGetAddressMapDefnList function.

AddressMapDefnGetPropertyAssignList

Declaration

```
void* AddressMapDefnGetPropertyAssignList(void* AddressMapDefn)
```

Description

Returns a list of properties of the AddressMapDefn object returned by the ComponentGetAddresssMapDefnByName or ComponentGetAddressMapDefnList function.

AddressMapDefnGetRDLName

Declaration

```
char* AddressMapDefnGetRDLName(void* AddressMapDefn);
```

Description

Returns the RDL name of the AddressMapDefn object returned by the ComponentGetAddresssMapDefnByName or ComponentGetAddressMapDefnList function.

AddressMapDefnGetRsvdset

Declaration

```
int AddressMapDefnGetRsvdset(void* AddressMapDefn);
```

Description

Returns the rsvdset information (1 or 0) for the AddressMapDefn object returned by the ComponentGetAddresssMapDefnByName or ComponentGetAddressMapDefnList function.

AddressMapDefnGetRsvdsetX

Declaration

```
int AddressMapDefnGetRsvdsetX(void* AddressMapDefn);
```

Description

Returns the rsvdsetX information (1 or 0) for the AddressMapDefn object returned by the ComponentGetAddresssMapDefnByName or ComponentGetAddressMapDefnList function.

AddressMapDefnGetScope

Declaration

```
char* AddressMapDefnGetScope(void* AddressMapDefn);
```

Description

Returns the scope information for the AddressMapDefn object returned by the ComponentGetAddresssMapDefnByName or ComponentGetAddressMapDefnList function.

AddressMapDefnGetSharedExtBus

Declaration

```
int AddressMapDefnGetSharedExtBus(void* AddressMapDefn);
```

Description

Returns the SharedExtBus information (1 or 0) for the AddressMapDefn object returned by the ComponentGetAddresssMapDefnByName or ComponentGetAddressMapDefnList function.

Adhoc Connection

The Adhoc Connection object has the following design hierarchy:

comRoot -> Design -> Component Instance -> Connection -> AdHoc Connection

comRoot -> Design -> Connection -> AdHoc Connection

Use the following API functions to process the Adhoc Connection objects:

AdHocConnectionGetCompInst

Declaration

```
void* AdHocConnectionGetCompInst(void* aconn);
```

Description

Returns the component instance for the Adhoc Connection *aconn*.

Arguments

`aconn`

Adhoc Connection Item object.

Returns

Component Instance for the Adhoc Connection *aconn*.

Notes

Use the Component Instance-related API functions to process component instances.

AdHocConnectionGetLSB

Declaration

```
const char* AdHocConnectionGetLSB(void* aconn);
```

Description

Returns the LSB of the Adhoc Connection *aconn*.

Arguments

`aconn`

Adhoc Connection Item object.

Returns

LSB of the Adhoc Connection *aconn*.

AdHocConnectionGetMSB

Declaration

```
const char* AdHocConnectionGetMSB(void* aconn);
```

Description

Returns the MSB of the Adhoc Connection *aconn*.

Arguments

`aconn`

Adhoc Connection Item object.

Returns

MSB of the Adhoc Connection *aconn*.

AdHocConnectionGetTerminal

Declaration

```
void* AdHocConnectionGetTerminal(void* aconn);
```

Description

Returns the terminal of the Adhoc Connection *aconn*.

Arguments

`aconn`

Adhoc Connection Item object.

Returns

Terminal for the Adhoc Connection *aconn*.

Notes

Use the Terminal-related API functions to process terminals.

AdHocConnectionIsTerminalAPort

Declaration

```
int AdHocConnectionIsTerminalAPort(void* aconn);
```

Description

Returns 1 if the terminal of the Adhoc Connection *aconn* is a primary port. Otherwise returns 0.

Arguments

`aconn`

Adhoc Connection Item object.

Returns

1 if the terminal of the Adhoc Connection *aconn* is a primary port. Otherwise returns 0.

Alias Logical Port

The Alias Logical Port object has the following design hierarchy:

comRoot -> Design -> Logical Port -> Alias Logical Port

comRoot -> Component -> Logical Port -> Alias Logical Port

comRoot -> Interface Definition -> Logical Port -> Alias Logical Port

Use the following API functions to process the Alias objects:

AliasLogicalPortGetLSB

Declaration

```
const char* AliasLogicalPortGetLSB(void* aliasLport);
```

Description

Returns the LSB of Alias Logical Port *aliasLport*.

Arguments

`aliasLport`

> Alias Logcial Port object returned by the LogicalPortGetAliasLogicalPortByName/ LogicalPortGetAliasLogicalPortList function.

Returns

LSB of Alias Logical Port *aliasLport*.

AliasLogicalPortGetMSB

Declaration

```
const char* AliasLogicalPortGetMSB(void* aliasLport);
```

Description

Returns the MSB of Alias Logical Port *aliasLport*.

Arguments

`aliasLport`

> Alias Logcial Port object returned by the LogicalPortGetAliasLogicalPortByName/ LogicalPortGetAliasLogicalPortList function.

Returns

MSB of Alias Logical Port *aliasLport*.

AliasLogicalPortGetName

Declaration

```
const char* AliasLogicalPortGetName(void* aliasLport);
```

Description

Returns the name of Alias Logical Port *aliasLport*.

Arguments

`aliasLport`

> Alias Logical Port object returned by the LogicalPortGetAliasLogicalPortByName/ LogicalPortGetAliasLogicalPortList function.

Returns

Name of Alias Logical Port *aliasLport*.

BitEnumDefn

The BitEnumDefn object has the following design hierarchy:

comRoot -> Component -> BitEnumDefn

Use the following API functions to process the BitEnumDefn objects:

BitEnumDefnGetEnumName

Declaration
```
char* BitEnumDefnGetEnumName (void* bitEnumDefn);
```

Description

Returns the name of the BitEnumDefn object returned by the ComponentGetBitEnumDefnList or ComponentGetBitEnumDefnByName function.

BitEnumDefnGetEnumTokenDefnByName

Declaration
```
void* BitEnumDefnGetEnumTokenDefnByName (void* bitEnumDefn);
```

Description

Returns the named enum token of the BitEnumDefn object returned by the ComponentGetBitEnumDefnList or ComponentGetBitEnumDefnByName function.

BitEnumDefnGetEnumTokenDefnList

Declaration
```
void* BitEnumDefnGetEnumTokenDefnList (void* bitEnumDefn);
```

Description

Returns a list of the EnumTokenDefnList objects of the BitEnumDefn object returned by the ComponentGetBitEnumDefnList or ComponentGetBitEnumDefnByName function.

BitEnumDefnGetRDLName

Declaration

```
char* BitEnumDefnGetRDLName(void* BitEnumDefn);
```

Description

Returns the RDL name of the BitEnumDefn object returned by the ComponentGetBitEnumDefnList or ComponentGetBitEnumDefnByName function.

BitEnumValue

The BitEnumValue object has the following design hierarchy:

comRoot -> Component -> Register Object -> Register Group -> Register Data -> BitField -> BitEnumValue

comRoot -> Component -> Register Object -> Register Data -> BitField -> BitEnumValue

Use the following API functions to process the BitEnumValue objects:

BitEnumValueGetAccessType

Declaration

```
const char* BitEnumValueGetAccessType (void* bitEnumValue);
```

Arguments

`bitEnumValue`

BitEnumValue object returned by the BitFieldGetBitEnumValueByName/ BitFieldGetBitEnumValuesList function.

Returns

The allowed access such as read or write for the BitEnum *bitEnumValue*.

BitEnumValueGetMaximum

Declaration

```
int BitEnumValueGetMaximum (void* bitEnumValue);
```

Arguments

`bitEnumValue`

BitEnumValue object returned by the BitFieldGetBitEnumValueByName/ BitFieldGetBitEnumValuesList function.

Returns

The maximum value of BitEnum *bitEnumValue*.

BitEnumValueGetMinimum

Declaration

```
int BitEnumValueGetMinimum (void* bitEnumValue);
```

Arguments

`bitEnumValue`

BitEnumValue object returned by the BitFieldGetBitEnumValueByName/ BitFieldGetBitEnumValuesList function.

Returns

The minimum value of BitEnum *bitEnumValue*.

BitEnumValueGetName

Declaration

```
const char* BitEnumValueGetName(void* bitEnumValue);
```

Description

Returns the name of BitEnum Value *bitEnumValue*.

Arguments
`bitEnumValue`

BitEnumValue object returned by the BitFieldGetBitEnumValueByName/ BitFieldGetBitEnumValuesList function.

Returns

Name of BitEnum Value *bitEnumValue*.

BitEnumValueGetRDLName

Declaration
```
char* BitEnumValueGetRDLName(void* bitEnumValue);
```

Description

Returns the RDL name of BitEnum Value *bitEnumValue*.

Arguments
`bitEnumValue`

BitEnumValue object returned by the BitFieldGetBitEnumValueByName/ BitFieldGetBitEnumValuesList function.

Returns

RDL name of BitEnum Value *bitEnumValue*.

BitEnumValueGetValue

Declaration
```
const char* BitEnumValueGetValue(void* bitEnum);
```

Description

Returns the value for BitEnum Value *bitEnumValue*.

Arguments

`bitEnum`

BitEnum Value object returned by the BitFieldGetBitEnumValueByName/ BitFieldGetBitEnumValuesListfunction.

Returns

Value of BitEnum Value *bitEnumValue*.

BitField

The BitField object has the following design hierarchy:

comRoot -> Component -> Register Object -> Register Group> Register Data -> BitField

comRoot -> Component -> Register Object -> Register Data -> BitField

Use the following API functions to process the BitField objects:

BitFieldGetAccessType

Declaration

```
const char* BitFieldGetAccessType(void* bitField);
```

Description

Returns the access type of the Bitfield *bitField*.

Arguments

`bitField`

BitField object returned by the RegisterDataGetBitFieldByName/ RegisterDataGetBitFieldList function.

Returns

Access type of the Bitfield *bitField* as one of the following values:

EXECUTE	NOT_ACCESSIBLE	PROGRAM_MEMORY
READ	READ_CLEAR	READ_EXECUTE
READ_WRITE	READ_WRITE_CLEAR	READ_WRITE_EXEC

READ_WRITE_SET	RESERVED	STICKY
UNDEF	WRITE	WRITE_CLEAR

BitFieldGetActualReset

Declaration

```
const char* BitFieldGetActualReset(void* bitField);
```

Description

Returns the actual reset value of Bitfield *bitField*.

If the reset value of the bitfield is not specified, undef is returned.

The return value of BitFieldGetActualReset function can be used to determine whether the value returned by the BitFieldGetReset function is the actual reset value or the default reset value.

Arguments

`bitField`

BitField object returned by the RegisterDataGetBitFieldByName/ RegisterDataGetBitFieldList function.

Returns

Actual Reset Value of Bitfield *bitField*.

BitFieldGetBitEnumValueByName

Declaration

```
void* BitFieldGetBitEnumValueByName  (void* bitField, const char*
bitEnumValueName);
```

Description

Returns the named enum value *bitEnumValueName* of Bitfield *bitField*.

Arguments

`bitField`

BitField object returned by the RegisterDataGetBitFieldByName/ RegisterDataGetBitFieldList function.

`bitEnumValueName`

Named BitEnum value

Returns

Named BitEnum Value *bitEnumValueName* of Bitfield *bitField*.

Notes

Use the BitEnumValue-related API functions to process the BitEnum Values.

BitFieldGetBitEnumValuesList

Declaration

```
void* BitFieldGetBitEnumValuesList(void* bitField);
```

Description

Returns the list (*itr*) of enum values of Bitfield *bitField*.

Arguments

`bitField`

BitField object returned by the RegisterDataGetBitFieldByName/ RegisterDataGetBitFieldList function.

Returns

List (*itr*) of BitEnum Values of Bitfield *bitField*..

Notes

Use the BitEnumValue-related API functions to process the BitEnum Values.

BitFieldGetClockSignal

Declaration

```
const char* BitFieldGetClockSignal (void* bitField)
```

Description

Returns value of a clock signal of the specified Bitfield *bitField*.

Arguments

```
bitField
```

BitField object returned by the RegisterDataGetBitFieldByName/ RegisterDataGetBitFieldList function.

Returns

Bitfield clock signal value

BitFieldGetCounterProperty

Declaration

```
char* BitFieldGetCounterProperty(void* bf, const char* prop_name)
```

Description

Returns the counter property.

Arguments

```
bitField
```

BitField object returned by the RegisterDataGetBitFieldByName/ RegisterDataGetBitFieldList function.

Returns

Bitfield clock signal value

BitFieldGetDontCompare

Declaration

```
Int BitFieldGetDontCompare(void* bf)
```

Description

Returns the DontCompare value of a bitfield of a SystemRDL register object.

BitFieldGetDontTest

Declaration

```
Int BitFieldGetDontTest(void* bf)
```

Description

Returns the DontTest value of a bitfield of a SystemRDL register object.

BitFieldGetEncode

Declaration

```
char* BitFieldGetEncode(void* bitField)
```

Description

Returns the encode value of a bitfield of a SystemRDL register object.

Arguments

`bitField`

BitField object returned by the RegisterDataGetBitFieldByName/ RegisterDataGetBitFieldList function.

BitfieldGetField

Declaration

```
char* BitfieldGetField(void* bitField)
```

Description

Returns the field value of a bitfield of a SystemRDL register object.

Arguments
`bitField`

BitField object returned by the RegisterDataGetBitFieldByName/ RegisterDataGetBitFieldList function.

BitFieldGetFunctionType

Declaration
```
const char* BitFieldGetFunctionType(void* bitField);
```

Description

Returns the function type of Bitfield *bitField*.

Arguments
`bitField`

BitField object returned by the RegisterDataGetBitFieldByName/ RegisterDataGetBitFieldList function.

Returns

Function type of Bitfield *bitField* as one of the following values:

ADDRESS	COMMAND	CONSTANT	CONTROL	COUNTER
DATA	MEM	RESERVED	STATUS	UNDEF

BitFieldGetHw

Declaration
```
char* BitFieldGetHw(void* bitField)
```

Description

Returns the HW value of the bitfiled of a SystemRDL register object. HW value can be any of the following:

rw	wr	r	w	na

Arguments
`bitField`

BitField object returned by the RegisterDataGetBitFieldByName/ RegisterDataGetBitFieldList function.

BitFieldGetHWAccessProperty

Declaration
`Char* BitFieldGetHWAccessProperty(void* bf, const char* prop_name)`

Description

Returns the value of the prop_name.

Arguments
`prop-name`

Name of the property object.

Returns

Returns the value of the prop_name.

BitFieldGetInterruptProperty

Declaration
`Char* BitFieldGetInterruptProperty(void* bf, const char* prop_name)`

Description

Returns the value of the prop_name.

Arguments
`prop-name`

Name of the property object.

Returns

Returns the value of the prop_name.

BitFieldGetIsVolatile

Declaration

```
const char* BitFieldGetIsVolatile(void* bitField);
```

Description

Returns 1 if Bitfield *bitField* is volatile. Otherwise, returns 0.

Arguments

`bitField`

BitField object returned by the RegisterDataGetBitFieldByName/ RegisterDataGetBitFieldList function.

Returns

1 if Bitfield *bitField* is volatile. Otherwise, returns 0.

BitFieldGetLSB

Declaration

```
const char* BitFieldGetLSB(void* bitField);
```

Description

Returns the LSB of Bitfield *bitField*.

Arguments

`bitField`

BitField object returned by the RegisterDataGetBitFieldByName/ RegisterDataGetBitFieldList function.

Returns

LSB of Bitfield *bitField*.

BitFieldGetMSB

Declaration

```
const char* BitFieldGetMSB(void* bitField);
```

Description

Returns the MSB of Bitfield *bitField*.

Arguments

`bitField`

BitField object returned by the RegisterDataGetBitFieldByName/ RegisterDataGetBitFieldList function.

Returns

MSB of Bitfield *bitField*.

BitFieldGetName

Declaration

```
const char* BitFieldGetName(void* bitField);
```

Description

Returns the name of Bitfield *bitField*.

Arguments

`bitField`

BitField object returned by the RegisterDataGetBitFieldByName/ RegisterDataGetBitFieldList function.

Returns

Name of Bitfield *bitField*.

BitFieldGetPrecedence

Declaration

```
char* BitFieldGetPrecedence(void* bitField)
```

Description

Returns the precedence value, hw or sw, of a bitfield of a SystemRDL register object.

Arguments

`bitField`

BitField object returned by the RegisterDataGetBitFieldByName/ RegisterDataGetBitFieldList function.

BitFieldGetPropertyAssignByName

Declaration

```
void* RegisterDataGetPropertyAssignByName(void* bitField, char name)
```

Description

Returns named property of the SystemRDL bitfield object *bitField*.

Arguments

`bitField`

BitField object returned by the RegisterDataGetBitFieldByName/ RegisterDataGetBitFieldList function.

`name`

Object name

BitFieldGetPropertyAssignList

Declaration

```
void* BitFieldGetPropertyAssignList(void* bitField)
```

Description

Returns a list of properties of the SystemRDL bitfield object *bitField*.

Arguments

`bitField`

BitField object returned by the RegisterDataGetBitFieldByName/ RegisterDataGetBitFieldList function.

BitfieldGetRDLName

Declaration

```
char* BitfieldGetRDLName(void* bitField)
```

Description

Returns the RDL name of a bitfield of a SystemRDL bitfield object.

Arguments

`bitField`

BitField object returned by the RegisterDataGetBitFieldByName/ RegisterDataGetBitFieldList function.

BitFieldGetReserved

Declaration

```
int BitFieldGetReserved(void* bitField);
```

Description

Returns 1 if the bitfield, *bitField*, is reserved and returns 0 if the bitfield is not reserved.

Arguments

`bitField`

BitField object returned by the RegisterDataGetBitFieldByName/ RegisterDataGetBitFieldList function.

BitFieldGetReset

Declaration

```
const char* BitFieldGetReset(void* bitField);
```

Description

Returns the reset value of Bitfield *bitField*.

If the reset value of the bitfield is not specified explicitly, the default reset value (0 or X) of that bitfield is returned.

Arguments

`bitField`

BitField object returned by the RegisterDataGetBitFieldByName/ RegisterDataGetBitFieldList function.

Returns

Reset Value of Bitfield *bitField*.

BitFieldGetResetMask

Declaration

```
const char* BitFieldGetResetMask(void* bitField);
```

Description

Returns reset mask value for the bitfield *bitField*.

Arguments

`bitField`

BitField object returned by the RegisterDataGetBitFieldByName/ RegisterDataGetBitFieldList function.

BitFieldGetResetSignal

Declaration

```
char* BitFieldGetResetSignal(obj);
```

Description

Returns the reset signal of the bitfield *bitField*.

Arguments

`bitField`

BitField object returned by the RegisterDataGetBitFieldByName/ RegisterDataGetBitFieldList function.

Returns

Reset signal of the bitfield.

BitFieldGetRType

Declaration

```
const char* BitFieldGetRType (void* bitField);
```

Description

Returns the RType of the bitfiled. RType can have any of the following values:

R	NA	RtoClr	Rreturn0s
Rreturn1s	Rreturns	RreturnsRemote	RSpecial

Arguments

`bitField`

BitField object returned by the RegisterDataGetBitFieldByName/ RegisterDataGetBitFieldList function.

BitFieldGetRTypeBitFieldRef

Declaration

```
const char* BitFieldGetRTypeBitFieldRef(void* bitField);
```

Description

Returns the bitfield reference for the bitfield, *bitField*, if RType of the bitfield is RreturnsRemote.

Arguments

`bitField`

BitField object returned by the RegisterDataGetBitFieldByName/ RegisterDataGetBitFieldList function.

BitFieldGetRTypeRegisterRef

Declaration

```
const char* BitFieldGetRTypeRegisterRef(void* bitField);
```

Description

Returns the register reference for the bitfield, *bitField*, if RType of the bitfield is RreturnsRemote.

Arguments

`bitField`

BitField object returned by the RegisterDataGetBitFieldByName/ RegisterDataGetBitFieldList function.

BitFieldGetRTypeValue

Declaration

```
const char* BitFieldGetRTypeValue(void* bitField);
```

Description

Returns the value of the bitfield, *bitField*, if the RType of the bitfield is Rreturns.

Arguments

`bitField`

BitField object returned by the RegisterDataGetBitFieldByName/ RegisterDataGetBitFieldList function.

BitFieldGetSticky

Declaration

```
char* BitFieldGetSticky(void* bitField);
```

Description

Returns the sticky information of the bitfield *bitField*.

Arguments

`bitField`

BitField object returned by the RegisterDataGetBitFieldByName/ RegisterDataGetBitFieldList function.

Returns

Returns the sticky information of the bitfield.

BitFieldGetSw

Declaration

```
char* BitFieldGetSw(void* bitField)
```

Description

Returns the SW value of the bitfiled of a SystemRDL register object. SW value can be any of the following:

rw	wr	r	w	na

Arguments

`bitField`

BitField object returned by the RegisterDataGetBitFieldByName/ RegisterDataGetBitFieldList function.

BitFieldGetSWAccessProperty

Declaration

```
Char* BitFieldGetSWAccessProperty(void* bf, const char* prop_name)
```

Description

Returns the value of the prop_name.

Arguments

`prop-name`

Name of the property object.

Returns

Returns the value of the prop_name.

BitFieldGetVType

Declaration

```
const char* BitFieldGetVType(void* bitField);
```

Description

Returns the VType field value for the bitfield *bitField*.

Arguments

```
bitField
```

BitField object returned by the RegisterDataGetBitFieldByName/ RegisterDataGetBitFieldList function.

BitFieldGetWType

Declaration

```
const char* BitFieldGetWType (void* bitField);
```

Description

Returns the WType of the bitfiled. WType can have any of the following values:

W	NA	WtoClr	W0toClr
W1toClr	W0toSet	W1toSet	WCtoClr
WCtoSet	WToggle	WSWReset	Woco
WSpecial			

Arguments

```
bitField
```

BitField object returned by the RegisterDataGetBitFieldByName/ RegisterDataGetBitFieldList function.

BitFieldGetWTypeBitFieldRef

Declaration
```
const char* BitFieldGetWTypeBitFieldRef(void* bitField);
```

Description

Returns the bitfield reference for the bitfield, *bitField*, if WType of the bitfield is W1toSet/ W0toSet.

Arguments
```
bitField
```

BitField object returned by the RegisterDataGetBitFieldByName/ RegisterDataGetBitFieldList function.

BitFieldGetWTypeRegisterRef

Declaration
```
const char* BitFieldGetWTypeRegisterRef(void* bitField);
```

Description

Returns the register reference for the bitfield, *bitField*, if WType of the bitfield is W1toSet/ W0toSet.

Arguments
```
bitField
```

BitField object returned by the RegisterDataGetBitFieldByName/ RegisterDataGetBitFieldList function.

Bridge

The Bridge type item object has the following design hierarchy:

comRoot -> Component -> Bridge

Use the following API functions to process the Bus objects:

BusBridgeGetBridgeMemoryMap

Declaration
```
void* BusBridgeGetBridgeMemoryMap(void* obj);
```

Description

Returns the bridge memory map pointer corresponding to the bus bridge object, *obj*.

Arguments

`obj`

Bus bridge object

BusBridgeGetBridgePathList

Declaration

```
void* BusBridgeGetBridgePathList(void * obj);
```

Description

Returns a list of bridge paths associated with the bus bridge object, *obj*.

Arguments

`obj`

Bus bridge object

BusBridgeGetSlaveInterfaceName

Declaration

```
const char* BusBridgeGetSlaveInterfaceName (void* obj);
```

Description

Returns the slave interface name of the bus bridge object, *obj*.

Arguments

`obj`

Bus bridge object

BitFieldDefn

The BitFieldDefn object has the following design hierarchy:

comRoot -> Component -> BitField Definition

Use the following API functions to process the BitFieldDefn objects:

BitFieldDefnGetClockSignal

Declaration

```
const char* BitFieldDefnGetClockSignal (void* BitFieldDefn)
```

Description

Returns the value of a clock signal of the BitFieldDefn object returned by the ComponentGetBitFieldDefnList or ComponentGetBitFieldDefnByName function.

BitFieldDefnGetCounterProperty

Declaration

```
Char* BitFieldDefnGetCounterProperty(void* bf, const char* prop_name)
```

Description

Returns the value of the prop_name.

Arguments

`prop-name`

Name of the property object.

Returns

Returns the value of the prop_name.

BitFieldDefnGetDontCompare

Declaration

```
Int BitFieldDefnGetDontCompare(void* bf)
```

Description

Returns the DontCompare value for the BitFieldDefn object.

BitFieldDefnGetDontTest

Declaration

```
Int BitFieldDefnGetDontTest(void* bf)
```

Description

Returns the DontTest value for the BitFieldDefn object.

BitFieldDefnGetEncode

Declaration

```
char* BitFieldDefnGetEncode(void* BitFieldDefn)
```

Description

Returns the encode of the BitFieldDefn object returned by the ComponentGetBitFieldDefnList or ComponentGetBitFieldDefnByName function.

BitFieldDefnGetField

Declaration

```
char* BitFieldDefnGetField(void* BitFieldDefn)
```

Description

Returns the field information of the BitFieldDefn object returned by the ComponentGetBitFieldDefnList or ComponentGetBitFieldDefnByName function.

BitFieldDefnGetHw

Declaration

```
char* BitFieldDefnGetHw(void* BitFieldDefn)
```

Description

Returns the hw information (rw, wr, r, w, or na) of the BitFieldDefn object returned by the ComponentGetBitFieldDefnList or ComponentGetBitFieldDefnByName function.

BitFieldDefnGetHWAccessProperty

Declaration

```
Char* BitFieldDefnGetHWAccessProperty(void* bf, const char* prop_name)
```

Description

Returns the value of the prop_name.

Arguments

`prop-name`

Name of the property object.

Returns

Returns the value of the prop_name.

BitFieldDefnGetInterruptProperty

Declaration

```
Char* BitFieldDefnGetInterruptProperty(void* bf, const char* prop_name)
```

Description

Returns the value of the prop_name.

Arguments

`prop-name`

Name of the property object.

Returns

Returns the value of the prop_name.

BitFieldDefnGetPrecedence

Declaration

```
char* BitFieldDefnGetPrecedence(void* BitFieldDefn)
```

Description

Returns the precedence value (hw or sw) of the BitFieldDefn object returned by the ComponentGetBitFieldDefnList or ComponentGetBitFieldDefnByName function.

BitFieldDefnGetPropertyAssignByName

Declaration

```
void* BitFieldDefnGetPropertyAssignByName(BitFieldDefn* SignalDefns, char
name)
```

Description

Returns named property for the BitFieldDefn object returned by the ComponentGetBitFieldDefnList or ComponentGetBitFieldDefnByName function.

BitFieldDefnGetPropertyAssignList

Declaration

```
void* BitFieldDefnGetPropertyAssignList(void* BitFieldDefn)
```

Description

Returns a list of properties for the BitFieldDefn object returned by the ComponentGetBitFieldDefnList or ComponentGetBitFieldDefnByName function.

BitFieldDefnGetRDLName

Declaration

```
char* BitFieldDefnGetRDLName(void* BitFieldDefn)
```

Description

Returns the RDL name of the BitFieldDefn object returned by the ComponentGetBitFieldDefnList or ComponentGetBitFieldDefnByName function.

BitFieldDefnGetResetSignal

Declaration
```
char* BitFieldDefnGetResetSignal(void* BitFieldDefn)
```

Description

Returns the the name of the reset signal for the BitFieldDefn object returned by the ComponentGetBitFieldDefnList or ComponentGetBitFieldDefnByName function.

BitFieldDefnGetSticky

Declaration
```
char* BitFieldDefnGetSticky(void* BitFieldDefn)
```

Description

Returns the sticky information (nonsticky, sticky, or stickybit) for the BitFieldDefn object returned by the ComponentGetBitFieldDefnList or ComponentGetBitFieldDefnByName function.

BitFieldDefnGetSw

Declaration
```
char* BitFieldDefnGetSw(void* BitFieldDefn)
```

Description

Returns the sw information (rw, wr, r, w, or na) of the BitFieldDefn object returned by the ComponentGetBitFieldDefnList or ComponentGetBitFieldDefnByName function.

BitFieldDefnGetSWAccessProperty

Declaration
```
Char* BitFieldDefnGetSWAccessProperty(void* bf, const char* prop_name)
```

Description

Returns the value of the prop_name.

Arguments

`prop-name`

Name of the property object.

Returns

Returns the value of the prop_name.

Bridge Memory Map

The Bridge Memory Map type Item object has the following design hierarchy:

comRoot -> Component -> Bridge Memory Map

Use the following API functions to process the Bridge Memory Map objects:

BridgeMemoryMapGetName

Declaration

```
const char* BridgeMemoryMapGetName (void* obj);
```

Description

Returns the name of the bridge memory map object, *obj*.

Arguments

`obj`

Bridge memory map object

Returns

Name of the bridge memory map object

BridgeMemoryMapGetRemapStateList

Declaration

`void* BridgeMemoryMapGetRemapStateList (void* obj);`

Description

Returns the list of bridge remap states associated with a bridge memory map object, *obj*.

Arguments

`obj`

Bridge memory map object

Returns

List of bridge remap states associated with a bridge memory map

Bridge Remap State

The Bridge Remap State type item object has the following design hierarchy:

comRoot -> Component -> Bridge Memory Map -> Bridge Remap State

Use the following API functions to process the Bridge Remap State objects:

RemapStateTableGetMasterOffsetPairList

Declaration

`void* RemapStateTableGetMasterOffsetPairList(void* obj);`

Description

Returns the list of master interface/base address pair associated with each bridge remap state.

Arguments

`obj`

Bridge remap state object

Returns

List of master interface/base address pair associated with each bridge remap state.

RemapStateTableGetName

Declaration
`const char* RemapStateTableGetName (void* obj);`

Description

Returns the name of the bridge remap state, *obj*.

Arguments
`obj`

Bridge remap state object

Returns

Name of bridge remap state

Bridge Path

The Bridge Path type item object has the following design hierarchy:

comRoot -> Component -> Bridge -> Bridge Path

Use the following API functions to process the Bridge Path objects:

BridgePathGetBaseAddress

Declaration
`const char* BridgePathGetBaseAddress (void* obj);`

Description

Returns the base address associated with the bridge path object, *obj*.

Arguments
`obj`

Bridge path object

BridgePathGetBitOffset

Declaration
```
const char* BridgePathGetBitOffset (void* obj);
```

Description

Returns the bit offset associated with the bridge path object, *obj*.

Arguments
`obj`

Bridge path object

BridgePathGetLAU

Declaration
```
const char* BridgePathGetLAU (void* obj);
```

Description

Returns the LAU associated with the bridge path object, *obj*.

Arguments
`obj`

Bridge path object

BridgePathGetMasterInterfaceName

Declaration
```
const char* BridgePathGetMasterInterfaceName(void* obj);
```

Description

Returns the target master interface name associated with the bridge path object, *obj*.

Arguments
`obj`

Bridge path object

BridgePathGetPathType

Declaration

```
const char* BridgePathGetPathType (void* obj);
```

Description

Returns the path type associated with the bridge path object, *obj*.

Arguments

`obj`

Bridge path object

BridgePathGetRange

Declaration

```
const char* BridgePathGetRange (void* obj);
```

Description

Returns the range associated with the bridge path object, *obj*.

Arguments

`obj`

Bridge path object

BridgePathGetWidth

Declaration

```
const char* BridgePathGetWidth (void* obj);
```

Description

Returns the width associated with the bridge path object, *obj*.

Arguments

`obj`

Bridge path object

Bus Channel

The Bus Channel Type Item object has the following design hierarchy:

comRoot -> Bus Channel

Use the following API functions to process the Bus Channel objects:

BusChannelGetMaxMasters

Declaration
```
const char* BusChannelGetMaxMasters(void* channel);
```

Description

Returns the maximum number of masters of the channel object, *channel*.

Arguments
```
channel
```

Channel object

Returns

Maximum number of masters of the channel object

BusChannelGetMaxSlaves

Declaration
```
const char* BusChannelGetMaxSlaves(void* channel);
```

Description

Returns the maximum number of slaves of the channel object, *channel*.

Arguments
```
channel
```

Channel object

Returns

Maximum number of slaves of the channel object

BusChannelGetMirroredMasterInterfaceList

Declaration

```
void* BusChannelGetMirroredMasterInterfaceList(void* channel);
```

Description

Returns a list of mirrored master interfaces for the bus channel object, *channel*.

Arguments

`channel`

Channel object

Returns

List of mirrored master interfaces for the bus channel object

BusChannelGetMirroredSlaveInterfaceList

Declaration

```
void* BusChannelGetMirroredSlaveInterfaceList(void* channel);
```

Description

Returns a list of mirrored slave interfaces for the bus channel object, *channel*.

Arguments

`channel`

Channel object

Returns

List of mirrored slave interfaces for the bus channel object

BusChannelGetName

Declaration

```
const char* BusChannelGetName(void* channel);
```

Description

Returns the name of the channel object, *channel*.

Arguments

`channel`

Channel object

Returns

Name of the channel object

Channel

The Channel type item object has the following design hierarchy:

comRoot -> Component -> Channel

Use the following API functions to process the Channel objects:

ChannelMirroredMasterInterfaceGetName

Declaration

```
const char* ChannelMirroredMasterInterfaceGetName(void* obj);
```

Description

Returns the name of the specified mirrored master interface object, *obj*.

Arguments

`obj`

Mirrored master interface object

Returns

Name of the mirrored master interface object

ChannelMirroredSlaveInterfaceGetName

Declaration
```
const char* ChannelMirroredSlaveInterfaceGetName(void* obj);
```

Description

Returns the name of the specified mirrored slave interface object, *obj*.

Arguments
`obj`

Mirrored slave interface object

Returns

Name of the mirrored slave interface object

ChannelMirroredSlaveInterfaceGetRange

Declaration
```
const char* ChannelMirroredSlaveInterfaceGetRange(void* obj);
```

Description

Returns the range of the specified mirrored slave interface object, *obj*.

Arguments
`obj`

Mirrored slave interface object

Returns

Range of the mirrored slave interface object

ChannelMirroredSlaveInterfaceGetRemapStateList

Declaration
```
void* ChannelMirroredSlaveInterfaceGetRemapStateList(void* obj);
```

Description

Returns the list of remapstates associated with the specified mirrored slave interface object, *obj*.

Arguments

`obj`

Mirrored slave interface object

Returns

List of remapstates of the specified mirrored slave interface object

ChannelRemapStateGetBaseAddress

Declaration

```
const char* ChannelRemapStateGetBaseAddress (void* obj);
```

Description

Returns the base address of the specified channel remapstate object, *obj*.

Arguments

`obj`

Channel remapstate object

Returns

Base address of the specified channel remapstate object

ChannelRemapStateGetName

Declaration

```
const char* ChannelRemapStateGetName (void* obj);
```

Description

Returns the name of the specified channel remapstate object, *obj*.

Arguments

`obj`

Channel remapstate object

Returns

Name of the specified channel remapstate object

CommonConnection

The Common Connection Type Item object has the following design hierarchy:

comRoot -> Design -> Component Instance -> Connection -> Common Connection

Use the following API functions to process the Common Connection objects:

CommonConnectionGetCompInst

Declaration

```
void* CommonConnectionGetCompInst(void* commonconn);
```

Description

Returns the component instance for common connection *commonconn*.

Arguments

```
commoncon
```

Connection object returned by the ComponentInstGetConnectionsList function.

Returns

Component Instance for common connection *commonconn*.

Notes

Use the Component Instance-related API functions to process component instances.

CommonConnectionGetLSB

Declaration

```
const char* CommonConnectionGetLSB(void* commonconn);
```

Description

Returns the LSB of common connection *commonconn*.

Arguments

`commoncon`

Connection object returned by the ComponentInstGetConnectionsList function.

Returns

LSB of common connection *commonconn*.

CommonConnectionGetMSB

Declaration

```
const char* CommonConnectionGetMSB(void* commonconn);
```

Description

Returns the MSB of common connection *commonconn*.

Arguments

`commoncon`

Connection object returned by the ComponentInstGetConnectionsList function.

Returns

MSB of common connection *commonconn*.

CommonConnectionGetPortTermRef

Declaration

```
void* CommonConnectionGetPortTermRef  (void* commonconn);
```

Description

Returns the port/terminal reference for common connection *commonconn*.

Arguments

`commoncon`

Connection object returned by the ComponentInstGetConnectionsList function.

Returns

Port/Terminal reference of common connection *commonconn*.

CommonConnectionGetTiedValue

Declaration
```
const char* CommonConnectionGetTiedValue  (void* commonconn);
```

Description

Returns the TiedOff value for common connection *commonconn*.

Arguments
```
commoncon
```

Connection object returned by the ComponentInstGetConnectionsList function.

Returns

TiedOff value of common connection *commonconn*.

Bank

The Bank object has the following design hierarchy:

comRoot -> Component > Bank

Use the following API functions to process Bank objects:

BankGetArraySize

Declaration
```
const char* BankGetArraySize(void* Bank);
```

Description

Returns the offset of Bank Bank.

Arguments
```
Bank
```

Bank object.

BankGetBankElementByName

Declaration

```
void* BankGetBankElementByName(void* Bank , const char* BankElementName);
```

Description

Returns the named BankElement Name for Bank Bank.

Arguments

`Bank`

Bank object.

`BankElementName`

Name of the BankElement.

Note:

Use the BankElement-related API functions to process the BankElement objects.

BankGetBankElementList

Declaration

```
void* BankGetBankElementList(void* Bank);
```

Description

Returns the list (itr) of BankElement for Bank Bank.

Arguments

`Bank`

Bank object.

Note:

Use the BankElement-related API functions to process the BankElement objects.

BankGetHead

Declaration
`const char* BankGetHead(void* Bank);`

Description

Returns the Head of Bank Bank.

Arguments
`Bank`

Bank object.

BankGetMode

Declaration
`const char* BankGetMode(void* Bank);`

Description

Returns the Mode of Bank Bank.

Arguments
`Bank`

Bank object.

BankGetName

Declaration
`const char* BankGetName(void* Bank);`

Description

Returns the name of the Bank Bank.

Arguments
`Bank`

Bank object.

BankGetReadControl

Declaration
```
const char* BankGetReadControl(void* Bank);
```

Description

Returns the ReadControl of Bank Bank.

Arguments
`Bank`

Bank object.

BankGetShadowDepth

Declaration
```
const char* BankGetShadowDepth(void* Bank);
```

Description

Returns the ShadowDepth of Bank Bank.

Arguments
`Bank`

Bank object.

BankGetTail

Declaration
```
const char* BankGetTail(void* Bank);
```

Description

Returns the Tail of Bank Bank.

Arguments
`Bank`

Bank object.

BankGetWriteControl

Declaration
```
const char* BankGetWriteControl(void* Bank);
```

Description

Returns the WriteControl of Bank Bank.

Arguments
`Bank`

Bank object.

Bank Element

The Bank Element object has the following design hierarchy:

comRoot -> Component > Bank Element

Use the following API functions to process Bank Element objects.

BankElementGetArraySize

Declaration
```
const char* BankElementGetArraySize(void* BankElement);
```

Description

Returns the ArraySize of BankElement is BankElement.

Arguments
`BankElement`

BankElement object.

BankElementGetBank

Declaration
```
void* BankElementGetBank(void* BankElement);
```

Description

Returns the Bank if BankElement type of BankElement is BANK.

Arguments

`BankElement`

BankElement object.

BankElementGetMemoryBlock

Declaration

```
void* BankElementGetMemoryBlock(void* BankElement);
```

Description

Returns the MemoryBlock if BankElement type of BankElement is MEMORY.

Arguments

`BankElement`

BankElement object.

BankElementGetOffset

Declaration

```
const char* BankElementGetOffset(void* BankElement);
```

Description

Returns the Offset of BankElement BankElement.

Arguments

`BankElement`

BankElement object.

BankElementGetRegisterData

Declaration

```
void* BankElementGetRegisterData(void* BankElement);
```

Description

Returns the register if BankElement type of BankElement is REGISTER.

Arguments

`BankElement`

BankElement object.

BankElementGetType

Declaration

```
const char* BankElementGetType(void* BankElement);
```

Description

Returns the type of Bank Element BankElement .

Arguments

`BankElement`

BankElement object.

Component

The Component object has the following design hierarchy:

comRoot -> Component

Use the following API functions to process the Component objects:

ComponentGetAddressInterfaceList

Declaration

```
void* ComponentGetAddressInterfaceList(void* comp, const char* req)
```

Description

Returns the elaborated address interface nodes list for a given requestor *req* on the component *comp*.

Arguments

`comp`

Component object returned by the comRootGetComponentByName/ comRootGetComponentList/comRootGetDesignByName function.

`req`

Requestor

ComponentGetAddressMapDefnList

Declaration

```
void* ComponentGetAddressMapDefnList(void* comp)
```

Description

Returns a list of AddressMapDefn objects for the component *comp*.

Arguments

`comp`

Component object returned by the comRootGetComponentByName/ comRootGetComponentList/comRootGetDesignByName function.

ComponentGetAddresssMapDefnByName

Declaration

```
void* ComponentGetAddresssMapDefnByName(void* comp, char name)
```

Description

Returns named address map object for the component *comp*.

Arguments

`comp`

Component object returned by the comRootGetComponentByName/ comRootGetComponentList/comRootGetDesignByName function.

`name`

Object name

ComponentGetBankByName

Declaration

```
void* ComponentGetBankByName(void* comp , const char* BankName);
```

Description

Returns the named bank BankName for the component *comp*.

Arguments

`comp`

Component object returned by the comRootGetComponentByName/ comRootGetComponentList/comRootGetDesignByName function.

`BankName`

Name of the Bank.

Note:
Use the Bank-related API functions to process the Bank objects.

ComponentGetBankList

Declaration

```
void* ComponentGetBankList(void* comp);
```

Description

Returns the list (itr) of Banks for the component *comp*.

Arguments

`comp`

Component object returned by the comRootGetComponentByName/ comRootGetComponentList/comRootGetDesignByName function.

Note:
Use the Bank-related API functions to process the Bank objects.

ComponentGetBitEnumDefnByName

Declaration

`void* ComponentGetBitEnumDefnByName(void* comp, char* name)`

Description

Returns named bitenum definition object

Arguments

`comp`

Component object returned by the comRootGetComponentByName/ comRootGetComponentList/comRootGetDesignByName function.

`name`

Object name

ComponentGetBitEnumDefnList

Declaration

`void* ComponentGetBitEnumDefnList(void* comp)`

Description

Returns list of biteum definitions for component *comp*.

Arguments

`comp`

Component object returned by the comRootGetComponentByName/ comRootGetComponentList/comRootGetDesignByName function.

ComponentGetBitFieldDefnByName

Declaration

`void* ComponentGetBitFieldDefnByName(void* comp, char* name)`

Description

Returns named bitfield definition object

Arguments

`comp`

Component object returned by the comRootGetComponentByName/ comRootGetComponentList/comRootGetDesignByName function.

`name`

Object name

ComponentGetBitFieldDefnList

Declaration

```
void* ComponentGetBitFieldDefnList(void* comp)
```

Description

Returns list of bitfield definitions for component *comp*.

Arguments

`comp`

Component object returned by the comRootGetComponentByName/ comRootGetComponentList/comRootGetDesignByName function.

ComponentGetBridgeMemoryMapByName

Declaration

```
void* ComponentGetBridgeMemoryMapByName(void* comp, const char* name)
```

Description

Returns the bridge memory map of the specified name.

Arguments

`comp`

Component object returned by the comRootGetComponentByName/ comRootGetComponentList/comRootGetDesignByName function.

`name`

Name of the bridge memory map

ComponentGetBridgeMemoryMapList

Declaration

`void* ComponentGetBridgeMemoryMapList(void* comp)`

Description

Returns the list of memory maps of bus bridges for a component *comp*.

Arguments

`comp`

Component object returned by the comRootGetComponentByName/ comRootGetComponentList/comRootGetDesignByName function.

ComponentGetBusBridgeList

Declaration

`void* ComponentGetBusBridgeList(void* comp)`

Description

Returns the list of bus bridges for a component *comp*.

Arguments

`comp`

Component object returned by the comRootGetComponentByName/ comRootGetComponentList/comRootGetDesignByName function.

ComponentGetBusChannelByName

Declaration

`void* ComponentGetBusChannelByName(void* comp, const char* name)`

Description

Returns the channel of the specified name.

Arguments
`comp`

Component object returned by the comRootGetComponentByName/ comRootGetComponentList/comRootGetDesignByName function.

`name`

Name of the channel

ComponentGetBusChannelList

Declaration
```
void* ComponentGetBusChannelList(void* comp)
```

Description

Returns the list of bus channels for a component *comp*.

Arguments
`comp`

Component object returned by the comRootGetComponentByName/ comRootGetComponentList/comRootGetDesignByName function.

ComponentGetClkSourceList

Declaration
```
void* ComponentGetClkSourceList(void* comp);
```

Description

Returns the list (*itr*) of Clock Sources for component *comp*.

Arguments
`comp`

Component object returned by the comRootGetComponentByName/ comRootGetComponentList/comRootGetDesignByName function.

Returns

List (*itr*) of Clock Sources for component *comp*.

ComponentGetClockDestinationList

Declaration

```
void* ComponentGetClockDestinationList(void* comp);
```

Description

Returns the list (*itr*) of Clock destinations for component *comp*.

Arguments

`comp`

Component object returned by the comRootGetComponentByName/ comRootGetComponentList/comRootGetDesignByName function.

Returns

List (*itr*) of Clock Destinations for component *comp*.

ComponentGetComponentType

Declaration

```
const char* ComponentGetComponentType(void* comp);
```

Description

Returns the Component Type of component *comp*.

Arguments

`comp`

Component object returned by the comRootGetComponentByName/ comRootGetComponentList/comRootGetDesignByName function.

Returns

Component Type of component *comp* as one of the following values:

BUSLOGIC	CPU	MEMORY	OTHERS
PERIPHERAL	SUBSYSTEM	UNDEF	

ComponentGetEventDestinationList

Declaration

```
void* ComponentGetEventDestinationList(void* comp);
```

Description

Returns the list (*itr*) of Event Destinations for component *comp*.

Arguments

```
comp
```

Component object returned by the comRootGetComponentByName/ comRootGetComponentList/comRootGetDesignByName function.

Returns

List (*itr*) of Event Destinations for component *comp*.

ComponentGetEventSourceList

Declaration

```
void* ComponentGetEventSourceList(void* comp);
```

Description

Returns the list (*itr*) of Event Sources for component *comp*.

Arguments

```
comp
```

Component object returned by the comRootGetComponentByName/ comRootGetComponentList/comRootGetDesignByName function.

Returns

List (*itr*) of Event Sources for component *comp*.

ComponentGetInterfaceByName

Declaration

```
void* ComponentGetInterfaceByName  (void* comp, const char*
interfaceName);
```

Description

Returns the named interface *interfaceName* for component *comp*.

Arguments

`comp`

Component object returned by the comRootGetComponentByName/ comRootGetComponentList/comRootGetDesignByName function.

`interfaceName`

Name of the interface.

Returns

Named interface *interfaceName* for component *comp*.

Notes

Use the Interface-related API functions to process the Interface objects.

ComponentGetInterfaceList

Declaration

```
void* ComponentGetInterfaceList(void* comp);
```

Description

Returns the list (*itr*) of Interfaces for component *comp*.

Arguments

`comp`

Component object returned by the comRootGetComponentByName/ comRootGetComponentList/comRootGetDesignByName function.

Returns

List (*itr*) of Interfaces for component *comp*.

Notes

Use the Interface-related API functions to process the Interface objects.

ComponentGetLogicalPortByName

Declaration

```
void* ComponentGetLogicalPortByName  (void* comp, const char* lportName);
```

Description

Returns the named logical port *lportName* for component *comp*.

Arguments

`comp`

Component object returned by the comRootGetComponentByName/ comRootGetComponentList/comRootGetDesignByName function.

`lportName`

Name of the logical port.

Returns

Named logical port *lportName* for component *comp*.

Notes

Use the Logical Port-related API functions to process the Logical Port objects.

ComponentGetLogicalPortList

Declaration

```
void* ComponentGetLogicalPortList(void* comp);
```

Description

Returns the list (*itr*) of Logical Ports for component *comp*.

Arguments

`comp`

Component object returned by the comRootGetComponentByName/ comRootGetComponentList/comRootGetDesignByName function.

Returns

List (*itr*) of Logical Ports for component *comp*.

Notes

Use the Logical Port-related API functions to process the Logical Ports objects.

ComponentGetMemoryBlockByName

Declaration

```
void* ComponentGetMemoryBlockByName(void* comp, const char* memBlock);
```

Description

Returns the memory block *memBlock* for component *comp*.

Arguments

`comp`

Component object returned by the comRootGetComponentByName/ comRootGetComponentList/comRootGetDesignByName function.

`memBlock`

Name of the memory block object.

ComponentGetMemoryBlockList

Declaration

```
void* ComponentGetMemoryBlockList (void* comp);
```

Description

Returns the list (*itr*) of memory blocks for component *comp*.

Arguments

`comp`

Component object returned by the v function.

ComponentGetMemoryMapByName

Declaration

```
void* ComponentGetMemoryMapByName (void* comp, char name);
```

Description

Returns named memory map object

Arguments

`comp`

Component object returned by the comRootGetComponentByName/ comRootGetComponentList/comRootGetDesignByName function.

`name`

Object name

ComponentGetMemoryMapList

Declaration

```
void* ComponentGetMemoryMapList (void* comp);
```

Description

Returns a list of memory map objects.

Arguments

`comp`

Component object returned by the comRootGetComponentByName/ comRootGetComponentList/comRootGetDesignByName function.

ComponentGetMRSXML

Declaration

```
const char* ComponentGetMRSXML(void* comp);
```

Description

Returns the name of the Source MRS file for component *comp*.

Arguments

`comp`

Component object returned by the comRootGetComponentByName/ comRootGetComponentList/comRootGetDesignByName function.

Returns

Name of the Source MRS file for component *comp*.

ComponentGetName

Declaration

```
const char* ComponentGetName(void* comp);
```

Description

Returns the name of component *comp*.

Arguments

`comp`

Component object returned by the comRootGetComponentByName/ comRootGetComponentList/comRootGetDesignByName function.

Returns

Name of component *comp*.

ComponentGetParameterTable

Declaration

```
void* ComponentGetParameterTable(void* comp);
```

Description

Returns the Parameter table for component *comp*.

Arguments

`comp`

Component object returned by the comRootGetComponentByName/ comRootGetComponentList/comRootGetDesignByName function.

Returns

Parameter Table for component *comp*.

ComponentGetParent

Declaration

```
void* ComponentGetParent(void* comp);
```

Description

Returns the parent design for component *comp*.

Arguments

`comp`

Component object returned by the comRootGetComponentByName/ comRootGetComponentList/comRootGetDesignByName function.

Returns

Parent design for component *comp*.

ComponentGetPartition

Declaration

```
const char* ComponentGetPartition(void* comp)
```

Description

Returns the partition name for the component, *comp*.

Arguments
`comp`

Component object returned by the comRootGetComponentByName/ comRootGetComponentList/comRootGetDesignByName function.

Returns

Partition name for the component, *comp*.

ComponentGetPortByName

Declaration
```
void* ComponentGetPortByName  (void* comp, const char* portName);
```

Description

Returns the named port *portName* for component *comp*.

Arguments
`comp`

Component object returned by the comRootGetComponentByName/ comRootGetComponentList/comRootGetDesignByName function.

`portName`

Name of the port.

Returns

Named port *portName* for component *comp*.

Notes

Use the Port-related API functions to process the Port objects.

ComponentGetPortList

Declaration
```
void* ComponentGetPortList(void* comp);
```

Description

Returns the list (*itr*) of Ports for component *comp*.

Arguments

`comp`

Component object returned by the comRootGetComponentByName/ comRootGetComponentList/comRootGetDesignByName function.

Returns

List (*itr*) of Ports for component *comp*.

Notes

Use the Port-related API functions to process the Port objects.

ComponentGetPropertyDefnList

Declaration

```
void* ComponentGetPropertyDefnList(void* comp);
```

Description

Returns a list of PropertyDefn objects for component *comp*.

Arguments

`comp`

Component object returned by the comRootGetComponentByName/ comRootGetComponentList/comRootGetDesignByName function.

Returns

List of PropertyDefn objects for component *comp*.

ComponentGetProptertyDefnByName

Declaration

```
void* ComponentGetProptertyDefnByName(void* comp, char name);
```

Description

Returns namedPropertyDefn object for component *comp*.

Arguments

`comp`

Component object returned by the comRootGetComponentByName/ comRootGetComponentList/comRootGetDesignByName function.

`name`

Object name

Returns

Named PropertyDefn object for component *comp*.

ComponentGetRegfileDefnByName

Declaration

```
void* ComponentGetRegfileDefnByName(void* comp, char name)
```

Description

Returns named Regfile object for the component *comp*.

Arguments

`comp`

Component object returned by the comRootGetComponentByName/ comRootGetComponentList/comRootGetDesignByName function.

`name`

Object name

ComponentGetRegfileDefnList

Declaration

```
void* ComponentGetRegfileDefnList(void* comp)
```

Description

Returns a list of RegfileDefn for the component *comp*.

Arguments

`comp`

Component object returned by the comRootGetComponentByName/ comRootGetComponentList/comRootGetDesignByName function.

ComponentGetRegisterAccessSequence

Declaration

```
const char* ComponentGetRegisterAccessSequence  (void* comp);
```

Description

Returns the Register Access sequence for component *comp*.

Arguments

`comp`

Component object returned by the comRootGetComponentByName/ comRootGetComponentList/comRootGetDesignByName function.

Returns

Register Access sequence for component *comp*.

ComponentGetRegisterDataByName

Declaration

```
void* ComponentGetRegisterDataByNmae(void* comp , const char*
RegisterName);
```

Description

Returns the named Register Name for component *comp*.

Arguments

`comp`

Component object returned by the comRootGetComponentByName/ comRootGetComponentList/comRootGetDesignByName function.

`RegisterName`

Name of the Register.

Returns

Returns the named Register RegisterName for component *comp*.

Note:
Use the Register-related API functions to process the Register objects.

ComponentGetRegisterDataList

Declaration

```
void* ComponentGetRegisterDataList(void* comp);
```

Description

Returns the list (itr) of Registers for component *comp*.

Arguments

`comp`

Component object returned by the comRootGetComponentByName/ comRootGetComponentList/comRootGetDesignByName function.

Returns

List (itr) of Registers for component *comp*.

Note:
Use the Register-related API functions to process the Register objects.

ComponentGetRegisterObjectByName

Declaration

```
void* ComponentGetRegisterObjectByName(void* comp, const char*
regObjName)
```

Description

Returns the named register object *regObjName* for component *comp*.

Arguments

`comp`

Component object returned by the comRootGetComponentByName/ comRootGetComponentList/comRootGetDesignByName function.

`regObjName`

Name of the register object.

ComponentGetRegisterObjectList

Declaration

```
void* ComponentGetRegisterObjectList (void* comp);
```

Description

Returns the list (*itr*) of register objects for component *comp*.

Arguments

`comp`

Component object returned by the comRootGetComponentByName/ comRootGetComponentList/comRootGetDesignByName function.

ComponentGetRequestorList

Declaration

```
void* ComponentGetRequestorList (void* comp);
```

Description

Returns the list of requestor interfaces on the component *comp*.

Arguments

`comp`

Component object returned by the comRootGetComponentByName/ comRootGetComponentList/comRootGetDesignByName function.

ComponentGetResetDestinationList

Declaration
```
void* ComponentGetResetDestinationList(void* comp);
```

Description

Returns the list (*itr*) of Reset Destinations for component *comp*.

Arguments
```
comp
```

Component object returned by the comRootGetComponentByName/ comRootGetComponentList/comRootGetDesignByName function.

Returns

List (*itr*) of Reset Destinations for component *comp*.

ComponentGetRstSourceList

Declaration
```
void* ComponentGetRstSourceList(void* comp);
```

Description

Returns the list (*itr*) of Reset Sources for component *comp*.

Arguments
```
comp
```

Component object returned by the comRootGetComponentByName/ comRootGetComponentList/comRootGetDesignByName function.

Returns

List (*itr*) of Reset Sources for component *comp*.

ComponentGetSignalAssignList

Declaration

```
void* ComponentGetSignalAssignList(void* comp);
```

Description

Returns the list (*itr*) of signal assignments for component *comp*.

Arguments

`comp`

Component object returned by the comRootGetComponentByName/ comRootGetComponentList/comRootGetDesignByName function.

Returns

List (*itr*) of signal assignments for component *comp*.

ComponentGetSignalDefnByName

Declaration

```
void* ComponentGetSignalDefnByName(void* comp, char name);
```

Description

Returns the named signal object for component *comp*.

Arguments

`comp`

Component object returned by the comRootGetComponentByName/ comRootGetComponentList/comRootGetDesignByName function.

`comp`

Object name

Returns

Named SystemRDL signal object for component *comp*

ComponentGetSignalDefnList

Declaration

```
void* ComponentGetSignalDefnList(void* comp);
```

Description

Returns the list of SystemRDL signals for component *comp*.

Arguments

`comp`

Component object returned by the comRootGetComponentByName/ comRootGetComponentList/comRootGetDesignByName function.

Returns

List of SystemRDL signals for component *comp*

ComponentGetTableByName

Declaration

```
void* ComponentGetTableByName  (void* comp, const char* childTableName);
```

Description

Returns the named table *childTableName* of component *comp*.

Note:
The ComponentGetTableByName API function works only for tabs present in the GenSys standard schema.

Arguments

`comp`

Component object returned by the comRootGetComponentByName/ comRootGetComponentList/comRootGetDesignByName function.

`tableName`

Name of the table to be fetched.

Returns

Named table *childTableName* of component *comp*.

ComponentGetTableList

Declaration
```
void* ComponentGetTableList(void* comp);
```

Description

Returns the list of tables (*itr*) for component *comp*.

Note:
The ComponentGetTableList API function works only for tabs present in the GenSys standard schema.

Arguments
```
comp
```

Component object returned by the comRootGetComponentByName/comRootGetComponentList/comRootGetDesignByName function.

Returns

List (*itr*) of Tables for component *comp*.

ComponentGetThirdPartyGenerator

Declaration
```
const char* ComponentGetThirdPartyGenerator  (void* comp);
```

Description

Returns the third party generator for component *comp*.

Arguments
```
comp
```

Component object returned by the comRootGetComponentByName/comRootGetComponentList/comRootGetDesignByName function.

Returns

Third Party Generator for component *comp*.

ComponentGetUserParameterTable

Declaration

```
void* ComponentGetUserParameterTable(void* comp);
```

Description

Returns the User-defined Parameter Table for component *comp*.

Arguments

`comp`

Component object returned by the comRootGetComponentByName/ comRootGetComponentList/comRootGetDesignByName function.

Returns

User-defined Parameter Table for component *comp*.

ComponentGetVersionedName

Declaration

```
void* ComponentGetVersionedName(void* comp);
```

Description

Returns the Versioned Name (*vnlv*) for component *comp*.

Arguments

`comp`

Component object returned by the comRootGetComponentByName/ comRootGetComponentList/comRootGetDesignByName function.

Returns

Versioned Name (*vnlv*) for component *comp*.

Notes

Use the VersionedNameClass-related API functions to process the Versioned Name objects.

ComponentHasMultipleInstantiations

Declaration

```
int ComponentHasMultipleInstantiations(void* comp);
```

Description

Specifies if the component *comp* is instantiated once or multiple times.

Arguments

`comp`

Component object returned by the comRootGetComponentByName/ comRootGetComponentList/comRootGetDesignByName function.

Returns

Returns 0 if component *comp* is instantiated once.

Returns 1 if component *comp* is instantiated multiple times.

ComponentIsADesign

Declaration

```
int ComponentIsADesign(void* comp);
```

Description

Returns 1 if component *comp* is a design. Otherwise returns 0.

Arguments

`comp`

Component object returned by the comRootGetComponentByName/ comRootGetComponentList/comRootGetDesignByName function.

Returns

1 if component *comp* is a design. Otherwise returns 0.

ComponentIsDummy

Declaration

```
int ComponentIsDummy(void* comp);
```

Description

Returns 1 if component *comp* is a dummy component. Otherwise, returns 0.

ComponentIsInternallyGenerated

Declaration

```
int ComponentIsInternallyGenerated(void* comp);
```

Description

Returns 1 if component *comp* is an internally generated component. Otherwise, returns 0.

Arguments

```
comp
```

Component object returned by the comRootGetComponentByName/ comRootGetComponentList/comRootGetDesignByName function.

Returns

1 if component *comp* is an internally generated component. Otherwise, returns 0.

Component Instance

The Component Instance object has the following design hierarchy:

comRoot -> Design -> Component Instance

Use the following API functions to process the Component Instance objects:

ComponentInstanceElaborate

Declaration

```
void ComponentInstanceElaborate(void* compInst);
```

Description

Returns the parameterized values of component instance *compInst* after elaboration.

Notes

The elaborated component instance has the following design hierarchy:

comRoot -> Design -> Component Instance -> Elaborated View of Component Instance

In the Elaborated view of component instance, the parameters and expressions defined in the master component get substituted with the actual value of the parameters given in the instance.

Consider an example where a component C1 has a parameter param1 used in the port definition of Port port1 with LSB=0 and MSB=param1. Also, component C1 is instantiated in a design with value 5 for param1. In the elaborated view of the component instance, if you access the MSB of the port, you will get the value 5 (defined for param1 in the instance).

In the unelaborated view of the component instance, if you access the MSB of the port, you will get the default value of param1 as defined in the master. If the default value is not defined in the master, you will get param1 instead of value 5.

ComponentInstanceGetPartition

Declaration

```
const char* ComponentInstanceGetPartition(void* compInst);
```

Description

Returns the partition name for the component instance *compInst*.

ComponentInstanceGetPowerDomain

Declaration

```
const char* ComponentInstanceGetPowerDomain(void* obj)
```

Description

Returns the power domain information of component instance *obj*.

ComponentInstanceGetTableByName

Declaration

```
void* ComponentInstanceGetTableByName  (void* compInst, const char*
childTableName);
```

Description

Returns the named table *childTableName* of component instance *compInst*.

Note:
> The ComponentInstanceGetTableByName API function works only for tabs present in the GenSys standard schema.

ComponentInstanceGetTableList

Declaration
```
void* ComponentInstanceGetTableList(void* compInst);
```

Description

Returns the list of tables for the component instance *compInst*.

ComponentInstanceGetVoltage

Declaration
```
const char* ComponentInstanceGetVoltage(void* compInst);
```

Description

Returns the voltage value of the component instance *compInst*.

ComponentInstanceUnElaborate

Declaration
```
void ComponentInstanceUnElaborate(void* compInst);
```

Description

Returns the parameterized values of component instance *compInst* before elaboration.

Notes

The unelaborated component instance has the following design hierarchy:

comRoot -> Design -> Component Instance -> Unelaborated View of Component Instance

In the unelaborated view of the component instance, the parameters defined in the master component are not substituted with the actual values given in the instance. Instead, the default value defined in the master component is displayed. If the default value is not defined, the parameters appear as expressions.

ComponentInstGetAdHoc2AdHocConnectionsList

Declaration

```
void* ComponentInstGetAdHoc2AdHocConnectionsList  (void* compInst);
```

Description

Returns the list (*itr*) of Adhoc-to-Adhoc connections for component instance *compInst*.

Use the Adhoc Connection-related API functions to process the Adhoc-to-Adhoc connection objects.

ComponentInstGetAdHoc2TieOffConnectionsList

Declaration

```
void* ComponentInstGetAdHoc2TieOffConnectionsList  (void* compInst);
```

Description

Returns the list (*itr*) of Adhoc-to-Tieoff connections for component instance *compInst*.

Use the Adhoc Connection-related API functions to process the first connection type item of Adhoc-to-Tieoff connection objects. Use the TieOff Connection-related API functions to process the second connection type item of Adhoc-to-Tieoff connection objects.

ComponentInstGetConnectionsList

Declaration

```
void* ComponentInstGetConnectionsList(void* compInst);
```

Description

Returns the list (*itr*) of connections for component instance *compInst*.

ComponentInstGetFunction

Declaration

```
const char* ComponentInstGetFunction(compInst);
```

Description

Returns the function for component instance *compInst*.

ComponentInstGetGeneratedComponentFile

Declaration

```
const char* ComponentInstGetGeneratedComponentFile  (compInst);
```

Description

Returns the generated component file for component instance *compInst*.

ComponentInstGetInterface2InterfaceConnectionList

Declaration

```
void*  ComponentInstGetInterface2InterfaceConnectionList    (void*
compInst);
```

Description

Returns the list (*itr*) of Interface-to-Interface connections for component instance *compInst*.

ComponentInstGetInterfaceInstByName

Declaration

```
void* ComponentInstGetInterfaceInstByName  (void* compInst, const char*
interfaceInstname)
```

Description

Returns the named interface instance *interfaceInstName* for component instance *compInst*.

ComponentInstGetInterfaceInstList

Declaration

```
void* ComponentInstGetInterfaceInstList  (void* compInst);
```

Description

Returns the list (*itr*) of interface instances for component instance *compInst*.

ComponentInstGetIsInternallyGenerated

Declaration

```
int ComponentInstGetIsInternallyGenerated  (void* compInst);
```

Description

Returns 1 if component instance *compInst* is internally generated. Otherwise, returns 0.

ComponentInstGetLogical2LogicalConnectionsList

Declaration

```
void* ComponentInstGetLogical2LogicalConnectionsList  (void* compInst);
```

Description

Returns the list (*itr*) of Logical-to-Logical connections for component instance *compInst*.

ComponentInstGetLogical2TieOffConnectionsList

Declaration

```
void* ComponentInstGetLogical2TieOffConnectionsList  (void* compInst);
```

Description

Returns the list (*itr*) of Logical-to-Tieoff connections for component instance *compInst*.

ComponentInstGetMasterComponent

Declaration
```
void* ComponentInstGetMasterComponent(void* compInst);
```

Description

Returns the master component for component instance *compInst*.

ComponentInstGetName

Declaration
```
const char* ComponentInstGetName(void* compInst);
```

Description

Returns the name of component instance *compInst*.

ComponentInstGetNumberOfInterfaceInst

Declaration
```
int ComponentInstGetNumberOfInterfaceInst  (void* compInst);
```

Description

Returns the number of interface instances for component instance *compInst*.

ComponentInstGetNumberOfTerminals

Declaration
```
int ComponentInstGetNumberOfTerminals(void* compInst);
```

Description

Returns the number of terminals for component instance *compInst*.

ComponentInstGetParameter

Declaration

```
void* ComponentInstGetParameter(void* compInst);
```

Description

Returns the parameter for component instance *compInst*.

ComponentInstGetScope

Declaration

```
void* ComponentInstGetScope(void* compInst);
```

Description

Returns the scope for component instance *compInst*.

ComponentInstGetTerminalByName

Declaration

```
void* ComponentInstGetTerminalByName  (void* compInst, const char*
termName);
```

Description

Returns the named terminal *termName* of component instance *compInst*.

ComponentInstGetTerminalList

Declaration

```
void* ComponentInstGetTerminalList(void* compInst);
```

Description

Returns the list (*itr*) of Terminals for component instance *compInst*.

ComponentInstGetVoltage

Declaration

```
const char* ComponentInstGetVoltage(void* obj)
```

Description

Returns the voltage information of component instance *obj*.

ComponentInstResetParameter

Declaration

```
int ComponentInstResetParameter(void* compInst);
```

Description

Returns the Reset Parameter for component instance *compInst*.

comRoot

The comRoot object is the root of the design hierarchy. Use the following API functions to process the comRoot object:

comRootGetCompInstByName

Declaration

```
void* comRootGetCompInstByName  (void* root, const char* compInstName);
```

Description

Returns the named Component Instance *compInstName* under the comRoot *root*.

Arguments

`root`

The comRoot object.

`compInstName`

Name of the component instance.

Returns

Component instance *compInstName* under the comRoot *root*.

Notes

Use the Component Instance-related API functions to process the component instances.

comRootGetComponentByName

Declaration

```
void* comRootGetComponentByName(root, compName)
```

Description

Returns the named Component *compName* under the comRoot *root*.

Arguments

`root`

THE COMROOT OBJECT.

`compName`

Name of the component.

Returns

Component *compName* under the comRoot *root*.

Notes

1. The comRootGetComponentByName function returns the first-found component with the specified name. Use the comRootGetDesignByName function to fetch the named component of a specific library/vendor/version.
2. Use the Component-related API functions to process the components.

comRootGetComponentByVNLV

Declaration

```
void* comRootGetComponentByVNLV  (void* root, const char* version,  const
char* compName, const char* libName,  const char* vendorName);
```

Description

Returns the named Component *compName* under the comRoot *root* from library *libName* of vendor *vendorName* and of version *version*.

Arguments

`root`

The comRoot object.

`version`

Version number.

`compName`

Name of the component.

`libName`

Library name.

`vendorName`

Vendor name.

Returns

Component *compName* under the comRoot *root* from library *libName* of vendor *vendorName* and of version *version*.

Notes

Use the Component-related API functions to process the components.

comRootGetComponentList

Declaration

```
void* comRootGetComponentList(void* root);
```

Description

Returns the list (*itr*) of Components under the comRoot *root*.

Arguments

`root`

The comRoot object.

Returns

List (*itr*) of components under the comRoot *root*.

Notes

Use the Component-related API functions to process the components.

comRootGetDesignByName

Declaration

```
void* comRootGetDesignByName(root, desName);
```

Description

Returns the named Design *desName* under the comRoot *root*.

Arguments

```
root
```

The comRoot object.

Returns

Design *desName* under the comRoot *root*.

Notes

Use the Design-related API functions to process the designs.

comRootGetDesignByVNLV

Declaration

```
void* comRootGetDesignByVNLV  (void* root, const char* version,  const
char* desName, const char* libName,  const char* vendorName);
```

Description

Returns the named Design *desName* under the comRoot *root* from library *libName* of vendor *vendorName* and of version *version*.

Arguments

`root`

The comRoot object.

`version`

Version number.

`desName`

Name of the design.

`libName`

Library name.

`vendorName`

Vendor name.

Returns

Design *desName* under the comRoot *root* from library *libName* of vendor *vendorName* and of version *version*.

Notes

Use the Design-related API functions to process the designs.

comRootGetDesignList

Declaration

```
void* comRootGetDesignList(void* root);
```

Description

Returns the list (*itr*) of Designs under the comRoot *root*.

Arguments

`root`

The comRoot object.

Returns

List (*itr*) of designs under the comRoot *root*.

Notes

Use the Design-related API functions to process the designs.

comRootGetInterfaceDefByName

Declaration

```
void* comRootGetInterfaceDefByName  (root, interfaceDefName);
```

Description

Returns the named Interface Definition *interfaceDefName* under the comRoot *root*.

Arguments

`root`

The comRoot object.

`interfaceDefName`

Name of the interface definition.

Returns

Interface Definition *interfaceDefName* under the comRoot *root*.

Notes

Use the Interface Definition-related API functions to process the interface definitions.

comRootGetInterfaceDefByVNLV

Declaration

```
void* comRootGetInterfaceDefByVNLV  (void* root, const char*
version,  const char* interfaceDefName, const char* libName,  const char*
vendorName);
```

Description

Returns the named Interface Definition *interfaceDefName* under the comRoot *root* from library *libName* of vendor *vendorName* and of version *version*.

Arguments

`root`

The comRoot object.

`version`

Version number.

`interfaceDefName`

Name of the interface definition.

`libName`

Library name.

`vendorName`

Vendor name.

Returns

Interface Definition *interfaceDefName* under the comRoot *root* from library *libName* of vendor *vendorName* and of version *version*.

Notes

Use the Interface Definition-related API functions to process the interface definitions.

comRootGetInterfaceDefList

Declaration

```
void* comRootGetInterfaceDefList(void* root);
```

Description

Returns the list (*itr*) of Interface Definitions under the comRoot *root*.

Arguments

`root`

The comRoot object.

Returns

List (*itr*) of interface definitions under the comRoot *root*.

Notes

Use the Interface Definition-related API functions to process the interface definitions.

comRootGetIPXACTTableWithVLNV

Declaration

```
comTableClass* comRootGetIPXACTTableWithVLNV(string vnlv);
```

Description

Returns an IP-XACT table of the specified VNLV.

Arguments

`vnlv`

VNLV string

Returns

IP-XACT table of the specified VNLV

comRootSetIPXACTTableWithVLNV

Declaration

```
void comRootSetIPXACTTableWithVLNV(string vnlv, comTableClass* table);
```

Description

Adds an IP-XACT table of the specified VNLV string in the specified COM table.

Arguments

`vnlv`

VNLV string

`table`

comTableClass object

Returns

Nothing

Connection

The Connection object has the following design hierarchy:

comRoot -> Design -> Component Instance -> Connection

comRoot -> Design -> Connection

Use the following API functions to process the Connection objects:

ConnectionGetAlign

Declaration

```
const char* ConnectionGetAlign(void* conn);
```

Description

Returns the align values of connection *conn*.

ConnectionGetBackRef

Declaration

```
const char* ConnectionGetBackRef (void* conn);
```

Description

Returns string containing source of the connection (Tcl file name and line number or perl package/function name).

ConnectionGetCommand

Declaration

```
const char* ConnectionGetCommand (void* conn);
```

Description

Returns string containing the Tcl command for the connection *conn*.

ConnectionGetDeltaDelay

Declaration
```
const char* ConnectionGetDeltaDelay (void* conn);
```

Description

Returns string containing the delta delay information added to the connection *conn*.

ConnectionGetFirstConnectionTypeItem

Declaration
```
void* ConnectionGetFirstConnectionTypeItem  (void* conn);
```

Description

Returns the first connection type item of connection *conn*.

ConnectionGetIsOpen

Declaration
```
int ConnectionGetIsOpen(void* conn);
```

Description

Returns 1 if the given connection *conn* (logical-logical or interface-logical connection) is an open connection. Otherwise, returns 0.

Note:
The ConnectionGetIsOpen API is called on Connection objects returned by DesignGetConnectionList or ComponentInstGetConnectionsList API functions.

ConnectionGetIsTemporary

Declaration
```
int ConnectionGetIsTemporary(void* conn);
```

Description

Returns 1 if the given connection *conn* (tieoff or open connection) is a temporary connection. Otherwise, returns 0.

Note:

The ConnectionGetIsOpen API is called on Connection objects returned by DesignGetConnectionList or ComponentInstGetConnectionsList API functions.

ConnectionGetMoreBackRef

Declaration

```
const char* ConnectionGetMoreBackRef (void* conn);
```

Description

Returns string containing more information about the connection *conn*, if any (added during elaboration).

ConnectionGetPlane

Declaration

```
const char* ConnectionGetPlane(void* conn)
```

Description

Returns the plane name of the connection *conn*.

ConnectionGetSecondConnectionTypeItem

Declaration

```
void* ConnectionGetSecondConnectionTypeItem  (void* conn);
```

Description

Returns the second connection type item of the connection *conn*.

ConnectionIsElaborationGenerated

Declaration

```
int ConnectionIsElaborationGenerated(void* conn);
```

Description

Returns 1 if the given connection *conn* was generated during design connections elaboration. Otherwise, returns 0.

Note:

The ConnectionIsElaborationGenerated API is called on Connection objects returned by DesignGetConnectionList or ComponentInstGetConnectionsList API functions.

ConnectionTypeItemGetType

Declaration

```
const char* ConnectionTypeItemGetType(void* connType);
```

Description

Returns the type for the connection type item *connType*.

Constraints Info

The Constraints Info object has the following design hierarchy:

comRoot -> Component -> Interface -> Logical Port -> Constraints Info

comRoot -> Design -> Interface -> Logical Port -> Constraints Info

Use the following API functions to process the Constraints Info objects:

ConstraintsInfoGetOnMasterInfo

Declaration

```
void * ConstraintsInfoGetOnMasterInfo(void* constraintsInfo )
```

Description

Returns the OnMaster constraints information of the specified object.

ConstraintsInfoGetOnSlaveInfo

Declaration
```
void * ConstraintsInfoGetOnSlaveInfo(void* constraintsInfo )
```

Description

Returns the OnSlave constraints information of the specified object.

ConstraintsInfoGetOnSystemInfoList

Declaration
```
void * ConstraintsInfoGetOnSystemInfoList(void* constraintsInfo )
```

Description

Returns the OnSystem constraints information list of the specified object.

Design

The Design object has the following design hierarchy:

comRoot -> Design

Use the following API functions to process the Design objects:

DesignElaborate

Declaration
```
void DesignElaborate(void* des);
```

Description

Elaborates the design connections into scalar adhoc connections.

Arguments
```
des
```

Design object returned by the comRootGetDesignByName, comRootGetDesignList, comRootGetDesignByVNLV functions.

DesignGetAddressInterfaceList

Declaration

```
void* DesignGetAddressInterfaceList(void* des, const char* req);
```

Description

Returns the elaborated address interface node list for a given requestor, *req*.

Arguments

`des`

Design object returned by the comRootGetDesignByName, comRootGetDesignList, comRootGetDesignByVNLV functions.

`req`

Requestor object

Returns

List of elaborated address interface nodes.

DesignGetAdHoc2AdHocConnectionList

Declaration

```
void* DesignGetAdHoc2AdHocConnectionList(void* des);
```

Description

Returns the list (*itr*) of Adhoc-to-Adhoc connections for design *des*.

Arguments

`des`

Design object returned by the comRootGetDesignByName, comRootGetDesignList, comRootGetDesignByVNLV functions.

Returns

List (*itr*) of Adhoc-to-Adhoc connections under design *des*.

Notes

Use the Adhoc Connection-related API functions to process the Adhoc-to-Adhoc connections.

DesignGetAdHoc2TieOffConnectionList

Declaration

```
void* DesignGetAdHoc2TieOffConnectionList(void* des);
```

Description

Returns the list (*itr*) of Adhoc-to-Tieoff connections for design *des*.

Arguments

`des`

Design object returned by the comRootGetDesignByName, comRootGetDesignList, comRootGetDesignByVNLV functions.

Returns

List (*itr*) of Adhoc-to-Tieoff connections under design *des*.

Notes

Use the Adhoc Connection-related API functions to process the Adhoc Connection item of Adhoc-to-Tieoff connections and the TieOff Connection-related API functions to process the Tieoff Connection item of Adhoc-to-Tieoff connections.

DesignGetClockSrcsList

Declaration

```
void* DesignGetClockSrcsList(void* des);
```

Description

Returns the list (*itr*) of Clock Sources of design *des*.

Arguments

`des`

Design object returned by the comRootGetDesignByName, comRootGetDesignList, comRootGetDesignByVNLV functions.

Returns

List (*itr*) of clock sources under design *des*.

DesignGetCompInstByHierName

Declaration

```
void* DesignGetCompInstByHierName  (void* obj, const char* compInstName);
```

Description

Returns instance object, given its hierarchical name as *compInstName* under the design *obj*.

DesignGetCompInstByName

Declaration

```
void* DesignGetCompInstByName  (void* des, const char* compInstName);
```

Description

Returns the named component instance *compInstName* under design *des*.

Arguments

`des`

Design object returned by the comRootGetDesignByName, comRootGetDesignList, comRootGetDesignByVNLV functions.

`compInstName`

Name of the component instance.

Returns

Component Instance *compInstName* under design *des*.

Notes

Use the Component Instance-related API functions to process the component instances.

DesignGetCompInstByNameAndScope

Declaration

```
void* DesignGetCompInstByNameAndScope  (void* des, const char*
compInstName,   const char* scope);
```

Description

Returns the named Component Instance *compInstName* under design *des* for scope *scope*.

Arguments

`des`

Design object returned by the comRootGetDesignByName, comRootGetDesignList, comRootGetDesignByVNLV functions.

`compInstName`

Name of the component instance.

`scope`

Name of the scope.

Returns

Component Instance *compInstName* under design *des* for scope *scope*.

Notes

Use the Component Instance-related API functions to process the component instances.

DesignGetCompInstList

Declaration

```
void* DesignGetCompInstList(void* des);
```

Description

Returns the list (*itr*) of Component Instances for design *des*.

Arguments
`des`

> Design object returned by the comRootGetDesignByName, comRootGetDesignList, comRootGetDesignByVNLV functions.

Returns

List (*itr*) of component instances for design *des*.

Notes

Use the Component Instance-related API functions to process the component instances.

DesignGetConnectionList

Declaration
```
void* DesignGetConnectionList(void* des);
```

Description

Returns the list (*itr*) of Connections for design *des*.

Arguments
`des`

> Design object returned by the comRootGetDesignByName, comRootGetDesignList, comRootGetDesignByVNLV functions.

Returns

List (*itr*) of connections for design *des*.

Notes

Use the Connection-related API functions to process the connections.

DesignGetDesignList

Declaration
```
void* DesignGetDesignList(void* des);
```

Description

Returns the list (*itr*) of designs instantiated under design *des*.

Arguments
`des`

> Design object returned by the comRootGetDesignByName, comRootGetDesignList, comRootGetDesignByVNLV functions.

Returns

List (*itr*) of designs instantiated under design *des*.

DesignGetDesViewLibrary

Declaration
```
const char * DesignGetDesViewLibrary(void* des)
```

Description

Returns the library of a design view object.

DesignGetDesViewName

Declaration
```
const char * DesignGetDesViewName(void* des)
```

Description

Returns the name of a design view object.

DesignGetDesViewVendor

Declaration
```
const char * DesignGetDesViewVendor(void* des)
```

Description

Returns the vendor of a design view object.

DesignGetDesViewVersion

Declaration

```
const char * DesignGetDesViewVersion(void* des)
```

Description

Returns the version of a design view object.

DesignGetDesViewVersionedName

Declaration

```
void * DesignGetDesViewVersionedName(void* des)
```

Description

Returns a version identifier for a design view object.

DesignGetInterface2InterfaceConnectionList

Declaration

```
void* DesignGetInterface2InterfaceConnectionList  (void* des);
```

Description

Returns the list (*itr*) of Interface-to-Interface connections under design *des*.

Arguments

`des`

Design object returned by the comRootGetDesignByName, comRootGetDesignList, comRootGetDesignByVNLV functions.

Returns

List (*itr*) of Interface-to-Interface connections under design *des*.

Notes

Use the Interface Connection-related API functions to process the Interface-to-Interface connections.

DesignGetInterfaceByName

Declaration

```
void* DesignGetInterfaceByName  (void* des, const char*
interfaceInstName);
```

Description

Returns the named Interface Instance *interfaceInstName* under design *des*.

Arguments

`des`

Design object returned by the comRootGetDesignByName, comRootGetDesignList, comRootGetDesignByVNLV functions.

`interfaceInstName`

Name of the interface instance.

Returns

Interface Instance *interfaceInstName* under design *des*.

Notes

Use the Interface Instance-related API functions to process the interface instances.

DesignGetInterfaceList

Declaration

```
void* DesignGetInterfaceList(void* des);
```

Description

Returns the list (*itr*) of interfaces for design *des*.

Arguments

`des`

Design object returned by the comRootGetDesignByName, comRootGetDesignList, comRootGetDesignByVNLV functions.

Returns

List (*itr*) of Interfaces for design *des*.

Notes

Use the Interface-related API functions to process the interfaces.

DesignGetLogical2LogicalConnectionList

Declaration

```
void* DesignGetLogical2LogicalConnectionList  (void* des);
```

Description

Returns the list (*itr*) of Logical-to-Logical connections for design *des*.

Arguments

`des`

Design object returned by the comRootGetDesignByName, comRootGetDesignList, comRootGetDesignByVNLV functions.

Returns

List (*itr*) of Logical-to-Logical connections under design *des*.

Notes

Use the Logical Connection-related API functions to process the Logical-to-Logical connections.

DesignGetLogical2TieOffConnectionList

Declaration

```
void* DesignGetLogical2TieOffConnectionList  (void* des);
```

Description

Returns the list (*itr*) of Logical-to-Tieoff connections for design *des*.

Arguments

`des`

Design object returned by the comRootGetDesignByName, comRootGetDesignList, comRootGetDesignByVNLV functions.

Returns

List (*itr*) of Logical-to-Tieoff connections under design *des*.

Notes

Use the Logical Connection-related API functions to process the Logical Connection item of Logical-to-Tieoff connections and the TieOff Connection-related API functions to process the Tieoff Connection item of Logical-to-Tieoff connections.

DesignGetLogicalPortByName

Declaration

```
void* DesignGetLogicalPortByName  (void* des, const char* lportName);
```

Description

Returns the named Logical Port *lportName* under design *des*.

Arguments

`des`

Design object returned by the comRootGetDesignByName, comRootGetDesignList, comRootGetDesignByVNLV functions.

`lportName`

Name of the logical port.

Returns

Logical Port *lportName* of design *des*.

Notes

Use the Logical Port-related API functions to process the design logical ports.

DesignGetLogicalPortList

Declaration

```
void* DesignGetLogicalPortList(void* des);
```

Description

Returns the list (*itr*) of Logical Ports of design *des*.

Arguments

`des`

Design object returned by the comRootGetDesignByName, comRootGetDesignList, comRootGetDesignByVNLV functions.

Returns

List (*itr*) of logical ports of design *des*.

Notes

Use the Logical Port-related API functions to process the design logical ports.

DesignGetMRSXML

Declaration

```
const char* DesignGetMRSXML(void* des);
```

Description

Returns the name of the source MRS file for design *des*.

Arguments

`des`

Design object returned by the comRootGetDesignByName, comRootGetDesignList, comRootGetDesignByVNLV functions.

Returns

Name of the MRS file for design *des*.

DesignGetPortByName

Declaration

```
void* DesignGetPortByName  (void* des, const char* portName);
```

Description

Returns the named Port *portName* under design *des*.

Arguments

`des`

Design object returned by the comRootGetDesignByName, comRootGetDesignList, comRootGetDesignByVNLV functions.

`portName`

Name of the port.

Returns

Port *portName* of design *des*.

Notes

Use the Port-related API functions to process the design ports.

DesignGetPortList

Declaration

```
void* DesignGetPortList(void* des);
```

Description

Returns the list (*itr*) of Ports of design *des*.

Arguments

`des`

Design object returned by the comRootGetDesignByName, comRootGetDesignList, comRootGetDesignByVNLV functions.

Returns

List (*itr*) of ports of design *des*.

Notes

Use the Port-related API functions to process the design ports.

DesignGetRequestorList

Declaration

```
void* DesignGetRequestorList(void* des);
```

Description

Returns the list of requestor instance/interface names on the give design *des*.

Arguments

`des`

Design object returned by the comRootGetDesignByName, comRootGetDesignList, comRootGetDesignByVNLV functions.

DesignGetResetSrcsList

Declaration

```
void* DesignGetResetSrcsList(void* des);
```

Description

Returns the list (*itr*) of Reset Sources of design *des*.

Arguments

`des`

Design object returned by the comRootGetDesignByName, comRootGetDesignList, comRootGetDesignByVNLV functions.

Returns

List (*itr*) of reset sources under design *des*.

DesignGetRTLFilesSet

Declaration

```
void * DesignGetRTLFilesSet(void* obj)
```

Description

Returns iterator to a list of all RTL files created for each unique partition set.

DesignGetSocName

Declaration

```
void* DesignGetSocName(void* des);
```

Description

Returns the SoC name (Versioned Name) for design *des*.

Arguments

`des`

Design object returned by the comRootGetDesignByName, comRootGetDesignList, comRootGetDesignByVNLV functions.

Returns

SOC Name (vnlv) of design *des*.

DesignGetSocNameinVersion

Declaration

```
const char* DesignGetSocNameinVersion(void* des);
```

Description

Returns the Versioned Soc name for design *des*.

Arguments

`des`

Design object returned by the comRootGetDesignByName, comRootGetDesignList, comRootGetDesignByVNLV functions.

Returns

Versioned SOC Name (string) of design *des*.

Notes

Use the VersionedNameClass-related API functions to process the Versioned Names.

DesignGetSpaceConnectionList

Declaration

```
void* DesignGetSpaceConnectionList(void* des);
```

Description

Returns the list (*itr*) of Space Connections of design *des*.

Arguments

`des`

Design object returned by the comRootGetDesignByName, comRootGetDesignList, comRootGetDesignByVNLV functions.

Returns

List (*itr*) of space connections under design *des*.

DesignGetTableByName

Declaration

```
void* DesignGetTableByName  (void* des, const char* childtableName);
```

Description

Returns the named table *tableName* of design *des*.

Note:

The DesignGetTableByTable API function works for only the tabs present in the GenSys standard schema.

Arguments

`des`

Design object returned by the comRootGetDesignByName, comRootGetDesignList, comRootGetDesignByVNLV functions.

`childtableName`

Name of the table to be fetched.

Returns

Table *childtableName* of design *des*.

DesignGetTableList

Declaration

```
void* DesignGetTableList(void* des);
```

Description

Returns the list (*itr*) of tables in design *des*.

Note:

The DesignGetTableList API function works for only the tabs present in the GenSys standard schema.

Arguments

`des`

Design object returned by the comRootGetDesignByName, comRootGetDesignList, comRootGetDesignByVNLV functions.

Returns

List (*itr*) of tables of design *des*.

DesignUnElaborate

Declaration

```
void DesignUnElaborate(void* des);
```

Description

Un elaborates the design connections.

Arguments

`des`

Design object returned by the comRootGetDesignByName, comRootGetDesignList, comRootGetDesignByVNLV functions.

EnumTokenDefn

The EnumTokenDefn object has the following design hierarchy:

comRoot -> Component -> BitEnumDefn -> EnumTokenDefn

Use the following API functions to process the EnumTokenDefn objects:

EnumTokenDefnGetMnemonic

Declaration

```
char* EnumTokenDefnGetMnemonic(void* EnumTokenDefn);
```

Description

Returns the mnemonic information of the *EnumTokenDefn* object returned by the BitEnumDefnGetEnumTokenDefnByName or BitEnumDefnGetEnumTokenDefnList function.

EnumTokenDefnGetRDLName

Declaration

```
char* EnumTokenDefnGetRDLName(void* EnumTokenDefn);
```

Description

Returns the RDL name of the *EnumTokenDefn* object returned by the BitEnumDefnGetEnumTokenDefnByName or BitEnumDefnGetEnumTokenDefnList function.

EnumTokenDefnGetValue

Declaration
```
char* EnumTokenDefnGetValue(void* EnumTokenDefn);
```

Description

Returns the value of the *EnumTokenDefn* object returned by the BitEnumDefnGetEnumTokenDefnByName or BitEnumDefnGetEnumTokenDefnList function.

Interface Connection

The Interface Connection Type Item object has the following design hierarchy:

comRoot -> Design -> Component Instance -> Connection -> Interface Connection

comRoot -> Design -> Connection -> Interface Connection

Use the following API functions to process the Interface Connection Type Item objects:

InterfaceConnectionGetAdHocConnList

Declaration
```
void* InterfaceConnectionGetAdHocConnList  (void* iconn);
```

Description

Returns the list (*itr*) of Adhoc Connections for Interface Connection *iconn*.

Arguments
```
iconn
```

Interface Connection object returned by the ComponentInstGetInterface2InterfaceConnectionList function or the DesignGetPortList function.

Returns

List (*itr*) of Adhoc Connections for Interface Connection *iconn*.

InterfaceConnectionGetCompInst

Declaration

```
void* InterfaceConnectionGetCompInst(void* iconn);
```

Description

Returns the Component Instance for Interface Connection *iconn*.

Arguments

`iconn`

Interface Connection object returned by the ComponentInstGetInterface2InterfaceConnectionList function or the DesignGetPortList function.

Returns

Component Instance for Interface Connection *iconn*.

Notes

Use the Component Instance-related function to process the component instances.

InterfaceConnectionGetInterfaceInst

Declaration

```
void* InterfaceConnectionGetInterfaceInst  (void* iconn);
```

Description

Returns the Interface Instance for Interface Connection *iconn*.

Arguments

`iconn`

Interface Connection object returned by the ComponentInstGetInterface2InterfaceConnectionList function or the DesignGetPortList function.

Returns

Interface Instance for Interface Connection *iconn*.

Notes

Use the Interface Instance-related function to process the interface instances.

InterfaceConnectionGetMSB

Declaration

```
const char* InterfaceConnectionGetMSB(void* iconn);
```

Description

Returns the MSB of Interface Connection *iconn*.

Arguments

`iconn`

Interface Connection object returned by the ComponentInstGetInterface2InterfaceConnectionList function or the DesignGetPortList function.

Returns

MSB of Interface Connection *iconn*.

InterfaceConnectionGetLSB

Declaration

```
const char* InterfaceConnectionGetLSB(void* iconn);
```

Description

Returns the LSB of Interface Connection *iconn*.

Arguments

`iconn`

Interface Connection object returned by the ComponentInstGetInterface2InterfaceConnectionList function or the DesignGetPortList function.

Returns

LSB of Interface Connection *iconn*.

InterfaceConnectionGetSplices

Declaration

```
void* InterfaceConnectionGetSplices(void* firstconn);
```

Description

Returns the splice ports for the Interface Connection for which the first side of the connection is *firstconn*.

Arguments

`firstconn`

First side of the interface connection returned by the ConnectionGetFirstConnectionTypeItem function.

Returns

Null if the connection is a pure interface connection, otherwise returns an iterator to a list of the splice ports for the interface connection.

InterfaceConnectionGetSpliceTieoffs

Declaration

void* InterfaceConnectionGetSpliceTieoffs
  (void* iconn);

Description

Returns the splice tieoff values for the Interface Connection *iconn*.

Arguments

`iconn`

Interface Connection object returned by the ComponentInstGetInterface2InterfaceConnectionList function or the DesignGetPortList function.

Returns

Splice tieoff values for the Interface Connection *iconn*.

InterfaceConnectionGetUniquefier

Declaration

```
const char* InterfaceConnectionGetUniquefier  (void* iconn);
```

Description

Returns the Uniquefier name for Interface Connection *iconn*.

Arguments

`iconn`

> Interface Connection object returned by the ComponentInstGetInterface2InterfaceConnectionList function or the DesignGetPortList function.

Returns

Uniquefier name for Interface Connection *iconn*.

InterfaceConnectionIsExport

Declaration

```
int InterfaceConnectionIsExport(void* iconn);
```

Description

Returns 1 if Interface Connection *iconn* is an exported interface connection. Otherwise, returns 0.

Arguments

`iconn`

> Interface Connection object returned by the ComponentInstGetInterface2InterfaceConnectionList function or the DesignGetPortList function.

Returns

1 if Interface Connection *iconn* is an exported interface connection. Otherwise, returns 0.

Interface

The Interface object has the following design hierarchy:

comRoot -> Design -> Interface

comRoot -> Component -> Interface

Use the following API functions to process the Interface objects:

InterfaceGetConnectionRequired

Declaration

```
bool InterfaceGetConnectionRequired(void* interface)
```

Description

Returns 0 or 1 to specify if a connection is required.

Arguments

`interface`

Interface object returned by the DesignGetInterfaceByName/DesignGetInterfaceList or ComponentGetInterfaceByName/ComponentGetInterfaceList function.

InterfaceGetControlClock

Declaration

```
void* InterfaceGetControlClock(void* interface);
```

Description

Returns the Control Clock for interface *interface*.

Arguments

`interface`

Interface object returned by the DesignGetInterfaceByName/DesignGetInterfaceList or ComponentGetInterfaceByName/ComponentGetInterfaceList function.

Returns

Control Clock for interface *interface*.

InterfaceGetEndianness

Declaration
```
const char * InterfaceGetEndianness  (void* interface);
```

Description

Returns the endianness of the interface.

Arguments
```
interface
```

Interface object returned by the DesignGetInterfaceByName/DesignGetInterfaceList or ComponentGetInterfaceByName/ComponentGetInterfaceList function.

InterfaceGetIsMirrored

Declaration
```
const char* InterfaceGetIsMirrored(void* interface);
```

Description

Returns a string (YES/NO/MONITOR) according to the mirror type of the interface *interface*.

When a mirrored interface is instantiated in a component or a design, the directions are reversed.

Arguments
```
interface
```

Interface object returned by the DesignGetInterfaceByName/DesignGetInterfaceList or ComponentGetInterfaceByName/ComponentGetInterfaceList function.

Returns

Returns YES/NO/MONITOR according to the mirror type of the interface.

InterfaceGetLocalLogicalPortByName

Declaration

```
void* InterfaceGetLocalLogicalPortByName  (void* interface, const char*
lportName);
```

Description

Returns the named local logical port *lportname* for interface *interface*.

Arguments

`interface`

Interface object returned by the DesignGetInterfaceByName/DesignGetInterfaceList or ComponentGetInterfaceByName/ComponentGetInterfaceList function.

`lportName`

Name of the local logical port.

Returns

Named local logical port *lportname* for interface *interface*.

Notes

1. Use the DesignGetInterfaceByName/DesignGetInterfaceList or ComponentGetInterfaceByName/ComponentGetInterfaceList function to check whether the logical port is a local port.
2. Use the Logical Port-related API functions to process logical ports.

InterfaceGetLocalLogicalPortList

Declaration

```
void* InterfaceGetLocalLogicalPortList  (void* interface);
```

Description

Returns the list (*itr*) of local logical ports for interface *interface*.

Arguments

`interface`

Interface object returned by the DesignGetInterfaceByName/DesignGetInterfaceList or ComponentGetInterfaceByName/ComponentGetInterfaceList function.

Returns

List (*itr*) of local logical ports for interface *interface*.

Notes

1. Use the InterfaceGetLogicalPortList function to get the list of logical ports for the interface and the InterfaceLogicalPortIsLocal function to check whether the logical port is a local port.
2. Use the Logical Port-related API functions to process logical ports.

InterfaceGetLogicalPortByName

Declaration

```
void* InterfaceGetLogicalPortByName  (interface, lportName);
```

Description

Returns the named logical port *lportName* (local logical port or Interface Definition's logical port) for interface *interface*.

Arguments

`interface`

Interface object returned by the DesignGetInterfaceByName/DesignGetInterfaceList or ComponentGetInterfaceByName/ComponentGetInterfaceList function.

`lportName`

Name of the logical port.

Returns

Named logical port *lportname* for interface *interface*.

Notes

Use the Logical Port-related API functions to process logical ports.

InterfaceGetLogicalPortList

Declaration

```
void* InterfaceGetLogicalPortList(void* interface);
```

Description

Returns the list (*itr*) of all logical ports (local logical ports and Interface Definition's logical ports) for interface *interface*.

Arguments

`interface`

> Interface object returned by the DesignGetInterfaceByName/DesignGetInterfaceList or ComponentGetInterfaceByName/ComponentGetInterfaceList function.

Returns

List (*itr*) of all logical ports (local logical ports and Interface Definition's logical ports) for interface *interface*.

Notes

Use the Logical Port-related API functions to process logical ports.

InterfaceGetName

Declaration

```
const char* InterfaceGetName(void* interface);
```

Description

Returns the name of interface *interface*.

Arguments

`interface`

> Interface object returned by the DesignGetInterfaceByName/DesignGetInterfaceList or ComponentGetInterfaceByName/ComponentGetInterfaceList function.

Returns

Name of interface *interface*.

InterfaceGetParentInterfaceDef

Declaration

```
void* InterfaceGetParentInterfaceDef(void* interface);
```

Description

Returns the Parent Interface Definition for interface *interface*.

Arguments

`interface`

Interface object returned by the DesignGetInterfaceByName/DesignGetInterfaceList or ComponentGetInterfaceByName/ComponentGetInterfaceList function.

Returns

Parent Interface Definition for interface *interface*.

Notes

Use the Interface Definition-related API functions to process interface definitions.

InterfaceGetPortMapList

Declaration

```
void* InterfaceGetPortMapList(void* interface);
```

Description

Returns the list (*itr*) of Interface Portmap items for interface *interface*.

Arguments

`interface`

Interface object returned by the DesignGetInterfaceByName/DesignGetInterfaceList or ComponentGetInterfaceByName/ComponentGetInterfaceList function.

Returns

List (*itr*) of Interface Portmap items for interface *interface*.

Notes

Use the PortMap-related API functions to process logical ports.

InterfaceGetPowerDomain

Declaration

```
const char* InterfaceGetPowerDomain(void* obj)
```

Description

Returns the power domain information of interface *obj*.

Arguments

`obj`

Interface object returned by the DesignGetInterfaceByName/DesignGetInterfaceList or ComponentGetInterfaceByName/ComponentGetInterfaceList function.

Returns

Power domain of interface *obj*.

InterfaceGetType

Declaration

```
const char* InterfaceGetType(void* interface);
```

Description

Returns the type (Master or Slave) of interface *interface*.

Arguments

`interface`

Interface object returned by the DesignGetInterfaceByName/DesignGetInterfaceList or ComponentGetInterfaceByName/ComponentGetInterfaceList function.

Returns

Type (MASTER or SLAVE) of interface *interface*.

InterfaceGetVoltage

Declaration

```
const char* InterfaceGetVoltage(void* obj)
```

Description

Returns the voltage information of interface *obj*.

Arguments

`obj`

Interface object returned by the DesignGetInterfaceByName/DesignGetInterfaceList or ComponentGetInterfaceByName/ComponentGetInterfaceList function.

Returns

Voltage of the interface *obj*.

InterfaceLogicalPortIsLocal

Declaration

```
int InterfaceLogicalPortIsLocal  (void* interface, void* lport);
```

Description

Returns 1 if logical port *lport* is a local logical port of interface *interface*. Otherwise, returns 0.

Arguments

`interface`

Interface object returned by the DesignGetInterfaceByName/DesignGetInterfaceList or ComponentGetInterfaceByName/ComponentGetInterfaceList function.

`lport`

Logical port returned by the InterfaceGetLogicalPortByName/ InterfaceGetLogicalPortList function.

Returns

1 if logical port *lport* is a local logical port of interface *interface*. Otherwise, returns 0.

Interface Definition

The Interface Definition object has the following design hierarchy:

comRoot -> Interface Definition

Use the following API functions to process the Interface Definition objects:

InterfaceDefGetLogicalPortByName

Declaration

```
void* InterfaceDefGetLogicalPortByName  (void* interfaceDef, const char*
lportName);
```

Description

Returns the named Logical Port *lportName* of Interface Definition *interfaceDef*.

Arguments

`interfaceDef`

Interface Definition object returned by the comRootGetInterfaceDefByName/ comRootGetInterfaceDefList/comRootGetInterfaceDefByVNLV function.

`lportName`

Name of the logical port.

Returns

Named logical port *lportname* for Interface Definition *interfaceDef*.

Notes

Use the Logical Port-related API functions to process logical ports.

InterfaceDefGetLogicalPortList

Declaration

```
void* InterfaceDefGetLogicalPortList  (void* interfaceDef);
```

Description

Returns the list (*itr*) of Logical Ports of Interface Definition *interfaceDef*.

Arguments

`interfaceDef`

Interface Definition object returned by the comRootGetInterfaceDefByName/ comRootGetInterfaceDefList/comRootGetInterfaceDefByVNLV function.

Returns

List (*itr*) of Logical Ports of Interface Definition *interfaceDef*.

Notes

Use the Logical Port-related API functions to process logical ports.

InterfaceDefGetMRSXML

Declaration

```
const char* InterfaceDefGetMRSXML(void* interfaceDef);
```

Description

Returns the Source MRS file name for Interface Definition *interfaceDef*.

Arguments

`interfaceDef`

Interface Definition object returned by the comRootGetInterfaceDefByName/ comRootGetInterfaceDefList/comRootGetInterfaceDefByVNLV function.

Returns

Name of the Source MRS file for Interface Definition *interfaceDef*.

InterfaceDefGetName

Declaration

```
const char* InterfaceDefGetName(void* interfaceDef);
```

Description

Returns the name of Interface Definition *interfaceDef*.

Arguments

`interfaceDef`

Interface Definition object returned by the comRootGetInterfaceDefByName/ comRootGetInterfaceDefList/comRootGetInterfaceDefByVNLV function.

Returns

Name of Interface Definition *interfaceDef*.

InterfaceDefGetTableByName

Declaration

```
void* InterfaceDefGetTableByName  (void* interfaceDef, const char*
childTableName);
```

Description

Returns the named table *childTableName* (from the interface library) for Interface Definition *interfaceDef*.

Note:
The InterfaceDefGetTableByName API function works for only the tabs present in the GenSys standard schema.

Arguments

`interfaceDef`

Interface Definition object returned by the comRootGetInterfaceDefByName/ comRootGetInterfaceDefList/comRootGetInterfaceDefByVNLV function.

`childtableName`

Name of the table to be fetched.

Returns

Table *childtableName* of Interface Definition *interfaceDef*.

InterfaceDefGetTableList

Declaration

```
void* InterfaceDefGetTableList(void* interfaceDef);
```

Description

Returns the list (*itr*) of tables (from the interface library) for Interface Definition *interfaceDef*.

Note:
The InterfaceDefGetTableList API function works for only the tabs present in the GenSys standard schema.

Arguments

`interfaceDef`

> Interface Definition object returned by the comRootGetInterfaceDefByName/ comRootGetInterfaceDefList/comRootGetInterfaceDefByVNLV function.

Returns

List (*itr*) of Tables of Interface Definition *interfaceDef*.

InterfaceDefGetVersionedName

Declaration

```
void* InterfaceDefGetVersionedName  (void* interfaceDef);
```

Description

Returns the versioned name for Interface Definition *interfaceDef*.

Arguments

`interfaceDef`

> Interface Definition object returned by the comRootGetInterfaceDefByName/ comRootGetInterfaceDefList/comRootGetInterfaceDefByVNLV function.

Returns

Versioned Name (*vnlv*) of Interface Definition *interfaceDef*.

Notes

Use the VersionedNameClass-related API functions to process the Versioned Name objects.

Interface Instance

The Interface Instance object has the following design hierarchy:

comRoot -> Design -> Component Instance -> Connection -> Interface Connection -> Interface Instance

comRoot -> Design -> Connection -> Interface Connection -> Interface Instance

Use the following API functions to process the Interface Instance objects:

InterfaceInstGetCompInst

Declaration

```
void* InterfaceInstGetCompInst(void* interfaceInst);
```

Description

Returns the Component Instance for Interface Instance *interfaceInst*.

Arguments

```
interfaceInst
```

Interface Instance object returned by the InterfaceConnectionGetInterfaceInst function.

Returns

Component Instance for Interface Instance *interfaceInst*.

Notes

Use the Component Instance-related API functions to process the component instance objects.

InterfaceInstGetConnectionList

Declaration

```
void* InterfaceInstGetConnectionList  (void* interfaceInst);
```

Description

Returns the list (*itr*) of Connections of Interface Instance *interfaceInst*.

Arguments

```
interfaceInst
```

Interface Instance object returned by the InterfaceConnectionGetInterfaceInst function.

Returns

List (*itr*) of Connections of Interface Instance *interfaceInst*.

Notes

Use the Connection-related API functions to process the connection objects.

InterfaceInstGetInterface

Declaration

```
void* InterfaceInstGetInterface(void* interfaceInst);
```

Description

Returns the Interface for Interface Instance *interfaceInst*.

InterfaceInstGetTerminalByName

Declaration

```
void* InterfaceInstGetTerminalByName  (void* interfaceInst, const char*
termName);
```

Description

Returns the named terminal *termName* of Interface Instance *interfaceInst*.

Arguments

`interfaceInst`

Interface Instance object returned by the InterfaceConnectionGetInterfaceInst function.

`termName`

Name of the terminal.

Returns

Named terminal *termName* of of Interface Instance *interfaceInst*.

Notes

Use the Terminal-related API functions to process the terminal objects.

InterfaceInstGetTerminalList

Declaration

```
void* InterfaceInstGetTerminalList  (void* interfaceInst);
```

Description

Returns the list (*itr*) of Terminals of Interface Instance *interfaceInst*.

Arguments

`interfaceInst`

Interface Instance object returned by the InterfaceConnectionGetInterfaceInst function.

Returns

List (*itr*) of Terminals of Interface Instance *interfaceInst*.

Notes

Use the Terminal-related API functions to process the terminal objects.

InterfaceInstIsExported

Declaration

```
int InterfaceInstIsExported(void* interfaceInst);
```

Description

Returns 1 if Interface Instance *interfaceInst* is the instance of an exported interface. Otherwise, returns 0.

Arguments

`interfaceInst`

Interface Instance object returned by the InterfaceConnectionGetInterfaceInst function.

Returns

1 if Interface Instance *interfaceInst* is the instance of an exported interface. Otherwise, returns 0.

COM Interface Library

The COM Interface Library object has the following design hierarchy:

comRoot -> COM Interface Library

Use the following API functions to process the COM Interface Library objects:

InterfaceLibGetBusDefVLNV

Declaration
`void* InterfaceLibGetBusDefVLNV(void* comInterfaceLib)`

Description

Returns the VLNV information of the bus definition.

Arguments
`comInterfaceLib`

COM Interface Library object

InterfaceLibGetDirectionConnection

Declaration
`bool InterfaceLibGetDirectionConnection(void* comInterfaceLib)`

Description

Returns 0 or 1 to specify if interfaces should be connected directly.

Arguments
`comInterfaceLib`

COM Interface Library object

InterfaceLibGetIsAddressable

Declaration
`bool InterfaceLibGetIsAddressable(void* comInterfaceLib)`

Description

Returns 0 or 1 to specify the *IsAddressable* value.

Arguments
`comInterfaceLib`

COM Interface Library object

InterfaceLibGetMaxMasters

Declaration

```
int InterfaceLibGetMaxMasters(void* comInterfaceLib)
```

Description

Returns the maximum number of masters of an interface library.

Arguments

`comInterfaceLib`

COM Interface Library object

InterfaceLibGetMaxSlaves

Declaration

```
int InterfaceLibGetMaxSlaves(void* comInterfaceLib)
```

Description

Returns the maximum number of slaves of an interface library.

Arguments

`comInterfaceLib`

COM Interface Library object

InterfaceLibGetSystemGroupNameList

Declaration

```
void* InterfaceLibGetSystemGroupNameList(void* comInterfaceLib)
```

Description

Returns the system group name list.

Arguments

`comInterfaceLib`

COM Interface Library object

Logical Connection

The Logical Connection Type Item object has the following design hierarchy:

comRoot -> Design -> Component Instance -> Connection -> Logical Connection

comRoot -> Design -> Connection -> Logical Connection

Use the following API functions to process the Logical Connection Type Item objects:

LogicalConnectionGetCompInst

Declaration

```
void* LogicalConnectionGetCompInst(void* lconn);
```

Description

Returns the Component Instance for Logical Connection *lconn*.

Arguments

`lconn`

Logical Connection item returned by the ComponentInstGetLogical2LogicalConnectionsList/ ComponentInstGetLogical2TieOffConnectionsList function or the DesignGetPortList/ DesignGetLogical2TieOffConnectionList function.

Returns

Component Instance for Logical Connection *lconn*.

Notes

Use the Component Instance-related API functions to process the component instance objects.

LogicalConnectionGetAdHocConnectionRef

Declaration

```
void* LogicalConnectionGetAdHocConnectionRef  (void* lconn);
```

Description

Returns the Adhoc Connection reference for Logical Connection *lconn*.

Arguments

`lconn`

Logical Connection item returned by the ComponentInstGetLogical2LogicalConnectionsList/ ComponentInstGetLogical2TieOffConnectionsList function or the DesignGetPortList/ DesignGetLogical2TieOffConnectionList function.

Returns

Adhoc Connection reference for Logical Connection *lconn*.

Notes

Use the Adhoc Connection-related API functions to process the Adhoc connection objects.

LogicalConnectionGetLogicalPortRef

Declaration

```
void* LogicalConnectionGetLogicalPortRef(lconn);
```

Description

Returns the Logical Port reference for Logical Connection *lconn*.

Arguments

`lconn`

Logical Connection item returned by the ComponentInstGetLogical2LogicalConnectionsList/ ComponentInstGetLogical2TieOffConnectionsList function or the DesignGetPortList/ DesignGetLogical2TieOffConnectionList function.

Returns

Logical Port Reference for Logical Connection *lconn*.

Notes

Use the Logical Port-related API functions to process the logical port objects.

LogicalConnectionGetLSB

Declaration

```
const char* LogicalConnectionGetLSB(void* lconn);
```

Description

Returns the LSB of Logical Connection *lconn*.

Arguments

`lconn`

Logical Connection item returned by the ComponentInstGetLogical2LogicalConnectionsList/ ComponentInstGetLogical2TieOffConnectionsList function or the DesignGetPortList/ DesignGetLogical2TieOffConnectionList function.

Returns

LSB of Logical Connection *lconn*.

LogicalConnectionGetMSB

Declaration

```
const char* LogicalConnectionGetMSB(void* lconn);
```

Description

Returns the MSB of Logical Connection *lconn*.

Arguments

`lconn`

Logical Connection item returned by the ComponentInstGetLogical2LogicalConnectionsList/ ComponentInstGetLogical2TieOffConnectionsList function or the DesignGetPortList/ DesignGetLogical2TieOffConnectionList function.

Returns

MSB of Logical Connection *lconn*.

LogicalConnectionGetPort

Declaration

```
void* LogicalConnectionGetPort(void* lconn);
```

Description

Returns the Port for Logical Connection *lconn*.

Arguments

`lconn`

Logical Connection item returned by the ComponentInstGetLogical2LogicalConnectionsList/ ComponentInstGetLogical2TieOffConnectionsList function or the DesignGetPortList/ DesignGetLogical2TieOffConnectionList function.

Returns

Port for Logical Connection *lconn*.

Notes

Use the Port-related API functions to process the port objects.

LogicalConnectionGetSignalType

Declaration

```
const char* LogicalConnectionGetSignalType  (void* lconn);
```

Description

Returns the Signal type (clock, reset, event, or others) of Logical Connection *lconn*.

Arguments

`lconn`

Logical Connection item returned by the ComponentInstGetLogical2LogicalConnectionsList/ ComponentInstGetLogical2TieOffConnectionsList function or the DesignGetPortList/ DesignGetLogical2TieOffConnectionList function.

Returns

Signal type of Logical Connection *lconn* as one of the following values:

CLK	RST	EVT	OTHERS	UNDEFPORT

LogicalConnectionPortIsLogical

Declaration

```
int LogicalConnectionPortIsLogical(void* lconn);
```

Description

Returns 1 if the port referenced by Logical Connection *lconn* is a logical port. Otherwise, returns 0.

Arguments

`lconn`

Logical Connection item returned by the ComponentInstGetLogical2LogicalConnectionsList/ ComponentInstGetLogical2TieOffConnectionsList function or the DesignGetPortList/ DesignGetLogical2TieOffConnectionList function.

Returns

1 if the port referenced by Logical Connection *lconn* is a logical port. Otherwise, returns 0.

Logical Port

The Logical Port object has the following design hierarchy:

comRoot -> Design -> Logical Port

comRoot -> Component -> Logical Port

comRoot -> Interface Definition -> Logical Port

Use the following API functions to process the Logical Port objects:

LogicalPortGetActualDir

Declaration

```
const char* LogicalPortGetActualDir(void* lport);
```

Description

Returns the Direction (in, out, inout, or undef) of Logical Port *lport*. If the direction of the port is not set then this API returns undef.

Arguments

`lport`

Logical Port object returned by the DesignGetLogicalPortByName/ DesignGetLogicalPortList, ComponentGetLogicalPortByName/ ComponentGetLogicalPortList, or InterfaceDefGetLogicalPortByName/ InterfaceDefGetLogicalPortList function.

Returns

Direction of Logical Port *lport* as one of the following values:

IN_DIR	OUT_DIR	INOUT	UNDEF

LogicalPortGetActualPortName

Declaration

```
const char* LogicalPortGetActualPortName(void* lport);
```

Description

Returns the name of the actual port corresponding to Logical Port *lport*.

Arguments

`lport`

Logical Port object returned by the DesignGetLogicalPortByName/ DesignGetLogicalPortList, ComponentGetLogicalPortByName/ ComponentGetLogicalPortList, or InterfaceDefGetLogicalPortByName/ InterfaceDefGetLogicalPortList function.

Returns

Name of the actual port corresponding to Logical Port *lport*.

LogicalPortGetAliasLogicalPortByName

Declaration

```
void* LogicalPortGetAliasLogicalPortByName  (void* lport, const char*
aliasLportName);
```

Description

Returns the named Alias Logical Port *aliasPortName* for Logical Port *lport*.

Arguments

`lport`

Logical Port object returned by the DesignGetLogicalPortByName/ DesignGetLogicalPortList, ComponentGetLogicalPortByName/ ComponentGetLogicalPortList, or InterfaceDefGetLogicalPortByName/ InterfaceDefGetLogicalPortList function.

`aliasLportName`

Name of the Alias Logical Port.

Returns

Alias Logical Port *aliasPortName* for Logical Port *lport*.

Notes

Use the Alias Logical Port-related API functions to process the Alias Logical Port objects.

LogicalPortGetAliasLogicalPortList

Declaration

```
void* LogicalPortGetAliasLogicalPortList(void* lport);
```

Description

Returns the list (*itr*) of Alias Logical Ports for Logical Port *lport*.

Arguments

`lport`

Logical Port object returned by the v function.

Returns

List (*itr*) of Alias Logical Ports for Logical Port *lport*.

Notes

Use the Alias Logical Port-related API functions to process the Alias Logical Port objects.

LogicalPortGetConstraintsInfo

Declaration

```
void * LogicalPortGetConstraintsInfo(void* lport);
```

Description

Returns the constraints information of the specified logical port object.

Arguments

`lport`

Logical Port object returned by the DesignGetLogicalPortByName/ DesignGetLogicalPortList, ComponentGetLogicalPortByName/ ComponentGetLogicalPortList, or InterfaceDefGetLogicalPortByName/ InterfaceDefGetLogicalPortList function.

LogicalPortGetDefaultValue

Declaration

```
const char* LogicalPortGetDefaultValue(void* lport);
```

Description

Returns the Default Value of Logical Port *lport*.

Arguments

`lport`

Logical Port object returned by the DesignGetLogicalPortByName/ DesignGetLogicalPortList, ComponentGetLogicalPortByName/ ComponentGetLogicalPortList, or InterfaceDefGetLogicalPortByName/ InterfaceDefGetLogicalPortList function.

Returns

Default Value of Logical Port *lport*.

LogicalPortGetDir

Declaration

```
const char* LogicalPortGetDir(void* lport);
```

Description

Returns the Direction (in, out, inout, or undef) of Logical Port *lport*.

Arguments

`lport`

Logical Port object returned by the DesignGetLogicalPortByName/ DesignGetLogicalPortList, ComponentGetLogicalPortByName/ ComponentGetLogicalPortList, or InterfaceDefGetLogicalPortByName/ InterfaceDefGetLogicalPortList function.

Returns

Direction of Logical Port *lport* as one of the following values:

IN_DIR	OUT_DIR	INOUT	UNDEF

LogicalPortGetDriverType

Declaration

```
const char * LogicalPortGetDriverType(void* lport);
```

Description

Returns the type of driver required by the logical port.

Arguments

`lport`

Logical Port object returned by the DesignGetLogicalPortByName/ DesignGetLogicalPortList, ComponentGetLogicalPortByName/ ComponentGetLogicalPortList, or InterfaceDefGetLogicalPortByName/ InterfaceDefGetLogicalPortList function.

LogicalPortGetLSB

Declaration

```
const char* LogicalPortGetLSB(void* lport);
```

Description

Returns the LSB of Logical Port *lport*.

Arguments

`lport`

Logical Port object returned by the DesignGetLogicalPortByName/ DesignGetLogicalPortList, ComponentGetLogicalPortByName/ ComponentGetLogicalPortList, or InterfaceDefGetLogicalPortByName/ InterfaceDefGetLogicalPortList function.

Returns

LSB of Logical Port *lport*.

LogicalPortGetMSB

Declaration

```
const char* LogicalPortGetMSB(void* lport);
```

Description

Returns the MSB of Logical Port *lport*.

Arguments

`lport`

Logical Port object returned by the DesignGetLogicalPortByName/ DesignGetLogicalPortList, ComponentGetLogicalPortByName/ ComponentGetLogicalPortList, or InterfaceDefGetLogicalPortByName/ InterfaceDefGetLogicalPortList function.

Returns

MSB of Logical Port *lport*.

LogicalPortGetName

Declaration

```
const char* LogicalPortGetName(void* lport);
```

Description

Returns the name of Logical Port *lport*.

Arguments

`lport`

Logical Port object returned by the DesignGetLogicalPortByName/ DesignGetLogicalPortList, ComponentGetLogicalPortByName/ ComponentGetLogicalPortList, or InterfaceDefGetLogicalPortByName/ InterfaceDefGetLogicalPortList function.

Returns

Name of Logical Port *lport*.

LogicalPortGetOptional

Declaration

```
const char* LogicalPortGetOptional(void* lport);
```

Description

Returns the value of the Optional field of Logical Port *lport*.

Arguments

`lport`

Logical Port object returned by the DesignGetLogicalPortByName/ DesignGetLogicalPortList, ComponentGetLogicalPortByName/ ComponentGetLogicalPortList, or InterfaceDefGetLogicalPortByName/ InterfaceDefGetLogicalPortList function.

Returns

Value of the Optional field for Logical Port *lport*.

LogicalPortGetRequiresDriver

Declaration

```
const char * LogicalPortGetRequiresDriver(void* lport);
```

Description

Returns if the logical port requires a driver.

Arguments

`lport`

Logical Port object returned by the DesignGetLogicalPortByName/ DesignGetLogicalPortList, ComponentGetLogicalPortByName/ ComponentGetLogicalPortList, or InterfaceDefGetLogicalPortByName/ InterfaceDefGetLogicalPortList function.

LogicalPortGetSignalType

Declaration

```
const char* LogicalPortGetSignalType(void* lport);
```

Description

Returns the Signal Type (clock, reset, event, data, control,...) of Logical Port *lport*.

Arguments

`lport`

Logical Port object returned by the DesignGetLogicalPortByName/ DesignGetLogicalPortList, ComponentGetLogicalPortByName/ ComponentGetLogicalPortList, or InterfaceDefGetLogicalPortByName/ InterfaceDefGetLogicalPortList function.

Returns

Signal type of Logical Port *lport* as one of the following values:

ADDR	ANALOG	CLK	CONFIG	CONTROL
CORE	CORE_INPUT	CORE_OUTPUT	DATA	DFT
EVT	FUNCTION_IN	IO	IO_OEN	IO_PAD

IO_PULLEN	IO_SELECT	OSCCTL	PVT	PWRDN
RST	SCANIN	SCANOUT	SLEWRATE	TEST_IN
TIEOFF	UNDEFPORT			

LogicalPortGetWidth

Declaration

```
int LogicalPortGetWidth(void* lport);
```

Description

Returns the Width of Logical Port *lport*.

Arguments

```
lport
```

Logical Port object returned by the DesignGetLogicalPortByName/ DesignGetLogicalPortList, ComponentGetLogicalPortByName/ ComponentGetLogicalPortList, or InterfaceDefGetLogicalPortByName/ InterfaceDefGetLogicalPortList function.

Returns

Width of Logical Port *lport*.

LogicalPortIsRegistered

Declaration

```
const char* LogicalPortIsRegistered(void* lport);
```

Description

Returns 1 if Logical Port *lport* is registered port. Otherwise, returns 0.

Arguments

```
lport
```

Logical Port object returned by the DesignGetLogicalPortByName/ DesignGetLogicalPortList, ComponentGetLogicalPortByName/

ComponentGetLogicalPortList, or InterfaceDefGetLogicalPortByName/ InterfaceDefGetLogicalPortList function.

Returns

1 if Logical Port *lport* is registered port. Otherwise, returns 0.

Examples of Using API Functions to Process Logical Port Objects

The following example shows how you can use the API functions given above to process logical port objects.

Example1

The following subroutine actual_portinfo takes information about a logical port (including the lsb and msb) and returns the lsb and msb of its corresponding actual port.

```
sub actual_portinfo{   my ($logical_port,$lsb,$msb) =
@_;   if(!$logical_port)   {      return;   }   my $aport =
LogicalPortGetActualPortName($logical_port);   my $alsb =
LogicalPortGetLSB($logical_port);   #actual port lsb/msb corresponding to logical
port's   #lsb/msb would be alsb+lsb/alsb+msb
   my $retval = $aport."[".($alsb+$lsb).":".($alsb+$msb)."]";   return $retval;}
```

For example, if logical ports L1 and L2 have the following actual port mappings:

L1[0:31] ==> P1[0:31] and

L2[0:31] ==> P1[32:63]

The actual_portinfo subroutine will return the actual mappings as given below:

L1[0:0] => P1[0:0], L1[1:1] => P1[1:1].. and

L2[0:0] => P1[32:32], L1[1:1] => p1[33:33] ..

Master Offset Pair

The Master Offset Pair object has the following design hierarchy:

comRoot -> Component -> Bridge Memory Map ->
Bridge Remap State -> Master Offset Pair

Use the following API functions to process the Master Offset Pair objects:

MasterOffsetPairGetBaseAddress

Declaration
```
const char* MasterOffsetPairGetBaseAddress (void* obj);
```

Description

Returns the base address of the master interface associated with a master/base address pair.

Arguments
`obj`

Master offset pair object.

MasterOffsetPairGetBitOffset

Declaration
```
const char* MasterOffsetPairGetBitOffset (void* obj);
```

Description

Returns the bit offset of the master interface associated with a master/base address pair.

Arguments
`obj`

Master offset pair object.

MasterOffsetPairGetMasterName

Declaration
```
const char* MasterOffsetPairGetMasterName (void* obj);
```

Description

Returns the name of the master interface associated with a master/base address pair.

Arguments
`obj`

Master offset pair object.

Memory Block

The Memory Block object has the following design hierarchy:

comRoot -> Component -> Memory Block

Use the following API functions to process the Memory Block objects:

MemoryBlockGetAccessType

Declaration

```
const char* MemoryBlockGetAccessType(void* memBlock);
```

Description

Returns the Access Type of Memory Block *memBlock*.

Arguments

`memBlock`

Memory Block object returned by the ComponentGetClkSourceList function.

Returns

Access Type of Memory Block *memBlock* as one of the following values:

EXECUTE	NOT_ACCESSIBLE	PROGRAM_MEMORY
READ	READ_CLEAR	READ_EXECUTE
READ_WRITE	READ_WRITE_CLEAR	READ_WRITE_EXEC
READ_WRITE_SET	RESERVED	STICKY
UNDEF	WRITE	WRITE_CLEAR

MemoryBlockGetArraySize

Declaration

```
int MemoryBlockGetArraySize(void* memBlock);
```

Description

Returns the array size of Memory Block *memBlock*.

Arguments

`memBlock`

Memory Block object returned by the ComponentGetClkSourceList function.

MemoryBlockGetBankAlignType

Declaration

```
const char* MemoryBlockGetBankAlignType(void* memBlock);
```

Description

Returns the alignment of Memory Block *memBlock*.

Arguments

`memBlock`

Memory Block object returned by the ComponentGetClkSourceList function.

MemoryBlockGetEndianType

Declaration

```
const char* MemoryBlockGetEndianType(void* memBlock);
```

Description

Returns the Endian Type of Memory Block *memBlock*.

Arguments

`memBlock`

Memory Block object returned by the ComponentGetClkSourceList function.

Returns

Endian Type of Memory Block *memBlock* as one of the following values:

BIG	SMALL	UNDEF

MemoryBlockGetMaxDataWidth

Declaration

```
const char* MemoryBlockGetMaxDataWidth  (void* memBlock);
```

Description

Returns the Maximum Data Width of Memory Block *memBlock*.

Arguments

`memBlock`

Memory Block object returned by the ComponentGetClkSourceList function.

Returns

Maximum Data Width of Memory Block *memBlock*.

MemoryBlockGetName

Declaration

```
const char* MemoryBlockGetName(void* memBlock);
```

Description

Returns the name of Memory Block *memBlock*.

Arguments

`memBlock`

Memory Block object returned by the ComponentGetClkSourceList function.

Returns

Name of Memory Block *memBlock*.

MemoryBlockGetOffset

Declaration

```
const char* MemoryBlockGetOffset(void* memBlock);
```

Description

Returns the Offset of Memory Block *memBlock*.

Arguments

`memBlock`

Memory Block object returned by the ComponentGetClkSourceList function.

Returns

Offset of Memory Block *memBlock*.

MemoryBlockGetSize

Declaration

```
MemoryBlockGetSize(void* memBlock);
```

Description

Returns the size of Memory Block *memBlock*.

Arguments

`memBlock`

Memory Block object returned by the ComponentGetClkSourceList function.

MemoryBlockIsVolatile

Declaration

```
const char* MemoryBlockIsVolatile(void* memBlock);
```

Description

Returns 1 if Memory Block *memBlock* is volatile. Otherwise, returns 0.

Arguments

`memBlock`

Memory Block object returned by the ComponentGetClkSourceList function.

Returns

1 if Memory Block *memBlock* is volatile. Otherwise, returns 0.

Memory Map

The Memory map object has the following design hierarchy:

comRoot -> Component -> Memory Map

Use the following API functions to process the Memory ma objects:

MemoryMapDontCompare

Declaration

```
Int MemoryMapGetDontCompare(void* map)
```

Description

Returns the DontCompare of the memMap object.

MemoryMapGetAddressing

Declaration

```
char* MemoryMapGetAddressing(void* memMap);
```

Description

Returns addressing information (compact, regalign, or fullalign) of the memMap object returned by the ComponentGetMemoryMapList or ComponentGetMemoryMapByName function.

MemoryMapGetAlignment

Declaration

```
char* MemoryMapGetAlignment(void* memMap);
```

Description

Returns the alignment information of the memMap object returned by the ComponentGetMemoryMapList or ComponentGetMemoryMapByName function.

MemoryMapGetDontTest

Declaration

```
Int MemoryMapGetDontTest(void* map)
```

Description

Returns the DontTest of the memMap object.

MemoryMapGetEndianess

Declaration

```
char* MemoryMapGetEndianess(void* memMap);
```

Description

Returns endianness information (bigendian or littleendian) of the memMap object returned by the ComponentGetMemoryMapList or ComponentGetMemoryMapByName function.

MemoryMapGetMap

Declaration

```
char* MemoryMapGetMap(void* memMap);
```

Description

Returns the map name of the memMap object returned by the ComponentGetMemoryMapList or ComponentGetMemoryMapByName function.

MemoryMapGetName

Declaration

```
char* MemoryMapGetName(void* memMap);
```

Description

Returns the name of the memMap object returned by the ComponentGetMemoryMapList or ComponentGetMemoryMapByName function.

MemoryMapGetOrientation

Declaration

```
char* MemoryMapGetOrientation(void* memMap);
```

Description

Returns orientation information (lsb0 or msb0) of the memMap object returned by the ComponentGetMemoryMapList or ComponentGetMemoryMapByName function.

MemoryMapGetPropertyAssignByName

Declaration

```
void* MemoryMapGetPropertyAssignByName (void* memMap, char name);
```

Description

Returns named property of the memMap object returned by ComponentGetMemoryMapList or ComponentGetMemoryMapByName function.

MemoryMapGetPropertyAssignList

Declaration

```
void* MemoryMapGetPropertyAssignList (void* memMap);
```

Description

Returns a list of properties of the memMap object returned by ComponentGetMemoryMapList or ComponentGetMemoryMapByName function.

MemoryMapGetRDLName

Declaration

```
char* MemoryMapGetRDLName(void* memMap);
```

Description

Returns the RDL name of the memMap object returned by the ComponentGetMemoryMapList or ComponentGetMemoryMapByName function.

MemoryMapGetRsvdset

Declaration

```
int MemoryMapGetRsvdset(void* memMap);
```

Description

Returns the Rsvdset information (1 or 0) for the memMap object returned by the ComponentGetMemoryMapList or ComponentGetMemoryMapByName function.

MemoryMapGetRsvdsetX

Declaration

```
int MemoryMapGetRsvdsetX(void* memMap);
```

Description

Returns the RsvdsetX information (1 or 0) for the memMap object returned by the ComponentGetMemoryMapList or ComponentGetMemoryMapByName function.

MemoryMapGetSharedExtBus

Declaration

```
int MemoryMapGetSharedExtBus(void* memMap);
```

Description

Returns the sharedextbus information (1 or 0) for the memMap object returned by the ComponentGetMemoryMapList or ComponentGetMemoryMapByName function.

Parameter

The Parameter object has the following design hierarchy:

comRoot -> Component -> Parameter

comRoot -> Design -> Component Instance -> Parameter

Use the following API functions to process the Parameter objects:

ParameterGetTypeTable

Declaration

```
void* ParameterGetTypeTable(void* param);
```

Description

Returns the TypeTable for parameter *param*.

Arguments

`param`

Parameter object returned by the ComponentGetParameterTable/ ComponentGetUserParameterTable/ComponentInstGetParameter function.

ParameterGetTypeTreeByName

Declaration

```
void* ParameterGetTypeTreeByName  (void* param, const char*
typeTreeName);
```

Description

Returns the named TypeTree *typeTreeName* of parameter *param*.

Arguments

`param`

Parameter object returned by the ComponentGetParameterTable/ ComponentGetUserParameterTable/ComponentInstGetParameter function.

`typeTreeName`

Name of the TypeTree.

Returns

TypeTree *typeTreeName* of parameter *param*.

Notes

Use the TypeTree-related API functions to process TypeTree objects.

ParameterGetTypeTreeList

Declaration

```
void* ParameterGetTypeTreeList(void* param);
```

Description

Returns the list (*itr*) of TypeTrees of parameter *param*.

Arguments

`param`

Parameter object returned by the ComponentGetParameterTable/ ComponentGetUserParameterTable/ComponentInstGetParameter function.

Returns

List (*itr*) of TypeTrees of parameter *param*.

Notes

Use the TypeTree-related API functions to process TypeTree objects.

Returns

TypeTable for parameter *param*.

ParameterGetValueTable

Declaration

```
void* ParameterGetValueTable(void* param);
```

Description

Returns the ValueTable for parameter *param*.

Arguments

`param`

> Parameter object returned by the ComponentGetParameterTable/ ComponentGetUserParameterTable/ComponentInstGetParameter function.

Returns

ValueTable for parameter *param*.

ParameterGetValueTreeByName

Declaration

```
void* ParameterGetValueTreeByName  (void* param, const char*
valueTreeName);
```

Description

Returns the named ValueTree *valueTreeName* of parameter *param*.

Arguments

`param`

> Parameter object returned by the ComponentGetParameterTable/ ComponentGetUserParameterTable/ComponentInstGetParameter function.

`valueTreeName`

> Name of the ValueTree.

Returns

ValueTree *valueTreeName* of parameter *param*.

ParameterGetValueTreeList

Declaration

```
void* ParameterGetValueTreeList(void* param);
```

Description

Returns the list (*itr*) of ValueTrees of parameter *param*.

Arguments
`param`

Parameter object returned by the ComponentGetParameterTable/ ComponentGetUserParameterTable/ComponentInstGetParameter function.

Returns

List (*itr*) of ValueTrees of parameter *param*.

Notes

Use the ValueTree-related API functions to process ValueTree objects.

ParameterSearchBasicValueTreeByName

Declaration
```
void* ParameterSearchBasicValueTreeByName  (void* param, const char*
Name);
```

Description

Returns the named BasicValueTree *bvalueTreename* for parameter *param*.

Arguments
`param`

Parameter object returned by the ComponentGetParameterTable/ ComponentGetUserParameterTable/ComponentInstGetParameter function.

Returns

BasicValueTree *bvalueTreename* for parameter *param*.

ParameterSearchValueTreeByHierarchy

Declaration
```
void* ParameterSearchValueTreeByHierarchy  (void* param, const char*
valueTreeName,  const char* hier);
```

Description

Returns the named ValueTree *valueTreeName* under hierarchy *hier*.

Arguments

`param`

Parameter object returned by the ComponentGetParameterTable/ ComponentGetUserParameterTable/ComponentInstGetParameter function.

`valueTreeName`

Named ValueTree

`hier`

Hierarchy

Returns

ValueTree *valueTreeName* under hierarchy *hier*.

ParamType

The ParamType object has the following design hierarchy:

comRoot -> Component -> Parameter -> TypeTree -> ParamType

comRoot -> Design -> Component Instance -> Parameter -> TypeTree -> ParamType

Use the following API functions to process the ParamType objects:

ParamTypeGetName

Declaration

```
const char* ParamTypeGetName(void* paramType);
```

Description

Returns the name of ParamType *paramType*.

Arguments

`paramType`

ParamType object returned by the TypeTreeGetParamTypeTreeByName/ TypeTreeGetParamTypeList function.

Returns

Name of ParamType *paramType*.

ParamTypeGetParamTypeEnumString

Declaration

```
const char* ParamTypeGetParamTypeEnumString  (void* paramType);
```

Description

Returns the Enum string for ParamType *paramType*.

Arguments

`paramType`

ParamType object returned by the TypeTreeGetParamTypeTreeByName/ TypeTreeGetParamTypeList function.

Returns

Enum string of ParamType *paramType*.

ParamTypeGetType

Declaration

```
const char* ParamTypeGetType(void* paramType);
```

Description

Returns the type of ParamType *paramType*.

Arguments

`paramType`

ParamType object returned by the TypeTreeGetParamTypeTreeByName/ TypeTreeGetParamTypeList function.

Returns

Type of the ParamType *paramType* as one of the following values:

ArrayT	BasicT	EnumT	UndefT

Port

The Port object has the following design hierarchy:

comRoot -> Design -> Port

comRoot -> Component -> Port

Use the following API functions to process the Port objects:

PortGetAlign

Declaration
```
const char* PortGetAlign(void* obj)
```

Description

Returns the align field data of the port *obj*.

PortGetAllLogicalDirectionsAllowed

Declaration
```
int PortGetAllLogicalDirectionsAllowed(void* obj)
```

Description

Returns 0 or 1 to specify if a port can be mapped to a port in the abstraction definition (logical port) with a different direction.

PortGetClockRate

Declaration
```
const char* PortGetClockRate(void* obj)
```

Description

Returns the ClockRate field data of the port *obj*.

PortGetControlClock

Declaration

```
void* PortGetControlClock(void* port);
```

Description

Returns the Control Clock of Port *port*.

PortGetControlReset

Declaration

```
void* PortGetControlReset(port);
```

Description

Returns the Control Reset of Port *port*.

PortGetDefValue

Declaration

```
const char* PortGetDefValue(void* port);
```

Description

Returns the Default Value of Port *port*.

PortGetDirection

Declaration

```
const char* PortGetDirection(void* port);
```

Description

Returns the Port direction (in, out, inout, or undef) for Port *port*.

PortGetHDLType

Declaration

```
const char* PortGetHDLType(void* obj)
```

Description

Returns HDL type of the port *obj*.

PortGetHDLTypeLanguage

Declaration

```
const char* PortGetHDLTypeLanguage(void* obj)
```

Description

Returns the language of the HDL type *obj*.

PortGetHDLTypeLibrary

Declaration

```
const char* PortGetHDLTypeLibrary(void* obj)
```

Description

Returns the library of the HDL type *obj*.

PortGetHDLTypeLSB

Declaration

```
const char* PortGetHDLTypeLSB(void* obj)
```

Description

Returns the LSB of the HDL type *obj*.

PortGetHDLTypeMSB

Declaration
```
const char* PortGetHDLTypeMSB(void* obj)
```

Description

Returns the MSB of the HDL type *obj*.

PortGetHDLTypePackage

Declaration
```
const char* PortGetHDLTypePackage(void* obj)
```

Description

Returns the package of the HDL type *obj*.

PortGetInterfaceGroup

Declaration
```
const char* PortGetInterfaceGroup(void* obj)
```

Description

Returns interface group for the port *obj*.

PortGetLogicalName

Declaration
```
const char* PortGetLogicalName(void* port);
```

Description

Returns the Logical Name of Port *port*.

PortGetLSB

Declaration
```
const char* PortGetLSB(void* port);
```

Description

Returns the LSB of Vector Port *port*.

PortGetMSB

Declaration
```
const char* PortGetMSB(void* port);
```

Description

Returns the MSB of Vector Port *port*.

PortGetName

Declaration
```
const char* PortGetName(void* port);
```

Description

Returns the Name of Port *port*.

PortGetNthScalarPort

Declaration
```
void* PortGetNthScalarPort(void* port, int N);
```

Description

Returns the *N*th portbit (by index number) of Vector Port *port*.

PortGetParent

Declaration
```
void* PortGetParent(void* port);
```

Description

Returns the Parent (Design or Component) of Port *port*.

PortGetParentInterfaceDefList

Declaration
```
void* PortGetParentInterfaceDefList(void* port);
```

Description

Returns the list (*itr*) of Parent Interface Definitions of Port *port*.

PortGetParentInterfacesList

Declaration
```
void* PortGetParentInterfacesList(void* port);
```

Description

Returns the list (*itr*) of Parent Interfaces of Port *port*.

PortGetPowerDomain

Declaration
```
const char* PortGetPowerDomain(void* obj)
```

Description

Returns the power domain information of design port *obj*.

PortGetScalarPortList

Declaration
```
void* PortGetScalarPortList(void* port);
```

Description

Returns the list (*itr*) of PortBits of Vector Port *port*.

PortGetStatus

Declaration
```
const char* PortGetStatus(void* port);
```

Description

Returns the status (open or empty string) of Port *port*.

PortGetType

Declaration
```
const char* PortGetType(void* port);
```

Description

Returns the Port type (clock, reset, event, data, control,...) of Port *port*.

PortGetVoltage

Declaration
```
const char* PortGetVoltage(void* obj)
```

Description

Returns the voltage of design port *obj*.

PortGetWidth

Declaration

```
const char* PortGetWidth(void* port);
```

Description

Returns the width of Vector Port *port*.

PortIsOpen

Declaration

```
const char* PortIsOpen(void* obj)
```

Description

Returns the Open field data of the port *obj*.

PortIsScalar

Declaration

```
int PortIsScalar(void* port);
```

Description

Returns 1 if Port *port* is a scalar port. Otherwise, returns 0.

PortMap

The PortMap object has the following design hierarchy:

comRoot -> Design -> Interface -> PortMap

comRoot -> Component -> Interface -> PortMap

```
Use the following API functions to process the PortMap objects:
```

PortGetStatus

Declaration

```
const char* PortGetStatus(void* portMap);
```

Description

Returns the status of the Actual Port for Interface Port Map item *portMap*.

PortMapGetActualPortName

Declaration

```
const char* PortMapGetActualPortName(void* portMap);
```

Description

Returns the name of the Actual Port for Interface Port Map item *portMap*.

PortMapGetLogicalPortName

Declaration

```
const char* PortMapGetLogicalPortName(void* portMap);
```

Description

Returns the name of the Logical Port for Interface Port Map item *portMap*.

PropertyAssign

The PropertyAssign object has the following design hierarchy:

comRoot -> Component -> Register Object -> Register Group -> Register Data -> PropertyAssign

comRoot -> Component -> Register Object -> Register Group -> Register Data -> BitField -> PropertyAssign

comRoot -> Component -> Register Object -> Register Data -> BitField -> PropertyAssign

comRoot -> Component -> AddressMapDefn -> PropertyAssign comRoot -> Component -> RegfileDefn -> PropertyAssign

comRoot -> Component -> SignalDefns -> PropertyAssign

comRoot -> Component -> BitFieldDefn -> PropertyAssign

`Use the following API functions to process the PropertyAssign objects:`

PropertyAssignGetProperty

Declaration

```
char* PropertyAssignGetProperty(void* PropertyDefn);
```

Description

Returns the property name assigned to the Register Data -> BitField, AddressMapDefn, RegfileDefn, SignalDefns, or BitFieldDefn object.

PropertyAssignGetValue

Declaration

```
char* PropertyAssignGetValue(void* PropertyDefn);
```

Description

Returns the property value assigned to the Register Data -> BitField, AddressMapDefn, RegfileDefn, SignalDefns, or BitFieldDefn object.

PropertyDefn

The PropertyDefn object has the following design hierarchy:

comRoot -> Component -> PropertyDefn

`Use the following API functions to process the PropertyDefn objects:`

PropertyDefnGetComponent

Declaration

```
char* PropertyDefnGetComponent(void* PropertyDefn);
```

Description

Returns the component information (field, reg, regfile, addrmap, or all) of the PropertyDefn object returned by the ComponentGetProptertyDefnByName or ComponentGetPropertyDefnList functions.

PropertyDefnGetDefault

Declaration

```
int PropertyDefnGetDefault(void* PropertyDefn);
```

Description

Returns the default information (1 or 0) of the PropertyDefn object returned by the ComponentGetProptertyDefnByName or ComponentGetPropertyDefnList functions.

PropertyDefnGetDefaultDefault

Declaration

```
char* PropertyDefnGetDefaultDefault(void* PropertyDefn);
```

Description

Returns the default value of the PropertyDefn object returned by the ComponentGetProptertyDefnByName or ComponentGetPropertyDefnList functions.

PropertyDefnGetProperty

Declaration

```
char* PropertyDefnGetProperty(void* PropertyDefn);
```

Description

Returns the property name of the PropertyDefn object returned by the ComponentGetProptertyDefnByName or ComponentGetPropertyDefnList functions.

PropertyDefnGetRDLName

Declaration

```
char* PropertyDefnGetRDLName(void* PropertyDefn);
```

Description

Returns the RDL name of the PropertyDefn object returned by the ComponentGetProptertyDefnByName or ComponentGetPropertyDefnList functions.

PropertyDefnGetType

Declaration

```
char* PropertyDefnGetType(void* PropertyDefn);
```

Description

Returns the type information (string, number, boolean, or ref) of the PropertyDefn object returned by the ComponentGetProptertyDefnByName or ComponentGetPropertyDefnList functions.

RegfileDefn

The RegfileDefn Object object has the following design hierarchy:

comRoot -> Component -> RegfileDefn

Use the following API functions to process the RegfileDefn Object objects:

RegfileDefnGetAlignment

Declaration

```
char* RegfileDefnGetAlignment(void* RegfileDefn)
```

Description

Returns the alignment information for the *regfileDefn* object returned by the ComponentGetRegfileDefnByName or ComponentGetRegfileDefnList function.

RegfileDefnGetAllocationOperator

Declaration
`char* RegfileDefnGetAllocationOperator(void* RegfileDefn)`

Description

Returns the allocation operator information (at, incr, next_at) for the *regfileDefn* object returned by the ComponentGetRegfileDefnByName or ComponentGetRegfileDefnList function.

RegfileDefnGetDontCompare

Declaration
`Int RegFileDefnGetDontCompare(void* bf)`

Description
Returns the DontCompare value for the *regfileDefn* object.

RegfileDefnGetDontTest

Declaration
`Int RegFileDefnGetDontTest(void* bf)`

Description
Returns the DontTest value for the *regfileDefn* object

RegfileDefnGetPropertyAssignByName

Declaration
`void* RegfileDefnGetPropertyAssignByName(void* RegfileDefn, char name)`

Description

Returns named property of the *regfileDefn* object returned by the ComponentGetRegfileDefnByName or ComponentGetRegfileDefnList function.

RegfileDefnGetPropertyAssignList

Declaration

```
void* RegfileDefnGetPropertyAssignList(void* RegfileDefn)
```

Description

Returns a list of properties of the *regfileDefn* object returned by the ComponentGetRegfileDefnByName or ComponentGetRegfileDefnList function.

RegfileDefnGetRDLName

Declaration

```
char* RegfileDefnGetRDLName(void* RegfileDefn)
```

Description

Returns the RDL name of the *regfileDefn* object returned by the ComponentGetRegfileDefnByName or ComponentGetRegfileDefnList function.

RegfileDefnGetRegfile

Declaration

```
char* RegfileDefnGetRegfile(void* RegfileDefn)
```

Description

Returns the name of the *regfileDefn* object returned by the ComponentGetRegfileDefnByName or ComponentGetRegfileDefnList function.

RegfileDefnGetSharedExtBus

Declaration

```
int RegfileDefnGetSharedExtBus(void* RegfileDefn)
```

Description

Returns SharedExtBus information (1 or 0) for the *regfileDefn* object returned by the ComponentGetRegfileDefnByName or ComponentGetRegfileDefnList function.

Register Object

The Register Object object has the following design hierarchy:

comRoot -> Component -> Register Object

Use the following API functions to process the Register Object objects:

RegisterObjectGetName

Declaration

```
const char* RegisterObjectGetName (void* regObject)
```

Description

Returns the name of the register object *regObject*.

RegisterObjectGetRegisterData

Declaration

```
void* RegisterObjectGetRegisterData(void* regObject)
```

Description

Returns pointer to the register data of the register object *regObject*.

RegisterObjectGetRegisterGroup

Declaration

```
void* RegisterObjectGetRegisterGroup (void* regObject)
```

Description

Returns the register group of the register object *regObject*.

RegisterObjectGetType

Declaration
```
char* RegisterObjectGetType(void* regObject);
```

Description

Returns type of Register Object. Returns *REGISTER*, if Register Object is a Register Data and returns *GROUP*, if Register Object is a Register Group.

Register Data

The Register Data object has the following design hierarchy:

comRoot -> Component -> Register Object -> Register Data

Use the following API functions to process the Register Data objects:

RegisterDataGetAccessType

Declaration
```
const char* RegisterDataGetAccessType(void* regObj);
```

Description

Returns the access type of the register object, regObj. This API can return the following access types:

READ	WRITE	READ_WRITE	OTHER

RegisterDataGetAccessWidth

Declaration
```
int RegisterDataGetAccessWidth(void* regDataobj)
```

Description

Returns the access width of the SystemRDL register object.

RegisterDataGetArraySize

Declaration

```
int RegisterDataGetArraySize(void* regData);
```

Description

Returns the array size of Register Data *regData*.

RegisterDataGetBitFieldByName

Declaration

```
void* RegisterDataGetBitFieldByName  (void* regData, const char*
bitFieldName);
```

Description

Returns the named Bit-field *bitFieldName* for Register Data *regData*.

RegisterDataGetBitFieldList

Declaration

```
void* RegisterDataGetBitFieldList(void* regData);
```

Description

Returns the list (*itr*) of Bit-fields for Register Data *regData*.

RegisterDataGetDefnName

Declaration

```
char* RegisterDataGetDefnName(void* regDataobj)
```

Description

Returns the DefnName of the SystemRDL register object.

RegisterDataGetDontCompare

Declaration

```
Int RegisterDataGetDontCompare(void* reg)
```

Description

Returns the DontCompare of the SystemRDL register object.

RegisterDataGetDontTest

Declaration

```
Int RegisterDataGetDontTest(void* reg)
```

Description

Returns the DontTest of the SystemRDL register object.

RegisterDataGetErrExtBus

Declaration

```
char* RegisterDataGetErrExtBus(void* regDataobj)
```

Description

Returns the ErrExtBus of the SystemRDL register object.

RegisterDataGetExternal

Declaration

```
int RegisterDataGetExternal(void* regDataobj)
```

Description

Returns the external value, 1 or 0, of the SystemRDL register object.

RegisterDataGetName

Declaration

```
const char* RegisterDataGetName(void* regData);
```

Description

Returns the name of Register Data *regData*.

RegisterDataGetOffset

Declaration

```
const char* RegisterDataGetOffset(void* regData);
```

Description

Returns the Offset of Register Data *regData*.

RegisterDataGetOrder

Declaration

```
char* RegisterDataGetOrder(void* regDataobj)
```

Description

Returns the order, lsb0 or msb0, of the SystemRDL register object.

RegisterDataGetPropertyAssignByName

Declaration

```
void* RegisterDataGetPropertyAssignByName(void* regDataobj, char name)
```

Description

Returns named property of the SystemRDL register data object *regData*.

RegisterDataGetPropertyAssignList

Declaration

```
void* RegisterDataGetPropertyAssignList(void* regDataobj)
```

Description

Returns a list of properties of the SystemRDL register data object *regData*.

RegisterDataGetRDLName

Declaration

```
char* RegisterDataGetRDLName(void* regDataobj)
```

Description

Returns the RDL name of the SystemRDL register object.

RegisterDataGetRegWidth

Declaration

```
int RegisterDataGetRegWidth(void* regDataobj)
```

Description

Returns the width of the SystemRDL register object.

RegisterDataGetReserved

Declaration

```
Int RegisterDataGetReserved(void* regObj);
```

Description

Returns 1 if the register, *regObj*, is reserved and returns 0 if the register is not reserved.

RegisterDataGetReset

Declaration

```
const char* RegisterDataGetReset (void* regData);
```

Description

Returns the reset value for Register Data *regData*.

RegisterDataGetResetMask

Declaration

```
const char* RegisterDataGetResetMask (void* regData);
```

Description

Returns the reset mask for Register Data *regData*.

RegisterDataGetShared

Declaration

```
int RegisterDataGetShared(void* regDataobj)
```

Description

Returns the shared value, 1 or 0, of the SystemRDL register object.

RegisterDataGetWidth

Declaration

```
const char* RegisterDataGetWidth(void* regData);
```

Description

Returns the width of Register Data *regData*.

RegisterDataIsVolatileData

Declaration

```
const char* RegisterDataIsVolatileData (void* regData);
```

Description

Specifies if the Register Data *regData* is of C volatile type.

Register Group

The Register Group object has the following design hierarchy:

comRoot -> Component -> Register Object -> Register Group

Use the following API functions to process the Register Group objects:

RegisterGroupGetArrayOffset

Declaration

```
const char* RegisterGroupGetArrayOffset(void* regGrp);
```

Description

Returns the array offset value for Register Group, *regGrp*. The array offset is the space between two elements of a register group array.

RegisterGroupGetArraySize

Declaration

```
int RegisterGroupGetArraySize(void* regGrp);
```

Description

Returns the array size of Register Group *regGrp*.

RegisterGroupGetName

Declaration

```
const char* RegisterGroupGetName(void* regGrp);
```

Description

Returns the name of Register Group *regGrp*.

RegisterGroupGetOffset

Declaration

```
int RegisterGroupGetOffset(void* regGrp);
```

Description

Returns the offset for the register group, *regGrp*.

RegisterGroupGetRegisterByName

Declaration

```
void* RegisterGroupGetRegisterByName(void* regGrp, const char* regName)
```

Description

Returns the named Register *regName* for Register Group *regGrp*.

RegisterGroupGetRegisterList

Declaration

```
void* RegisterGroupGetRegisterList (void* regGrp);
```

Description

Returns the list (*itr*) of Registers for Register Group *regGrp*.

SignalDefns

The SignalDefns object has the following design hierarchy:

comRoot -> Component -> SignalDefns

Use the following API functions to process the SignalDefns objects:

SignalDefnGetActive

Declaration
```
char* SignalDefnGetActive(void* SignalDefns);
```

Description

Returns activelow, activehigh, or unconstrained values for the SignalDefns object returned by the ComponentGetSignalDefnByName or ComponentGetSignalDefnList function.

SignalDefnGetCpuifReset

Declaration
```
int SignalDefnGetCpuifReset(void* SignalDefns);
```

Description

Returns 1 or 0 value for the SignalDefns object returned by the ComponentGetSignalDefnByName or ComponentGetSignalDefnList function.

SignalDefnGetFieldReset

Declaration
```
int SignalDefnGetFieldReset(void* SignalDefns);
```

Description

Returns 1 or 0 value for the SignalDefns object returned by the ComponentGetSignalDefnByName or ComponentGetSignalDefnList function.

SignalDefnGetPropertyAssignByName

Declaration
```
void* SignalDefnGetPropertyAssignByName(void* SignalDefns, char name)
```

Description

Returns named property for the SignalDefns object returned by the ComponentGetSignalDefnByName or ComponentGetSignalDefnList function.

SignalDefnGetPropertyAssignList

Declaration

```
void* SignalDefnGetPropertyAssignList(void* SignalDefns)
```

Description

Returns a list of properties for the SignalDefns object returned by the ComponentGetSignalDefnByName or ComponentGetSignalDefnList function.

SignalDefnGetRDLName

Declaration

```
char* SignalDefnGetRDLName(void* SignalDefns);
```

Description

Returns the RDL name of the SignalDefns object returned by the ComponentGetSignalDefnByName or ComponentGetSignalDefnList function.

SignalDefnGetSignal

Declaration

```
char* SignalDefnGetSignal(void* SignalDefns);
```

Description

Returns the signal name of the SignalDefns object returned by the ComponentGetSignalDefnByName or ComponentGetSignalDefnList function..

SignalDefnGetSignalWidth

Declaration

```
char* SignalDefnGetSignalWidth(void* SignalDefns);
```

Description

Returns the width of the SignalDefns object returned by the ComponentGetSignalDefnByName or ComponentGetSignalDefnList function..

SignalDefnGetSync

Declaration

```
char* SignalDefnGetSync(void* SignalDefns);
```

Description

Returns the sync, async, or unconstrained values for the SignalDefns object returned by the ComponentGetSignalDefnByName or ComponentGetSignalDefnList function.

SignalAssign

The SignalAssign object has the following design hierarchy:

comRoot -> Component -> SignalAssign

Use the following API functions to process the SignalAssign objects:

SignalAssignGetDestBf

Declaration

```
char* SignalAssignGetDestBf(void* SignalAssign)
```

Description

Returns the signal destination field of the SignalAssign object returned by the ComponentGetSignalAssignList function.

SignalAssignGetDestPin

Declaration

```
char* SignalAssignGetDestPin(void* SignalAssign)
```

Description

Returns the signal destination pin of the SignalAssign object returned by the ComponentGetSignalAssignList function.

SignalAssignGetDestPort

Declaration

```
char* SignalAssignGetDestPort(void* SignalAssign)
```

Description

Returns the signal destination port of the SignalAssign object returned by the ComponentGetSignalAssignList function.

SignalAssignGetDestReg

Declaration

```
char* SignalAssignGetDestReg(void* SignalAssign)
```

Description

Returns the signal destination register of the SignalAssign object returned by the ComponentGetSignalAssignList function.

SignalAssignGetPropertyAssignByName

Declaration

```
void* SignalAssignGetPropertyAssignByName (void* SignalAssign, char
name);
```

Description

Returns named property of the SignalAssign object returned by ComponentGetSignalAssignList function.

SignalAssignGetPropertyAssignList

Declaration

```
void* SignalAssignGetPropertyAssignList (void* SignalAssign);
```

Description

Returns a list of properties of the SignalAssign object returned by ComponentGetSignalAssignList function.

SignalAssignGetSourceBf

Declaration

```
char* SignalAssignGetSourceBf(void* SignalAssign)
```

Description

Returns the signal source field value for the SignalAssign object returned by the ComponentGetSignalAssignList function.

SignalAssignGetSourcePin

Declaration

```
char* SignalAssignGetSourcePin(void* SignalAssign)
```

Description

Returns the signal source pin value for the SignalAssign object returned by the ComponentGetSignalAssignList function.

SignalAssignGetSourcePort

Declaration

```
char* SignalAssignGetSourcePort(void* SignalAssign)
```

Description

Returns the signal source port value for the SignalAssign object returned by the ComponentGetSignalAssignList function.

SignalAssignGetSourceReg

Declaration

```
char* SignalAssignGetSourceReg(void* SignalAssign)
```

Description

Returns the signal source register value for the SignalAssign object returned by the ComponentGetSignalAssignList function.

SignalAssignGetSourceValue

Declaration

```
char* SignalAssignGetSourceValue(void* SignalAssign)
```

Description

Returns the signal source value of the SignalAssign object returned by the ComponentGetSignalAssignList function.

Splice

The Splice object has the following design hierarchy:

comRoot -> Design -> Component Instance -> Connection -> Interface Connection -> splice

comRoot -> Design -> Connection -> Interface Connection -> splice

Use the following API functions to process the Splice objects:

SpliceGetLSB

Declaration

```
int SpliceGetLSB(void* spl);
```

Description

Returns the LSB value for the splice port *spl*.

SpliceGetMSB

Declaration
```
int SpliceGetMSB(void* spl);
```

Description

Returns the MSB value for the splice port *spl*.

SpliceGetPortName

Declaration
```
const char* SpliceGetPortName(void* spl);
```

Description

Returns the port name for the splice port *spl*.

Terminal

The Terminal object has the following design hierarchy:

comRoot -> Design -> Component Instance > Terminal

comRoot -> Design -> Interface Instance > Terminal

Use the following API functions to process the Terminal objects:

RTLTerminalGetFaninList

Declaration
```
void* RTLTerminalGetFaninList(void* term);
```

Description

Returns the list (*itr*) of terminals connected (fanin) to Component Instance Terminal *term*.

RTLTerminalGetFanoutList

Declaration

```
void* RTLTerminalGetFanoutList(void* term);
```

Description

Returns the list (*itr*) of terminals connected (fanout) to Component Instance Terminal *term*.

ScalarTerminalGetBitPosition

Declaration

```
int ScalarTerminalGetBitPosition(void* obj);
```

Description

Returns the bit position of the scalar terminal in its parent vector terminal.

ScalarTerminalGetConnectionList

Declaration

```
void* ScalarTerminalGetConnectionList(void* term);
```

Description

Returns the list (*itr*) of connections for Scalar Component Instance Terminal *term* or termbit *term* of Vector Component Instance Terminal.

ScalarTerminalGetMoreBackRef

Declaration

```
const char* ScalarTerminalGetMoreBackRef(void* term);
```

Description

Returns string containing more information about the terminal *term*, if any (added during elaboration).

ScalarTerminalGetOverriddenConnectionList

Declaration

```
const char* ScalarTerminalGetOverriddenConnectionList(void* term);
```

Description

Returns the list of overridden connections for the terminal *term*, if any.

ScalarTerminalGetStatus

Declaration

```
const char* ScalarTerminalGetStatus(void* term);
```

Description

Returns the status value of the scalar terminal *term*. Possible values are not connected, connected, tieoff, default tieoff, temporary tieoff, open, temporary open, default open, and glue.

TerminalGetParent

Declaration

```
void* TerminalGetParent(void* term);
```

Description

Returns the Parent of Component Instance Terminal *term*.

TerminalGetPort

Declaration

```
void* TerminalGetPort(void* term);
```

Description

Returns the Component Port corresponding to Component Instance Terminal *term*.

TerminalIsScalar

Declaration
```
int TerminalIsScalar(void* term);
```

Description

Returns 1 if Component Instance Terminal *term* is a scalar terminal. Otherwise, returns 0.

VectorTerminalGetConnectionList

Declaration
```
void* VectorTerminalGetConnectionList(void* term);
```

Description

Returns the list (*itr*) of Connections of Vector Component Instance Terminal *term*.

VectorTerminalGetLSB

Declaration
```
const char* VectorTerminalGetLSB(void* term);
```

Description

Returns the LSB of Vector Component Instance Terminal *term*.

VectorTerminalGetMSB

Declaration
```
const char* VectorTerminalGetMSB(void* term);
```

Description

Returns the MSB of Vector Component Instance Terminal *term*.

VectorTerminalGetNthScalarTerminal

Declaration
```
void* VectorTerminalGetNthScalarTerminal  (void* term, int N);
```

Description

Returns the *N*th Termbit (by index number) of Vector Component Instance Terminal *term*.

VectorTerminalGetScalarTerminalList

Declaration

```
void* VectorTerminalGetScalarTerminalList(void* term);
```

Description

Returns the list (*itr*) of termbits of Vector Component Instance Terminal *term*.

TieOff Connection

The TieOff Connection Type Item object has the following design hierarchy:

comRoot -> Design -> Component Instance -> Connection -> TieOff Connection

comRoot -> Design -> Connection -> TieOff Connection

Use the following API functions to process the TieOff Connection Type Item objects:

TieOffConnectionGetTieOffValue

Declaration

```
const char* TieOffConnectionGetTieOffValue  (void* tconn);
```

Description

Returns the Tie-off Value for TieOff Connection *tconn*.

TypeInfo

The TypeInfo object has the following design hierarchy:

comRoot -> Component -> Parameter -> TypeTree -> TypeInfo

comRoot -> Design -> Component Instance -> Parameter -> TypeTree -> TypeInfo

Use the following API functions to process the TypeInfo objects:

TypeInfoGetParamName

Declaration
```
const char* TypeInfoGetParamName(void* typeInfo);
```

Description

Returns the Parameter Name of the TypeInfo item *typeInfo*.

TypeInfoGetType

Declaration
```
void* TypeInfoGetType(void* typeInfo);
```

Description

Returns the Type of the TypeInfo item *typeInfo*.

TypeInfoGetValueList

Declaration
```
void* TypeInfoGetValueList(void* typeInfo);
```

Description

Returns the list (*itr*) of Values of the TypeInfo item *typeInfo*.

TypeTree

The TypeTree object has the following design hierarchy:

comRoot -> Component -> Parameter -> TypeTree

comRoot -> Design -> Component Instance -> Parameter -> TypeTree

Use the following API functions to process the TypeTree objects:

TypeTreeGetParameterLevel

Declaration

```
int TypeTreeGetParameterLevel(void* typeTree);
```

Description

Returns the Parameter Level for the TypeTree item *typeTree*.

TypeTreeGetParamName

Declaration

```
const char* TypeTreeGetParamName(void* typeTree);
```

Description

Returns the Parameter Name for the TypeTree item *typeTree*.

TypeTreeGetParamTypeList

Declaration

```
void* TypeTreeGetParamTypeList(void* typeTree);
```

Description

Returns the list (*itr*) of Parameter Types for the Type Tree item *typeTree*.

TypeTreeGetParamTypeTree

Declaration

```
void* TypeTreeGetParamTypeTree(void* typeTree);
```

Description

Returns the Parameter TypeTree for the TypeTree item *typeTree*.

TypeTreeGetParamTypeTreeByName

Declaration

```
void* TypeTreeGetParamTypeTreeByName  (void* typeTree, const char* Name);
```

Description

Returns the Named Parameter Typetree *name* for the TypeTree item *typeTree*.

TypeTreeGetPerlFunc

Declaration

```
const char* TypeTreeGetPerlFunc(void* typeTree);
```

Description

Returns the Name of the Perl function for the TypeTree item *typeTree*.

This Perl function helps to activate/deactivate the corresponding value tree.

ValueTree

The ValueTree object has the following design hierarchy:

comRoot -> Component -> Parameter -> ValueTree

comRoot -> Design -> Component Instance -> Parameter -> ValueTree

Use the following API functions to process the ValueTree objects:

ValueTreeFillValue

Declaration

```
void ValueTreeFillValue  (void* valueTree, const char* value);
```

Description

Sets the specified value *value* for the Value Tree item *valueTree*.

Arguments

`valueTree`

ValueTree object returned by the ParameterGetValueTreeByName/ ParameterGetValueTreeList function.

`value`

Value specified for the ValueTree item.

Returns

Nothing

ValueTreeGetActualValue

Declaration

```
const char* ValueTreeGetActualValue(void* valueTree);
```

Description

Returns the Actual Value of the ValueTree item *valueTree*.

Arguments

`valueTree`

ValueTree object returned by the ParameterGetValueTreeByName/ ParameterGetValueTreeList function.

Returns

Actual Value of the ValueTree item *valueTree*.

ValueTreeGetBasicType

Declaration

```
const char* ValueTreeGetBasicType(void* valueTree);
```

Description

Returns the Basic Object Type of the ValueTree item *valueTree*.

Arguments

`valueTree`

ValueTree object returned by the ParameterGetValueTreeByName/ ParameterGetValueTreeList function.

Returns

Basic Object Type of the ValueTree item *valueTree*.

ValueTreeGetChildByName

Declaration

```
void* ValueTreeGetChildByName  (void* valueTree, const char* childName);
```

Description

Returns the Named Child *childName* of the ValueTree item *valueTree*.

Arguments

`valueTree`

ValueTree object returned by the ParameterGetValueTreeByName/ ParameterGetValueTreeList function.

`childName`

Child of the ValueTree item.

Returns

Child *childName* of the ValueTree item *valueTree*.

ValueTreeGetChildList

Declaration

```
void* ValueTreeGetChildList(void* valueTree);
```

Description

Returns the list (*itr*) of Child Names for the ValueTree item *valueTree*.

Arguments

`valueTree`

ValueTree object returned by the ParameterGetValueTreeByName/ ParameterGetValueTreeList function.

Returns

List (*itr*) of Child Names for the ValueTree item *valueTree*.

ValueTreeGetChildNum

Declaration

```
int ValueTreeGetChildNum(void* valueTree);
```

Description

Returns the Number of Children of the ValueTree item *valueTree*.

Arguments

```
valueTree
```

ValueTree object returned by the ParameterGetValueTreeByName/ ParameterGetValueTreeList function.

Returns

Number of Children of the ValueTree item *valueTree*.

ValueTreeGetDependencyList

Declaration

```
void* ValueTreeGetDependencyList(void* valueTree);
```

Description

Returns the list (*itr*) of Dependency Items for the ValueTree item *valueTree*.

Arguments

```
valueTree
```

ValueTree object returned by the ParameterGetValueTreeByName/ ParameterGetValueTreeList function.

Returns

List (*itr*) of Dependency Items for the ValueTree item *valueTree*.

ValueTreeGetEnumList

Declaration

```
void* ValueTreeGetEnumList(void* valueTree);
```

Description

Returns the list (*itr*) of Enum items for the ValueTree item *valueTree*.

Arguments

`valueTree`

ValueTree object returned by the ParameterGetValueTreeByName/ ParameterGetValueTreeList function.

Returns

List (*itr*) of Enum items for the ValueTree item *valueTree*.

ValueTreeGetEnumSize

Declaration

```
int ValueTreeGetEnumSize(void* valueTree);
```

Description

Returns the number of Enums for the ValueTree item *valueTree*.

Arguments

`valueTree`

ValueTree object returned by the ParameterGetValueTreeByName/ ParameterGetValueTreeList function.

Returns

Number of Enums for the ValueTree item *valueTree*.

ValueTreeGetIsConst

Declaration

```
Int ValueTreeGetIsConst(void* valueTree);
```

Description

Returns the value of the const field for a parameter

Arguments

`valueTree`

ValueTree object returned by the ParameterGetValueTreeByName/ ParameterGetValueTreeList function.

Returns

Value of the const field for a parameter. It returns 1 for Y and 0 for N.

ValueTreeGetIsRTL

Declaration

```
Int ValueTreeGetIsRTL(void* valueTree);
```

Description

Returns the value of the RTL field for a parameter

Arguments

`valueTree`

ValueTree object returned by the ParameterGetValueTreeByName/ ParameterGetValueTreeList function.

Returns

Value of the RTL field for a parameter. It returns 1 for Y and 0 for N.

ValueTreeGetParamInfo

Declaration

```
void* ValueTreeGetParamInfo(void* valueTree);
```

Description

Returns the Parameter Information for the ValueTree item *valueTree*.

Arguments

`valueTree`

ValueTree object returned by the ParameterGetValueTreeByName/ ParameterGetValueTreeList function.

Returns

Parameter Information for the ValueTree item *valueTree*.

ValueTreeGetParamName

Declaration

```
const char* ValueTreeGetParamName(void* valueTree);
```

Description

Returns the Parameter Name for the ValueTree item *valueTree*.

Arguments

`valueTree`

ValueTree object returned by the ParameterGetValueTreeByName/ ParameterGetValueTreeList function.

Returns

Name of the Parameter for the ValueTree item *valueTree*.

Notes

Use the Parameter-related API functions to process parameter objects.

ValueTreeGetType

Declaration

```
const char* ValueTreeGetType(void* valueTree);
```

Description

Returns the TypeTree of the ValueTree item *valueTree*.

Arguments
`valueTree`

ValueTree object returned by the ParameterGetValueTreeByName/ ParameterGetValueTreeList function.

Returns

TypeTree of the ValueTree item *valueTree*.

Notes

Use the TypeTree-related API functions to process TypeTree objects.

ValueTreeGetValue

Declaration
```
const char* ValueTreeGetValue(void* valueTree);
```

Description

Returns the Value of the ValueTree item *valueTree*.

Arguments
`valueTree`

ValueTree object returned by the ParameterGetValueTreeByName/ ParameterGetValueTreeList function.

Returns

Value of the ValueTree item *valueTree*.

Wire Port OnMaster Info

The Wire Port OnMaster Info object has the following design hierarchy:

comRoot -> Component -> Interface -> Logical Port -> Constraints Info -> Wire Port OnMaster Info

comRoot -> Design -> Interface -> Logical Port -> Constraints Info -> Wire Port OnMaster Info

Use the following API functions to process the Wire Port OnMaster Info objects:

ConstraintsInfoGetOnMasterInfo

Declaration
```
const char* WirePortOnMasterGetPresence(void* wirePortOnMasterInfo )
```

Description

Returns presence constraint information of the specified object.

WirePortOnMasterGetDirection

Declaration
```
const char * WirePortOnMasterGetDirection(void* wirePortOnMasterInfo )
```

Description

Returns direction constraint information of the specified object.

WirePortOnMasterGetWidth

Declaration
```
const char * WirePortOnMasterGetWidth(void* wirePortOnMasterInfo )
```

Description

Returns width constraint information of the specified object.

Wire Port OnSlave Info

The Wire Port OnSlave Info object has the following design hierarchy:

comRoot -> Component -> Interface -> Logical Port -> Constraints Info -> Wire Port OnSlave Info

comRoot -> Design -> Interface -> Logical Port -> Constraints Info -> Wire Port OnSlave Info

Use the following API functions to process the Wire Port OnSlave Info objects:

WirePortOnSlaveGetDirection

Declaration
```
const char * WirePortOnSlaveGetDirection(void* wirePortOnSlaveInfo)
```

Description

Returns direction constraint information of the specified object.

WirePortOnSlaveGetPresence

Declaration
```
const char* WirePortOnSlaveGetPresence(void* wirePortOnSlaveInfo)
```

Description

Returns presence constraint information of the specified object.

WirePortOnSlaveGetWidth

Declaration
```
const char * WirePortOnSlaveGetWidth(void* wirePortOnSlaveInfo)
```

Description

Returns width constraint information of the specified object.

Wire Port OnSystem Info

The Wire Port OnSystem Info object has the following design hierarchy:

comRoot -> Component -> Interface -> Logical Port -> Constraints Info -> Wire Port OnSystem Info

comRoot -> Design -> Interface -> Logical Port -> Constraints Info -> Wire Port OnSystem Info

Use the following API functions to process the Wire Port OnSystem Info objects:

WirePortOnSystemGetDirection

Declaration

```
const char * WirePortOnSystemGetDirection(void* wirePortOnSystemInfo)
```

Description

Returns direction constraint information of the specified object.

WirePortOnSystemGetGroup

Declaration

```
const char* WirePortOnSystemGetGroup(void* wirePortOnSystemInfo)
```

Description

Returns OnSystem group information of the specified object.

WirePortOnSystemGetPresence

Declaration

```
const char* WirePortOnSystemGetPresence(void* wirePortOnSystemInfo)
```

Description

Returns presence constraint information of the specified object.

WirePortOnSystemGetWidth

Declaration

```
const char * WirePortOnSystemGetWidth(void* wirePortOnSystemInfo)
```

Description

Returns width constraint information of the specified object.

BaseClass API Functions

BaseClass and its related type store the common information about GenSys objects.

API functions relating to BaseClass and its related types are available in the following categories:

- BaseClass
- Attribute
- DataSourceClass
- VersionedNameClass

BaseClass

The BaseClass object stores the common data for all object types and has the following design hierarchy:

comRoot -> BaseClass

Use the following API functions to process the BaseClass objects:

BaseClassAddAttribute

Declaration

```
int BaseClassAddAttribute  (void* obj, const char* attrName, const char*
type,  const char* value, const char* category,  const char* validrange,
const char* help);
```

Description

Adds a new attribute *attrName* to object *obj*.

Other fields specify the properties of the attribute being added.

BaseClassGetAttributeByName

Declaration

```
void* BaseClassGetAttributeByName  (void* obj, const char* attrName);
```

Description

Returns the attribute *attrName* for object *obj*.

BaseClassGetAttributesList

Declaration

```
void* BaseClassGetAttributesList(void* obj);
```

Description

Returns the list (*itr*) of attributes of object *obj*.

BaseClassGetDataSource

Declaration

```
void* BaseClassGetDataSource(void* obj);
```

Description

Returns the Data Source object.

BaseClassGetDescription

Declaration

```
const char* BaseClassGetDescription(void* obj);
```

Description

Returns the un-formatted description of object *obj*.

BaseClassGetFormattedDescription

Declaration

```
const char* BaseClassGetFormattedDescription(void* obj);
```

Description

Returns the formatted description of object *obj*.

For example, "Bold" string would be returned as "@b@Bold@/b@" string token.

BaseClassGetGroupName

Declaration

```
const char* BaseClassGetGroupName(void* obj);
```

Description

Returns the name of the group of object *obj*.

BaseClassGetShortName

Declaration

```
const char* BaseClassGetShortName(void* obj);
```

Description

Returns the short name of object *obj*.

BaseClassNodeGetParent

Declaration

```
void* BaseClassNodeGetParent(void* obj);
```

Description

Returns the parent object of the given object *obj*.

Attribute

The Attribute object has the following design hierarchy:

comRoot -> BaseClass -> Attribute

Use the following API functions to process the Attribute objects:

AbstractBaseGetAttrValue

Declaration

```
const char* AbstractBaseGetAttrValue  (void * obj, const char* attr);
```

Description

Returns the value of the attribute *attr* for the object *obj*.

AttributeGetCategory

Declaration

```
const char* AttributeGetCategory(void* attr);
```

Description

Returns the Category of Attribute *attr*.

AttributeGetEnumsList

Declaration

```
void* AttributeGetEnumsList(void* attr);
```

Description

Returns the list (*itr*) of ENUM choices for Attribute *attr*.

AttributeGetHelp

Declaration

```
const char* AttributeGetHelp(void* attr);
```

Description

Returns the Help text of Attribute *attr*.

AttributeGetName

Declaration

```
const char* AttributeGetName(void* attr);
```

Description

Returns the Name of Attribute *attr*.

AttributeGetStrScalarValue

Declaration

```
const char* AttributeGetStrScalarValue(void* attr);
```

Description

Returns the String scalar value of Attribute *attr*.

AttributeGetType

Declaration

```
const char* AttributeGetType(void* attr);
```

Description

Returns the Type of Attribute *attr*.

AttributeGetValue

Declaration

```
const char* AttributeGetValue(void* attr);
```

Description

Returns the Value of Attribute *attr*.

DataSourceClass

The DataSourceClass object has the following design hierarchy:

comRoot -> BaseClass -> DataSourceClass

Use the following API functions to process the DataSourceClass objects:

DataSourceGetChangeDateTime

Declaration

```
const char* DataSourceGetChangeDateTime  (void* dataSource);
```

Description

Returns the change date time for data source *dataSource*.

DataSourceGetChangeMethod

Declaration

```
const char* DataSourceGetChangeMethod  (void* dataSource);
```

Description

Returns the change method for data source *dataSource*.

DataSourceGetChangeReason

Declaration

```
const char* DataSourceGetChangeReason  (void* dataSource);
```

Description

Returns the change reason for data source *dataSource*.

DataSourceGetChangeType

Declaration

```
const char* DataSourceGetChangeType(void* dataSource);
```

Description

Returns the change type for data source *dataSource*.

DataSourceGetCreationDate

Declaration

```
const char* DataSourceGetCreationDate(void* dataSource);
```

Description

Returns the date of creation for data source *dataSource*.

DataSourceGetDataIsFrozen

Declaration

```
const char* DataSourceGetDataIsFrozen  (void* dataSource);
```

Description

Returns the Data Frozen value for data source *dataSource*.

DataSourceGetPersonName

Declaration

```
const char* DataSourceGetPersonName(void* dataSource);
```

Description

Returns the user name for data source *dataSource*.

VersionedNameClass

The VersionedNameClass object has the following design hierarchy:

comRoot -> BaseClass -> VersionedNameClass

Use the following API functions to process the VersionedNameClass objects:

VersionedNameClassGetLibrary

Declaration
```
const char* VersionedNameClassGetLibrary(void* vnlv);
```

Description

Returns the Library Name of the versioned item *vnlv*.

VersionedNameClassGetName

Declaration
```
const char* VersionedNameClassGetName(void* vnlv);
```

Description

Returns the Name of the versioned item *vnlv*.

VersionedNameClassGetVendorName

Declaration
```
const char* VersionedNameClassGetVendorName  (void* vnlv);
```

Description

Returns the Vendor Name of the versioned item *vnlv*.

VersionedNameClassGetVersionNumber

Declaration
```
const char* VersionedNameClassGetVersionNumber  (void* vnlv);
```

Description

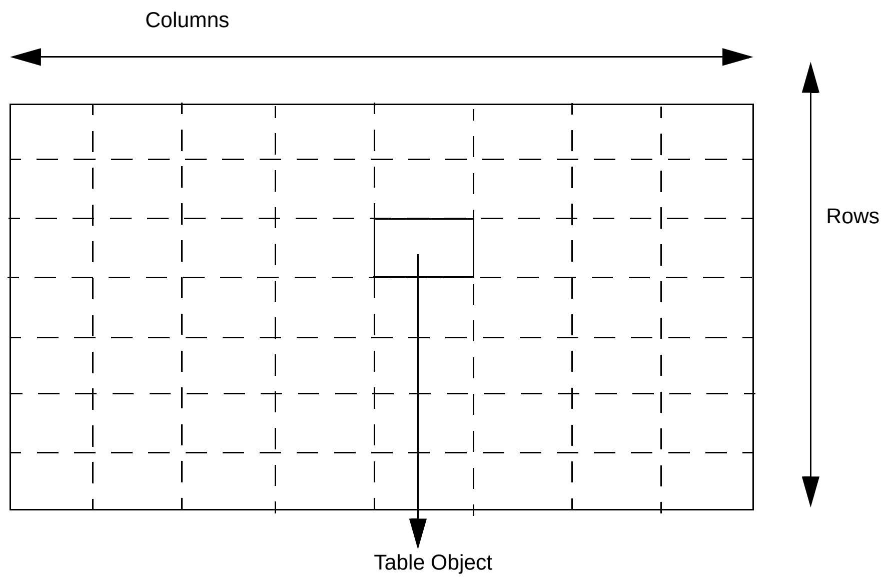
Returns the Version number of the versioned item *vnlv*.

Table Manipulation API Functions

This section provides the details about the various table APIs.

Table API Structure

The following figure depicts the structure of the table APIs:



comTableClass indicates the whole table
comTableValue indicates each

Following are the object names used for the table APIs:

Object Type	Description
comTableClass	Indicates the table in the design, component, or interface. Or (Derived from comTableValue) Indicates the cell, which is a subtable.
comTableValue	Indicates each table object present in the table.
comCellData	(Derived from comTableValue) Indicates the cell data type (INTEGER, STRING, or ENUM) in the table.

The *comCellData* object and *comTableClass* object have the following design hierarchy:

```
comTableClass -> comTableValue -> comCellDatacomTableClass ->
comTableValue -> comTableClass
```

General Table APIs

Following are some general table APIs:

API Function	Purpose
GetObjectType	Returns the object type
ComponentGetThirdPartyGenerator/ComponentGetTableList	Returns the named table/table list of a component
DesignGetTableByName/ DesignGetTableList	Returns the named table/table list of a design
InterfaceDefGetTableByName/ InterfaceDefGetTableList	Returns the named table/table list of an interface definition
ComponentInstanceGetTableByName	Returns the named table of a component instance

Note:

If a table name is not unique, you need to specify the full hierarchical (dot-separated) name. For example, when table A exists in both design.FOO1 and design.FOO2, you need to specify these tables as design.FOO1.A and design.FOO2.A respectively. Specifying design.A results in an error.

comTableValue API Functions

Use the following API functions to process the comTableValue objects:

comTableValueGetParent

Declaration

```
void* comTableValueGetParent(void* comTableValue);
```

Description

Returns the Parent Table for the cell *comTableValue*.

comTableValueGetIntScalarInScope

Declaration

```
int comTableValueGetIntScalarInScope  (void* comTableValue, const char*
scope);
```

Description

Returns the Integer value for the cell *comTableValue*.

Table Property APIs

TableCellGetType

Declaration

```
const char* TableCellGetType(void* table);
```

Description

Returns the type of the table cell data of table *table* (specified in the table schema) as INT, BOOL, STRING, ENUM, TABLE, or PERL.

TableGetName

Declaration

```
const char* TableGetName(void* table);
```

Description

Returns the name of table *table*.

TableGetNumColumns

Declaration

```
int TableGetNumColumns(void* table);
```

Description

Returns the number of columns in table *table*.

The number of columns may change dynamically if the table can be evaluated.

TableGetNumRows

Declaration

```
int TableGetNumRows(void* table);
```

Description

Returns the number of rows in table *table*.

TableGetSchema

Declaration

```
void* TableGetSchema(void* table);
```

Description

Returns table schema for the table *table*.

TableGetTableID

Declaration

```
int TableGetTableID(void* table);
```

Description

Returns the Table ID of table *table* (specified in the table schema).

TableIsEvaluatable

Declaration
```
int TableIsEvaluatable(void* table);
```

Description

Returns 1 if table *table* has any evaluatable property (specified in the table schema). Otherwise, returns 0.

TableIsModified

Declaration
```
int TableIsModified(table);
```

Description

Returns 1 if the table *table* has some changed local data. Otherwise, returns 0.

Use the TableIsModified function to decide whether the table needs to be saved.

TableIsReadOnly

Declaration
```
int TableIsReadOnly(void* table);
```

Description

Returns 1 if the table *table* is a read-only table (specified in the table schema).

Note:
Currently, the read-only property of table only affects the GUI view. API functions can change the values in all tables including read-only tables.

TableIsRowEmpty

Declaration
```
int TableIsRowEmpty(void* obj, int rowNum);
```

Description

Returns 1 if the row *rowNum* of the table *obj* is empty. Otherwise, returns 0.

TableIsTab

Declaration

```
int TableIsTab(void* table);
```

Description

Returns 1 if the table *table* is a subtable of another table.

TableSetReadOnly

Declaration

```
void TableSetReadOnly  (void* obj, const char* readonlyflag)
```

Description

Sets the table *obj* and all its subtable as read-only, when the *readonlyflag* is true. When the *readonlyflag* is false, the table *obj* and all its subtables will be write-enabled in the GUI.

Both the *obj* and *readonlyflag* arguments are mandatory.

Table Access APIs

TableGetColumnData

Declaration

```
void* TableGetColumnData(void* table, const char* colName);
```

Description

Returns the list (*itr*) of row data (type *comTableValue*) of Column *colName* of table *table*.

TableGetColumnNameList

Declaration

```
void* TableGetColumnNameList(void* table);
```

Description

Returns the list (*itr*) of Table Column names for table *table*.

TableGetData

Declaration

```
void* TableGetData  (void* table, const char* colName, int rowNum);
```

Description

Returns the table data for column *colName* and row *rowNum*.

Returns the Perl variable undef if the specified cell has NULL data.

TableGetRowData

Declaration

```
void* TableGetRowData(void* table, int rowNum);
```

Description

Returns the list (*itr*) of column data (type *comTableValue*) of all columns for row *rowNum* of table *table*.

TableGetStrData

Declaration

```
const char* TableGetStrData  (void* obj, const char* colName, int
rowNum);
```

Description

Returns the string data for the column *colName* and row *rowNum* of table *obj*.

TableValue and CellData APIs

TableCellGetEnumValues

Declaration

```
void* TableCellGetEnumValues(void* tableCellData);
```

Description

Returns the complete list of ENUM values stored in table cell *tableCellData* that is an ENUM type table cell.

TableCellGetIntData

Declaration

```
int TableCellGetIntData(void* tableCellData);
```

Description

Returns the Integer-equivalent of data stored in table cell *tableCellData*.

If the cell data is an Integer value, that value is returned.

If the cell data is a string value, such as P, and there is a parameter called P defined in the component or component instance, the current value of parameter P is returned.

If the cell data is an ENUM data, the ENUM integer (not the ENUM string) is returned.

If the cell data is a Perl Expression, the integer value from the evaluated Perl expression is returned or -1 is returned if integer value is not determinable.

TableCellGetLongLongIntData

Declaration

```
const char* TableCellGetLongLongIntData  (void* tableCellData);
```

Description

Returns an integer-equivalent value of data stored in table cell *tableCellData* as String.

Depending on the type of data stored in table cell, different values will be returned, as shown in the following table.

Table Cell Data Type	Returned Value
String value in the form of Ks, Ms, or Gs. For example 1K, 3M	Integer-equivalent value of data as string
String value, which is name of a parameter P defined in the component or component instance	Integer-equivalent value of parameter P as string
Perl expression	Integer-equivalent value of perl expression as string or -1 if integer value is not determinable
Invalid value	-1

TableCellGetStringData

Declaration

```
const char* TableCellGetStringData  (void* tableCellData);
```

Description

Returns the string-equivalent of the data stored in table cell *tableCellData*.

If the cell data is an integer value, such as 5, the string equivalent "5" is returned. If the cell data is an ENUM data, the ENUM string (not the ENUM integer) is returned. If the cell is empty, undef is returned.

TableCellIsEmpty

Declaration

```
int TableCellIsEmpty(void* CellData)
```

Description

Returns 1 if the table cell *CellData* is empty. Otherwise, returns 0.

TableCellIsEnum

Declaration

```
int TableCellIsEnum(void* tableCellData);
```

Description

Returns 1 if data stored in table cell *tableCellData* is ENUM type data. Otherwise, returns 0.

TableCellIsInteger

Declaration

```
int TableCellIsInteger(void* tableCellData);
```

Description

Returns 1 if data stored in table cell *tableCellData* is Integer type data. Otherwise, returns 0.

TableCellIsString

Declaration

```
int TableCellIsString(void* tableCellData);
```

Description

Returns 1 if data stored in table cell *tableCellData* is string type data. Otherwise, returns 0.

TableCellIsValidData

Declaration

```
int TableCellIsValidData(void* CellData *)
```

Description

Returns 1 if the data stored in the table cell *CellData* is valid. Otherwise, returns 0.

This API checks the current data to see if it is a valid data for the cell. Validity is checked from the schema VALID function.

TableValidate

Declaration

```
int TableValidate(void* table)
```

Description

Validates the given table and returns the number of invalid cells in the table *table*. Otherwise, returns 0.

This API checks all the cells in the table to see if they contain valid data. Validity is checked from the schema VALID function.

TableValueIsTable

Declaration

```
int TableValueIsTable(void* tableValue);
```

Description

Returns 1 if table cell *tableValue* is a sub-table of another table. Otherwise, returns 0.

Table Modification APIs

RefreshGuiTables

Declaration

```
void RefreshGuiTables(t_ptr_list);
```

Description

Refreshes the specified tables *t_ptr_list* (comma-separated list of table pointers).

Normally, the tables displayed in the GenSys GUI are refreshed when they are saved. Use the RefreshGuiTables function to refresh the specified tables without needing to save.

TableDeleteRow

Declaration

```
int TableDeleteRow(void* table, int rowNum);
```

Description

Deletes row *rowNum* of table *table*.

Returns 1 if row deletion is successful.

TableEvaluate

Declaration
```
void* TableEvaluate(void* table);
```

Description

Evaluates the dynamic table *table* and returns the new table if the table *table* could be evaluated. Otherwise, returns the original table *table*.

TableInsertRowAfter

Declaration
```
int TableInsertRowAfter(void* table, int rowNum);
```

Description

Inserts a new row after row *rowNum* of the current column.

TableInsertRowBefore

Declaration
```
int TableInsertRowBefore(void* table, int rowNum);
```

Description

Inserts a new row before row *rowNum* of the current column.

TableRowSetReadOnly

Declaration
```
void TableRowSetReadOnly  (void *tablePtr, int rowNo, const char*
readonlyflag)
```

Description

Sets a row <*rowNo*> of a table <*tablePtr*> in GUI as read-only if <*readonlyflag*> is set to true. If the <*readonlyflag*> is set to false, the row is set as write-enabled.

All the subtables of the table <*tablePtr*> corresponding to the row <*rowNo*> are also made read-only or write-enabled depending on the value of <*readonlyflag*>.

No row of the table other than <*rowNo*> is affected.

TableSelectRow

Declaration
```
void* TableSelectRow(void* table, valueList);
```

Description

Selects the list (*itr*) of rows where the specified columns have the specified values.

The <*value-list*> is a comma-separated list of column names and their values in the following format:

```
<col1-name> => <value1>, <col2-name> => <value2>,...
```

Thus, the TableSelectRow functions returns those rows where column <*col1-name*> has value <*value1*>, column <*col2-name*> has value <*value2*>, and so on.

TableSelectRowFirst

Declaration
```
int TableSelectRowFirst(void* table, valueList);
```

Description

Selects the first row where the specified columns have the specified values.

The <*value-list*> is a comma-separated list of column names and their values in the following format:

```
<col1-name> => <value1>, <col2-name> => <value2>,...
```

Thus, the TableSelectRow functions returns the first row where column <*col1-name*> has value <*value1*>, column <*col2-name*> has value <*value2*>, and so on.

TableSetData

Declaration

```
const char* TableSetData  (void* table, const char* colName, int
rowNum,   const char* data);
```

Description

Sets the specified cell (*colName* by *rowNum*) of table *table* to data *data*.

TableUpdateRow

Declaration

```
int TableUpdateRow(void* table, int rowNum, valueList);
```

Description

Updates the table row *rowNum* for the specified columns.

The <*value-list*> is a comma-separated list of column names and their values in the following format:

```
<col1-name> => <value1>, <col2-name> => <value2>,...
```

Thus, row *rowNum* of column <*col1-name*> is updated with value <*value1*>, the row *rowNum* of column <*col2-name*> is updated with value <*value2*>, and so on.

Table Traversal APIs

TableValueGetEnclosingScope

Declaration

```
void* TableValueGetEnclosingScope(void* tableValue);
```

Description

Returns the parent object (design, component, instance, or interface) of the table cell *tableValue*.

TableValueGetParentColumn

Declaration

```
const char* TableValueGetParentColumn  (void* tableValue);
```

Description

Returns the name of the Parent Column for table cell *tableValue*.

TableValueGetParentRow

Declaration

```
int TableValueGetParentRow(void* tableValue);
```

Description

Returns the Parent Row for table cell *tableValue*.

This API function is valid for all table cells including table type cells.

TableValueGetParentTable

Declaration

```
void* TableValueGetParentTable(void* tableValue);
```

Description

Returns the Parent Table for table cell *tableValue*.

This API function is valid for all table cells including table type cells.

TableValueGetTable

Declaration

```
void* TableValueGetTable(void* tableValue);
```

Description

Returns pointer to the subtable, if the table cell *tableValue* is a table. Otherwise, returns NULL.

Schema API Functions

This section provides details about the Schema APIs.

Table Schema APIs

TableSchemaGetColumnSchemaByName

Declaration

```
void* TableSchemaGetColumnSchemaByName  (void* obj1, const char* colName,
void* tptr)
```

Description

Searches for the column schema by the name passed in the table schema *obj1*.

Returns pointer to the column schema, if found. Otherwise, the table *tptr* is evaluated and the column is searched again.

TableSchemaGetColumnSchemaList

Declaration

```
void* TableSchemaGetColumnSchemaList  (void* tableSchema);
```

Description

Returns the list (*itr*) of columns schemas for the table schema *tableSchema*.

TableSchemaGetEvalRowFunction

Declaration

```
const char* TableSchemaGetEvalRowFunction(void* obj);
```

Description

Returns the eval row function associated with a table schema *obj*.

TableSchemaGetFile

Declaration
```
const char* TableSchemaGetFile(void* tableSchema);
```

Description

Returns the filename in which the table schema *tableSchema* is defined.

TableSchemaGetHelp

Declaration
```
const char* TableSchemaGetHelp(void* tableSchema);
```

Description

Returns the help for the table schema *tableSchema*.

TableSchemaGetHiddenColumnList

Declaration
```
void* TableSchemaGetHiddenColumnList  (void* tableSchema);
```

Description

Returns the list (*itr*) of columns names that are hidden for table schema *tableSchema*.

TableSchemaGetHierarchicalName

Declaration
```
const char* TableSchemaGetHierarchicalName  (void* tableSchema);
```

Description

Returns the hierarchical name of table schema *tableSchema*.

TableSchemaGetKey

Declaration

```
const char* TableSchemaGetKey(void* tableSchema);
```

Description

Returns the name of the key column for the table schema *tableSchema*.

TableSchemaGetLine

Declaration

```
unsigned int tableSchemaGetLine(void* tableSchema);
```

Description

Returns the line number at which the table schema *tableSchema* is defined.

TableSchemaGetName

Declaration

```
const char* TableSchemaGetName(void* tableSchema);
```

Description

Returns the name of table schema *tableSchema*.

TableSchemaGetNumRows

Declaration

```
int TableSchemaGetNumRows (void* obj, void* tptr)
```

Description

Returns the number of rows if the tables has EVAL ROWS. If the table does not have any EVAL ROW Perl Function in the schema, it returns -1.

This API takes the table schema pointer *obj* and the table pointer *tptr*, evaluates the EVAL ROW Perl function, if any, associated with the table schema.

TableSchemaGetPersonalityPerlFunction

Declaration

```
char* TableSchemaGetPersonalityPerlFunction
  (void* obj);
```

Description

Returns the personality Perl function of the table schema pointer *obj*.

TableSchemaGetRowType

Declaration

```
const char* TableSchemaGetRowType(void* tableSchema);
```

Description

Returns type of rows for the table schema *tableSchema* (specified as SINGLE or MULTIPLE.) Returns Eval where the schema specifies ROW_TYPE :: perl function() for the table.

TableSchemaGetVersion

Declaration

```
const char * TableSchemaGetVersion(void* obj)
```

Description

Returns the VERSION field if specified in the Table Schema *obj*.

TableSchemaIsAutoFill

Declaration
```
int TableSchemaIsAutoFill(void* obj, int recursive);
```

Description

Specifies whether the table is Autofill from the table schema *obj*. If the int variable *recursive* is set to 1, then it does a recursive traversal, otherwise, there is no recursive traversal.

TableSchemaIsBaseTable

Declaration
```
int TableSchemaIsBaseTable(void* tableSchema);
```

Description

Returns 1 if the table schema *tableSchema* is a base table. Otherwise, returns 0.

TableSchemaIsEvalAllowed

Declaration
```
int TableSchemaIsEvalAllowed(void* obj);
```

Description

Recursively checks for evaluatable columns in the table schema. Returns 1 if the table schema *obj* has evaluatable columns. Otherwise, returns 0.

TableSchemaIsEvalColPresent

Declaration
```
int  TableSchemaIsEvalColPresent(void* obj)
```

Description

Returns 1 if the table schema *obj* has evaluatable columns. Otherwise, 0.

TableSchemaIsExtentionTable

Declaration

```
int TableSchemaIsExtentionTable(void* tableSchema);
```

Description

Returns 1 if the table schema *tableSchema* is an extension table. Otherwise, returns 0.

TableSchemaIsFlatTable

Declaration

```
int TableSchemaIsFlatTable(void* tableSchema);
```

Description

Returns 1 if the table schema *tableSchema* is a flat table. Otherwise, returns 0.

TableSchemaIsHidden

Declaration

```
int TableSchemaIsHidden(void* tableSchema);
```

Description

Returns 1 if the table schema *tableSchema* is hidden. Otherwise, returns 0.

TableSchemaIsReadOnly

Declaration

```
int TableSchemaIsReadOnly(void* tableSchema);
```

Description

Returns 1 if the table schema *tableSchema* is read-only. Otherwise, returns 0.

TableSchemaIsTab

Declaration
```
int TableSchemaIsTab(void* tableSchema);
```

Description

Returns 1 if the table schema *tableSchema* has been specified as tab. Otherwise, returns 0.

TableSchemaIsValidateAllowed

Declaration
```
int TableSchemaIsValidateAllowed(void* obj)
```

Description

Recursively searches for validation columns in the table schema. Returns 1 if validation columns in table schema are found. Otherwise, 0.

TableSchemaIsValidationColPresent

Declaration
```
int TableSchemaIsValidationColPresent(void* obj)
```

Description

Returns 1 if the table schema *obj* has a validation column. Otherwise, returns 0.

Column Schema APIs

ColumnSchemaAddValidPerlFunction

Declaration
```
int ColumnSchemaAddValidPerlFunction  (void* obj, const char*
functionName, int status)
```

Description

Adds a validate perl function, functionName, to the column schema, obj. In addition, the perl function is enabled if status is set to 1 or disabled if the status is set to 0.

ColumnSchemaGetAutoFillSchema

Declaration

```
void* ColumnSchemaGetAutoFillSchema(void* obj)
```

Description

Returns the Autofill Schema associated with a Column Schema *obj*.

ColumnSchemaGetCellEnumList

Declaration

```
void* ColumnSchemaGetCellEnumList(void* colSchema);
```

Description

Returns a list (*itr*) of enum values for column schema *colSchema*, when the column type is ENUM_TYPE.

ColumnSchemaGetCellType

Declaration

```
const char* ColumnSchemaGetCellType(void* colSchema);
```

Description

Returns the cell type of column schema *colSchema* (specified as INT_TYPE, BOOL_TYPE, STRING_TYPE, ENUM_TYPE, or TABLE_TYPE.)

ColumnSchemaGetChangePerlFunction

Declaration

```
const char* ColumnSchemaGetChangePerlFunction  (void* obj)
```

Description

Returns the change Perl function, if any, associated with the column schema *obj*.

ColumnSchemaGetDependList

Declaration

```
void* ColumnSchemaGetDependList(colSchema);
```

Description

Returns list (*itr*) of hierarchical column names on which the column schema *colSchema* is dependant.

ColumnSchemaGetFile

Declaration

```
const char* ColumnSchemaGetFile(void* colSchema);
```

Description

Returns the filename in which the column schema *colSchema* is defined.

ColumnSchemaGetHelp

Declaration

```
const char* ColumnSchemaGetHelp(void* colSchema);
```

Description

Returns the help of column schema *colSchema*.

ColumnSchemaGetHierarchicalName

Declaration

```
const char* ColumnSchemaGetHierarchicalName(void* colSchema);
```

Description

Returns the hierarchical name of column schema *colSchema*.

ColumnSchemaGetIsEval

Declaration

```
int ColumnSchemaGetIsEval(void* colSchema);
```

Description

Returns 1 if the column schema *colSchema* is specified as an evaluatable column. Otherwise, returns 0.

ColumnSchemaGetIsExtension

Declaration

```
int ColumnSchemaGetIsExtension(void* colSchema);
```

Description

Returns 1 if the column schema *colSchema* is an extension. Otherwise, returns 0.

ColumnSchemaGetIsHidden

Declaration

```
int ColumnSchemaGetIsHidden(void* colSchema);
```

Description

Returns 1 if the column schema *colSchema* is hidden. Otherwise, returns 0.

ColumnSchemaGetIsKey

Declaration

```
int ColumnSchemaGetIsKey(void* colSchema);
```

Description

Returns 1 if the column schema *colSchema* is a key column. Otherwise, returns 0.

ColumnSchemaGetIsReadOnly

Declaration

```
int ColumnSchemaGetIsReadOnly(void* colSchema);
```

Description

Returns 1 if the column schema *colSchema* is read-only. Otherwise, returns 0.

ColumnSchemaGetIsValidate

Declaration

```
int ColumnSchemaGetIsValidate(void* colSchema);
```

Description

Returns 1 if validate Perl function has been specified for column schema *colSchema*. Otherwise, returns 0.

ColumnSchemaGetKeyColumnList

Declaration

```
void* ColumnSchemaGetKeyColumnList  (void* obj, void* tptr)
```

Description

Returns the key columns for dynamic columns, associated with a Column Schema *obj* and table *tptr*.

ColumnSchemaGetLine

Declaration

```
unsigned int ColumnSchemaGetLine(void* colSchema);
```

Description

Returns the line number at which the column schema *colSchema* is defined.

ColumnSchemaGetName

Declaration

```
const char* ColumnSchemaGetName(void* colSchema);
```

Description

Returns the name of column schema *colSchema*.

ColumnSchemaGetPerlFunction

Declaration

```
const char* ColumnSchemaGetPerlFunction(void* obj)
```

Description

Returns the Perl function, if any, associated with the column schema *obj*.

ColumnSchemaGetPersonalityPerlFunction

Declaration

```
char* ColumnSchemaGetPersonalityPerlFunction  (void* obj)
```

Description

Returns the personality Perl function of the column schema pointer *obj*.

ColumnSchemaGetTableSchema

Declaration

```
void* ColumnSchemaGetTableSchema(void* colSchema);
```

Description

Returns table schema for column schema *colSchema* having type TABLE_TYPE.

ColumnSchemaGetValidPerlFunction

Declaration

```
const char* ColumnSchemaGetValidPerlFunction  (void* obj)
```

Description

Returns the validation Perl function, if any, associated with the Column Schema *obj*.

ColumnSchemaSetValidPerlFunctionStatus

Declaration

```
int ColumnSchemaSetValidPerlFunctionStatus  (void* obj, const char*
functionname, int status)
```

Description

Sets the status of the validate perl function, functionName, of the column schema, obj. The perl function is enabled if status is set to 1 or disabled if the status is set to 0.

AutoFill Schema APIs

AutoFillSchemaGetDefaultValue

Declaration

```
const char* AutoFillSchemaGetDefaultValue(void* obj)
```

Description

Returns the Default Value of the Autofill Schema *obj*.

AutoFillSchemaGetPerlFunction

Declaration

```
const char* AutoFillSchemaGetPerlFunction(void* obj)
```

Description

Returns the Perl Function of the Autofill Schema *obj*.

Parameter Manipulation API Functions

ParameterFillValueTree

Declaration

```
void ParameterFillValueTree(param, valList);
```

Description

Stores the values *valList* (comma-separated list of values) in the ValueTree of Parameter *param*.

Post-Elaboration API Functions

The following post-elaboration API functions are available:

RTLInstanceGetFaninList

Declaration

```
void* RTLInstanceGetFaninList(void* rtlInst);
```

Description

Returns the list (*itr*) of input/inout terminals of RTL instance *rtlInst*.

Vector terminals are scalarized and reported accordingly.

RTLInstanceGetFanoutList

Declaration

```
void* RTLInstanceGetFanoutList(void* rtlInst);
```

Description

Returns the list (*itr*) of output/inout terminals of RTL instance *rtlInst*.

Vector terminals are scalarized and reported accordingly.

RTLTerminalGetFanoutList

Declaration

```
void* RTLTerminalGetFanoutList(rtlTerm);
```

Description

Returns the list (*itr*) of terminals/ports that are in the immediate fanout of the output scalar terminal *rtlTerm*.

RTLTerminalGetFaninList

Declaration

```
void* RTLTerminalGetFaninList (rtlTerm);
```

Description

Returns the list (*itr*) of terminals/ports that are in the immediate fanin of the input scalar terminal *rtlTerm*.

Generated RTL API Functions

The following API functions for the generated RTL are available:

genRTLNetNamingConvention

Declaration

```
void genRTLNetNamingConvention(const char* FuncName);
```

Description

Specifies the net naming convention for the generated RTL that is the returned value of Perl function *funcName*.

genRTLPortNamingConvention

Declaration

```
void genRTLPortNamingConvention(const char* FuncName);
```

Description

Specifies the port naming convention (prefix) for the generated RTL that is the returned value of Perl function *funcName*.

GetRTLFileDumpPathName

Declaration

```
const char* GetRTLFileDumpPathName(void* obj);
```

Description

Returns the path where RTL files are generated for the given unique partition set. *obj* is the object stored in the list returned by API DesignGetRTLFilesSet.

Returns

The name of the path where RTL files are generated for *obj*.

GetRTLFileLanguage

Declaration

```
const char* GetRTLFileLanguage(void* obj);
```

Description

Returns the language of the RTL files generated for each unique partition set. *obj* is the object stored in the list returned by DesignGetRTLFilesSet() API.

Returns

The language of the RTL files for *obj*. For example, Verilog, VHDL, SV, or SystemC.

GetRTLFileName

Declaration

```
const char* GetRTLFileName(void* obj);
```

Description

Returns the RTL file name. obj is the object that is stored in the list returned by API DesignGetRTLFilesSet().

Returns

The name of the RTL file for *obj*.

GetRTLFilePartitionName

Declaration

```
const char* GetRTLFilePartitionName(void* obj);
```

Description

Returns underscore (_) separated name of the unique partition set used in RTL file generation. *obj* is the object stored in the list returned by DesignGetRTLFilesSet() API.

Returns

The name of the unique partition set for *obj*.

GetRTLFilesForEachSet

Declaration

```
void* GetRTLFilesForEachSet(void* obj);
```

Description

Returns the iterator of RTL files of the *obj* partition set. *obj* is the object stored in the list returned by API DesignGetRTLFilesSet().

Returns

Iterator to a list of all RTL files which are generated for the unique partition set *obj*.

getRTLOption

Declaration

```
const char* getRTLOption(const char *option);
```

Description

Returns value of RTL option *option* required for RTL generation. These options are set using the set_rtl_option Tcl command. Any of the following values can be passed as *option*:

- header_template

- rtl_comment
- params
- defval_rtldump
- vectorize_net
- print_conn_comment
- delta_delay

  Note:
  The option names should not be prefixed with a -. For example, to get the value for the header_template RTL option, the argument should be passed as header_template and not -header_template.

Returns

Value of the RTL option *option*.

Message API Functions

The following API functions can be used to suppress or un-suppress messages that appear in pbd.log file and GenSys console.

SuppressMessage

Declaration

```
void SuppressMessage(char* <messageDetails>);
```

Description

Suppresses a GenSys message with the given message details <*messageDetails*>*.* The suppressed message does not appear in the pbd.log file and console of the particular GenSys session.

Arguments

`messageDetails`

> If a GenSys internal message needs to be suppressed, <*messageDetails*> is set to the message ID of that message. However, if a user-defined message needs to be suppressed, <*messageDetails*> is set to a string value that has the following format:

<*pkgName*>-<*msgID*>

Returns

Nothing

Notes

The effect of the SuppressMessage function is for the current GenSys session only and will not be maintained for the next sessions.

There will not be any effect on a message that has the given message ID but belongs to other packages.

UnSuppressMessage

Declaration

```
void UnSuppressMessage(char* <messageDetails>);
```

Description

Un-suppresses a suppressed GenSys message with the given message details <*messageDetails*>. After this, the message appears in the pbd.log file and console of the particular GenSys session.

Arguments

`messageDetails`

> If a GenSys internal message needs to be unsuppressed, <*messageDetails*> is set to the message ID of that message. However, if a user-defined message needs to be unsuppressed, <*messageDetails*> is set to a string value that has the following format:

<*pkgName*>-<*msgID*>

Returns

Nothing

Notes

By default, all GenSys error and warning messages are in the unsuppressed state. If the message with the given <*msgID*> is already unsuppressed, there will not be any effect and the default behavior is maintained.

There will not be any effect on a suppressed message that has the given message ID but belongs to other packages.

Message Handler API Functions

The following API functions can be used to register or de-register a message handler using a Perl subroutine specified in genesis.pl file.

DeregisterMessageHandler

Declaration

```
void DeregisterMessageHandler();
```

Description

De-registers a message handler and the messages generated by the previously registered message handler is no longer reported.

Note that GenSys reports an error when the RegisterMessageHandler() API has been called twice without calling the DeregisterMessageHandler() API in between.

Returns

Nothing

Examples

Following example de-registers the message handler *myhandler*:

```
RegisterMessageHandler("myhandler");                        // register a
message handler named “myhandler”  // definition of perl subroutine
“myhandler”sub myhandler {    ...    ...    ...}
// Call a generator (such as, IO generator)
DeregisterMessageHandler();
```

RegisterMessageHandler

Declaration

```
void RegisterMessageHandler  (const char* <handlerName>);
```

Description

Registers the message handler <*handlerName*>. The <*handlerName*> Perl subroutine is called when a message is generated by GenSys in the pbd.log file.

Ensure that the Perl subroutine with the name <*handlerName*> has been defined in the given scope. Following arguments will be passed to the Perl subroutine (in the order specified) by GenSys:

- Filename, in which the message has been flagged
- Line number, at which the message has been flagged
- Message ID of the reported message
- Message ID prefix of the message
- Severity of the message
- Message string

Note that GenSys reports an error when the <*handlerName*> Perl subroutine is passed as ““ (NULL).

Returns

Nothing

Examples

Following example registers a message handler *myhandler*:

```
RegisterMessageHandler("myhandler");                        // registers
a message handler named “myhandler”  // definition of perl subroutine
“myhandler”sub myhandler {    my $file = $_[0];    my $line =
$_[1];    my $msgId = $_[2];    my $msgIdPrefix = $_[3];    my $severity
= $_[4];    my $message = $_[5];    printg("***File =
$file\n");    printg("***Line = $line\n");    printg("***MsgId =
$msgId\n");    printg("***MsgIdPrefix =
$msgIdPrefix\n");    printg("***Severity =
$severity\n");    printg("***Message = $message\n");    return;}
```

Perl Callback API Functions for Tcl Commands

The following API functions can be used to register Perl callbacks for Tcl commands.

RegisterPreCommand

Declaration

```
void RegisterPreCommand  (const char* cmdName, const char*
perlCallback,  int priority);
```

Description

Registers pre-command Perl callback function <*perlCallback*> for Tcl command <*cmdName*> with specified priority <*priority*>.

This executes the Perl callback function before executing the Tcl command for which the function is registered. Perl callback for pre-command is always executed. The Tcl command will be executed only on successful execution of the Perl callback for pre-command.

A Tcl command is passed as an argument to the registered callback function. The user is expected to get the active object on which the current callback is acting using the GetActiveObject API.

If two Perl callbacks are registered for the same command, the callback with lesser priority value is executed first.

Returns

None

Usage Notes

You can pass the Perl callback function in any of the following forms (for registering and de-registering Tcl commands):

```
RegisterPreCommand("cmdName",\&perlCallback,priority);
```

OR

```
RegisterPreCommand("cmdName","perlCallback",priority);
```

OR

```
$ref = \&perlCallback;RegisterPreCommand(cmdName,$ref,priority);
```

Limitations

Re-registeration/re-deregistration will generate error/warning if the previous registration/ deregistration has been done similar to the following examples:

```
RegisterPreCommand("cmdName",\&perlCallback,priority);
RegisterPreCommand("cmdName","perlCallback",priority);
```

Such re-registration for the same Tcl command will NOT display an error.

However, consider the following examples:

```
RegisterPreCommand("cmdName",\&perlCallback,priority);
$ref = \&perlCallback;RegisterPreCommand("cmdName",$ref,priority);
```

Re-registration as done in the above example will display an error.

Consider more examples:

If the Perl callback has been registered as shown below:

```
RegisterPreCommand("cmdName",\&perlCallback,priority);
```

Or

```
$ref = \&perlCallback;RegisterPreCommand("cmdName",$ref, priority);
```

And de-registered as shown below:

```
RegisterPreCommand("cmdName","perlCallback",-1);
```

This will display warning and de-regisration will not take place.

In this case, you will need to de-register as shown below:

```
RegisterPreCommand("cmdName",\&perlCallback,-1);
```

Or

```
$ref = \&perlCallback;RegisterPreCommand("cmdName",$ref, -1);
```

RegisterPostCommand

Declaration

```
void RegisterPostCommand  (const char* cmdName, const char*
perlCallback,  int priority, const char* execCondition);
```

Description

Register post-command Perl callback function <*perlCallback*> for the Tcl command <*cmdName*> with the specified priority <*priority*>. The <*execCondition*> can be 'ALWAYS' | 'SUCCESS' | 'FAILURE'.

The RegisterPostCommand API executes the Perl callback function after executing the registered Tcl command.

The Perl callback functions for post-command may be executed as per the value specified for <*execCondition*>. If you specify 'ALWAYS' as <*execCondition*>, the Perl callback is executed irrespective of whether the Tcl command execution failed or passed. If you specify 'FAILURE' as <*execCondition*>, the Perl callback is executed only on failure of the Tcl command execution. If you specify 'SUCCESS' as <*execCondition*>, the Perl callback is executed only on success of the Tcl command execution.

A Tcl command is passed as an argument to the registered callback function. The user is expected to get the active object on which the current callback is acting using the GetActiveObject API.

To de-register the post-command Perl callback, use the RegisterPostCommand API with -1 as priority.

Note:

The <execCondition> 'FAILURE' will work only when the command line option -stopTclInterpOnErrors is given. Otherwise, the Perl Callback will not execute and the tcl commands will return 0 and display error or warning message, where applicable.

If two Perl callbacks are registered for the same command, the callback with lesser priority value will be executed first.

Returns

None

Example

The following example registers a Perl API, func1, to be executed every time the add_instance Tcl command is executed.

```
RegisterPostCommand(“add_instance”,\&func1,0,'ALWAYS');
```

Limitations

The 'FAILURE' execute condition will work only when the command line option -stopTclInterpOnErrors is given. Otherwise, the Perl Callback will not execute and the tcl commands will return 0 and display error or warning message, where applicable.

API Functions to Get Active Objects

You can use the following API functions to get the active objects.

GetActiveFile

Declaration

```
const char* GetActiveFile();
```

Returns

Returns currently active RTL file which is being written by a generator.

GetActiveLang

Declaration

```
const char* GetActiveLang();
```

Returns

Returns language of the currently active RTL file as VHDL or VERILOG.

GetActiveObject

Declaration

```
void* GetActiveObject(const char* type);
```

Description

Returns pointer to the respective comObject of type *type*. *type* can have any of the following values:

design, component, register, bitfield, instance, interfacedef, typetree, interface, bitenum, port, connection, table, subtable, partitiontable, partitiontablehierarchy.

GetCurrentRow

Declaration

```
int GetCurrentRow();
```

Description

Returns current row number as integer.

Template Generator API Functions

You can use the following API functions to work with Template Generator.

pptif::RegisterPttInitVars

Declaration

```
pptif::RegisterPttInitVars(const char* config_param);
```

Description

Sets up additional initialization information for Template Generator. By default, a template is setup with the following configuration parameters:

```
ABSOLUTE => 1RELATIVE => 1COMPILE_DIR => <value of $TMPDIR, if defined,
               else CWD>COMPILE_EXT => '.ttc2'CACHE_SIZE => 64
```

You can add one or more configuration parameter config_param to this list or change values of the existing parameters using this API. You can specify config_param, as a name-value pair, in the following format:

<config_param_name> => *<config_param_value>*

For example, to define a directory for OUTPUT_PATH configuration parameter, so that you can generate more than one file from your template, using redirect, use the following API call:

```
pptif::RegisterPttInitVars( OUTPUT_PATH => '.' );
```

In the above example, the specified directory must exist. If it does not exist, it will not be created by the API.

To set more than one configuration parameter, you can call this API multiple times, but all such calls must be made before template generation, which depends on these values. To ensure this, you can make all such calls at the time of initialization of GenSys.

You can also wrap the call(s) in a subroutine, but you must make a call to that subroutine in the beginning of your Perl script. For example, genesis.pl.

Note:
To understand the use of this API, it is recommended to read and understand the Template Toolkit section on configuration at http://www.template-toolkit.org/docs/manual/Config.html.

pptif::RegisterPttVars

Declaration

```
pptif::RegisterPttVars(const char* var);
```

Description

Sets up additional variable information for Template Generator. By default, a template is setup with just one variable, $root, which is the root of the GenSys object model. You can add more variables or functions to this list using this API.

You can specify a comma-separated list of variables as name-value pairs, in the following format:

<var_name> => *<value>*

For example, if you want to add a constant called PI and a subroutine to give a random number, you can call the API as given below:

```
pptif::RegisterPttVars( PI => 3.14159, Random => sub { return rand; } );
```

To set more than one variable, you can call this API multiple times, but all such calls must be made before template generation, which depends on these variables. To ensure this, you can make all such calls at the time of initialization of GenSys.

You can also wrap the call(s) in a subroutine, but you must make a call to that subroutine in the beginning of your Perl script. For example, genesis.pl.

Note:

To understand the use of this API, it is recommended to read and understand the Template Toolkit section on configuration at http://www.template-toolkit.org/docs/manual/variables.html.

RTL database API Functions

During Verilog/VHDL RTL generation process, RTL components are created in the RTL database. These RTL components have a direct one-to-one correlation with COM components and designs except in some cases like partitioning. Partitioning might have intermediate RTL components that are not present in the COM database.

GenSys provides the following Perl APIs to access RTL database and to control the RTL generation process.

GetActiveRTLComponent

Declaration

```
void* GetActiveRTLComponent()
```

Description

```
Returns  the currently active RTL component pointer. The active RTL
component is set while generating RTL file for a component and is reset
to NULL when RTL generation is complete.
```

Returns

Pointer to the currently active RTL component

RTLComponentGetCOMComponent

Declaration

```
void* RTLComponentGetCOMComponent(void* obj)
```

Description

```
Returns the COM component/design pointer for the RTL component. In case
of partitioning, if RTL component does not have any corresponding COM
component, its parent COM component is returned.
```

Returns

Pointer to the COM component/design for the RTL component

RTLComponentGetName

Declaration

```
const char* RTLComponentGetName(void* obj)
```

Description

```
Returns name of the RTL component.
```

Returns

Name of the RTL component

RTLComponentGetRTLInstanceList

Declaration

```
void* RTLComponentGetRTLInstanceList(void* obj)
```

Description

```
Returns the list of RTL instances for the RTL component.
```

Returns

List of RTL instances for the RTL component

RTLComponentGetRTLPortList

Declaration

```
void* RTLComponentGetRTLPortList(void* obj)
```

Description

```
Returns list of RTL ports for the RTL component.
```

Returns

List of RTL ports for the RTL component

RTLInstanceGetMasterRTLComponent

Declaration

```
void* RTLInstanceGetMasterRTLComponent(void* obj)
```

Description

```
Returns master RTL component for the RTL instance.
```

Returns

Master RTL component for the RTL instance

RTLInstanceGetName

Declaration

```
const char* RTLInstanceGetName(void* obj)
```

Description

`Returns name of RTL instance`

Returns

RTL instance name

RTLInstanceGetParentRTLComponent

Declaration

```
void* RTLInstanceGetParentRTLComponent(void* obj)
```

Description

`Returns parent RTL component for the RTL instance.`

Returns

Parent RTL component for the RTL instance

RTLPortGetName

Declaration

```
const char* RTLPortGetName(void* obj)
```

Description

`Returns name of the RTL port.`

Returns

RTL port name

API Functions to Uniquify Names

For hierarchical designs, the complete hierarchical names of interface instances and registers in instances can be quite large (for e.g., SS0::SS1::inst1::interface1). For such large hierarchical element names (interface instances and registers), GenSys enables you to create short unique names.

Use the following API functions related with unique names:

ClearUniquifiedNames

Declaration

`void ClearUniquifiedNames()`

Description

`Clears all the previously created unique names.`

Returns

Nothing

GetHierNameFromUniqueName

Declaration

`const char* GetHierNameFromUniqueName(name)`

Description

`Returns full hierarchical name for the given unique name,` *`name`*`. If an invalid unique name is specified, GenSys returns "".`

Returns

Hierarchical name for the given unique name

GetUniqueNameFromHierName

Declaration

```
const char* GetUniqueNameFromHierName(void* name)
```

Description

`Returns unique name for the hierarchical name` *`name`*`. If an invalid hierarchical name is specified, GenSys returns "".`

Returns

Unique name for the given hierarchical name

UniquifyNames

Declaration

```
void UniquifyNames()
```

Description

`Create unique names for all the hierarchical names for interface instances and registers/memories.`

Returns

Nothing

Custom Report APIs

The Custom Report Perl APIs listed in this section are available in the gensysReport package. Use these for generating a formatted report output or for taking interactive user inputs. APIs in the report body function.

CreateDesignBrowser

Declaration

```
const char* CreateDesignBrowser (const char* design, const char* title)
```

Description

Displays the Hierarchy Design Browser dialog for the design. Use:

- title to specify the title of the Hierarchy Design Browser dialog
- design to specify the name of the design

Returns

Returns the full hierarchical name, separated by '::', of the selected instance. If the specified design is not found, this API returns an empty string.

Note:
 In batch mode, this API always returns an empty string.

CreateDetailsDialog

Declaration

```
bool CreateDetailsDialog(const char* design, const char* title)
```

Description

Displays the Check Box dialog to get the value to include detailed information in the report. Use:

- title to specify the title of the Check Box dialog
- design to specify the name of the design

Returns

Returns 1 if you checked the check box, else it returns 0.

Note:
In batch mode, this API always returns 1.

CreateHierarchyDialog

Declaration

```
bool CreateHierarchyDialog(const char* design, const char* $title)
```

Description

Displays the Check Box dialog to get the value to include hierarchy details in the report. Use:

- title to specify the title of the Check Box dialog
- design to specify the name of the design

Returns

Returns 1 if you selected the check box, else it returns 0.

Note:
In batch mode, this API always returns 1.

DesignBrowserResultPointer

Declaration

```
void* DesignBroserResultPointer()
```

Description

Display the Hierarchy Design Browser dialog for the active design.

Returns

Returns an array containing the following items:

- r_design: The reference to the selected master component of the instance. If the selected instance is the active design or if the active object is component, this variable returns the reference of the active object.
- type: The instance type can have the following values:
    - 0: Selected object is top design.
    - 1: Active object is component.
    - -1: Selected object is an instance.
    - r_instance: Reference to the selected instance. If no instance is selected, this value is NULL.

SetReportFileName

Declaration

```
void SetReportFileName(const char * fileName, const char * command)
```

Description

Sets the report output file name and its command header information. You can customize the file name and command header information while executing the custom report function, rather than specifying a fixed file name or module name in register_report. The file name information specified in register_report is overwritten by this function call. Refer to Registering the Custom Generator for details on register_report.

Note:
The command string appears in the header information of the report in the command field.

Returns

Nothing

WriteReportBody

Declaration

```
void WriteReportBody(const char * text, int lineWrap)
```

Description

Writes text in a single line, if the integer value lineWrap is not available. Otherwise, it attempts to wrap the text using the maximum single line length lineWrap.

Returns

Nothing

WriteReportTableHeader

Declaration

```
void WriteReportTableHeader(columnHeaderList)
```

Description

Writes the table columns header. Multiple consecutive calls to this API are not permitted.

Returns

Nothing

WriteReportTableRow

Declaration

```
void WriteReportTableRow(rowDataList)
```

Description

Writes the row data to the report table. Call the WriteReportTableHeader API before the WriteReportTableRow API to add a row to the last table created. The numbers of data more than the number of columns are ignored.

Returns

Nothing

WriteReportTitle

Declaration

```
void WriteReportTitle( const char* title)
```

Description

Sets the title of the report file and the report window. Multiple calls to this API are not allowed in the same custom report function.

Returns

Nothing

GUI API Functions

GenSys GUI API functions control GUI functions from custom generators.

isModeBatch

Declaration

```
isModeBatch();
```

Description

Checks whether GenSys is working in the GUI mode or the Batch mode.

Returns

0 if GenSys is running in the GUI mode or returns 1 if GenSys is running in the Batch mode.

menuConfigureItem

Declaration

```
menuConfigureItem(<menu_name>,<options>...);
```

Description

Modifies the properties of the menu item.

The valid options with the menuConfigureItem API is given below:

```
 -text <string>       -underline <int> -command <string> -enabled {0|1}
-visible {0|1} -tearoff {0|1} -accelerator <string> -tooltip <string>
-menuitems <list>
```

Note:

The ability to toggle the -enabled property takes higher priority; all others options are secondary.

`-text <string>`

Specifies the label that appears on the action/cascade. This option is required for all action and cascade menu items.

`-underline <int>`

Specifies the index of the character within the label string that will be used as the shortcut key. This character in the label is underlined on the display. If this option is not specified, there is no shortcut key.

`-command <string>`

Specifies the Tcl command to invoke the action.

`-sub <subroutine>`

Specifies the Perl subroutine to invoke the action. This may be a subroutine name, a CODE reference, or an ARRAY reference (Tk-style callback convention).

Note:

The -command and -sub options apply only to action menu items. A given menu item may specify a -command option or a -sub option, but not both. If neither option is specified, no action will be taken by the menu item.

`-enabled {0|1}`

Specifies whether the action can be invoked. If set to 0, the label will be grayed out and clicking on the action will have no effect. The default is 1.

`-visible {0|1}`

Specifies whether the action will be visible on the menu. If set to 0, the action will not appear. The default is 1.

`-tearoff {0|1}`

Specifies whether the submenu for a cascade menu item will have a tear-off line.

`-accelerator <string>`

Specifies a key combination which invokes the action. The string appears on the menu to the right of the label. The default is no keyboard accelerator.

`-tooltip <string>`

Specifies the text to be displayed when the cursor hovers over the menu item for an action. By default, no text is displayed.

`-menuitems <list>`

Specifies the contents of the submenu for a 'cascade' menu item. The default is an empty list.

menuGetItems

Declaration

```
$menu_desc = menuGetItems(<menu_name>);
```

Description

Returns a reference to an array containing the menu description for the named menu. Otherwise, returns undef on error.

menuInsertItems

Declaration

```
menuInsertItems(<complete_menu_name>,                -menuitems =>
<list>,                {-before => <menu-name>|                  -after
=> <menu-name>});
```

Description

Inserts the menu items described in the list in the named menu, at the specified location. Returns 1 if successful, 0 otherwise.

Example

Following example shows the usage of menuInsertItems API:

```
menuInsertItems('Generator->Any->Name',-menuitems =>   [['action', -text
=> 'Allocate NewIO',                  -underline => 0,-sub => \&myfunc],
    ['action',-text => 'Generate NewIO',                 -underline =>
0,-command => 'runmycmd']] -before=>'Some Action');
```

menuSetItems

Declaration

```
menuSetItems(<menu_name>,<list>);
```

Description

Builds the entire menu with the specified hierarchical menu name <*menu_name*> and a reference to an array <*list*> describing the contents of the menu.

The hierarchical menu name is a string containing (sub)menu names separated by the -> symbol.

Note:
This API can be used to create a menu item under the Generators menu.

For example, the following specification
Generator->Subsystems
refers to the submenu labeled SubSystems within the Generator menu. A menu description is passed as a Perl array reference. Menu items are elements within the array. Each menu item is an array containing the options that describe the menu item.

The syntax for describing a menu item is given below:

```
[ <type>, <options>... ]
```

where <*type*> is action, cascade, or separator.

The menu description supports the following options:

`-text <string>`

Specifies the label that appears on the action/cascade menu item. This option is required for all action and cascade menu items.

`-name <string>`

Specifies the real name of the menu item. This name can be different from -text and is used with run command and other references, such as in menuConfigureItem and menuInsertItems. This allows the display name to be different from the actual name.

For example, display name of a menu can be Template... whereas its actual name can be Template.

`-underline <int>`

Specifies the index of the character within the label string that will be used as the shortcut key. This character in the label will be underlined on the display. If this option is not specified, no shortcut key will be specified for the menu item.

`-command <string>`

Specifies the Tcl command to invoke the action for the menu item.

`-sub <subroutine>`

Specifies the Perl subroutine to invoke the action. This may be a subroutine name, a code reference, or an array reference.

Note:
The -command and -sub options apply only to action menu items. A given menu item may specify a -command option or a -sub option, but not both. If neither option is specified, no action will be taken by the menu item.

The specified subroutine for a menu item can return the following types of values:

- Reference to a hash with appropriate fields. This helps in returning a single value. For example,

```
my $ret=
{
   ResType => $resType,   ResSubType => $resSubType,   ResValue =>
$resValue};
```

- Reference to array of references to hashes. This helps in returning multiple values. For example,

```
my $ret = [   {       ResType      => 'file',       ResSubType  =>
'verilog',      ResValue    => 'rtl/psc.v;rtl/pll.v;rtl/intc.v',   },
{       ResType     => 'message',       ResSubType  => 'warning',
ResValue    => 'No SYSCLK source specified',   },   {       ResType
=> 'message',       ResSubType  => 'info',       ResValue    => '3
Verilog files generated',   }, ]return $ret;
```

In the above example, a message box or an output file dialog is displayed accordingly.

`-enabled {0|1}`

Specifies whether the action can be invoked. If set to 0, the label will be grayed-out and clicking the menu item will have no effect. The default value for -enabled option is 1.

`-visible {0|1}`

Specifies whether the action will be visible on the menu. If set to 0, the action will not appear. The default value of -visible option is 1.

`-menuitems <list>`

Specifies the contents of the submenu for a cascade menu item. The default is an empty list.

Returns

Returns 1 if successful; Otherwise, returns 0.

Examples

Consider the following example showing the usage of the menuSetItems API:

```
menuSetItems('Generator', [  ['cascade', -text => 'Subsystems',
-underline => 0,     -menuitems => [      ['action', -text =>
'X_Generator',           -underline => 0,-sub =>
\&buildXdialog,],      ['action', -text => 'Y_Generator',
         -underline => 0, -sub => \&buildYdialog, ],       ['action',
-text => 'Z_Generator',           -underline => 0, -sub =>
\&buildZdialog,           -enabled => 0,  # grayed
out      ],    ],  ],  ['cascade', -text => 'IO', -underline => 0,
      -menuitems => [      ['action', -text => 'Allocate IO',
         -underline => 0, -command =>
'runIOAlloc',      ],      ['action', -text => 'Generate IO',
        -underline => 0, -command => 'runIOgen',
      ],  ],  ],  ['separator'],  ['cascade', -tearoff => 1, -text =>
'Checkers',       -underline => 0, -menuitems => [      ['action', -text
=> 'Sanity', -underline => 0,          -command =>
'runSanityCheck',          -accelerator =>
'CTRL+O',      ],      ['action', -text => 'Connections',
          -underline => 0,          -command =>
'runConnectionCheck',          -accelerator =>
'CTRL+K',      ],      ['action', -text => 'Opens',            -underline
=> 0, -command => 'runOpenCheck',       ],      ['action', -text =>
'RTL', -underline => 0,            -command =>
'runRTLHealthCheck',      ],    ],  ],  ['cascade', -text => 'RTL',
-underline => 1,           -menuitems => [      ['action', -text =>
'Verilog', -underline => 0,          -command =>
'runVerilogGenerator',      ],      ['action', -text => 'VHDL', -visible
=> 1 ,            -tooltip => 'Runs VHDL Generator',           -underline
=> 1, -command => 'help',      ],     ],  ],]);
```

registerDefaultAllowPortName

Declaration

```
registerDefaultAllowPortName(void* funcref)
```

Description

Registers the Perl subroutine, *funcref*, which returns a suggested port name for an IP that has the allow_create property set to Y. This subroutine is called with the following three arguments:

- Instance name (String)
- Pin/interface name (String)
- 0 for interface and 1 for pin (Integer)

funcref returns a string value, which is the suggested port name.

Returns

```
Nothing
```

setProgress

Declaration

```
setProgress(int progress);
```

Description

Sets the integer value, progress, to specify the progress of the generator in the progress bar. You can specify any value between 1 to 100. Once the value becomes greater than 99, the progress is assumed to be complete and the progress bar gets closed.

Returns

Nothing

showImageViewer

Declaration

```
showImageViewer(String <imageFile>);
```

Description

Displays an image file <*imageFile*> in the GenSys Image Viewer.

Returns

Nothing

showInfoDialog

Declaration

```
showInfoDialog($header,$message);
```

Description

Generates an Information Dialog displaying $header as header text and $message as the message text.

Returns

Nothing

showListDialog

Declaration

```
showListDialog($header, $message,     $list_ref, $multiselect);
```

Description

Generates a List Dialog displaying $header as header text and $message as the message text having $list_ref as the list data.

Specify $multiselect as 0 to allow selection of only one list item or as 1 to allow multiple selections.

showProgressBar()

Declaration

```
showProgressBar();
```

Description

Displays the progress bar in the GenSys GUI. After displaying, the progress bar, you can also set/reset its status using the setProgress API. Once the value becomes greater than 99, the progress is assumed to be complete and the progress bar gets closed.

Returns

Nothing

showQuestionDialog

Declaration

```
showQuestionDialog($header,$message,$showcancel);
```

Description

Generates a Question Dialog displaying $header as header text and $message as the message text having Ok and Cancel buttons.

Returns

2 when the user clicks the Cancel button or returns 1 when the user clicks the Ok button of the Question Dialog. Otherwise, returns 0.

Menu Item API Functions

Following APIs enable you to set, insert, get, and configure menu items in the GUI.

Notes

- The menu description specifies the menu items in the order that they will appear.
- Each menu item may be one of the following types:

  - Action (a command binding)

  - Separator (a thin, horizontal rule)

  - Cascade (a submenu)
- Menus support keyboard shortcuts to allow menu navigation from the keyboard. Shortcut keys are indicated visually by underlined characters in the Action/Cascade names.
- Action items also support keyboard accelerators (Ctrl- combinations with characters or function keys), to allow actions to be invoked directly from the keyboard without having to navigate through the menu at all. The accelerator key combination is displayed to the right of the action's label on the menu.
- All menus and cascades should have a tear-off line, by default. If tear-off capability is not desired, it can be disabled by an option in the menu description.

Global API Functions

In addition to the global API functions described in this section, you can view the API functions related to file and fileSet from fileSet and file.

CheckLicense

Declaration

```
int CheckLicense(const char * product)
```

Description

Checks the license of the product specified in the product.

Returns

Returns 1 if license is available for specified product, else return 0

ConnectionGetBits

Declaration

```
const char* ConnectionGetBits(obj)
```

Returns

Returns any of the following constant character string for a connection object:

- normal

  This string indicates that the connection is a normal connection.

- reversed

  This string indicates that connectivity is reversed. For example, if P1[1:0] and P2[1:0] are reverse connected, it means that P1[0] will be connected to P2[1] and P1[1] will be connected to P2[0].

DecompileOutput

Declaration

```
void DecompileOutput(void* obj);
```

Description

Prints debug information about the specified object *obj*.

Returns

Nothing

DeleteComponent

Declaration

```
int DeleteComponent(char* name, char* vendor, char* version);
```

Description

Deletes the component with the specified name, vendor, and version. This API searches the root for the component with the given VNV (Vendor, Name & Version) and deletes the component from the root.

The DeleteComponent API also deletes all the instances of the specified component from the root along with the connections associated with those instances.

Note:

> You need to specify all the three arguments of the DeleteComponent API, i.e.'name', 'version', 'vendor'. If you do not specify vendor and version, the you must specify empty strings for the corresponding arguments as given below:

```
DeleteComponent("comp","","");
```

Returns

1 if the component is deleted; otherwise returns 0.

DelLibraryObjForFile

Declaration

```
int DelLibraryObjForFile(const char* fileName)
```

Description

```
Removes the object source file fileName from the library. The fileName
can be specified with absolute or relative path.
```

Returns

1 on success and 0 on failure.

DelLibraryObjForVNLV

Declaration

```
int DelLibraryObjForVNLV(const char* type, const char* name, const char*
version, const char* vendorName, const char* libName)
```

Description

`Deletes an object of type type, with name name, from library libName of vendor vendorName and of version version.`

Note:

type is a mandatory argument for this API. If only type is specified, all the files of that type are deleted. If other parameters are also specified with type, the files matching that vnlv are deleted.

Returns

1 on success and 0 on failure.

ElaborateIpxact

Declaration

```
ElaborateIpxact()
```

Description

Elaborates GenSys database with IP-XACT schema tables.

GenerateAll

Declaration

```
GenerateAll()
```

Description

Internally runs the PDF Document Generator and Word ML Generator.

Note:

The GenerateAll() API can be used as an alternative to the -gen all switch. For example, consider the following command:

```
genesis -batch -cmd "run GenerateAll" *.xml
```

The above command is equivalent to the following command:

```
genesis -batch -gen all *.xml
GenerateMemoryMap
```

GenerateMemoryMap

Declaration

```
void GenerateMemoryMap()
```

Description

Generates memory map for active design/component.

GetColumnSchemaByName

Declaration

```
void* GetColumnSchemaByName(const char* type)
```

Description

Returns the column schema for the hierarchical column name *type*.

GetcomRoot

Declaration

```
void* GetcomRoot();
```

Returns

The comRoot object.

GetDefaultAlign

Declaration

```
const char* GetDefaultAlign()
```

Description

Returns the default alignment set by using the set_elab_option Tcl command.

For more details on set_elab_option Tcl command, see *Tcl Command Reference Guide*.

Returns

LSB or MSB if the default alignment is set to LSB or MSB, respectively, or returns UNDEF if the default alignment is not set at all.

GetDefaultDataSource

Declaration

```
void* GetDefaultDataSource()
```

Returns

Returns the default datasource from the top of the default datasource stack.

getGenSysVersion

Declaration

```
const char* getGeneSysVersion()
```

Returns

GenSys version number

getInstIndex

Declaration

```
int char* getInstIndex(const char *type);
```

Returns

The index for the specified type, type. type can be any of interface or instance.

getInstPrefix

Declaration

```
const char* getInstPrefix(const char *type);
```

Returns

The prefix for the specified type, type. type can be any of interface or instance.

getInstSuffix

Declaration

```
const char* getInstSuffix(const char *type);
```

Returns

The suffix for the specified type, type. type can be any of interface or instance.

GetLibraryObj

Declaration

```
const char* GetLibraryObj(const char* type, const char* name, const char*
version, const char* vendor, const char* library);
```

Returns

Absolute path of the specified object with name as name, which is of type type, is of version version, is provided by a vendor vendor, and exists in library library.

Note:

name and type are mandatory arguments for this API.

GetLibraryObjByType

Declaration

```
void* GetLibraryObjByType(const char* type);
```

Returns

Returns reference to an array of hash containing information of all the objects for specified type. Type can be specified as design, component, or interface. Each entry of the array gives information of an object specifying name, library, vendor, version, and file.

getLogDir

Declaration

```
const char* getLogDir()
```

Returns

The log directory name

GetObjectType

Declaration

```
const char* GetObjectType(void* obj);
```

Returns

The type of the specified object *obj* as one of the following values:

```
ROOTBASE_CLASS  ACT_GRAPH  ACTION_INFO  ADDRESS    MEMORY_BLOCK    REGIST
ER_BLOCK  ADDRESS_INTERFACE  ALIAS  ATTRIBUTE  BIT_ENUM_VALUE  BIT_FIELD_
CLASS  BLOCK_SIZE  COM_PARAMETER_BASE_CLASS    PARAMETER    PARAM_TYPE
 TYPE_INFO    TYPE_TREE    VALUE_TREE  COMPONENT    DESIGN  CONNECTION  D
ATA_SOURCE  DEP_GRAPH  DEP_INFO  DEP_OPERAND    BIT_FIELD_OPERAND    PORT
_OPERAND  DEPENDENCY_CLASS  GROUP_ALIAS  INTERFACE  INTERFACE_DEF  INTERF
ACE_INST  LOGICAL_PORT  PORT  PORT_MAP  REG_GRP_INST  REGISTER_DATA  REQU
ESTER  SPECIAL_CONNECT  VALUE_RANGE  VERSIONED_NAME_CLASSCOM_CONNECTION_T
YPE_ITEM  ADHOC_CONNECTION  INTERFACE_CONNECTION  LOGICAL_CONNECTION  TIE
_OFF_CONNECTIONCOM_TABLE_VALUE  TABLE  PIX_MAP_SCALARTERMINAL  SCALAR_TER
MINAL  VECTOR_TERMINAL
```

Note:

A type name highlighted in Red color is returned when an object is expected to be of one of the type names listed under the highlighted type name but cannot be inferred to be of one of these types. For example, a terminal should be inferred as either SCALAR_TERMINAL or VECTOR_TERMINAL failing which its type is reported as TERMINAL (highlighted).

getOutputDir

Declaration

```
const char* getOutputDir();
```

Returns

Returns the path to the output directory.

getReportDir

Declaration

```
const char* getReportDir();
```

Returns

Returns the path to the report directory set using the set_report_dir Tcl command.

Note:
Standard Perl generators (Sanity, Statistics, Connections, Openconnections, and Autoconnect) will now create output report files in the directory set using the set_report_dir Tcl command.

GetString

Declaration

```
char* GetString(vptr);
```

Returns

The equivalent char pointer for the specified void pointer *vptr*.

GetTableSchemaByName

Declaration

```
void* GetTableSchemaByName(const char* type)
```

Description

Returns the table schema for the hierarchical table name *type*.

getTclValue

Declaration

```
const char* getTclValue($tcl_var_name)
```

Description

Returns the value of the Tcl variable *$tcl_var_name*.

If a Tcl command calls a Perl function which further calls getTclValue function to access a Tcl variable, the variable is first searched in the scope of the original Tcl command. If the variable is not found in that scope then it is searched in global scope. If the Tcl variable is not defined, *undef* is returned.

getVNLVString

Declaration

```
string getVNLVString(comVersionedNameClass* vnlv, string objectType)
```

Returns

Returns a VNLV string of an IP-XACT table. This string is used to locate the IP-XACT table stored in the comRoot object.

getWorkDir

Declaration

```
const char* getWorkDir();
```

Returns

Returns the path to the current working directory.

getXMLDir

Declaration

```
const char* getXMLDir();
```

Returns

Returns the path to the directory where XML files are stored.

nonblock_system

Declaration

```
void nonblock_system(string cmd)
```

Description

Executes a system shell command *cmd*.

The nonblock_system API is similar to the Perl system command. However, it does not block the parent process to wait for the completion of the child process started with command *cmd*. You can work on the parent process simultaneously when a child process started with *cmd* is running.

printg

Declaration

```
void printg(const char* msgStr, int msgId=-1,   const char* Severity="",
  const char* generatorName ="");
```

Description

Prints the specified message *message* to the Tcl Window and pbd.log file.

By default, the message ID is -1 and the severity is "" (NULL). If you do not specify either message ID or severity, the default values are used while displaying messages from printg().

You can specify message IDs between the range of 4001-6000. If an ID supplied to the printg()API is not in this range, GenSys generates an error.

GenSys internally supplies messages with severity – DEBUG/INFO/WARNING/ERROR. You can use one of these message severities when calling printg(), or can use any other strings, such as FATAL, USER_INFO, and so on, as required.

Also, the information related to the message (such as, filename, line number, message ID, message severity, and message string) will be passed to the message handler (if any) [registered using the RegisterMessageHandler API].

By default, no header will be appended to the messages using printg().

The generatorName specifies the name of the generator. The format of the message when generatorName is given is shown below:

```
<severity>[<generatorName>-<msgId>]: <msgStr>
```

If the generatorName is not given or is passed as "" (NULL), the format of the message will be:

```
<severity>[<msgId>]: <msgStr>
```

The default value of generatorName is NULL or "".

Note:
Standard error messages (stderr) from Perl and Tcl will continue to use the default 'ERROR[6000]' prefixed to the messages.

Returns

Nothing

printToPerfLog

Declaration

```
printToPerfLog(const char *text,int print_time_mem);
```

Description

Prints the given message *text* or time and memory details to the file, gensys.perflog, in the log directory.

You can choose to print a string or elapsed time and memory usage details or both.

Arguments

`text`

Text message to be printed to gensys.perflog

`print_time_mem`

If set to 1, prints time and memory details to gensys.perflog, else does not print time and memory details.

Examples

The following example prints the given message:

```
printToPerfLog("This is a sample text\n", 0)
```

The following example prints time and memory details:

```
printToPerfLog("", 1)
```

The following example prints both the given message and time and memory details:

```
printToPerfLog("This is a sample text\n", 1)
```

RegisterRule

Declaration

```
RegisterRule(id, rule_flag, severity, environment, category, subcategory,
short_desc, long_desc, perl_func_name);
```

Description

Enables you to create your own rules and register them for a GenSys session.

Arguments

`id`

Specifies the name or ID of the rule. The ID should be unique and should not have been used for any of the existing rules even if the rules are disabled.

`run_flag`

Enables or disables a rule. Set to on to run the rule in the specified GenSys environment(s).

`severity`

Specifies the severity of the message(s) flagged from the rule when the required condition is not met.

`environment`

Specifies the GenSys environments in which the GenSys rule is to be run.

`category`

Specifies the category of the rule.

`subcategory`

Specifies the subcategory of the rule.

`short_desc`

Specifies a one liner description about the rule.

`long_desc`

Specifies the detailed description of the rule.

`perl_func_name`

Specifies the name of the Perl subroutine that implements the rule functionality including flagging of violation messages. This function would pass id and severity arguments and should work on the current design/component/interface using existing Perl APIs.

You need to specify the path of the Perl file where you have written the subroutine while invoking GenSys using the -I command line option.

Initially, all the existing rules in GenSys would be registered in a special file under SPYGLASS_HOME/auxi/Genesis/registerRules.pl with default severity/environment(s) for the rules. All the rules being enabled by default. If you want to disable specific rules or change any attributes, such as severity/environment, use the update_rule Tcl command or GUI rule setup.

Rename

Declaration

```
int Rename(void* obj, const char* newName );
```

Description

Renames the specified object *obj* (design, component, or interface objects) to the new name *newName*.

Returns

0 on success and 1 on failure.

SaveAll

Declaration

```
int SaveAll();
```

Description

Saves all open objects (designs, components, and interfaces).

Returns

Nothing

SetRefreshEnabled

Declaration

```
void SetRefreshEnabled(const char* str);
```

Description

Enables refreshing of GUI tables when "true"/"TRUE". Disables refreshing of GUI tables when "false"/"FALSE".

For example,

```
setRefreshEnabled("true");
```

The above command allows refreshing of GUI tables.

Returns

Nothing

TableCellGetExpression

Declaration

```
const char* TableCellGetExpression(void* tableCell);
```

Description

Returns the live formula expression string value in a cell *tableCell*, if the cell contains a live formula, else returns "".

VectorizeElaboratedConnections

Declaration

```
void VectorizeElaboratedConnections(void* obj, const char* isElaborated,
const char* isRecursive)
```

Description

This API performs the following steps:

1. Elaborates the connectivity database to get the actual scalar connection (as seen in RTL). At this point, all the overlapping and overridden connections are taken care of based on the elaboration rules of precedence. Only scalar adhoc connections, scalar tieoff connections, and scalar open connections remain after elaboration.
2. Calls a vectorization routine which performs the best possible vectorization of the given scalar connections, and convert them to vector adhoc connections wherever possible. The new vectorized connections are added the GenSys design, and the previous connections (all of the previous user connections and the elaborated scalar connections) are deleted. After this, user can use normal GenSys Perl APIs to access these connections.

After calling this API, the design connectivity database gets changed and it is the responsibility of the user to save it properly.

Note:
This API changes the GenSys connection database.

Arguments

`obj`

Specifies the design object pointer

If you do not specify the design object pointer, this API considers the active design.

`isElaborated`

Specifies a boolean value

By default, this argument is set to false and this API performs elaboration.

If the design is already elaborated, specify the value of this argument to true to stop internal elaboration.

`isRecursive`

Specifies a boolean value

By default, this argument is set to true, which means that this API would work in a recursive manner. That is, this API would go to each subsystem of the design, elaborate, perform vectorization of the elaborated connections, and add them to that subsystem after flushing the previous user specified connections.

Set the value of this argument to false to stop this recursive behavior.

Returns

Nothing

---

fileSet

The fileSet object has the following design hierarchy:

```
comRoot -> Component -> fileSet
comRoot -> Design -> fileSet
```

Use the following API functions to process fileSet objects.

ComponentGetFileSetByName

Declaration

```
void*   ComponentGetFileSetByName(void* comp, const char* fName)
```

Description

Returns the fileSet that has the name fName in the component/design comp.

Argument

comp

The component object returned by the comRootGetComponentByName, comRootGetComponentList, or comRootGetComponentByVNLV functions.

Or, the design object returned by the comRootGetDesignByName, comRootGetDesignList, or comRootGetDesignByVNLV functions.

fName

The name of the fileSet that is to be searched

Returns

fileSet having name fName in the component/design comp.

ComponentGetFileSetList

Declaration

```
void* ComponentGetFileSetList(void* comp)
```

Description

Returns the list of fileSet of the component/design comp.

Argument

comp

The component object returned by comRootGetComponentByName, comRootGetComponentList, or comRootGetComponentByVNLV functions.

Or, the design object returned by the comRootGetDesignByName, comRootGetDesignList, or comRootGetDesignByVNLV functions.

Returns

List of fileSet of the component/design comp.

FileSetGetDependency

Declaration
```
const char* FileSetGetDependency(void* fileSet)
```

Description
Returns the dependency of the fileSet that is specified in the argument

Argument
fileSet

The fileSet for which dependency is required

Returns
Dependency of the fileSet that is specified in the argument.

FileSetGetFileByName

Declaration
```
void* FileSetGetFileByName(void* fileSet, const char* fileName)
```

Description
Returns the File that has the name fileName of the fileSet specified in the argument

Argument
fileSet

The fileset in which the File that has the name fileName is to be searched.

fileName

The name of the File which is to be searched.

Returns
File that has the name fileName of the fileSet specified in the argument.

FileSetGetFileList

Declaration

```
void* FileSetGetFileList(void* fileSet)
```

Description

Returns the FileList of the fileSet that is specified in the argument.

Argument

fileSet

The fileSet for which FileList is required

Returns

FileList of the fileSet that is specified in the argument.

FileSetGetGroupList

Declaration

```
void* FileSetGetGroupList(void* fileSet)
```

Description

Returns the GroupList of the fileSet that is specified in the argument

Argument

fileSet

The fileSet for which GroupList is required

Returns

GroupList of the fileSet that is specified in the argument.

FileSetGetName

Declaration
```
const char* FileSetGetName(void* fileSet)
```

Description

Returns the name of the fileSet that is specified in the argument

Argument

fileSet

The fileSet for which the name is required

Returns

The name of the fileSet that is specified in the argument.

file

The file object has the following design hierarchy:

```
comRoot -> Component -> fileSet -> file
comRoot -> Design -> fileSet -> file
```

Use the following API functions to process file objects.

FileGetDefAttrByName

Declaration
```
void* FileGetDefAttrByName(void* file,  const char* AttrName)
```

Description

Returns the define attribute of file that has the name AttrName in the file.

Argument

file

The file for which the define attribute is required

Returns

Define attribute of file having name AttrName in the file

FileGetDefAttrList

Declaration
`void* FileGetDefAttrList(void* file)`

Description

Returns the define attribute list of the file that is specified in the argument

Argument

file

The file for which the define attribute list is required

Returns

Define attribute list of the file that is specified in the argument

FileGetDependency

Declaration
`const char* FileGetDependency(void* file)`

Description

Returns the dependency of the file that is specified in the argument

Argument

file

The file for which the dependency is required

Returns

Dependency of the file that is specified in the argument

FileGetFileID

Declaration

```
const char* FileGetFileID(void* file)
```

Description

Returns the FileID of the file that is specified in the argument

Argument

file

The file for which FileID is required

Returns

FileID of the file that is specified in the argument

FileGetFileType

Declaration

```
const char* FileGetFileType(void* file)
```

Description

Returns the FileType of the file that is specified in the argument

Argument

file

The file for which fileType is required

Returns

FileType of the file that is specified in the argument

FileGetIsInclude

Declaration

```
const char* FileGetIsInclude(void* file)
```

Description

Returns the IsInclude of the file that is specified in the argument. It denotes whether the file is an included file in RTL. It can either be true or false.

Argument

file

The file for which IsInclude is required

Returns

IsInclude of the file that is specified in the argument

FileGetIsPackage

Declaration

```
const char* FileGetIsPackage(void* file)
```

Description

Returns the IsPackage of the file that is specified in the argument. It denotes whether the file is a package file. It can either be true or false.

Argument

file

The file for which IsPackage is required

Returns

IsPackage of the file that is specified in the argument

FileGetLogicalName

Declaration

```
const char* FileGetLogicalName(void* file)
```

Description

Returns the logical name of the file that is specified in the argument

Argument

file

The file for which the logical name is required

Returns

Logical name of the file that is specified in the argument

FileGetName

Declaration

```
const char* fileGetName( void* file)
```

Description

Returns the name of the file that is specified in the argument

Argument

file

The file for which name is required

Returns

Name of the file that is specified in the argument

Other APIs

This section covers the getAbsoluteFilePath API.

getAbsoluteFilePath

Returns an absolute file path

Declaration

```
const char* getAbsoluteFilePath (char* fPath);
```

Description

Returns an absolute path for the given relative path.

Following is the usage of this API in the Perl file:

```
genesisapis::getAbsoluteFilePath(<file_path>);
```

Note:
Ensure that the path specified in the <file_path> argument exist.

Arguments

`fPath`

Input file name.

Returns

Absolute path for the given relative path

Deprecated APIs

Following table provides a list of deprecated APIs:

API Function	Purpose
AddRow	Adds a row
BaseClassGetDataSourceChangeMethod	Returns the Data Source Change Method value. Replacement APIs that can be used are: BaseClassGetDataSource/DataSourceGetChangeMethod
BaseClassGetDataSourceChangeReason	Returns the Data Source Change Reason value. Replacement APIs that can be used are: BaseClassGetDataSource/DataSourceGetChangeReason
BaseClassGetDataSourceChangeFrozen	Returns the Data Source Change Frozen value. Replacement APIs that can be used are: BaseClassGetDataSource/DataSourceGetDataIsFrozen
BaseClassGetSessionFileName	Returns the name of the session file.

API Function	Purpose
BaseClassGetFormattedString	Returns the formatted description. Replacement API that can be used is: BaseClassGetDescription
BaseClassGetPermissionList	Returns the list (itr) of permissions
RegisterBlockGetShortName	Returns the short name of Register Block. Replacement API that can be used is: BaseClassGetShortName
RegisterDataGetShortName	Returns the short name of Register Data. Replacement API that can be used is: BaseClassGetShortName
ColumnGetData	Data for column
ColumnGetRowList	Row list for column
ComponentGetConfigComponent	Configuration component for the component
ComponentGetConstraintTable	Constraints Table for the component
ComponentGetGeneratedComponentFileName	Generated component filename
ComponentGetGeneratedComponentFileType	Generated component file type
ComponentGetGenRTLName	Generated RTL name
ComponentGetLibraryName	Library name
ComponentGetVendorName	Component’s Vendor name
ComponentGetVersionNumber	Component’s version number
ComponentInstGetElaborate	
ComponentInstGetExtnTable	Extension table for the component instance
ComponentInstGetIntfInstByName	Named interface instance for the component instance

API Function	Purpose
ComponentInstGetMasterComponentFile	Master component file for the component instance
ComponentInstGetRTLScope	RTL scope for the component instance
comRootGetComponentByFileName(Name)	Component by file name under the comRoot
comRootGetComponentByRTLName	Component by RTL name under the comRoot
comRootGetInterfaceDefByFileName(Name)	Interface Definition by file name under the comRoot
DeleteRow	Deletes the row
DesignGetRTLConnection (inst_name, port_name, lsb, msb)	List of RTL connections to the named port/terminal of the design/named component instance corresponding to the specified bit-range msb:lsb (inst_name is NULL for the design)
DesignGetRTLConnectionList	List of RTL connections
EditCol	
EvaluateTable	Evaluates the table
InsertRow	
InsertSubTable	Inserts the subtable
LogicalPortGetAliasPortList	
LogicalPortGetAliasPortByName	
PortGetPortName	Name of the port
PortGetPortStatus	Port's status (open or empty string)
PortGetPortType	Port type (clock, reset, event, data, control, ...)
PortIsPortScalar	1 if the port is a scalar port. Otherwise 0
RegGrpInstGetName	Returns the name of Register Group Instance

API Function	Purpose
RegGrpInstGetRegisterList	Returns the list of Registers for Register Group Instance
RegGrpInstGetRegisterByName	Returns the named Register for Register Group Instance
RegGrpInstGetOffset	Returns the Offset for Register Group Instance
RegGrpInstGetNumIns	Returns the NumInst value for Register Group Instance
RegGrpInstGetGrpAlias	Returns the Group Alias for Register Group Instance
RegGrpInstGetInstAddr	Returns the Instance Address for Register Group Instance
RegisterDataGetIsVolatileData	Returns 1 if Register Data is volatile. Otherwise, returns 0.
RegisterDataGetArrayDim	Returns the Array Dimensions of Register Data
RegisterDataGetAliasReg	Returned the Alias Register for Register Data
RegisterObjectGetRegister	Returned pointer to Register Data, if the Register Object was a Register Data. Otherwise, returned NULL
RegisterObjectGetGroup	Returned pointer to Register Group, if the Register Object was a Register Group. Otherwise, returned NULL.
ComponentGetAliasesList	Returned the list of Aliases for component
AliasGetName	Returned the Name of the Group Alias
AliasGetPhysicalRegister	Returned the Physical Register corresponding to the Group Alias
AliasGetAliasRegsList	Returned the list of Alias Registers for the Group Alias
AliasGetAliasRegByName	Returned the named physical register regName corresponding to alias
ComponentGetGrpAliasesList	Returned the list of Group Aliases for component
GrpAliasGetName	Returned the Name of the Group Alias
GrpAliasGetPhysicalRegister	Returned the Physical Register corresponding to the Group Alias
GrpAliasGetAliasRegsList	Returned the list of Alias Registers for the Group Alias

API Function	Purpose
GrpAliasAliasGetAliasRegGrpByName	Returned the list of Alias Registers for the Group Alias
ComponentInstGetAddressInterfaceList	Returned the list (itr) of Address Interfaces for component instance
AddressInterfaceGetTargetInst	Returned the target instance pointer of address interface object
TableCellGetPerlExpr	Returned the Perl expression associated with a table cell of type PERL
BitEnumValueGetValueRange	Returned the value range for bitenum
ValueRangeGetMinimum	Returned the minimum value of ValueRange
ValueRangeGetMaximum	Returned the maximum value of ValueRange
RegisterBlockGetRegGrpList	List of Register Group Instances under the Register Block
SelectRow	Select the row
SelectSubTable	Selects the subtable
SelectTable	Selects the table
TableAddRow	Adds the row in the table
TableCellGetData	Data for the table cell
TableCellGetEnumChoiceList	Enum choice list for table cell
TableCellGetParentTable	Parent table for table cell
TableGetChildTable	Child table of table
TableGetColumnByName	Column by name for table
TableGetColumnList	Column list for table
TableGetGroupId	Group ID for table
TableGetID	ID for table
TableGetIthColumn	Ith column for table

API Function	Purpose
TableGetNumOfCols	Number of columns for table
TableGetNumOfRows	Number of rows for table
TableGetParentTable	Parent table of table
TableGetSubTableList	Subtable list of table
GetColNameInParentTable	Parent column. This is valid for every table cell. The table cell could also be a sub-table.
GetRowNumInParentTable	Parent row. This is valid for every table cell. The table cell could also be a sub-table
isObjectATable	1 if the object is a table (object of type comTableClass). Otherwise 0.
UpdateRow	
VectorTerminalGetLSBForVectorTerminal	LSB of the vector component instance terminal
VectorTerminalGetMSBForVectorTerminal	MSB of the vector component instance terminal
AddressGetCondition	Condition of the Address object
BitFieldGetResetDependency	Reset dependency for Bitfield object
AttributeGetEnumChoiceList	Enum Choice List for the attribute
GenericDesGetFormattedDes	Generic description (formatted)
GenericDesGetUnFormattedDes	Generic description (unformatted)
ComponentInstGetRTLConnectionsList	RTL Connections for the component instance
ComponentInstGetClkConnectionsList	Clock Connections for the component instance
ComponentInstGetRstConnectionsList	Reset Connections for the component instance

API Function	Purpose
ComponentInstGetEvtConnectionsList	Event Connections for the component instance
ComponentInstGetConfigAdHocConnectionsList	Configuration AdHoc Connections for the component instance
ComponentInstGetConfigTieOffConnectionsList	Configuration Tieoff Connections for the component instance
ConnectionGetConnectionForTerminal	Terminal for the connection
AddressInterfaceGetMemDescription	Memory description for the address interface
AddressInterfaceGetFormattedString	Formatted description for the address interface
comTableValueGetPerlExpr	Perl expression for the comTableValue
BaseClassGetXMLFile	XML file for the BaseClass object
BaseClassGetTableList	Table list for the BaseClass object
BaseClassGetTableForTab	Table for the tab for the BaseClass object
InterfaceGetConstraintTable	Returns the Constraints table for interface
PortGetPropertiesTab	Returns the Properties Tab of Port
PortGetConstraintTab	Returns the Constraint Tab of Port
comTableValueGetIntScalar	
comTableValueGetStringScalar	
comTableValueGetEnumChoices	
RegisterOperandGetRegData	
RegisterOperandGetColumn	
BitFieldOperandGetBitField	

API Function	Purpose
BitFieldOperandGetBitEnum	
BitFieldOperandGetColumn	
PortOperandGetOp	
RValueOperandGetDepOperand	
RValueOperandGetMaskVal	
RValueOperandGetPerlFunc	
RValueOperandGetExpr	
RValueOperandGetOperator	
RValueOperandGetChildRValueOperand	
ComponentInstGetInterface2LogicalConnectionsList	
AddressInterfaceGetName	Returns the name of the address interface
AddressInterfaceGetAddressInstanceName	Returns the name of the address instance for the address interface
AddressInterfaceGetMemoryStartAddress	Returns the starting address of the memory associated with the address interface
AddressInterfaceGetMemoryEndAddress	Returns the ending address of the memory associated with the address interface
AddressInterfaceGetMemorySize	Returns the memory size of the address interface
AddressInterfaceGetParentInterface	Returns the parent interface of the address interface
AddressInterfaceGetRequester	Returns the requester of the address interface
ComponentInstGetRequestersList	Returns the list (itr) of Requesters for component instance

API Function	Purpose
DesignGetAddressInterfaceByName	Returns the memory map of named address interface that contains the register memory map of each instance of design
RequesterGetRequester	Returns the requester
RequesterGetRequesterName	Returns the name of the requester instance for a requester
RequesterGetMAU	Returns the MAU for a requester
RequesterGetAccessType	Returns the access type for a requester

Following APIs have been made defunc and will not work properly:

API Function	Purpose
BlockSizeGetWidth	Returns the width of Block Size
BlockSizeGetDepth	Returns the depth of Block Size
AddressGetName	Returns the name of the address object
AddressGetType	Returns the type of the address object
RegisterBlockGetName	Returns the name of Register Block
RegisterBlockGetRegDataList	Returns the list of Register Data items for Register Block
RegisterBlockGetRegObjectList	Returns the list of Register Objects (both Register Data and Register Group) for a Register Block in the order defined by the user
RegisterBlockGetRegDataByName	Returns the named Register Data item for a Register Block
RegisterBlockGetRegGrpInstList	Returns the list of Register Group Instances for Register Block
RegisterBlockGetRegGrpInstByName	Returns the named Register Group Instance for a Register Block
InterfaceGetAddressList	Returns the list of Addresses for interface
ComponentGetAddressList	Returns the list of Addresses for component

API Function	Purpose
ComponentGetAddressByName	Returns the named Address for component
RegGrpInstGetSubRegGrpInstList	Returns the list of Sub-Register Group Instances for Register Group Instance
RegGrpInstGetSubRegGrpInstByName	Returns the named Sub-Register Group Instance for Register Group Instance
AddressInterfaceGetTargetIntfName	Returns the target interface name of address interface object
AddressInterfaceGetRequestorInst	Returns the requestor instance pointer of address interface object
AddressInterfaceGetRequestorInstName	Returns the requestor instance name of address interface object
AddressInterfaceGetRequestorIntfName	Returns the requestor interface name of address interface object
MemoryBlockGetAccessNumBytes	Returns the list of Accessed Bytes of Memory Block
MemoryBlockGetIncrOffset	Returns the Incremental Offset of Memory Block
MemoryBlockGetBlockSize	Returns the size of Memory Block