Leda
C Interface Guide

Version 2006.06
June 2006

Comments?
E-mail your comments about this manual to
leda-support@synopsys.com.

SYNOPSYS®

Copyright Notice and Proprietary Information

Destination Control Statement

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

Disclaimer

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

Registered Trademarks (®)

Synopsys, AMPS, Arcadia, C Level Design, C2HDL, C2V, C2VHDL, Cadabra, Calaveras Algorithm, CATS, CSim, Design Compiler, DesignPower, DesignWare, EPIC, Formality, HSPICE, Hypermodel, iN-Phase, in-Sync, Leda, MAST, Meta, Meta-Software, ModelAccess, ModelTools, NanoSim, OpenVera, PathMill, Photolynx, Physical Compiler, PowerMill, PrimeTime, RailMill, Raphael, RapidScript, Saber, SiVL, SNUG, SolvNet, Stream Driven Simulator, Superlog, System Compiler, Testify, TetraMAX, TimeMill, TMA, VCS, Vera, and Virtual Stepper are registered trademarks of Synopsys, Inc.

Trademarks (™)

abraCAD, abraMAP, Active Parasitics, AFGen, Apollo, Apollo II, Apollo-DPII, Apollo-GA, ApolloGAII, Astro, Astro-Rail, Astro-Xtalk, Aurora, AvanTestchip, AvanWaves, BCView, Behavioral Compiler, BOA, BRT, Cedar, ChipPlanner, Circuit Analysis, Columbia, Columbia-CE, Comet 3D, Cosmos, CosmosEnterprise, CosmosLE, CosmosScope, CosmosSE, Cyclelink, Davinci, DC Expert, DC Expert Plus, DC Professional, DC Ultra, DC Ultra Plus, Design Advisor, Design Analyzer, Design Vision, DesignerHDL, DesignTime, DFM-Workbench, DFT Compiler, Direct RTL, Direct Silicon Access, Discovery, DW8051, DWPCI, Dynamic-Macromodeling, Dynamic Model Switcher, ECL Compiler, ECO Compiler, EDAnavigator, Encore, Encore PQ, Evaccess, ExpressModel, Floorplan Manager, Formal Model Checker, FoundryModel, FPGA Compiler II, FPGA Express, Frame Compiler, Galaxy, Gatran, HDL Advisor, HDL Compiler, Hercules, Hercules-Explorer, Hercules-II, Hierarchical Optimization Technology, High Performance Option, HotPlace, HSPICE-Link, iN-Tandem, Integrator, Interactive Waveform Viewer, i-Virtual Stepper, Jupiter, Jupiter-DP, JupiterXT, JupiterXT-ASIC, JVXtreme, Liberty, Libra-Passport, Library Compiler, Libra-Visa, Magellan, Mars, Mars-Rail, Mars-Xtalk, Medici, Metacapture, Metacircuit, Metamanager, Metamixsim, Milkyway, ModelSource, Module Compiler, MS-3200, MS-3400, Nova Product Family, Nova-ExploreRTL, Nova-Trans, Nova-VeriLint, Nova-VHDLlint, Optimum Silicon, Orion_ec, Parasitic View, Passport, Planet, Planet-PL, Planet-RTL, Polaris, Polaris-CBS, Polaris-MT, Power Compiler, PowerCODE, PowerGate, ProFPGA, ProGen, Prospector, Protocol Compiler, PSMGen, Raphael-NES, RoadRunner, RTL Analyzer, Saturn, ScanBand, Schematic Compiler, Scirocco, Scirocco-i, Shadow Debugger, Silicon Blueprint, Silicon Early Access, SinglePass-SoC, Smart Extraction, SmartLicense, SmartModel Library, Softwire, Source-Level Design, Star, Star-DC, Star-MS, Star-MTB, Star-Power, Star-Rail, Star-RC, Star-RCXT, Star-Sim, Star-SimXT, Star-Time, Star-XP, SWIFT, Taurus, Taurus-Device, Taurus-Layout, Taurus-Lithography, Taurus-Process, Taurus-Topography, Taurus-Visual, Taurus-Workbench, TimeSlice, TimeTracker, Timing Annotator, TopoPlace, TopoRoute, Trace-On-Demand, True-Hspice, TSUPREM-4, TymeWare, VCS Express, VCSi, Venus, Verification Portal, VFormal, VHDL Compiler, VHDL System Simulator, VirSim, and VMC are trademarks of Synopsys, Inc.

Service Marks (℠)

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

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

Contents

Tables

Preface

About This Manual

This manual is designed for engineers who want to develop design netlist rules and Synopsys Design Constraints rules for checking in Leda using the C interface. This manual is intended for use by design and quality assurance engineers who are already familiar with C or C++.

Related Documents

This manual is part of the Leda documentation set. To see a complete listing, refer to the *Leda Document Navigator*.

Manual Overview

This manual contains the following chapters:

Preface	Describes the manual and lists the typographical conventions and symbols used in it. Tells how to get technical assistance.
Chapter 1 Using C-based Rules	Overview of the Leda C-based interface for writing custom design netlist checking rules. Explains how to write, test, and run C-based rules.
Chapter 2 C API DQL Reference	An API reference for the DQL C interface, which includes definitions of the predefined constants, typedefs, functions, and returned data types.
Chapter 3 C API CQL Reference	An API reference for the CQL C interface, which includes definitions of the predefined constants, typedefs, enumeration types, functions and returned data types.

Typographical and Symbol Conventions

The following conventions are used throughout this document:

Table 1: Documentation Conventions

Convention	Description and Example
%	Represents the UNIX prompt.
Bold	User input (text entered by the user). `% cd $LMC_HOME/hdl`
`Monospace`	System-generated text (prompts, messages, files, reports). `No Mismatches: 66 Vectors processed: 66 Possible"`
Italic or *Italic*	Variables for which you supply a specific value. As a command line example: `% setenv LMC_HOME prod_dir` In body text: In the previous example, *prod_dir* is the directory where your product must be installed.
| (Vertical rule)	Choice among alternatives, as in the following syntax example: `-effort_level low | medium | high`
[ ] (Square brackets)	Enclose optional parameters: `pin1 [pin2 ... pinN]` In this example, you must enter at least one pin name (*pin1*), but others are optional ([*pin2* … *pinN*]).
TopMenu > SubMenu	Pulldown menu paths, such as: **File > Save As …**

Getting Leda Help

For help with Leda, send a detailed explanation of the problem, including contact information, to leda-support@synopsys.com.

The Synopsys Web Site

General information about Synopsys and its products is available at this URL:

http://www.synopsys.com

1 Using C-based Rules

Introduction

The Leda C interface allows you to write custom netlist checker rules using Design Query Language (DQL) and Synopsys Design Constraints (SDC) rules using Constraint Query Language (CQL). You use these APIs to implement C or C++ programs that you can use to check your HDL designs or SDC files. You can also write design netlist checking rules and Synopsys Design Constraints (SDC) rules in Tcl (see the *Leda Tcl Interface Guide*).

Hint

Design and SDC rules that you write in C/C++ run 25 to 100 times faster than the same design and SDC rules in Tcl.

If you want to develop language-based coding rules, use the VRSL and VeRSL rule specification languages. See the *VeRSL Reference Guide* (for Verilog coding rules) or *VRSL Reference Guide* (for VHDL coding rules). Note that there are lots of prepackaged coding rules built into Leda that you can use to implement most or all of your rule checking needs. You can customize and parameterize many of these rules. This chapter explains how to develop and use C-based rules in the following sections:

Writing C-based Rules using DQL

To write C-based custom design rules, you include the supplied dql.h file in your C or C++ source file, and implement your rules using the supplied functions, constants, and typedefs. Note that the dql string stands for Design Query Language (DQL). The entire contents of this header file are documented in the "C API DQL Reference" on page 33. You can find this header file in the install tree at $LEDA_PATH/rules/include/dql.h.

Writing C-based Rules using CQL

To write C-based custom SDC rules, you include the supplied cql.h file in your C or C++ source file, and implement your rules using the supplied functions, constants, and typedefs. Note that the cql string stands for Constraint Query Language (CQL). The entire contents of this header file are documented in the "C API CQL Reference" on page 29. You can find this header file in the install tree at $LEDA_PATH/rules/include/cql.h.

Writing Error Messages in C

When you specify a message in your C code and the VeRSL wrapper for a rule, Leda concatenates the two messages (see "Creating a Policy for C-based Rules" on page 29). For example, if you add a message in your C-based rule like the following:

```
strcpy(message,"C message");
EDB_add_messages( message );
```

and your VeRSL wrapper also specifies a message:

```
netlist_check "function" in "file.ext"
message "VeRSL message"
severity ERROR
```

Leda concatenates the two messages as follows:

```
VeRSL messageC message
```

Note that if you do not specify a VeRSL message in the wrapper file, your rule will not be visible in the Rule Wizard. Therefore, the recommended methodology is to specify a message in the VeRSL wrapper that clearly indicates the general error type, and use the message you write in the C source file to pick up specific parameterized information about the error.

Ensuring that C-based Rules Appear in Info Report

C-based rules only appear in the info report (leda.inf) if they are selected for checking. If you want your C-based rules to appear in the info report either way (whether they are selected for checking or not), wrap them with a needToRun command, as shown in the following example:

```
if (needToRun(tagName.c_str())) {
/* rule source code here */
...
  }
}
```

For more information, see “needToRun” on page 84.

Example C-based Rules using DQL

Following are some example C-based rules:

- C Example Rule (buffer on clock tree)
- C++ Example Rule (buffer on clock tree)
- Another C++ Example Rule (asynchronous inputs to clock system clocked twice)

C Example Rule

Here is an example rule written in C that checks for a buffer on a clock tree:

```
#include "dql.h"
#include <stdio.h>
#include <string.h>

void rule_USER2 () {
/* Declaration part */
dqll completePath;
dql inst_list,
clkPin_list, path;
dqh inst, clkPin,
path_elem; DQ_NODE* node;
DQ_OBJECT_TYPES gate_type;
DQ_NODE* pathNode;
char message [1024];

/* Algorithm part */
DQ_debug( "rule_USER2...\n" );
inst_list = DQ_get_all_instances();
forall_in_list(inst,inst_list)
{
  clkPin_list = DQ_get_all_clock_pins_in_instance(inst) ;
  forall_in_list (clkPin, clkPin_list)
  {
    completePath = DQ_complete_trace_backward_to_seq_and_pi ( clkPin,
DQ_GIVE_EDGE_INFO );
    forall_in_llist (path, completePath)
    {
      node = (DQ_NODE*) path;
      if (node->thing)
      {
        forall_in_list (path_elem,(dql)node->thing)
       {
          pathNode = (DQ_NODE*) path_elem;
        if( pathNode->type.DQ_STMT != 0 )
          {
```

```
                gate_type = DQ_get_gate_type (path_elem);
               if ( gate_type.DQ_BUFFERED != 0 )
               {
                  strcpy(message,"(USER2) No buffer on clock tree: ");
                 strcat (message,DQ_get_instance_location(path_elem));
                 strcat (message,"\n"); EDB_add_messages( message );
               }
             }
          }
        }
      }
      delete_DQ_LLIST(completePath);
    }
    delete_DQ_LIST(clkPin_list);
  }
  delete_DQ_LIST(inst_list);
  }
```

C++ Example Rule

Here is an example rule written in C++ that checks for a buffer on a clock tree:

```
#include "dql.h"
#include <stdio.h>
#include <string> /* C++ STL library */
using namespace std;

extern "C" {

void rule_USER1 () {
  /* Declaration part */
  dqll completePath;
  dql inst_list, clkPin_list, path;
  dqh inst, clkPin, path_elem;
  DQ_NODE* node;
  DQ_OBJECT_TYPES gate_type;
  DQ_NODE* pathNode;
  std::string message ;
  /* Algorithm part */
  DQ_debug( "rule_USER1...\n" );
  inst_list = DQ_get_all_instances();
  forall_in_list(inst,inst_list) {
    clkPin_list = DQ_get_all_clock_pins_in_instance(inst) ;
    forall_in_list (clkPin, clkPin_list) {
      completePath = DQ_complete_trace_backward_to_seq_and_pi(clkPin,
DQ_GIVE_EDGE_INFO );
      forall_in_llist (path, completePath) {
        node = (DQ_NODE*) path;
```

```
        if (node->thing) {
          forall_in_list (path_elem,(dql)node->thing) {
            pathNode = (DQ_NODE*) path_elem;
            if ( pathNode->type.DQ_STMT != 0 ) {
              gate_type = DQ_get_gate_type (path_elem);
              if ( gate_type.DQ_BUFFERED != 0 ) {
                message = string("(USER1) No buffer on clock tree for
clock <") + DQ_get_instance_name(clkPin) + "> " +
DQ_get_instance_location(path_elem);
                EDB_add_messages( message.c_str() );
              }
            }
          }
        }
      }
      delete_DQ_LLIST(completePath);
    }
    delete_DQ_LIST(clkPin_list);
  }
  delete_DQ_LIST(inst_list);
 }
}
```

Another C++ Example Rule

Here is another example rule written in C++ that checks to make sure all asynchronous inputs to a clock system are clocked twice:

```
// RULES RUN IN THIS FILE:
// NTL_CLK05: All async inputs to a clock system must be clocked twice.

#include "dql.h"
#include <stdio.h>
#include <string> /* C++ STL library */

static bool is_synchronized (dqh ff);

extern "C" {
  void rule_MCD () {
    string tagName = "NTL_CLK05";
    string text = "All async inputs to a clock system must be clocked
     twice";
    string message ;
    if (!needToRun(tagName.c_str())) return;
    DQ_debug( "rule_MCD...\n" );
    dql ff_list = DQ_get_all_ffs();
    dqh ff;
    forall_in_list(ff,ff_list) {
```

```
      dqh target_clock_origin = DQ_get_clock_origin(ff);
      DQ_OBJECT_TYPES target_clock_origin_polarity =
        DQ_get_clock_origin_polarity(ff);
      dql source_list = DQ_trace_backward_to_seq_and_pi(ff,false);
      dqh source;
      forall_in_list(source,source_list) {
       DQ_OBJECT_TYPES source_type = DQ_get_type(source);
       if (source_type.DQ_FLIPFLOP || source_type.DQ_LATCH) {
         dqh source_clock_origin = DQ_get_clock_origin(source);
         DQ_OBJECT_TYPES source_clock_origin_polarity =
          DQ_get_clock_origin_polarity(source);
         if ((DQ_get_node_id(source_clock_origin) !=
           DQ_get_node_id(target_clock_origin)) ||
            (source_clock_origin_polarity.DQ_POSEDGE !=
             target_clock_origin_polarity.DQ_POSEDGE)) {
           if (!is_synchronized(ff)) {
             message = "(" + tagName + ") " + text +
               DQ_get_instance_location(ff);
             EDB_add_messages( message.c_str() );
           }

         }
       }
      }
      delete_DQ_LIST(source_list);
    }
    delete_DQ_LIST(ff_list);
  }
}
/* Model of synchronizer pattern to filter:
                                 _________________
                    ---         / ---           --- \
                ----| |-----------| |-----------| |-----
                    | |   MCD     | |           | |
                    -^-           -^-           -^-
   Clk1-------------|              |-------------|---------- Clk2
*/
bool is_synchronized (dqh ff) {
  int nbFanout = DQ_getn_fanout(ff);
  // No fanout allowed in between the two synch-ffs
  if (nbFanout != 1) return false;
  dqh gate = DQ_get_fanout(ff,0);
  DQ_OBJECT_TYPES gate_type = DQ_get_type(gate);
  // Simple buffering allowed only
  if (gate_type.DQ_COMPLEX) return false;
  dqh output = DQ_get_output(gate);
  DQ_OBJECT_TYPES output_type = DQ_get_type(output);
  // It has to be a Flip-Flop
  if (output_type.DQ_FLIPFLOP == 0) return false;
```

```
  dqh target_clock_origin = DQ_get_clock_origin(ff);
  DQ_OBJECT_TYPES target_clock_origin_polarity =
    DQ_get_clock_origin_polarity(ff);
dqh synch_clock_origin = DQ_get_clock_origin(output);
DQ_OBJECT_TYPES synch_clock_origin_polarity =
DQ_get_clock_origin_polarity(output);
  // The synch has to have same clock as the target
  if ((DQ_get_node_id(target_clock_origin) !=
    DQ_get_node_id(synch_clock_origin)) ||
              (target_clock_origin_polarity.DQ_POSEDGE !=
               synch_clock_origin_polarity.DQ_POSEDGE)) {
    return false;
  }
  return true;
}
```

Compiling C-based Rules

This section applies to both DQL and CQL. You must use .c extension for C files and .cc/.cpp/.cxx extension for C++ files. Compile your completed C-based rules as shown in the following platform-specific examples:

Solaris

```
% g++ -O2 -Wall -I$LEDA_PATH/rules/include -fno-exceptions -c -o \
       rule_USER1-sol.o rule_USER1.cc
% gcc -O2 -Wall -I$LEDA_PATH/rules/include -c -o \
       rule_USER2-sol.o rule_USER2.c
```

To build a shared library (.so file) for Solaris:

```
g++ -G rule_USER1-sol.o rule_USER2-sol.o -o rule_USER.so
```

On Solaris, you can run the rule (+exec+, +patch) using the object file (.o) name.

Linux

```
% g++ -fPIC -O2 -Wall -I$LEDA_PATH/rules/include -fno-exceptions -c -o \
       rule_USER1-linux.o rule_USER1.cc
% gcc -fPIC -O2 -Wall -I$LEDA_PATH/rules/include -c -o \
       rule_USER2-linux.o rule_USER2.c
```

To build a shared library (.so file) for Linux (.o are not linkable):

```
% g++ -shared -Wl,-soname,rule_user-linux.so -o rule_user-linux.so \
       rule_USER2-linux.o rule_USER1-linux.o
```

On Linux, you must run the rule using the shared library (.so) name.

HP-UX

```
% aCC -AA -I$LEDA_PATH/rules/include -c -o rule_USER1-hp.o rule_USER1.cc
% cc  -Aa -I$LEDA_PATH/rules/include -c -o rule_USER2-hp.o rule_USER2.c
% aCC -b rule_USER2-hp.o rule_USER1-hp.o -o rule_user-hp.sl
```

On HP-UX, you must run the rule using the shared library (.sl) name.

Supported Operating Systems and Compiler Versions

For compiling C-based custom SDC or Netlist checker rules, you must use the same compiler version that was used to build Leda. The following compiler versions must be used with the 2006.06 release for the following operating systems.

Table 2: Compiler Versions for Operating Systems

Supported Operating Systems	Compiler Version
Sun Solaris 32-bit	GNU 2.95
Linux RedHat Enterprise 3.0 32-bit	GNU 2.9.6
Linux RHEL 3.0 Opteron AMD 64-bit	g++ 3.2.3
SUSE 32 bit	gcc3.3.3
SUSE 64 bit	gcc3.3.3
IBM - AIX 32-bit	Visual age C++ 6.0
HP-UX	A.03.33

Testing C-based Rules

This section applies to both DQL and CQL. It is a good idea to test a C-based rule before you add it to a policy. To test a C-based rule, use the following syntax in Leda batch mode. When you execute this command, the Checker also runs all rules in the NETLIST policy (depending on your configuration).

```
% $LEDA_PATH/bin/leda +exec+rule_file.ext+function \
      testcase.v -top top_level_unit
```

where:

rule_file.ext is the object or shared library file that implements the rule. The file name extension (*ext*) is platform-dependent:

Solaris—*rule_file*.o

Linux—*rule_file*.so

HP-UX—*rule_file*.sl

function is the name of the C function that implements the rule without the “rule_” prefix.

Creating a Policy for C-based Rules

This section applies to both DQL and CQL. To include C-based rules in the Leda environment, you use the same flow that you would if you were creating a new rule using the VeRSL or VRSL rule specification languages. But instead of using the VeRSL or VRSL commands and templates to implement a rule, you point to a compiled object or shared library file in your VeRSL code and specify the function that you wrote to implement a rule. The VeRSL code works as a wrapper to pull your C-based rule into the Leda environment. You use VeRSL wrappers regardless of whether the HDL files you want to check are in Verilog or VHDL. Here is the VeRSL wrapper syntax:

```
ruleset MY_RULESET is

Label:
netlist_check "function" in "file.ext"
message "message"
html_document "policy.html#Label"
severity severity

end ruleset
```

where:

Label	Specify the *Label* for the rule (for example, B6000). This *Label* appears in the Error Viewer when this rule is violated.
function	Specify the name of the C function that implements the rule without the “rule_” prefix. For example, if your function is named rule_B6000, specify the function as B6000. Note that the *Label* and *function* name must match exactly.
file.ext	Specify the full path to the object or shared library file where the specified function is implemented (see “Testing C-based Rules” on page 28). You don’t need to specify the path to the file if it is in the current working directory. You can also use an environment variable in the path to the file (for example, $MY_RULES/rules.sl).
message	Specify a general error message in this field; if you don’t, the rule will not be visible in the Rule Wizard. When you specify a message in this field, Leda concatenates it to any message specified in the C source code for the rule (see “Writing Error Messages in C” on page 20).
html_document	Optional. Specify the HTML help file for the policy that contains this rule. Use the rule label as an anchor in the HTML file.

severity	Specify the severity level. Choices include NOTE, WARNING, ERROR, and FATAL.

Put your completed VeRSL code that implements the C-based custom rule in a text file with a .sl extension. You can put as many rules as you want in one .sl VeRSL source file. From there, you can organize one or more rulesets into a policy (set of rules concerning a particular design verification topic).

Create a new custom policy using the Leda Specifier and add your .sl ruleset files to a new policy. Put all of your C-based rules in their own policy, instead of adding them to one of the prepackaged policies that come with the tool. For detailed procedures on how to create new policies for Leda, see the *Leda User Guide*.

Caution

If you add a C-based ruleset to one of the prepackaged policies, this means that you won't be able to configure any of the rules in that policy without deleting your new C-based rulesets.

Selecting C-based Rules for Checking

This section applies to both DQL and CQL. You select C-based rules for the Leda Checker just like any other prepackaged or custom rules. Use the Leda Rule Wizard (from the Leda main window, **Check > Configure**). Each C-based rule that you wrap inside a ruleset in a .sl source file appears as a separate rule that you can select for checking. Starting with version 4.1 netlist checks are on by default. For more information on selecting rules for the Checker, see the *Leda User Guide*.

Note

C-based rules are not configurable. This means you can't use the Leda Rule Wizard to change how the rule works.

Leda executes a C-based rule when the Checker is running only if the corresponding policy or ruleset is selected. To run just one C-based rule on your elaborated design database, select it alone in the Leda Rule Wizard before running the Checker.

Checking C-based Rules in Batch Mode

This section applies to both DQL and CQL. To check a C-based rule in batch mode, you use the same command-line options that you use to check other custom or prepackaged rules. For example, to run the batch mode Checker on a policy that contains a collection of C-based rules, use:

```
% $LEDA_PATH/bin/leda -p policy_name
```

To run the batch mode Checker on a ruleset that contains one rule use:

```
% $LEDA_PATH/bin/leda -r ruleset_name
```

2
C API DQL Reference

Introduction

This chapter provides API reference information for the C interface that you can use to write design netlist checking rules for Leda in C or C++. The C interface is supplied in one big header file named dql.h (located in $LEDA_PATH/rules/include/dql.h). The DQL stands for Design Query Language. Note that all the of the DQL API is “extern C” for the C++ compiler. This chapter documents the contents of the dgl.h file in the following sections:

Basic Data Structures

The Leda C API provides the following basic data structures for DQL:

struct _list_

The _list_ struct contains the DQ_LIST, which is the container list to chain DQ_NODE:

```
typedef struct _list_ {
     DQ_NODE* head;
     DQ_NODE* tail;
  } DQ_LIST;
```

struct DQ_NODE

The DQ_NODE struct implements a smart pointer that allows you to create lists of design objects such as signals and instances:

```
typedef struct DQ_NODE {
     nodeId               thing;
     struct DQ_NODE_TYPES  type;
     struct DQ_NODE *next;
  } DQ_NODE;
```

struct DQ_NODE_TYPES

The DQ_NODE_TYPES struct enumerates all the node types you can manipulate in DQL (see Table 3).

Table 3: DQL Node Types

Data Field	Description
int DQ_DEFINITION:1	Cell definition
int DQ_SIGNAL:1	Signal
int DQ_LIST_TYPE:1	List
int DQ_INSTANCE:1	Instance
int DQ_FILE:1	Source file
int DQ_STMT:1	Gate or statement-like assignment

struct DQ_OBJECT_TYPES

The DQ_OBJECT_TYPES struct contains all the design object types that you can manipulate with DQL (see Table 4).

Table 4: DQL Object Types

Data Field	Description
Instance-related Types	
int DQ_TOP_MODULE :1	Top module of the design.
int DQ_MEMORY :1;	Memory module.
int DQ_GATE_NETLIST :1;	Gate-level sub netlist.
int DQ_BLACKBOX :1	Black box.
int DQ_EXCLUDED :1	Object excluded from analysis.
int DQ_CELL :1	Technology cell.
Signal-related Types	
int DQ_PORT :1	Module port.
int DQ_I :1	Module input.
int DQ_O :1	Module output.
int DQ_IO :1	Module IO.
int DQ_PAD :1	PAD. A special library cell that handles the interface between the chip and the outside world (for example, voltage conversion and static charge protection).
int DQ_PI :1	Design primary input.
int DQ_PO :1	Design primary output.
int DQ_PIO:1	Design primary IO.
int DQ_CLOCK:1	Clock signal.
int DQ_S_SET :1	Synchronous set signal.
int DQ_S_RESET :1;	Synchronous reset signal.
int DQ_A_SET :1	Asynchronous set signal.
int DQ_A_RESET :1	Asynchronous reset signal.
int DQ_FLIPFLOP :1	Flip-flop

Table 4: DQL Object Types (Continued)

Data Field	Description
int DQ_LATCH :1	Latch
int DQ_Q :1	Q output of sequential element.
int DQ_QN :1	QN output of sequential element.
int DQ_DATA :1	Data in of sequential element.
int DQ_LOOP :1	Asynchronous loop.
int DQ_WRITE_DATA :1	Memory write data port.
int DQ_READ_DATA :1	Memory read data port.
int DQ_WRITE_ADDRESS :1	Memory write address port.
int DQ_READ_ADDRESS :1	Memory read address port.
int DQ_WRITE_EN :1	Memory write enable control signal.
int DQ_READ_EN :1	Memory read enable control signal.
int DQ_CONTROL :1	Control signal (other kind of control).
int DQ_TRISTATE :1	Three-stated signal (0, 1, Z).
int DQ_SCAN_IN :1	Scan insertion pin.
int DQ_SCAN_CLOCK :1	Scan clock pin.
int DQ_ENABLE :1	Enable pin.
int DQ_SCAN_ENABLE :1	Scan enable pin.
int DQ_NOTIFIER :1	Notifier pin.
Gate Properties (access with DQ_get_gate_type)	
int DQ_NON_INVERTED :1	Non-inverting gate or statement.
int DQ_INVERTED :1	Inverting gate or statement.
int DQ_COMPLEX :1	Complex gate or statement.
int DQ_BUFFERED :1	Buffered gate or statement.
Gate Types (accessed with DQ_get_gate_type)	
int DQ_AND :1	And gate or statement.

Table 4: DQL Object Types (Continued)

Data Field	Description
int DQ_OR :1	Or gate or statement.
int DQ_NAND :1	Nand gate or statement.
int DQ_NOR :1	Nor gate or statement.
int DQ_XOR :1	Exclusive or gate or statement.
int DQ_XNOR :1	Exclusive nor gate or statement.
int DQ_MUX21 :1	2:1 mux gate or statement.
int DQ_MUX41 :1	4:1 mux gate or statement.
int DQ_MUX_N :1	N:1 mux gate or statement.
int DQ_TRIEN :1	Tristate gate or statement.
int DQ_TRIENB :1	Tristate enable bar gate or statement.
int DQ_INV_TRIEN :1	Inverted tristate gate or statement.
int DQ_INV_TRIENB :1	Inverted tristate enable bar gate or statement.
int DQ_INVERTER :1	Inverter gate or statement.
int DQ_BUFFER :1	Buffer gate.
Sub Types For Signals	
int DQ_POSEDGE :1	Positive edge triggered.
int DQ_NEGEDGE :1	Negative edge triggered.
int DQ_ENABLE_HIGH :1	Enabled high (latch gate).
int DQ_ENABLE_LOW :1	Enabled low (latch gate).
int DQ_PRIORITY_PIN :1	Priority pin (for example, set over reset).

Algorithm Helper Data Structures

The Leda C API provides the following algorithm helper data structures for DQL:

struct DQ_STRING_S

The DQ_STRING_S struct is a string container that you can use for secure string manipulation:

```
struct DQ_STRING_S;
  typedef struct DQ_STRING_S* DQ_STRING;
DQ_STRING DQ_string_new();
  void DQ_string_delete(DQ_STRING s);
  void DQ_string_copy(DQ_STRING s, const char* text);
  void DQ_string_append(DQ_STRING s, const char* text);
  const char* DQ_string_get(DQ_STRING s);
```

struct DQ_SET_S

The DQ_SET_S struct is a set container that you can use to create a set of void*:

```
struct DQ_SET_S;
  typedef struct DQ_SET_S* DQ_SET;
DQ_SET  DQ_set_new();
  void DQ_set_delete(DQ_SET s);
  void DQ_set_insert(DQ_SET s, void* element);
  bool DQ_set_find  (DQ_SET s, void* element);
```

struct DQ_SET_STRING_S

The DQ_SET_STRING_S struct is a string container that you can use to create a set of strings:

```
struct DQ_SET_STRING_S;
  typedef struct DQ_SET_STRING_S* DQ_SET_STRING;
DQ_SET_STRING DQ_set_string_new();
  void DQ_set_string_delete(DQ_SET_STRING s);
  void DQ_set_string_insert(DQ_SET_STRING s, const char* element);
  bool DQ_set_string_find  (DQ_SET_STRING s, const char* element);
```

struct DQ_MAP_S

The DQ_MAP_S struct is a map of void* containers that you can use to create a map of void* (key, value).

```
struct DQ_MAP_S;
  typedef struct DQ_MAP_S* DQ_MAP;
DQ_MAP DQ_map_new();
  void DQ_map_delete(DQ_MAP s);
  void DQ_map_insert(DQ_MAP s, void* key, void* element);
  void* DQ_map_find  (DQ_MAP s, void* key);
```

struct DQ_MAP_STRING_S

The DQ_MAP_STRING_S struct is a map of string-void* containers that you can use to create a map of string-void* (key, value).

```
struct DQ_MAP_STRING_S;
  typedef struct DQ_MAP_STRING_S* DQ_MAP_STRING;
DQ_MAP_STRING DQ_map_string_new();
  void DQ_map_string_delete(DQ_MAP_STRING s);
  void DQ_map_string_insert(DQ_MAP_STRING s, const char* key, void*
    element);
  void* DQ_map_string_find (DQ_MAP_STRING s, const char* key);
```

Predefined Constants

The Leda C API provides the predefined constants listed in Table 5 for DQL.

Table 5: DQL Predefined Constants

#define	Value
false	0
FALSE	0
true	1
TRUE	1
Node-type Nesting	
node_type_is_DQ_DEFINITION(I)	((I) ? (((I)->type).DQ_DEFINITION != 0) :0)
node_type_is_DQ_SIGNAL(I)	((I) ? (((I)->type).DQ_SIGNAL != 0) :0)
node_type_is_DQ_LIST_TYPE(I)	((I) ? (((I)->type).DQ_LIST_TYPE != 0) :0)
node_type_is_DQ_INSTANCE(I)	((I) ? (((I)->type).DQ_INSTANCE != 0) :0)
node_type_is_DQ_FILE(I)	((I) ? (((I)->type).DQ_FILE != 0) :0)
node_type_is_DQ_STMT(I)	((I) ? (((I)->type).DQ_STMT != 0) :0)
DQ_forall_in_list(v, M)	for (v= head_DQ_LIST(M); v; v = next_DQ_NODE(v)) Creates an iterator on a list of dqh structs.
forall_in_list(v, M)	DQ_forall_in_list(v,M) For upward compatibility
DQ_forall_in_llist(v, M)	for (v= head_DQ_LLIST(M); v; v = next_DQ_NODE(v)) Creates an iterator on a list of lists.
forall_in_llist(v, M)	DQ_forall_in_llist(v,M) For upward compatibility
Tracing Options	
DQ_CONCISE_REPORT	0 Mode set by “report concise” command.
DQ_NORMAL_REPORT	1 Mode set by “report normal” command.

Table 5: DQL Predefined Constants (Continued)

#define	Value
DQ_VERBOSE_REPORT	2 Mode set by “report verbose” command.
DQ_GIVE_EDGE_INFO	0x00000001 This option makes the trace function generate statement information in between signals for objects in the design (for example, gates, blocks, instances, cells, flip-flops, or latches).
DQ_STOP_AT_PORT	0x00000002 This option makes the trace function stop at any port.
DQ_STOP_AT_ANY_SIGNAL	0x00000004 This option makes the trace function stop at any signal (step one signal at a time).
DQ_GIVE_ALL_PATHS	0x00000008 This option makes the trace function give all paths. Note that this can be memory intensive and time consuming.
DQ_COUNT_LOGIC_LEVEL	0x00000010 This option makes the DQ_scan_forward and DQ_getn_input functions give the maximum logic levels in between flip-flops.
DQ_COUNT_ALL_OCCURRENCES	0x00000020 This option makes the DQ_scan_forward and DQ_getn_input functions count the number of gates scanned.
DQ_CHECK_RECONVERGENT_FANOUT	0x00000040 This option makes the DQ_scan_forward and DQ_getn_input functions check for reconvergent fanout.
DQ_STOP_AT_COMPLEX	0x00000080 This option makes the DQ_scan_forward and DQ_getn_input functions stop at complex gates.
DQ_TRACK_INFO_TRACE	0x000000100 This option makes the trace function check the node number of the current visited node for track info.

Table 5: DQL Predefined Constants (Continued)

#define	Value
DQ_LIST_ARGUMENT	0x000000200 This option tells the function that the object passed is a dql instead of a dqh.
Constant Propagation	
DQ_v0	1 This is logical value 0, encoded as 0001. Use for constant propagation.
DQ_v1	2 This is logical value 1, encoded as 0010. Use for constant propagation.
DQ_vZ	4 This is logical value Z, encoded as 0100. Use for constant propagation.
DQ_vU	8 This is logical value U, encoded as 1000. If you query the value of a module, this is what you get.
DQ_vX	3 This is logical value X, encoded as 0011. Use for constant propagation.
Symbol Simulator	
DQ_POSITIONAL_SIGNAL_INDEX	0
DQ_SYMBOL_SIGNAL_INDEX	1
DQ_CLOCK_SIGNAL_INDEX	2

Typedefs

The Leda C API provides the typedefs listed in Table 6.

Table 6: DQL Typedefs

Data Type	Variable
int	BOOL
int	bool
unsigned long *	dqh
The dqh is the design query handle for an instance or signal.	
unsigned long *	dql
The dql is the design query list of handles.	
unsigned long *	dqll
The dqll is the design query list of lists of handles.	
void *	nodeId
The nodeId is the design query handle Id, instance, or signal Id.	
struct DQ_NODE	DQ_NODE
The DQ_NODE struct implements the smart pointer that allows you to create lists of design objects such as signals and instances.	
struct _list_	DQ_LIST
The DQ_LIST is the container list to chain DQ_NODE	
struct DQ_NODE_TYPES	DQ_NODE_TYPES
struct DQ_OBJECT_TYPES	DQ_OBJECT_TYPES

Design Functions

The Leda C API provides the following predefined design functions for DQL:

node_type_is_DQ_NO_TYPE

Prototype

```
bool node_type_is_DQ_NO_TYPE (DQ_NODE * dq_node)
```

Function

This function returns true if the specified *dq_node* does not match one of the DQL Node Types defined in the interface.

DQ_get_node_id

Prototype

```
nodeId DQ_get_node_id (dqh object)
```

Function

This function returns the nodeId for the specified *object.*

DQ_identical_nodes

Prototype

```
bool DQ_identical_nodes (dqh object1, dqh object2)
```

Function

This function returns true if the specified objects (*object1* and *object2*) are identical.

new_DQ_NODE

Prototype

```
DQ_NODE * new_DQ_NODE ( )
```

Function

This function returns an empty node (dqh).

make_DQ_STMT_NODE

Prototype

```
dqh make_DQ_STMT_NODE (nodeId id)
```

Function

This function returns an empty dqh statement node for the specified *id*.

make_DQ_SIGNAL_NODE

Prototype

```
dqh make_DQ_SIGNAL_NODE (nodeId id)
```

Function

This function returns an empty dqh signal node for the specified *id*. Here is the equivalence for DQ_SIGNAL type:

```
dqh h = make_DQ_SIGNAL_NODE(id);
<======== both do the same thing ==========>
dqh h = new_DQ_NODE();
 DQ_NODE * node = (DQ_NODE *) h;
 node->thing = id;
 node->type.DQ_SIGNAL = 1;
```

make_DQ_INSTANCE_NODE

Prototype

```
dqh make_DQ_INSTANCE_NODE (nodeId id)
```

Function

This function returns an empty dqh instance node for the specified *id*.

delete_DQ_NODE

Prototype

```
void delete_DQ_NODE (DQ_NODE * node)
```

Function

This function deletes the specified *node*.

copy_DQ_NODE

Prototype

```
dqh copy_DQ_NODE (dqh object)
```

Function

This function copies a dqh, where *object* is the node to copy.

new_DQ_LIST

Prototype

```
dql new_DQ_LIST ( )
```

Function

This function returns an empty list of dql.

copy_DQ_LIST

Prototype

```
dql copy_DQ_LIST (dql container)
```

Function

This function returns a copy of a list of dqh, where *container* is the original list of dqh. Returns a shallow copy of the list of dqh.

delete_DQ_LIST

Prototype

```
void  delete_DQ_LIST (dql container)
```

Function

This function deletes the list of dqh (garbage collection), where *container* is the list of dqh.

prepend_DQ_LIST

Prototype

```
void prepend_DQ_LIST (dql container, dqh object))
```

Function

This function prepends the object to the list of dqh, where *container* is the list to prepend to and *object* is the object to add to the list.

append_DQ_LIST

Prototype

```
void  append_DQ_LIST (dql container, dqh object)
```

Function

This function appends the object to the list of dqh, where *container* is the list to append to and *object* is the object to add to the list.

head_DQ_LIST

Prototype

```
dqh head_DQ_LIST (dql container)
```

Function

This function returns the head of the list specified by *container.*

concat_DQ_LIST

Prototype

```
void  concat_DQ_LIST (dql container, dql thing)
```

Function

This function appends the list *thing* to the list *container.*

new_DQ_LLIST

Prototype

```
dqll new_DQ_LLIST ( )
```

Function

This function returns an empty list of dqh.

copy_DQ_LLIST

Prototype

```
dqll copy_DQ_LLIST (dqll container)
```

Function

This function copies the list of lists, where *container* is the list of lists to copy. Returns the dqll.

delete_DQ_LLIST

Prototype

```
void  delete_DQ_LLIST (dqll container)
```

Function

This function deletes the list of lists, where *container* is the list of lists to delete.

append_DQ_LLIST

Prototype

```
void  append_DQ_LLIST (dqll container, dql thing)
```

Function

This function appends the list thing to the list of lists container, where *container* is the list to append to and *thing* is the list to append to the container.

prepend_DQ_LLIST

Prototype

```
void prepend_DQ_LLIST (dqll container, dql thing)
```

Function

This function prepends the list thing to the list of lists container, where *container* is the list to prepend to and *thing* is the list to prepend to the container.

head_DQ_LLIST

Prototype

```
dql head_DQ_LLIST (dqll container)
```

Function

This function returns the head of the list of lists specified by *container*.

next_DQ_NODE

Prototype

```
dqh next_DQ_NODE (dqh h)
```

Function

This function returns the next DQ node in the list.

DQ_get_value

Prototype

```
int  DQ_get_value (dqh signal)
```

Function

This function returns the encoded value of the specified *signal* (DQ_v0, DQ_v1, DQ_vZ, DQ_vU, or DQ_vX).

DQ_set_value

Prototype

```
void  DQ_set_value (dqh signal, int value)
```

Function

This function sets the *value* of the specified *signal* (DQ_v0, DQ_v1, DQ_vZ, DQ_vU, or DQ_vX).

DQ_unset_values

Prototype

```
void  DQ_unset_values ( )
```

Function

This function unsets all the values.

DQ_unset_value

Prototype

```
void DQ_unset_value (dqh signal)
```

Function

This function unsets the value of the specified *signal*.

DQ_set_test_hold

Prototype

```
void DQ_set_test_hold (int value dqh signal)
```

Function

This function constrains the test mode values on primary inputs. Use for compatibility with PrimeTime and DC constant propagation or case analysis. For the *signal* parameter, specify the signal that you want to constrain. For the *value* parameter, specify DQ_v0, DQ_v1, DQ_vZ, DQ_vU, or DQ_vX.

DQ_set_test_assume

Prototype

```
void DQ_set_test_assume (int value dqh signal)
```

Function

This function constrains the test mode values on an internal node. Use for compatibility with PrimeTime and DC constant propagation or case analysis. For the *signal* parameter, specify the signal that you want to constrain. For the *value* parameter, specify DQ_v0, DQ_v1, DQ_vZ, DQ_vU, or DQ_vX.

DQ_set_case_analysis

Prototype

```
void DQ_set_case_analysis (int value dqh signal)
```

Function

This function constrains the test mode values on any node. Use for compatibility with PrimeTime and DC constant propagation or case analysis. For the *signal* parameter, specify the signal that you want to constrain. For the *value* parameter, specify DQ_v0, DQ_v1, DQ_vZ, DQ_vU, or DQ_vX.

DQ_remove_case_analysis

Prototype

```
void DQ_remove_case_analysis (dqh signal)
```

Function

This function unconstrains the specified *signal*. Use for compatibility with PrimeTime and DC constant propagation or case analysis.

DQ_reset_design

Prototype

```
void DQ_reset_design ( )
```

Function

This function resets the design. Use for compatibility with PrimeTime and DC constant propagation or case analysis.

DQ_apply_test_values

Prototype

```
void DQ_apply_test_values ( )
```

Function

This function propagates the test mode values. Use for compatibility with PrimeTime and DC constant propagation.

DQ_apply_case_values

Prototype

```
void DQ_apply_case_values ( )
```

Function

This function propagates the case analysis values. Use for compatibility with PrimeTime and DC case analysis.

DQ_propagate_values

Prototype

```
void  DQ_propagate_values ( )
```

Function

This function propagates the test mode values.

DQ_get_type

Prototype

```
DQ_OBJECT_TYPES DQ_get_type (dqh instanceOrSignal)
```

Function

This function returns the type for the specified *instanceOrSignal*.

DQ_get_def_name

Prototype

```
const char* DQ_get_def_name (dqh instanceOrSignal)
```

Function

This function returns the definition name for the specified *instanceOrSignal*.

DQ_get_instance_name

Prototype

```
char* DQ_get_instance_name (dqh instanceOrSignal)
```

Function

This function returns the instance name for the specified *instanceOrSignal*.

DQ_get_signal_name_in_instance

Prototype

```
char* DQ_get_signal_name_in_instance (dqh signal, dqh instance)
```

Function

This function returns the instance name for the specified *signal* in the context of a particular *instance*.

DQ_get_signal_in_instance

Prototype

```
dqh DQ_get_signal_in_instance (dqh signal, dqh instance)
```

Function

This function returns a pointer for the specified *signal* in the context of a particular *instance*.

DQ_get_instance_location

Prototype

```
char* DQ_get_instance_location (dqh instanceOrSignal)
```

Function

This function returns a pointer to the EDB formatted location for the specified *instanceOrSignal*. The location is in the following format: *hiererchicalName* (File Line Module).

DQ_get_pin_location

Prototype

```
char* DQ_get_pin_location (dqh signal, dqh instance)
```

Function

This function returns the EDB formatted location for the specified pin. The location is in the following format: *hiererchicalName* (File Line Module).

DQ_demangle_instance_name

Prototype

```
char* DQ_demangle_instance_name (const char* mangledName)
```

Function

This function returns the demangled name of the *mangledName* (mangled for Tcl interface). Use a pointer to a signal or instance to specify the *mangledName*.

DQ_get_def_file_name

Prototype

```
const char* DQ_get_def_file_name (dqh instanceOrSignal)
```

Function

This function returns the definition file name for the specified *instanceOrSignal*.

DQ_get_def_line

Prototype

```
int  DQ_get_def_line (dqh instanceOrSignal)
```

Function

This function returns the definition file line for the specified *instanceOrSignal*.

DQ_get_instance_file_name

Prototype

```
const char* DQ_get_instance_file_name (dqh instanceOrSignal)
```

Function

This function returns the file name for the specified *instanceOrSignal*.

DQ_get_instance_line

Prototype

```
int DQ_get_instance_line (dqh instanceOrSignal)
```

Function

This function returns the instance file line for the specified *instanceOrSignal*. For signals, this means the definition line.

DQ_get_instance_parent_name

Prototype

```
const char* DQ_get_instance_parent_name (dqh instanceOrSignal)
```

Function

This function returns the instance parent definition name for the specified *instanceOrSignal*.

DQ_get_mangled_pointer

Prototype

```
char* DQ_get_mangled_pointer (dqh h)
```

Function

This function returns the mangled name for the specified object (*h*) for Tcl interpreter support.

DQ_get_nets_by_name

Prototype

```
dql DQ_get_nets_by_name (char* name, int flags)
```

Function

This function returns a list of signals matching the specified *name*, using Design Compiler (DC) naming conventions.

DQ_get_pins_by_name

Prototype

```
dql DQ_get_pins_by_name (char* name, int flags)
```

Function

This function returns a list of pins matching the specified *name*, using Design Compiler (DC) naming conventions.

DQ_get_ports_by_name

Prototype

```
dql DQ_get_ports_by_name (char* name, int flags)
```

Function

This function returns a list of ports matching the specified *name*, using Design Compiler (DC) naming conventions.

DQ_get_width

Prototype

```
int DQ_get_width (dqh signal)
```

Function

This function returns the width of the specified *signal*. It works for full vectors (top.q) and bits (top.q(1)).

DQ_get_bit

Prototype

```
int DQ_get_bit (dqh signal)
```

Function

This function returns the bit position of the specified *signal*.

DQ_get_mangled_pointer

Prototype

```
char*  DQ_get_mangled_pointer (dqh h)
```

Function

This function returns the mangled name for Tcl interpreter support.

DQ_get_definition

Prototype

```
dqh DQ_get_definition (const char* mangledName)
```

Function

This function returns the module definition for the specified *mangledName* (for the Tcl interpreter).

DQ_get_instance_parent

Prototype

```
dqh DQ_get_instance_parent (dqh instanceOrSignal)
```

Function

This function returns the instance parent for the specified *instanceOrSignal*.

DQ_get_instance_by_name

Prototype

```
dqh DQ_get_instance_by_name (char* hierarchicalName)
```

Function

This function returns the instance for the specified *hierarchicalName*.

DQ_get_top_instance

Prototype

```
dqh DQ_get_top_instance ( )
```

Function

This function returns the top-level instance in the design.

DQ_get_all_instances

Prototype

```
dql DQ_get_all_instances ( )
```

Function

This function returns all instances of the module definition in the design.

DQ_get_max_design_depth

Prototype

```
int DQ_get_max_design_depth ( )
```

Function

This function returns the maximum design depth.

DQ_get_sub_tree

Prototype

```
dql DQ_get_sub_tree (dqh instance)
```

Function

This function returns all the instances underneath the specified *instance* in the design.

DQ_get_all_instances_of

Prototype

```
dql DQ_get_all_instances_of (dqh definition)
```

Function

This function returns all the instances of the specified module *definition* in the design.

DQ_getn_sub_instances

Prototype

```
int  DQ_getn_sub_instances (dqh instance)
```

Function

This function returns the number of sub-instances of the specified *instance* (one level).

DQ_get_sub_instances

Prototype

```
dql DQ_get_sub_instances (dqh instance)
```

Function

This function returns a list of the sub-instances of the specified *instance* (one level).

DQ_get_sub_instance_by_name

Prototype

```
dqh DQ_get_sub_instance_by_name (dqh instance, const char* name)
```

Function

This function returns the sub-instance for the specified *name*.

DQ_get_sub_module_tree

Prototype

```
dqh DQ_get_sub_module_tree (dqh instance)
```

Function

This function returns the list of sub-module definitions under the specified *instance* (all levels).

DQ_get_nets_by_name

Prototype

```
dql DQ_get_nets_by_name (char* hierarchicalName, int flags)
```

Function

This function returns a list of names matching the specified *hierarchicalName*, using DC naming conventions.

DQ_get_pins_by_name

Prototype

```
dql DQ_get_pins_by_name (char* hierarchicalName, int flags)
```

Function

This function returns a list of pins matching the specified *hierarchicalName*, using DC naming conventions.

DQ_get_ports_by_name

Prototype

```
dql DQ_get_ports_by_name (char* hierarchicalName, int flags)
```

Function

This function returns a list of ports matching the specified *hierarchicalName*, using DC naming conventions.

DQ_get_signal_by_name

Prototype

```
dqh DQ_get_signal_by_name (char* hierarchicalName)
```

Function

This function returns the instanced signal for the specified *hierarchicalName*.

DQ_get_all_pis

Prototype

```
dql DQ_get_all_pis ( )
```

Function

This function returns a list of all the primary inputs in the design.

DQ_get_all_pos

Prototype

```
dql DQ_get_all_pos ( )
```

Function

This function returns a list of the primary outputs in the design.

DQ_get_all_pios

Prototype

```
dql DQ_get_all_pios ( )
```

Function

This function returns a list of the primary IOs in the design.

DQ_get_all_tristates

Prototype

```
dql DQ_get_all_tristates ( )
```

Function

This function returns a list of the tristate drivers in the design.

DQ_get_all_ffs

Prototype

```
dql DQ_get_all_ffs ( )
```

Function

This function returns a list of the flip-flops in the design.

DQ_get_all_latches

Prototype

```
dql DQ_get_all_latches ( )
```

Function

This function returns a list of the latches in the design.

DQ_get_all_signals

Prototype

```
dql DQ_get_all_signals ( )
```

Function

This function returns a list of the signals of the design.

DQ_get_all_clock_origins

Prototype

```
dql DQ_get_all_clock_origins ( )
```

Function

This function returns a list of the clock origins of the design. Clocks are traced backwards from clock pins through buffers and inverters.

DQ_get_all_clock_pins

Prototype

```
dql DQ_get_all_clock_pins ( )
```

Function

This function returns a list of all clock pins in the design (sequential block or cell clock pins).

DQ_get_all_set_pins

Prototype

```
dql DQ_get_all_set_pins ( )
```

Function

This function returns a list of the set pins in the design (sequential block or cell set pins).

DQ_get_all_set_origins

Prototype

```
dql DQ_get_all_set_origins ( )
```

Function

This function returns a list of the set origins in the design. Sets are traced backwards from set pins through buffers and inverters.

DQ_get_all_reset_pins

Prototype

```
dql DQ_get_all_reset_pins ( )
```

Function

This function returns a list of the reset pins in the design (sequential block or cell reset pins).

DQ_get_all_reset_origins

Prototype

```
dql DQ_get_all_reset_origins ( )
```

Function

This function returns a list of the reset origins in the design. Resets are traced backwards from reset pins through buffers and inverters.

DQ_get_all_clock_pins_in_instance

Prototype

```
dql DQ_get_all_clock_pins_in_instance (dqh instance)
```

Function

This function returns a list of the clocks pins in the specified *instance*.

DQ_get_all_set_pins_in_instance

Prototype

```
dql DQ_get_all_set_pins_in_instance (dqh instance)
```

Function

This function returns a list of the set pins in the specified *instance*.

DQ_get_all_reset_pins_in_instance

Prototype

```
dql DQ_get_all_reset_pins_in_instance (dqh instance)
```

Function

This function returns a list of the reset pins in the specified *instance*.

DQ_get_all_inputs_in_instance

Prototype

```
dql DQ_get_all_inputs_in_instance (dqh instance)
```

Function

This function returns a list of the inputs pins in the specified *instance*.

DQ_get_all_outputs_in_instance

Prototype

```
dql DQ_get_all_outputs_in_instance (dqh instance)
```

Function

This function returns a list of the outputs pins in the specified *instance*.

DQ_get_all_ios_in_instance

Prototype

```
dql DQ_get_all_ios_in_instance (dqh instance)
```

Function

This function returns a list of the IO pins in the specified *instance*.

DQ_get_all_tristates_in_instance

Prototype

```
dql DQ_get_all_tristates_in_instance (dqh instance)
```

Function

This function returns a list of the tristates drivers in the specified *instance*.

DQ_get_all_ctrl_from_tristate

Prototype

```
dql DQ_get_all_ctrl_from_tristate (dqh signal)
```

Function

This function returns a list of the control signals from the specified tristate *signal*.

DQ_get_all_signals_in_instance

Prototype

```
dql DQ_get_all_signals_in_instance (dqh instance)
```

Function

This function returns a list of the signals in the specified *instance*.

DQ_get_all_ffs_in_instance

Prototype

```
dql DQ_get_all_ffs_in_instance (dqh instance)
```

Function

This function returns a list of the flip-flops in the specified *instance*.

DQ_get_all_latches_in_instance

Prototype

```
dql DQ_get_all_latches_in_instance (dqh instance)
```

Function

This function returns a list of the latches in the specified *instance*.

DQ_getn_gates_in_instance

Prototype

```
int  DQ_getn_gates_in_instance (dqh instance)
```

Function

This function returns the number of gates (built-in primitives) in the specified *instance*.

DQ_get_all_gates_in_instance

Prototype

```
dql DQ_get_all_gates_in_instance (dql instance)
```

Function

This function returns a list of the gates (built-in primitives) in the specified *instance*.

DQ_get_clock_pin

Prototype

```
dql DQ_get_clock_pin (dql signal)
```

Function

This function returns the clock pin associated with the specified *signal*. The *signal* must be a flip-flop or latch.

DQ_get_clock_pins

Prototype

```
dqh DQ_get_clock_pins (dqh signal)
```

Function

This function returns the clock pins associated with the specified *signal*. The *signal* must be a flip-flop or latch. Use for cases of multi-clock flip-flops or latches.

DQ_get_clock_origin

Prototype

```
dqh DQ_get_clock_origin (dqh signal)
```

Function

This function returns the clock origin associated with the specified *signal*. The *signal* must be a flip-flop or latch.

DQ_get_reset_pin

Prototype

```
dqh DQ_get_reset_pin (dqh signal)
```

Function

This function returns the reset pin associated with the specified *signal*. The *signal* must be a flip-flop or latch.

DQ_get_reset_origin

Prototype

```
dqh DQ_get_reset_origin (dqh signal)
```

Function

This function returns the reset origin associated with the specified *signal*. The *signal* must be a flip-flop or latch.

DQ_get_set_pin

Prototype

```
dqh DQ_get_set_pin (dqh signal)
```

Function

This function returns the set pin associated with the specified *signal*. The *signal* must be a flip-flop or latch.

DQ_get_set_origin

Prototype

```
dqh DQ_get_set_origin (dqh signal)
```

Function

This function returns the set origin associated with the specified *signal*. The *signal* must be a flip-flop or latch.

DQ_get_enable_pin

Prototype

```
dqh DQ_get_enable_pin (dqh signal)
```

Function

This function returns the enable pin associated with the specified *signal*. The *signal* must be a flip-flop or latch.

DQ_get_data_pin

Prototype

```
dqh DQ_get_data_pin (dqh signal)
```

Function

This function returns the data pin associated with the specified *signal*. The *signal* must be a flip-flop or latch.

DQ_get_data_pins

Prototype

```
dql DQ_get_data_pins (dqh signal)
```

Function

This function returns a list of data pins associated with the specified *signal*. The *signal* must be a scan flip-flop or scan latch.

DQ_get_scan_enable_pin

Prototype

```
dqh DQ_get_scan_enable_pin (dqh signal)
```

Function

This function returns the scan enable pin associated with the specified *signal*. The *signal* must be a flip-flop or latch.

DQ_get_scan_in_pin

Prototype

```
dqh DQ_get_scan_in_pin (dqh signal)
```

Function

This function returns the scan in pin associated with the specified *signal*. The *signal* must be a flip-flop or latch.

DQ_get_scan_clock_pin

Prototype

```
dqh DQ_get_scan_clock_pin (dqh signal)
```

Function

This function returns the scan clock pin associated with the specified *signal*. The *signal* must be a flip-flop or latch.

DQ_get_scan_clock_pin_polarity

Prototype

```
DQ_OBJECT_TYPES DQ_get_scan_clock_pin_polarity (dqh signal)
```

Function

This function returns the polarity of the scan clock pin associated with the specified *signal*. The *signal* must be a flip-flop or latch. The returned polarity type is in the form:

```
{ DQ_OBJECT_TYPES.DQ_POSEDGE , DQ_OBJECT_TYPES.DQ_NEGEDGE }
```

DQ_get_clock_origin_polarity

Prototype

```
DQ_OBJECT_TYPES DQ_get_clock_origin_polarity (dqh signal)
```

Function

This function returns the polarity of the clock origin for the specified *signal*. The *signal* must be a flip-flop or latch. The returned polarity type is in the form:

```
{ DQ_OBJECT_TYPES.DQ_POSEDGE , DQ_OBJECT_TYPES.DQ_NEGEDGE }
```

DQ_get_reset_origin_polarity

Prototype

```
DQ_OBJECT_TYPES DQ_get_reset_origin_polarity (dqh signal)
```

Function

This function returns the polarity of the reset origin for the specified *signal*. The returned polarity type is in the form:

```
{ DQ_OBJECT_TYPES.DQ_POSEDGE , DQ_OBJECT_TYPES.DQ_NEGEDGE }
```

DQ_get_set_origin_polarity

Prototype

```
DQ_OBJECT_TYPES DQ_get_set_origin_polarity (dqh signal)
```

Function

This function returns the polarity of the set origin for the specified *signal*. The returned polarity type is in the form:

```
{ DQ_OBJECT_TYPES.DQ_POSEDGE , DQ_OBJECT_TYPES.DQ_NEGEDGE }
```

DQ_get_clock_pin_polarity

Prototype

```
DQ_OBJECT_TYPES DQ_get_clock_pin_polarity (dqh signal)
```

Function

This function returns the polarity of the clock pin for the specified *signal*. The returned polarity type is in the form:

```
{ DQ_OBJECT_TYPES.DQ_POSEDGE , DQ_OBJECT_TYPES.DQ_NEGEDGE }
```

DQ_get_reset_pin_polarity

Prototype

```
DQ_OBJECT_TYPES DQ_get_reset_pin_polarity (dqh signal)
```

Function

This function returns the polarity of the reset pin for the specified *signal*. The returned polarity type is in the form:

```
{ DQ_OBJECT_TYPES.DQ_POSEDGE , DQ_OBJECT_TYPES.DQ_NEGEDGE }
```

DQ_get_set_pin_polarity

Prototype

```
DQ_OBJECT_TYPES DQ_get_set_pin_polarity (dqh signal)
```

Function

This function returns the polarity of the set pin for the specified *signal*. The returned polarity type is in the form:

```
{ DQ_OBJECT_TYPES.DQ_POSEDGE , DQ_OBJECT_TYPES.DQ_NEGEDGE }
```

DQ_get_set_pin_type

Prototype

```
DQ_OBJECT_TYPES DQ_get_set_pin_type (dqh signal)
```

Function

This function returns the type (synchronous or asynchronous) of the set pin for the specified *signal*. The returned type is in the form:

```
{ DQ_OBJECT_TYPES.DQ_S_SET , DQ_OBJECT_TYPES.DQ_A_SET }
```

DQ_get_all_ffs_from_clock_origin

Prototype

```
dql DQ_get_all_ffs_from_clock_origin (dqh signal)
```

Function

This function returns a list of all flip-flops controlled from the clock origin of the specified *signal*.

DQ_get_all_ffs_from_reset_origin

Prototype

```
dqh DQ_get_all_ffs_from_reset_origin (dqh signal)
```

Function

This function returns a list of all flip-flops controlled from the reset origin or the specified *signal*.

DQ_get_all_ffs_from_set_origin

Prototype

```
dql DQ_get_all_ffs_from_set_origin (dqh signal)
```

Function

This function returns a list of the flip-flips controlled from the set origin of the specified *signal*.

DQ_get_all_latches_from_clock_origin

Prototype

```
dql DQ_get_all_latches_from_clock_origin (dqh signal)
```

Function

This function returns a list of the latches controlled from the clock origin of the specified *signal*.

DQ_get_all_latches_from_reset_origin

Prototype

```
dql DQ_get_all_latches_from_reset_origin (dql signal)
```

Function

This function returns a list of the latches controlled from the reset origin of the specified *signal*.

DQ_get_all_latches_from_set_origin

Prototype

```
dql DQ_get_all_latches_from_set_origin (dqh signal)
```

Function

This function returns a list of the latches controlled from the set origin of the specified *signal*.

DQ_scan_backward

Prototype

```
int DQ_scan_backward (dqh signal, int flags, DQ_OBJECT_TYPES gate_type);
```

Function

This function scans backward from the specified *signal* and returns the number of occurrences of the specified *gate_type*, where *gate_type* can be FLIPFLOP, LATCH, PI, PO, AND, OR, etc. This is a fast trace that checks for nodes of a particular type. Use the *flags* argument to specify scan options:

- DQ_STOP_AT_PORT
- DQ_COUNT_LOGIC_LEVEL
- DQ_COUNT_ALL_OCCURRENCES
- DQ_CHECK_RECONVERGENT_FANOUT
- DQ_LIST_ARGUMENT
- DQ_IGNORE_CONSTANT_SIGNALS

DQ_scan_forward

Prototype

```
int DQ_scan_forward (dqh signal, int flags, DQ_OBJECT_TYPES gate_type);
```

Function

This function scans forward from the specified *signal* and returns the number of occurrences of the specified *gate_type*, where *gate_type* can be FLIPFLOP, LATCH, PI, PO, AND, OR, etc. This is a fast trace that checks for nodes of a particular type. Use the *flags* argument to specify scan options:

- DQ_STOP_AT_PORT
- DQ_COUNT_LOGIC_LEVEL
- DQ_COUNT_ALL_OCCURRENCES
- DQ_CHECK_RECONVERGENT_FANOUT
- DQ_LIST_ARGUMENT

Note

This function correctly handles multi-output nodes such as splits. This affects trace forward results.

DQ_get_scan_matches

Prototype

```
dql DQ_get_scan_matches ( )
```

Function

This function returns a list of signals that matches the last DQ_scan_forward or DQ_scan_backward scan stopping options.

DQ_complete_trace_forward_to_complex_logic

Prototype

```
dqll DQ_complete_trace_forward_to_complex_logic (dql signal, int flags)
```

Function

This function returns a list of lists of paths with all intermediate signals, where *signal* is the signal to trace and *flags* are tracing options (DQ_GIVE_EDGE_INFO, DQ_STOP_AT_PORT, DQ_STOP_AT_ANY_SIGNAL, DQ_GIVE_ALL_PATHS, and DQ_TRACK_INFO_TRACE). This function stops when it encounters complex combinational logic.

DQ_complete_trace_forward_to_seq_and_po

Prototype

```
dqll DQ_complete_trace_forward_to_seq_and_po (dqh signal, int flags)
```

Function

This function returns a list of lists of paths with all intermediate signals, where *signal* is the signal to trace and *flags* are tracing options (DQ_GIVE_EDGE_INFO, DQ_STOP_AT_PORT, DQ_STOP_AT_ANY_SIGNAL, DQ_GIVE_ALL_PATHS, and DQ_TRACK_INFO_TRACE). This function stops when it encounters sequential elements and primary output signals.

DQ_complete_trace_backward_to_seq_and_pi

Prototype

```
dqll DQ_complete_trace_backward_to_seq_and_pi (dqh signal, int flags)
```

Function

This function returns a list of lists of paths with all intermediate signals, where *signal* is the signal to trace and *flags* are tracing options (DQ_GIVE_EDGE_INFO, DQ_STOP_AT_PORT, DQ_STOP_AT_ANY_SIGNAL, DQ_GIVE_ALL_PATHS, and DQ_TRACK_INFO_TRACE). This function stops when it encounters sequential elements and primary input signals.

DQ_complete_trace_backward_to_complex_logic

Prototype

```
dqll DQ_complete_trace_backward_to_complex_logic (dqh signal, int flags)
```

Function

This function returns a list of lists of paths with all intermediate signals, where *signal* is the signal to trace and *flags* are tracing options (DQ_GIVE_EDGE_INFO, DQ_STOP_AT_PORT, DQ_STOP_AT_ANY_SIGNAL, DQ_GIVE_ALL_PATHS, and DQ_TRACK_INFO_TRACE). This function stops when it encounters complex combinational logic.

DQ_getn_input

Prototype

```
int  DQ_getn_input (dqh gate)
```

Function

This function returns the number of inputs to the specified *gate*.

DQ_get_input

Prototype

```
dqh DQ_get_input (dqh gate, int index)
```

Function

This function returns the *index*-th input to the specified *gate*.

DQ_get_output

Prototype

```
dqh DQ_get_output (dqh gate)
```

Function

This function returns the output signal from the specified *gate*.

DQ_get_all_outputs

Prototype

```
dql DQ_get_all_outputs (dqh gate)
```

Function

This function returns all output signals from the specified *gate*.

DQ_getn_driver

Prototype

```
int DQ_getn_driver (dqh signal)
```

Function

This function returns the number of driving gates for the specified *signal*.

DQ_get_driver

Prototype

```
dqh DQ_get_driver (dqh signal, int index)
```

Function

This function returns the driver gate specified by the *signal* and *index*.

DQ_get_all_driver

Prototype

```
dql DQ_get_all_driver (dqh signal)
```

Function

This function returns a list of the driver gates for the specified *signal*, and appends fanouts from overlapping part selects to the list. This affects results from backward scans and traces.

DQ_getn_fanout

Prototype

```
int DQ_getn_fanout (dqh signal)
```

Function

This function returns the number of fanout gates for the specified *signal*.

DQ_get_fanout

Prototype

```
dqh DQ_get_fanout (dqh signal, int index)
```

Function

This function returns the fanout gate specified by the *index* for the *signal*.

DQ_get_all_fanout

Prototype

```
dql DQ_get_all_fanout (dqh signal)
```

Function

This function returns a list of the fanout gates for the specified *signal*, and appends fanouts from overlapping part selects to the list. This affects results from forward scans and traces.

DQ_get_gate_type

Prototype

```
DQ_OBJECT_TYPES DQ_get_gate_type (dqh gate)
```

Function

This function returns the specified *gate* type using DQ_OBJECT_TYPES.

DQ_trace_forward_to_seq_and_po

Prototype

```
dql DQ_trace_forward_to_seq_and_po (dqh signal, bool stopAtPort)
```

Function

This function returns a list of ending signals for the trace forward, stopping at primary output signals and sequential elements. The trace stops at the first port found.

DQ_trace_forward_to_complex_logic

Prototype

```
dql DQ_trace_forward_to_complex_logic (dqh signal, bool stopAtPort)
```

Function

This function returns the list of ending signals for the trace forward, stopping at complex logic elements. The trace stops at the first port found.

DQ_trace_backward_to_seq_and_pi

Prototype

```
dql DQ_trace_backward_to_seq_and_pi (dqh signal, bool stopAtPort)
```

Function

This function returns the list of ending signals for the trace backward, stopping at primary input signals and sequential elements The trace stops at the first port found.

DQ_trace_backward_to_complex_logic

Prototype

```
dql DQ_trace_backward_to_complex_logic (dqh signal, int flags)
```

Function

This function returns a list of paths with all intermediate signals, where *signal* is the signal to trace and *flags* are tracing options (DQ_GIVE_EDGE_INFO, DQ_STOP_AT_PORT, DQ_STOP_AT_ANY_SIGNAL, and DQ_GIVE_ALL_PATHS.) This function stops when it encounters complex combinational logic.

DQ_signature

Prototype

```
long DQ_signature (dqh signal)
```

Function

This function returns the signature of the specified *signal*. The signature is a long value that results from conversion of the *signal* name.

DQ_regexp

Prototype

```
bool DQ_regexp (const char* obj, const char format)
```

Function

This function returns true (1) if the string specified in *obj* matches the extended regular expression specified in *format*; else returns false (0).

DQ_debug

Prototype

```
void DQ_debug (const char* outputText)
```

Function

This function prints the *outputText* on STDOUT if debug mode is enabled.

DQ_debug_int

Prototype

```
void DQ_debug_int (const char* outputText, int outputInt)
```

Function

This function prints the *outputText* and integer value on STDOUT if debug mode is enabled.

DQ_setup_rule_statistics

Prototype

```
void DQ_setup_rule_statistics (const char* tagName, const char*
    mainText)
```

Function

This function initializes the time and memory data used to measure rule performance. Specify the rule label in the *tagName* argument and the rule warning message text in the *mainText* argument.

DQ_max_time_reached

Prototype

```
bool DQ_max_time_reached ( )
```

Function

This function allows you to break out of a for loop when the maximum time specified for a rule has been reached.

EDB_add_messages

Prototype

```
void EDB_add_messages (const char* formattedMessage)
```

Function

This function prints the specified *formattedMessage* to the log file. There are two Error Database (EDB) formats:

For single-line messages, use:

formattedMessage := (*Tag*) *Text Location*

The *Tag* number corresponds to the rule label.

Text can be any length with blanks.

The *Location* is a formatted location returned by the DQ_get_instance_location function in this form:

<*hierName*> (File Line Module)

For multi-line messages, use:

formattedMessage := (*Tag*) (Signature= <*sig*>) *Text Location*

TAB Text Location

TAB Text Location

The *Tag* number corresponds to the rule label.

TAB must be at least three spaces or a Tab character.

The *sig* must be computed with the hierarchical names of all the signals involved in the message. Use the procedure signature <*hier*> and add all the signatures together.

Text can be any length with blanks.

The *Location* is a formatted location returned by the DQ_get_instance_location function in this form:

<*hierName*> (File Line Module)

DQ_get_main_text

Prototype

```
const char* DQ_get_main_text (const char* text)
```

Function

This function returns a pointer to the error message text specified for a C-based rule in the C code itself. Note that Leda appends this message text to the text specified in the VeRSL wrapper used to integrate the compiled rule into the Leda environment (see "Writing Error Messages in C" on page 20).

needToRun

Prototype

```
bool needToRun (const char* tagName)
```

Function

This function returns true (1) if the rule specified by *tagName* needs to be executed.

DQ_get_rule_parameter

Prototype

```
const char* DQ_get_rule_parameter (const char* tagName, const char*
    paramName, const char* paramDefaultValue)
```

Function

This function returns the overridden parameter value for the specified *tagName*, *paramName* and *paramDefaultValue*, or the default value if no override was performed.

DQ_get_db_attribute

Prototype

```
const char* DQ_get_db_attribute (const char* sigInst, const char*
    attribute)
```

Function

This function returns the value of built-in or user-defined attributes in a .db library. Specify the signal instance (*sigInst*) and *attribute* name (for example, max_capacitance, max_fanout, function).

DQ_get_db_attribute_from_handle

Prototype

```
const char* DQ_get_db_attribute (dqh sigInst, const char* attribute)
```

Function

This function returns the value of built-in or user-defined attributes in a .db library. Specify the signal instance (*sigInst*) handle (dqh) and *attribute* name (for example, max_capacitance, max_fanout, function).

DQ_sfdb_get_all_source_files

Prototype

```
dql DQ_sfdb_get_all_source_files ( )
```

Function

This function returns a list of all source files in the design.

DQ_sfdb_get_all_library_files

Prototype

```
dql DQ_sfdb_get_all_library_files ( )
```

Function

This function returns a list of the library files in the design

DQ_sfdb_get_all_include_files

Prototype

```
dql DQ_sfdb_get_all_include_files ( )
```

Function

This function returns a list of the include files in the design

DQ_sfdb_get_file_name

Prototype

```
char* DQ_sfdb_get_file_name (dqh h)
```

Function

This function returns the file name with a char*. Note that you must free the string.

DQ_sfdb_get_file_and_line

Prototype

```
char* DQ_sfdb_get_file_and_line (const char* fileName, int line)
```

Function

This function returns the actual source line for the specified *fileName* and *line* number.

DQ_sfdb_get_fileHandle_and_line

Prototype

```
char* DQ_sfdb_get_fileHandle_and_line (dqh fileHandle, int line)
```

Function

This function returns the actual source line for the specified *fileHandle* and *line* number.

DQ_set_scan_path

Prototype

```
void DQ_set_scan_path (const char* path)
```

Function

This function creates a new scan chain for the specified *path*.

DQ_get_scan_paths

Prototype

```
void DQ_get_scan_paths ( )
```

Function

This function returns a list of a all the scan chains.

DQ_set_scan_signal

Prototype

```
void DQ_set_scan_signal (const char* path, const char* type, dqh signal)
```

Function

This function specifies a *signal* to be part of a particular scan chain (*path*), with a particular function (*type*), where *type* can be:

- test_clock
- test_clock_falling
- test_reset
- test_reset_inverted
- test_scan_clock

- test_scan_clock_a
- test_scan_clock_b
- test_scan_enable
- test_scan_enable_inverted
- test_scan_in
- test_scan_out

DQ_get_scan_signals

Prototype

```
dql DQ_get_scan_signals (const char* path, const char* type)
```

Function

This function returns a list of all signals of the specified *type* in the specified scan chain (*path*), where type can be:

- test_clock
- test_clock_falling
- test_reset
- test_reset_inverted
- test_scan_clock
- test_scan_clock_a
- test_scan_clock_b
- test_scan_enable
- test_scan_enable_inverted
- test_scan_in
- test_scan_out

DQ_get_scan_signal

Prototype

```
dqh DQ_get_scan_signal (const char* path, const char* type)
```

Function

This function returns a signal of the specified *type* in the specified scan chain (*path*), where type can be:

- test_clock
- test_clock_falling
- test_reset
- test_reset_inverted
- test_scan_clock
- test_scan_clock_a
- test_scan_clock_b
- test_scan_enable
- test_scan_enable_inverted
- test_scan_in
- test_scan_out

DQ_init_track_info

Prototype

```
void DQ_init_track_info ( )
```

Function

This function initializes the track information that is used to generate schematics in the Path Viewer. For more information on using the Path Viewer, see the *Leda User Guide*.

DQ_track_connect

Prototype

```
bool DQ_track_connect (dqh signalSource, dqh signalTarget)
```

Function

This function adds a connection between the *signalSource* and *signalTarget*. This function returns true (1) if the track information generates properly and is appended to the track information buffer successfully.

DQ_get_track_info

Prototype

```
const char* DQ_get_track_info ( )
```

Function

This function returns the track information that needs to be appended to the message in string form, if there is any valid track information after a recent call of the DQ_init_track_info function. To use this string, append it to an error database (edb) message, followed by a signal location.

DQ_add_path_delimiter

Prototype

```
void DQ_add_path_delimiter ( )
```

Function

This function adds a delimiter to the track information if needed.

DQ_track_path

Prototype

```
bool DQ_track_path (dql path)
```

Function

This function generates track information for the specified *path* dql data structure. It returns true (1) if the track information is generated properly and appended to the track information buffer successfully; else returns false (0).

DQ_track_path_list

Prototype

```
bool DQ_track_path_list (dqll pathList, dqh endNode)
```

Function

This function generates a list of track information for the specified *endNode*. If *endNode* is null, it generates track information for all paths in *pathList*. Specify the list of paths using the *pathList* argument. This function returns true (1) if the track information is generated properly and appended to the track information buffer successfully; else returns false (0).

DQ_track_object_connect

Prototype

```
bool DQ_track_object_connect (dqh source, dqh object, dqh target)
```

Function

This function generates track information for the connection between the *source* and *target* signals, including objects specified by dqh *object* data structures. In this context, objects means gates, blocks, instances, cells, flip-flops, or latches (anything that is not a net). Objects enable more informative schematics for analysis in Leda's Path Viewer (see Figure 1).

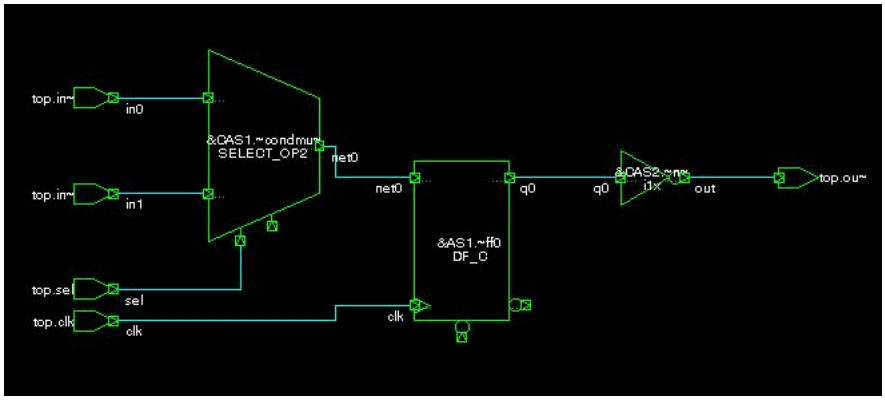

Figure 1: Path Viewer Schematic Example

For information on how to use the Path Viewer, see the *Leda User Guide*. This function returns true (1) if it completes successfully; else returns false (0).

DQ_track_object_path

Prototype

```
bool DQ_track_object_path (dql path)
```

Function

This function generates track information for the specified *path*, including objects in that *path* described by dql data structures. This function returns true (1) if it completes successfully; else returns false (0).

DQ_track_backward_path

Prototype

```
bool DQ_track_backward_path (dql path)
```

Function

This function generates track information for the specified backward *path* described by the dql data structure. This function returns true (1) if it completes successfully; else returns false (0).

DQ_track_backward_path_list

Prototype

```
bool DQ_track_backward_path_list (dqll pathList, dqh endNode)
```

Function

This function generates track information for the specified backward *pathList*. If *endNode* is null, it generates track information for all paths in *pathList*. This function returns true (1) if it completes successfully; else returns false (0).

DQ_track_object_backward_path

Prototype

```
bool DQ_track_object_backward_path (dql path)
```

Function

This function generates track information for the specified backward *path* described by the dql data structure, including objects in that *path*. This function returns true (1) if it completes successfully; else returns false (0).

DQ_track_object_backward_path_list

Prototype

```
bool DQ_track_object_backward_path_list (dqll pathList, dqh endNode)
```

Function

This function generates track information for the specified backward *pathList*, including objects in that *pathList*. If *endNode* is null, it generates track information for all paths in *pathList*. This function returns true (1) if it completes successfully; else returns false (0).

DQ_track_object_path_list

Prototype

```
bool DQ_track_object_path_list (dqll pathList, dqh endNode)
```

Function

This function generates track information for the specified *pathList*, including objects in that *pathList*. If *endNode* is null, it generates track information for all paths in *pathList*. This function returns true (1) if it completes successfully; else returns false (0).

Symbol Simulator Functions

This section provides prototypes, functions, and examples for the Tcl procedures supported in Leda for working with the symbol simulator. You can use the symbol simulator to check designs that have multiple or mixed clock domains. Such designs are common for chips that interface with standard bus protocols such as PCI.

The symbol simulator propagates tuples (symbols) through your design netlist and uses the propagated symbols to identify where signals clocked at different frequencies converge. Such checks are often referred to as clock domain crossing (CDC) checks.

With CDC checks, you want to make sure that two or more synchronized signals of different frequencies crossing clock domains converge only after a predetermined sequential depth in the new clock domain (e.g, often two or three flip-flops). This avoids metastability in the circuit.

Symbols can contain signal names and additional information such as the sequential depth of a circuit element through which a signal is propagated (from a boundary at which the clock frequency changes in the circuit description).

When you propagate a symbol using the commands documented in this section, Leda adds it to a list of zero or more symbols (tuples) currently identified with a circuit element, unless a symbol for the same signal is already in the list. If the symbol is already in the list, propagation of that redundant symbol stops. Propagation of symbols also stops depending on limits you define, such as sequential depth.

You can us the symbol simulator to propagate symbols and identify:

- A convergence point for differently clocked signals
- The location of gray coders
- The location of synchronizers such as flip-flops, by identifying the circuit elements where symbol propagation starts. The simulator checks the list of symbols that result from that symbol propagation.

DQ_init_symbol_simulation

Prototype

```
void DQ_init_symbol_simulation()
```

Function

This function initiates the symbol simulator.

DQ_set_symbol

Prototype

```
void DQ_set_symbol(dqh signal, dqh symbol, dqh clock)
```

Function

This function sets the symbol to simulate, where.

- *signal* is the positional signal
- *symbol* is the symbol to simulate
- *clock* is the clock for the symbol

DQ_simulate_symbols

Prototype

```
void DQ_simulate_symbols(int maxSequentialDepthForSimulation)
```

Function

This function runs the symbol simulation. Use the *maxSequentialDepthForSimulation* parameter to specify the maximum length of the simulation in terms of sequential depth. You can use the following predefined constants for *maxSequentialDepthForSimulation:*

- DQ_POSITIONAL_SIGNAL_INDEX 0

  Index of the positional signal in the result list obtained using the DQ_get_colliding_symbols or DQ_get_symbols_cone_of_influence functions.

- DQ_SYMBOL_SIGNAL_INDEX 1

  Index of the symbol signal in the result list obtained using the DQ_get_colliding_symbols or DQ_get_symbols_cone_of_influence functions.

  DQ_CLOCK_SIGNAL_INDEX 2

  Index of the clock signal in the result list obtained using the DQ_get_colliding_symbols or DQ_get_symbols_cone_of_influence functions.

DQ_get_colliding_symbols

Prototype

```
dqll DQ_get_colliding_symbols (int minSequentialDepth,
     int maxSequentialDepth)
```

Function

This procedure returns a list of lists for colliding symbols between the specified minimum and maximum sequential depths. Set the *minSequentialDepth* parameter to return symbols colliding with sequential depth greater than or equal to that number. Set the *maxSequentialDepth* parameter to return symbols colliding with sequential depth less than or equal to that number. This procedure returns the list of symbols in the following format:

```
{signal1, symbol1, clock1, symbol2, clock2, ... }
{signal2, symbol3, clock3, symbol4, clock4, ... } ...
```

DQ_get_symbols_cone_of_influence

Prototype

```
dqll DQ_get_symbols_cone_of_influence (int minSequentialDepth, int
     maxSequentialDepth)
```

Function

This function returns a list of lists for colliding symbols between the specified minimum and maximum sequential depths. Set the *minSequentialDepth* parameter to return symbols colliding with sequential depth greater than or equal to that number. Set the *maxSequentialDepth* parameter to return symbols colliding with sequential depth less than or equal to that number. This function returns the list of symbols in the following format:

```
{signal1, symbol1, clock1, symbol2, clock2, ... }
{signal2, symbol3, clock3, symbol4, clock4, ... } ...
```

Configuring Reset/PI Data for CDC Rules

When you run certain clock domain crossing (CDC) design rules (for example, NTL_29), the Leda netlist checker infers information about reset and primary input signals in a PI_RESET_CLOCK_MAP.tcl file in the current working directory. You can modify this file so that subsequent runs with the tool use your user-defined input. You can also change the file name using the REPORT_FILE parameter for rules that support it. For example, at the Leda Tcl prompt:

```
leda> rule_set_parameter -rule NTL_29 -parameter REPORT_FILE \
     -value MY_REPORT.tcl
```

You can use the following C functions to control reset and PI data for runs with clock domain crossing rules in the Design policy. For detailed information about the prepackaged netlist rules available in the Design policy, see the *Leda Design Rules Guide*.

DQ_reset_clock_drive_info

Prototype

```
void DQ_reset_clock_drive_info ( )
```

Function

This function resets the clock drive information.

DQ_parse_clock_drive_info

Prototype

```
void DQ_parse_clock_drive_info (const char* file)
```

Function

This function parses your annotated clock drive information *file*.

DQ_set_pi_drive_clock

Prototype

```
void DQ_set_pi_drive_clock ( dqh pi, dqh clock)
```

Function

This function associates the specified primary input (*pi*) with the specified *clock* for CDC checks.

DQ_set_reset_drive_clock

Prototype

```
void DQ_set_reset_drive_clock ( dqh reset, dqh clock)
```

Function

This function associates the specified *reset* with the specified *clock* for CDC checks.

DQ_get_pi_drive_clock

Prototype

```
dqll DQ_get_pi_drive_clock ( )
```

Function

This function returns a list of lists for all primary input/clock pairs in the database.

DQ_get_reset_drive_clock

Prototype

```
dqll DQ_get_reset_drive_clock ( )
```

Function

This function returns a list of lists for all reset/clock pairs in the database.

3
C API CQL Reference

Introduction

This chapter provides API reference information for the C interface that you can use to write Synopsys Design Constraint (SDC) rules for Leda in C or C++. The C interface is supplied in one big header file named cql.h (in $LEDA_PATH/rules/include/cql.h). The cql stands for Constraint Query Language. Note that all of the CQL API is “extern C” for the C++ compiler. This chapter documents the contents of the cql.h file in the following sections:

Basic Data Structures

The Leda C API provides the following basic data structures for CQL:

CQ_LIST

The CQ_LIST struct is the container list to chain CQ_NODE:

```
typedef struct CQ_LIST {
     CQ_NODE* head;
     CQ_NODE* tail;
  } CQ_LIST;
```

CQ_NODE

The CQ_NODE struct implements a smart pointer that allows you to create lists of SDC objects (units, constraints, paths, expressions):

```
typedef struct CQ_NODE {
     struct CQ_NODE *next;
     union {
       nodeId thing;
       float fvalue;
     };
     struct CQ_NODE_TYPES  type;
} CQ_NODE;
```

CQ_NODE_TYPES

The CQ_NODE_TYPES struct enumerates all the node types you can manipulate in CQL (see Table 7).

Table 7: CQL Node Types

Data Field	Description
int CQ_CONSTRAINT:1	Constraint
int CQ_FILE:1	Constraint file descriptor
int CQ_FLOAT:1	Float value
int CQ_HDL_EXP:1	HDL expression
int CQ_HDL_REF:1	HDL reference
int CQ_LIST_TYPE:1	List
int CQ_PATH:1	Path
int CQ_SDC_EXP:1	SDC expression
int CQ_SDC_SIGNAL:1	SDC signal
int CQ_SDC_UNIT:1	SDC unit

Predefined Constants

The Leda C API provides the predefined constants listed in Table 8 for CQL.

Table 8: CQL Predefined Constants

Predefined Constants	Meaning
CQ_NULL_HANDLE	Represents the null handle
CQ_NULL_LIST	Represents the null list
CQ_NULL_LLIST ((cqll) 0)	Represents the null list
CQ_forall_in_list(v, M)	creates an iterator on list of cqh
CQ_forall_in_llist(v, M)	creates an iterator on list of lists
CQ_is_empty_list(l)	test if the list l is empty or not
CQ_is_empty_llist(l)	test if the llist ll is empty or not
CQ_is_null_handle(h)	test if the handle h is null or not
CQ_is_null_list(l)	test if the list l is null or not
CQ_is_null_llist(ll)	test if the list ll is null or not
CQ_is_valid_handle(h)	test if the handle h is valid or not
_QL_TYPE_DEFINED	
node_type_is_CQ_CONSTRAINT(I)	node_type_is_CQ_CONSTRAINT
node_type_is_CQ_FILE(I)	node type is CQ_FILE
node_type_is_CQ_FLOAT(I)	node type is CQ_FLOAT
node_type_is_CQ_HDL_EXP(I)	node type is CQ_HDL_EXP
node_type_is_CQ_HDL_REF(I)	node type is CQ_HDL_REF
node_type_is_CQ_LIST_TYPE(I)	node type is CQ_LIST_TYPE
node_type_is_CQ_PATH(I)	node type is CQ_PATH
node_type_is_CQ_SDC_EXP(I)	node type is CQ_SDC_EXP
node_type_is_CQ_SDC_SIGNAL(I)	node type is CQ_SDC_SIGNAL
node_type_is_CQ_SDC_UNIT(I)	node_type_is_CQ_SDC_UNIT

Typedefs

The Leda C API provides the typedefs listed in Table 9 for CQL.

Table 9: CQL Typedefs

Data Type	Variable
int	BOOL
int	bool
CQ_NODE *	cqh
The cqh is the constraint query handle for an instance or signal.	
CQ_LIST *	cql
The cql is the constraint query list of handles.	
CQ_LIST *	cqll
The cqll is the constraint query list of lists of handles.	
void *	nodeId
The nodeId is the constraint query handle Id, constraint, signal or path Id.	
struct CQ_NODE	CQ_NODE
The CQ_NODE struct implements the smart pointer that allows you to create lists of design objects such as signals and instances.	
struct CQ_LIST	CQ_LIST
The CQ_LIST is the container list to chain CQ_NODE	
struct CQ_NODE_TYPES	CQ_NODE_TYPES

Enumeration Types

The Leda C API provides enumeration values for the CQ_CONSTRAINT_TYPE (see Table 10).

Table 10: Enumeration Types

Enumeration Values	Description
CQ_ALL_CMDS	groups all constraint types.
CQ_CLOCK_CMDS	groups create_clock and generated_clock constraint types.
CQ_CLOCK_CST_CMDS	groups constraint types working on clocks.
CQ_CASE_ANALYSIS_CMDS	groups set_case_analysis and remove_case_analysis constraint types.
CQ_SET_HIERARCHY_ SEPARATOR	set_hierarchy_separator constraint type.
CQ_CREATE_CLOCK	create_clock constraint types.
CQ_CREATE_GENERATED_ CLOCK	create_generated_clock constraint type.
CQ_SET_CLOCK_GATING_ CHECK	set_clock_gating_check constraint type.
CQ_SET_CLOCK_LATENCY	set_clock_latency constraint type.
CQ_SET_CLOCK_ TRANSITION	set_clock_transition constraint type.
CQ_SET_CLOCK_ UNCERTAINTY	set_clock_uncertainty constraint type.
CQ_SET_FALSE_PATH	set_false_path constraint type.
CQ_SET_INPUT_DELAY	set_input_delay constraint type.
CQ_SET_MAX_DELAY	set_max_delay constraint type.
CQ_SET_MIN_DELAY	set_min_delay constraint type.
CQ_SET_MULTICYCLE_PATH	set_multicycle_path constraint type.
CQ_SET_OUTPUT_DELAY	set_output_delay constraint type.
CQ_SET_PROPAGATED_D ELAY	set_propagated_delay constraint type.
CQ_SET_DISABLE_TIMING	set_disable_timing constraint type.

Table 10: Enumeration Types (Continued)

Enumeration Values	Description
CQ_SET_DATA_CHECK	set_data_check constraint type.
CQ_SET_MAX_DYNAMIC_ POWER	set_max_dynamic_power constraint type.
CQ_SET_MAX_LEAKAGE_ POWER	set_max_leakage_power constraint type.
CQ_SET_MAX_TIME_ BORROW	set_max_time_borrow constraint type.
CQ_SET_MIN_POROSITY	set_min_porosity constraint type.
CQ_SET_LOAD	set_load constraint type.
CQ_SET_DRIVE	set_drive constraint type.
CQ_SET_DRIVING_CELL	set_driving_cell constraint type.
CQ_SET_CASE_ANALYSIS	set_case_analysis constraint type.
CQ_SET_FANOUT_LOAD	set_fanout_load constraint type.
CQ_SET_INPUT_TRANSITION	set_input_transition constraint type.
CQ_SET_LOGIC_DC	set_logic_dc constraint type.
CQ_SET_LOGIC_ONE	set_logic_one constraint type.
CQ_SET_LOGIC_ZERO	set_logic_zero constraint type.
CQ_SET_MAX_AREA	set_max_area constraint type.
CQ_SET_MAX_ CAPACITANCE	set_max_capacitance constraint type.
CQ_SET_MAX_FANOUT	set_max_fanout constraint type.
CQ_SET_MAX_TRANSITION	set_max_transition constraint type.
CQ_SET_MIN_TRANSITION	set_min_transition constraint type.
CQ_SET_MIN_ CAPACITANCE	set_min_capacitance constraint type.
CQ_SET_MIN_FANOUT	set_min_fanout constraint type.
CQ_SET_OPERATING_C ONDITIONS	set_operating_conditions constraint type.

Table 10: Enumeration Types (Continued)

Enumeration Values	Description
CQ_SET_RESISTANCE	set_resistance constraint type.
CQ_SET_WIRE_LOAD_MIN_ BLOCK_SIZE	set_wire_load_min_block_size constraint type.
CQ_SET_WIRE_LOAD_MODE	set_wire_load_mode constraint type.
CQ_SET_WIRE_LOAD_ MODEL	set_wire_load_modal constraint type.
CQ_SET_WIRE_LOAD_ SELECTION_GROUP	set_wire_load_selection_group constraint type.
CQ_SET_IDEAL_TRANSITION	set_ideal_transition constraint type.
CQ_SET_IDEAL_LATENCY	set_ideal_latency constraint type.
CQ_GROUP_PATH	group_path constraint type.
CQ_RESET_PATH	reset_path constraint type.
CQ_REMOVE_DRIVING_CELL	remove_driving_cell constraint type.
CQ_REMOVE_MAX_TIME_ BORROW	remove_max_time_borrow constraint type.
CQ_REMOVE_CASE_ ANALYSIS	remove_case_analysis constraint type.
CQ_SET_CAPACITANCE	set_capacitance constraint type.
CQ_SET_DONT_TOUCH	set_dont_touch constraint type.
CQ_SET_DONT_TOUCH_ NETWORK	set_dont_touch_network constraint type.
CQ_SET_IDEAL_NET	set_ideal_net constraint type.
CQ_SET_IDEAL_NETWORK	set_ideal_network constraint type.

Design Functions

The Leda C API provides the following predefined design functions for CQL:

append_CQ_LIST

Prototype

```
void  append_CQ_LIST (cql container, cqh object)
```

Function

This function appends the object to the list of cqh, where *container* is the list to append to and *object* is the object to add to the list.

append_CQ_LLIST

Prototype

```
void  append_CQ_LLIST (cqll container, cql thing)
```

Function

This function appends the list thing to the list of lists container, where *container* is the list to append to and *thing* is the list to append to the container.

concat_CQ_LIST

Prototype

```
void  concat_CQ_LIST (cql container, cql thing)
```

Function

This function appends the list *thing* to the list *container*, where *container* is the list to append to and *thing* is the list to append to the container

copy_CQ_LIST

Prototype

```
cql copy_CQ_LIST (cql container)
```

Function

This function returns a copy of a list of cqh , where *container* is the original list of cqh . Returns a shallow copy of list of cqh.

copy_CQ_LLIST

Prototype

```
cqll copy_CQ_LLIST (cqll container)
```

Function

This function copies the list of lists, where *container* is the list of lists to copy. Returns the `cqll`.

copy_CQ_NODE

Prototype

```
cqh copy_CQ_NODE (cqh object)
```

Function

This function copies a `cqh`, where *object* is the node to copy. This function returns a copy.

CQ_get_all_constraints

Prototype

```
cql CQ_get_all_constraints (cqh SdcUnit, CQ_CONSTRAINT_TYPE,
ConstraintType)
```

Function

This function returns the list of constraints handles corresponding to the given constraint type in the given SDC unit, where *SdcUnit* is the enclosing unit handle and *ConstraintType* is the constraint type. Returns *list* of handles of type CQ_CONSTRAINT or empty list if no match or null *l*ist if not applicable.

Note

If *SdcUnit* is a null handle, then all SDC units in the SDC-B are scanned.

CQ_get_all_constraints_for_sdc_signal

Prototype

```
cql CQ_get_all_constraints_for_sdc_signal (cqh SdcSignal,
CQ_CONSTRAINT_TYPE ConstraintType)
```

Function

This function returns the list of constraints handles (i.e., with the CQ_CONSTRAINT type) corresponding to the given constraint type and constraining the given SDC signal, where *SdcSignal* is the SDC signal handle and *ConstraintType* is the constraint type. Returns *list* of handles of type CQ_CONSTRAINT or empty list if no match or null *l*ist if not applicable.

CQ_get_all_constraints_for_signal

Prototype

```
cql CQ_get_all_constraints_for_signal (cqh HdlRef, CQ_CONSTRAINT_TYPE
ConstraintType)
```

Function

This function returns the list of constraint handles (with the CQ_CONSTRAINT_Type) corresponding to the given constraint type and constraining the given HDL reference, where *HdlRef* is the HDL reference handle and *ConstraintType* is the constraint type. Returns *list* of handles of type CQ_CONSTRAINT or empty list if no match or null list if not applicable.

CQ_get_all_referenced_sdc_signals

Prototype

```
cql CQ_get_all_referenced_sdc_signal (cqh SdcExpr)
```

Function

This function returns the list of SDC signal handles matching the given SDC expression, where *SdcExpr* is the SDC expression handle. Returns *list* of handles of type CQ_SDC_SIGNAL or empty list if no match or null list if not applicable.

CQ_get_all_referenced_signals

Prototype

```
cql CQ_get_all_referenced_signals (cqh HdlExpr)
```

Function

This function returns the list of HDL reference handles matching the given HDL expression, where *HdlExpr* is the HDL expression handle. Returns *list* of handles of type CQ_HDL_REF or empty list if no match or null list if not applicable.

CQ_get_all_sdc_units_for_dimension

Prototype

```
cql CQ_get_all_sdc_units_for_dimension (char* DimensionName, char*
DimensionElementName)
```

Function

This function returns the list of unit handles linked to the dimension. A unit handle point on the top node of the SDC_DB hierarchy used to store all constraints for a given context. A dimension (e.g. MODE=TEST) might belong to several contexts. Therefore, a list of unit handle might be returned, where *DimensionName* is the dimension name and *DimensionElementName* is the name of the element in the dimension. Returns *list* of handles of type CQ_UNIT or empty list if no matching SDC unit.

CQ_get_clock

Prototype

```
cqh CQ_get_clock (cqh Constraint)
```

Function

This function returns the -clock option value of the given constraint, where *Constraint* is the constraint handle. Returns *handle* of type CQ_SDC_EXP or null handle if not applicable.

CQ_get_constraint_file_name

Prototype

```
char* CQ_get_constraint_file_name (cqh Constraint)
```

Function

This function returns the file name of the given constraint, where *Constraint* is the constraint handle. Returns non empty string or empty string if not applicable.

CQ_get_constraint_line

Prototype

```
int CQ_get_constraint_line (cqh Constraint)
```

Function

This function returns the line of the given constraint, where *Constraint* is the constraint handle. Returns non zero value or zero value if not applicable.

CQ_get_constraint_location

Prototype

```
char* CQ_get_constraint_location (cqh Constraint)
```

Function

This function returns the EDB formatted location of the given constraint, where *Constraint* is the constraint handle. Returns non empty string or empty string if not applicable.

CQ_get_divide_by

Prototype

```
int CQ_get_divide_by (cqh Constraint)
```

Function

This function returns the -divide_by option value of the given constraint, where *Constraint* is the constraint handle. Returns non zero value or @ertval zero value if not applicable.

CQ_get_duty_cycle

Prototype

```
float CQ_get_duty_cycle (cqh Constraint)
```

Function

This function returns the -duty_cycle option value of the given constraint, where *Constraint* is the constraint handle. Returns non zero value or @ertval zero value if not applicable.

CQ_get_edge_shift

Prototype

```
cql CQ_get_edge_shift (cqh Constraint)
```

Function

This function returns the -edge_shift option value of the given constraint, where *Constraint* is the constraint handle. Returns non empty list of handles of type CQ_FLOAT or empty list if no edge or null list if not applicable.

CQ_get_edges

Prototype

```
cql CQ_get_edges (cqh Constraint)
```

Function

This function returns the -edges option value of the given constraint, where *Constraint* is the constraint handle. Returns non empty list of handles of type CQ_FLOAT or empty list if no edge or null list if not applicable.

CQ_get_float_value

Prototype

```
float CQ_get_float_value (cqh Object)
```

Function

This function returns the float value.

CQ_get_from

Prototype

```
cql CQ_get_from (cqh Constraint)
```

Function

This function returns the list of HDL or SDC expressions. There is no mix list. Returns *list* of handles of type CQ_HDL_EXP or CQ_SDC_EXP or empty list handle if no object or null list handle if not applicable.

CQ_get_group_path

Prototype

```
cqh CQ_get_group_path (cqh Constraint)
```

Function

This function returns the -group_path option value of the given constraint, where *Constraint* is the constraint handle. Returns *handle* of type CQ_SDC_EXP or null handle if not applicable.

CQ_get_input_transition_fall

Prototype

```
float CQ_get_input_transition_fall (cqh Constraint)
```

Function

This function returns the positional float value of a given constraint, where *Constraint* is the constraint handle. Returns non zero float value or zero float value if not applicable.

CQ_get_input_transition_rise

Prototype

```
float CQ_get_input_transition_rise (cqh Constraint)
```

Function

This function returns the positional float value of a given constraint (set_driving_cell), where *Constraint* is the constraint handle. Returns non zero float value or zero float value if not applicable.

CQ_get_master_clock

Prototype

```
cqh CQ_get_master_clock (cqh Constraint)
```

Function

This function returns the -master_clock option value of the given constraint, where *Constraint* is the constraint handle. Returns *handle* of type CQ_SDC_SIGNAL or null handle if not applicable.

CQ_get_multiply_by

Prototype

```
int CQ_get_multiply_by (cqh Constraint)
```

Function

This function returns the -multiply_by option value of the given constraint, where *Constraint* is the constraint handle. Returns non zero value or @ertval zero value if not applicable.

CQ_get_name

Prototype

```
cqh CQ_get_name (cqh Constraint)
```

Function

This function returns the -name option value of the given constraint, where *Constraint* is the constraint handle. Returns *handle* of type CQ_SDC_SIGNAL or null handle if not applicable.

CQ_get_node_id

Prototype

```
nodeID CQ_get_node_id (cqh Object)
```

Function

This function returns the node id.

CQ_get_object_list

Prototype

```
cql CQ_get_object_list (cqh Constraint)
```

Function

This function returns the list of HDL or SDC expression. There is no mix list. Returns *list* of handles of type CQ_HDL_EXP or CQ_SDC_EXP or empty list handle if no object or null list handle if not applicable.

CQ_get_period

Prototype

```
float CQ_get_period (cqh Constraint)
```

Function

This function returns the -period parameter value of the given constraint, where *Constraint* is the constraint handle. Returns non zero float value or zero float value if not applicable.

CQ_get_referenced_sdc_signal

Prototype

```
cqh CQ_get_referenced_sdc_signal (cqh SdcExpr)
```

Function

This function returns the SDC signal handle matching the given SDC expression. Returns *handle* of type CQ_SDC_SIGNAL or empty handle if no match or multiple match or null handle if not applicable.

CQ_get_referenced_signal

Prototype

```
cqh CQ_get_referenced_signal (cqh HdlExpr)
```

Function

This function returns the HDL reference handle matching the given HDL expression. Returns *handle* of type CQ_HDL_REF or empty handle if no match or multiple match or null handle if not applicable.

CQ_get_simple_name

Prototype

```
char* CQ_get_simple_name (cqh Obj)
```

Function

This function returns the simple name of the given handle. Returns non empty string or empty string if not applicable.

CQ_get_source

Prototype

```
cqh CQ_get_source (cqh Constraint)
```

Function

This function returns the -source parameter value of the given constraint. Returns *handle* of type CQ_HDL_EXP or null handle if not applicable.

CQ_get_through

Prototype

```
cql CQ_get_through (cqh Constraint)
```

Function

This function returns the list of HDL or SDC expressions. There is no mix list. Returns *list* of handle of type CQ_HDL_EXP or CQ_SDC_EXP or empty list handle if no object or null list handle if not applicable.

CQ_get_to

Prototype

```
cql CQ_get_to (cqh Constraint)
```

Function

This function returns the list of HDL or SDC expressions. There is no mix list. Returns *list* of handle of type CQ_HDL_EXP or CQ_SDC_EXP or empty list handle if no object or null list handle if not applicable.

CQ_get_value

Prototype

```
float CQ_get_value (cqh Constraint)
```

Function

This function returns the positional float value of given constraint. Returns non zero float value or zero float value if not applicable.

CQ_get_waveform

Prototype

```
cql CQ_get_waveform (cqh Constraint)
```

Function

This function returns the -waveform option of the given constraint. Returns *list* of handles of type CQ_FLOAT or null list if not applicable.

CQ_has_add

Prototype

```
bool CQ_has_add (cqh Constraint)
```

Function

This function returns true if the -add option is supplied in the given constraint.

CQ_has_add_delay

Prototype

```
bool CQ_has_add_delay (cqh Constraint)
```

Function

This function returns true if the -add_delay option is supplied in the given constraint.

CQ_has_all

Prototype

```
bool CQ_has_all (cqh Constraint)
```

Function

This function returns true if the -all option is supplied in the given constraint.

CQ_has_clock_fall

Prototype

```
bool CQ_has_clock_fall (cqh Constraint)
```

Function

This function returns true if the -clock_fall option is supplied in the given constraint.

CQ_has_constraints_on_sdc_signal

Prototype

```
bool CQ_has_constraints_on_sdc_signal (cqh SdcSignal, CQ_CONSTRAINT_TYPE
ConstraintType)
```

Function

This function returns true if the given SDC signal is constrained by at least one constraint of type the given constraint type, where *SdcSignal* is the SDC signal handle and *ConstraintType* is the constraint type. Returns true if at least one constraint of *ConstraintType* exist or false if not exist.

CQ_has_constraints_on_signal

Prototype

```
bool CQ_has_constraints_on_signal (cqh HdlRef, CQ_CONSTRAINT_TYPE
ConstraintType)
```

Function

This function returns true if the given HDL object reference is constrained at least by one constraint of type *ConstraintType* or not, where *HdlRef* is the HDL reference handle and *ConstraintType* is the constraint type. Returns true if at least one constraint of *ConstraintType* exists or false if none exist.

CQ_has_early

Prototype

```
bool CQ_has_early (cqh Constraint)
```

Function

This function returns true if the -early option is supplied in the given constraint.

CQ_has_fall

Prototype

```
bool CQ_has_fall (cqh Constraint)
```

Function

This function returns true if the -fall option is supplied in the given constraint.

CQ_has_hold

Prototype

```
bool CQ_has_hold (cqh Constraint)
```

Function

This function returns true if the -hold option is supplied in the given constraint.

CQ_has_invert

Prototype

```
bool CQ_has_invert (cqh Constraint)
```

Function

This function returns true if the -invert option is supplied in the given constraint.

CQ_has_late

Prototype

```
bool CQ_has_late (cqh Constraint)
```

Function

This function returns true if the -late option is supplied in the given constraint.

CQ_has_level_sensitive

Prototype

```
bool CQ_has_level_sensitive (cqh Constraint)
```

Function

This function returns true if the -level_sensitive option is supplied in the given constraint.

CQ_has_max

Prototype

```
bool CQ_has_max (cqh Constraint)
```

Function

This function returns true if the -max option is supplied in the given constraint.

CQ_has_min

Prototype

```
bool CQ_has_min (cqh Constraint)
```

Function

This function returns true if the -min option is supplied in the given constraint.

CQ_has_name

Prototype

```
bool CQ_has_name (cqh Constraint)
```

Function

This function returns true if the -name option is supplied in the given constraint (create_clock or create_generated_clock).

CQ_has_network_latency_included

Prototype

```
bool CQ_has_network_latency_included (cqh Constraint)
```

Function

This function returns true if the -network_latency_included option is supplied in the given constraint.

CQ_has_period

Prototype

```
bool CQ_has_period (cqh Constraint)
```

Function

This function returns true if the -period option is supplied in the given constraint.

CQ_has_rise

Prototype

```
bool CQ_has_rise (cqh Constraint)
```

Function

This function returns true if the -rise option is supplied in the given constraint.

CQ_has_setup

Prototype

```
bool CQ_has_setup (cqh Constraint)
```

Function

This function returns true if the -setup option is supplied in the given constraint.

CQ_has_source

Prototype

```
bool CQ_has_source (cqh Constraint)
```

Function

This function returns true if the -source option is supplied in the given constraint.

CQ_has_source_latency_included

Prototype

```
bool CQ_has_source_latency_included (cqh Constraint)
```

Function

This function returns true if the -source_latency_included option is supplied in the given constraint.

CQ_identical_nodes

Prototype

```
bool CQ_identical_nodes (cqh object1, cqh object2)
```

Function

This function returns true if the specified objects (*object1* and *object2*) are identical.

delete_CQ_LIST

Prototype

```
void  delete_CQ_LIST (cql container)
```

Function

This function deletes the list of cqh (garbage collect), where *container* is the list of cqh.

delete_CQ_LLIST

Prototype

```
void  delete_CQ_LLIST (cqll container)
```

Function

This function deletes the list of lists, where *container* is the list of lists to delete.

delete_CQ_NODE

Prototype

```
void delete_CQ_NODE (CQ_NODE * node)
```

Function

This function deletes the specified *node*, where *node* is the node to delete.

find_CQ_LIST

Prototype

```
bool find_CQ_List (cql container, cqh thing)
```

Function

This function returns true if the thing pointed by *thing* is in the given list *container*, where *container* is the list to browse and *thing* is the element to find in the container. Returns a boolean value corresponding to the find result.

head_CQ_LIST

Prototype

```
cqh head_CQ_LIST (cql container)
```

Function

This function returns the head of the list specified by *container*, where *container* is the list of cqh. Returns the cqh.

head_CQ_LLIST

Prototype

```
cql head_CQ_LLIST (cqll container)
```

Function

This function returns the head of the list of lists specified by *container*, where *container* is the list of cql.

make_CQ_CONSTRAINT_NODE

Prototype

```
cqh make_CQ_CONSTRAINT_NODE (nodeId Id)
```

Function

This function returns a cqh constraint node, where *Id* is the node Id. Returns the cqh (the constraint node).

make_CQ_FLOAT_NODE

Prototype

```
cqh make_CQ_FLOAT_NODE (float value)
```

Function

This function returns a cqh float node, where *value* is the float value. Returns the cqh (the float node).

make_CQ_HDL_EXP_NODE

Prototype

cqh **make_CQ_HDL_EXP_NODE** (nodeId *Id*)

Function

This function returns a cqh HDL expression node, where *Id* is the node Id. Returns the cqh (the HDL expression node).

make_CQ_HDL_REF_NODE

Prototype

cqh **make_CQ_HDL_REF_NODE** (nodeId *Id*)

Function

This function returns a cqh HDL reference node, where *Id* is the node Id. Returns the cqh (the HDL reference node).

make_CQ_PATH_NODE

Prototype

cqh **make_CQ_PATH_NODE** (nodeId *Id*)

Function

This function returns a cqh path node, where *Id* is the node Id. Returns the cqh (the path node).

make_CQ_SDC_EXP_NODE

Prototype

cqh **make_CQ_SDC_EXP_NODE** (nodeId *Id*)

Function

This function returns a cqh SDC expression node, where *Id* is the node Id. Returns the cqh (the SDC expression node).

make_CQ_SDC_SIGNAL_NODE

Prototype

```
cqh make_CQ_SDC_SIGNAL_NODE (nodeId Id)
```

Function

This function returns a cqh SDC signal node, where *Id* is the node Id. Returns the cqh (the SDC signal node).

make_CQ_SDC_UNIT_NODE

Prototype

```
cqh make_CQ_SDC_UNIT_NODE (nodeId Id)
```

Function

This function returns a cqh SDC unit node, where *Id* is the node Id. Returns the cqh (the SDC unit node).

merge_CQ_LIST

Prototype

```
void merge_CQ_LIST (cql container, cql thing)
```

Function

This function merge the list thing to the list container removing doublons, where *container* is the list to merge to and *thing* is the list to append to the container.

new_CQ_LIST

Prototype

```
cql new_CQ_LIST ( )
```

Function

This function returns an empty list of cqh.

new_CQ_LLIST

Prototype

```
cqll new_CQ_LLIST ( )
```

Function

This function returns an empty list of lists. Returns the cqll.

new_CQ_NODE

Prototype

```
CQ_NODE * new_CQ_NODE ( )
```

Function

This function returns an empty node (cqh).

next_CQ_NODE

Prototype

```
cqh next_CQ_NODE (cqh h)
```

Function

This function returns the next node of the list.

node_type_is_CQ_NO_TYPE

Prototype

```
bool node_type_is_CQ_NO_TYPE (CQ_NODE * cq_node)
```

Function

This function returns true if the specified c*q_node* does not match one of the CQL Node Types defined in the interface.

Index

Symbols

A

B

C

D

E

F

G

H

I

L

M

N

P

R

S

T

V

W