cadence®

Liberate™ DataBase eXplorer 2.0 Reference Manual

Product Version LIBERATE 21.1
May 2021

Cadence Design Systems, Inc. (Cadence), 2655 Seely Ave., San Jose, CA 95134, USA.

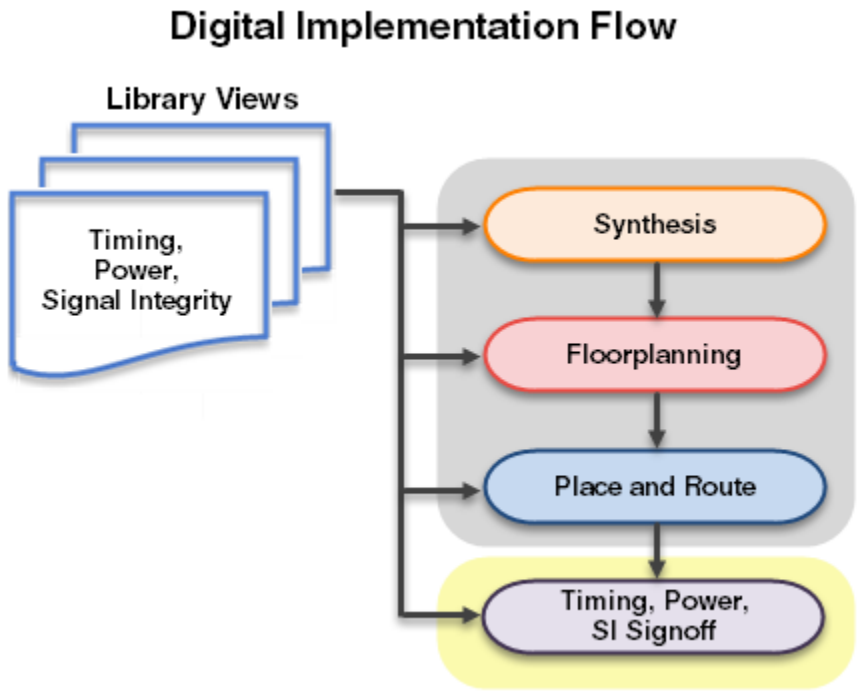
Contents

Preface

Introduction to Characterization

The Role and Importance of Libraries

Creation of electrical views is a prerequisite for any digital design flow. The electrical information stored in the library views is used throughout design implementation from logic synthesis, through design optimization to final signoff verification. Accurate library view creation is essential to ensure close correlation between the design intent and the final silicon.

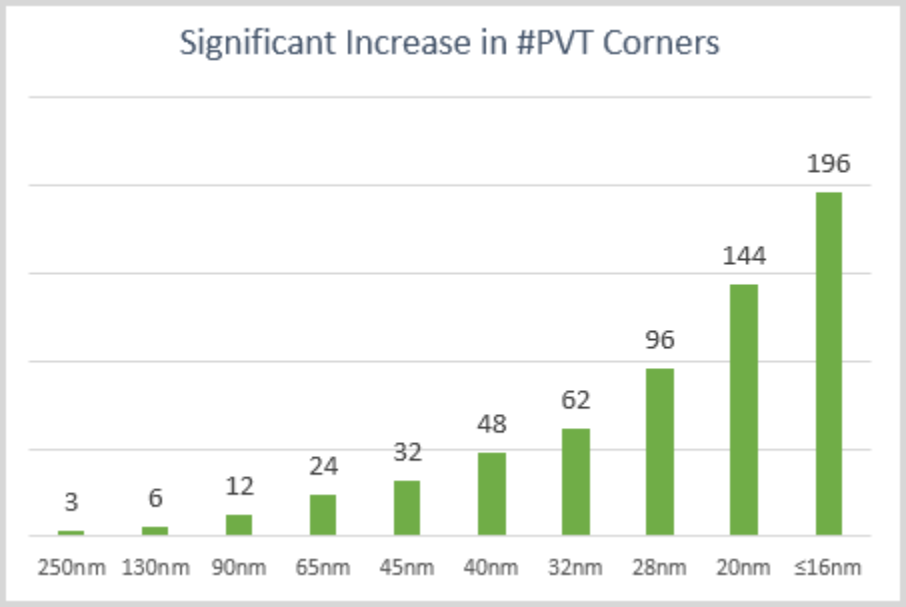


A Growing Problem

In nanometer geometries (65nm or below), the required number of library views is growing dramatically because of issues related to power leakage and process variation. To minimize

power leakage at deep submicron nodes, we see process variations such as LVT, RVT, and HVT (low/regular/high voltage) being utilized. For example, to manage power at 65nm, it is common to have library cells with two or three different threshold values (high threshold to reduce leakage power, lower thresholds to improve performance), and to use two or more on-chip supply voltages. In this scenario, the number of views needed for 65nm will be six times greater than what is needed for 130nm.

The figure below shows the growing trend that requires PVT corners to accurately model the circuit behavior:

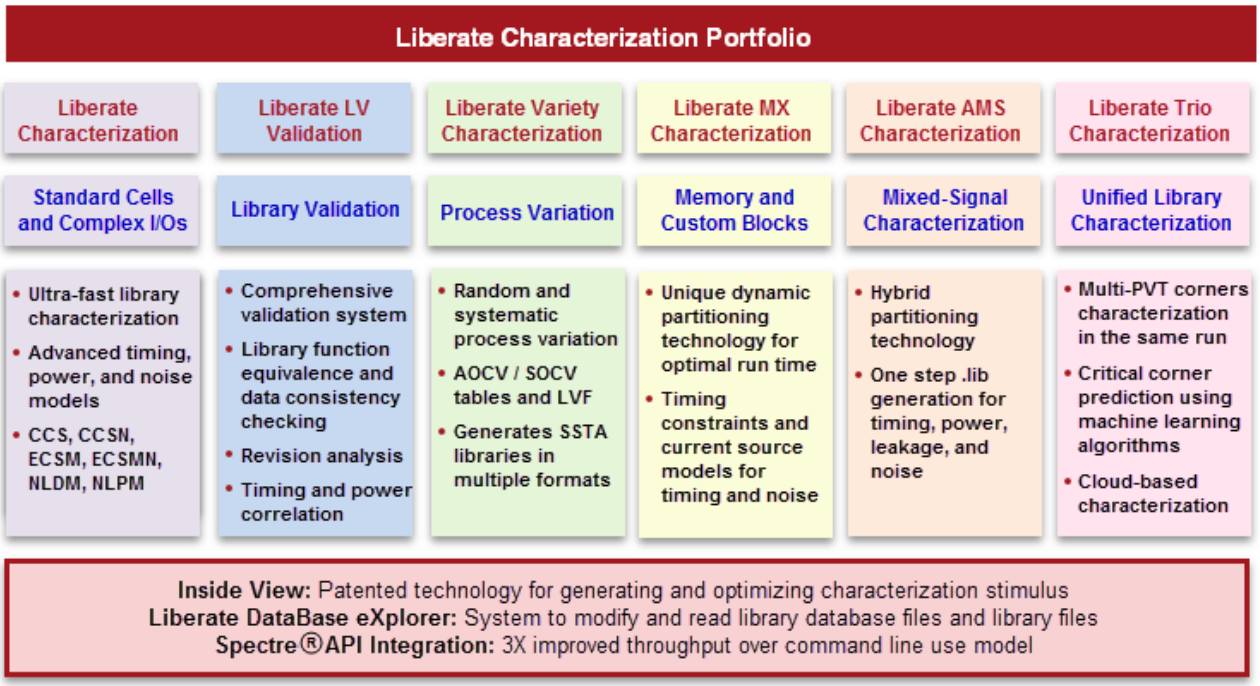


In addition, library views require more advanced models like:

- Current source models CCS and ECSM
- Statistical models – AOCV/SOCV/LVF
- Netlist extraction at various temperatures for Nanometer Process Nodes
- Support multiple foundries to assure flexibility for yield issues
- Support for many more functional designs – 1000+ STD cell, I/O, custom data path, memory and Analog IP

Liberate Characterization Portfolio

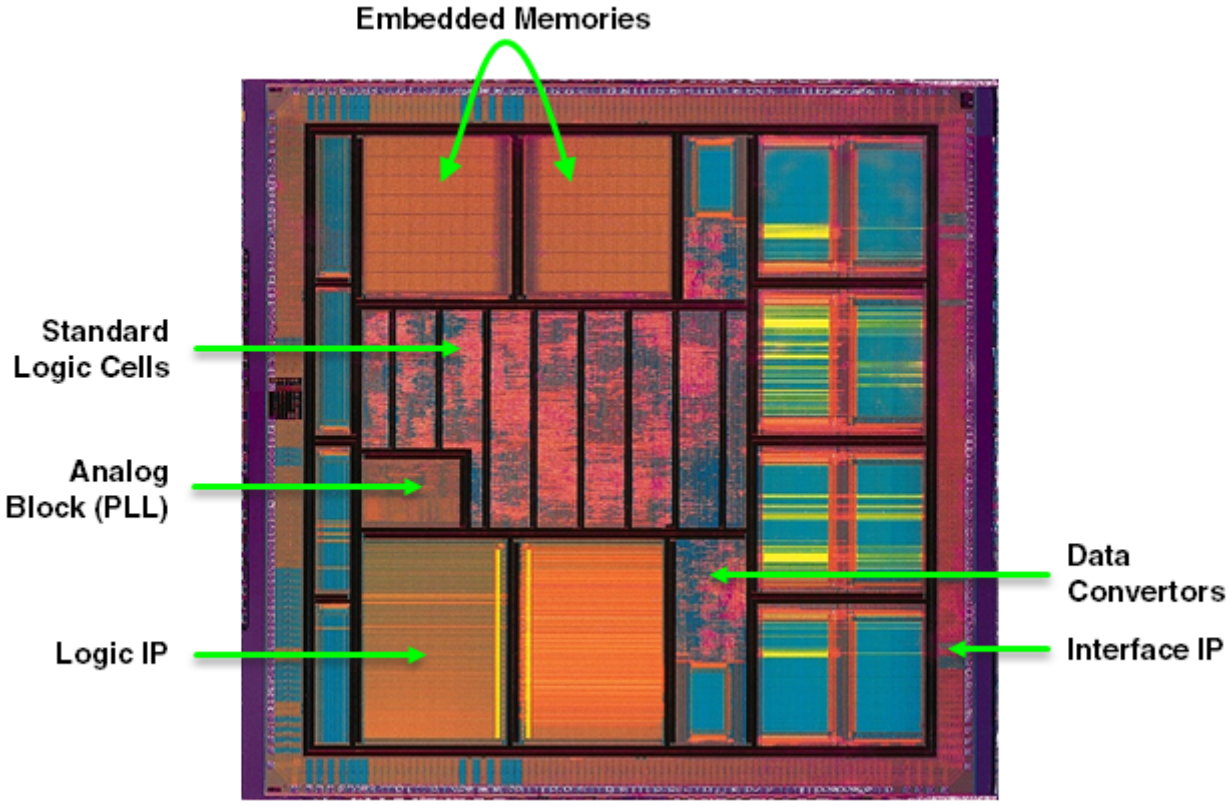
To address all the challenges, Cadence offers Liberate™ Characterization Portfolio that includes the complete set of characterization solutions given below:



The Liberate characterization portfolio intends to provide highly efficient and automated electrical view creation and validation for all IP blocks that including the following:

- Logic and I/O cells (GPIO, PCI, SSTL, PECL, and so on)
- Embedded Memory (SRAM, ROM, Register files, CAM, and so on)
- Custom digital blocks (custom cells, data path, cores, and so on)

- Interface IP and analog blocks (USB, Serdes, DDR, and so on)



In addition, the Liberate DataBase eXplorer (Liberate DBX) system of the Liberate characterization portfolio lets you load and manage the contents of library database files (`.ldb`) and library files (`.lib`).

System and Licensing Requirements

Refer to *LIBERATE Software Licensing and Configuration Guide* for information about the different types of software licenses available to use the products of Liberate characterization portfolio. This guide also describes how to configure the licenses for efficient utilization of the available server and client resources.

For detailed information about the system requirements, see Computing Platforms.

About This Manual

The *Liberate DataBase eXplorer Reference Manual* describes the Cadence® Liberate™ DBX tool. The manual includes opening chapters that describe what Liberate DBX does and how to get started with the tool. Later chapters discuss the commands that can be used with Liberate DBX and the various operations you can perform using Liberate DBX to manipulate data structures.

Audience Profile

This manual is aimed at developers and designers who want to understand the various methods of performing post-processing manipulations on characterized data values in the library database and library files. It assumes that you are familiar with:

- SPICE simulations
- Basic expected behavior of the design being used
- Tool of Liberate Characterization Portfolio from where Liberate DBX is being run
- Basics of the Tcl or Python programming languages

Additional Documents for Reference

For information about known problems and solutions, see *Liberate Characterization Portfolio Known Problems and Solutions*.

For a list of new features in a release, see *Liberate Characterization Portfolio What's New*.

For information about other products in Liberate characterization portfolio, refer to the following manuals:

- *Liberate Characterization Reference Manual* describes the Liberate characterization tool that creates electrical views (timing, power, and signal integrity) in formats such as the Synopsys Liberty (`.lib`) format. This manual also covers information about the Liberate Trio Characterization Suite.
- *Liberate LV Library Validation Reference Manual* describes the Liberate LV validation tool that provides a collection of capabilities used to validate and verify the data consistency, accuracy, and completeness of cell libraries.

- *Liberate Variety Statistical Characterization Reference Manual* describes Liberate Variety characterization tool that characterizes process variation aware timing models and generates libraries for multiple statistical static timing analyzers (SSTA) without requiring re-characterization for each unique format.
- *Liberate MX Memory Characterization Reference Manual* describes Liberate MX characterization tool that provides library creation capabilities to cover memory cores.
- *Liberate AMS Mixed-Signal Characterization Reference Manual* describes Liberate AMS characterization tool that provides library creation capabilities for Analog Mixed Signal (AMS) macro blocks.
- *Liberate API Reference Manual* describes a Tcl interface that allows access to the Liberate characterized Library DataBase (LDB).

Rapid Adoption Kits

Cadence provides Rapid Adoption Kits that demonstrate how to use Liberate characterization portfolio in your design flows. These kits contain design databases and instructions on how to run the design flow.

Typographic and Syntax Conventions

This section describes the typographic and syntax conventions used in this manual.

`text`	Indicates text that you must type as presented in the manual. Typically used to denote command, variable, routine, or argument names that must be typed literally.
`argument`	Indicates text that you must replace with an appropriate argument value.
`< >`	Angle brackets indicate text that you must replace with a single appropriate value. When used with vertical bars, they enclose a list of choices from which you must choose one.
`|`	Vertical bars separate a choice of values. They take precedence over any other character.
`-`	Hyphens denote arguments of commands or variables. Usually arguments denoted in this way are optional but, as noted in the syntax, some are required. The hyphen is part of the name and must be included when the argument is used.

`{ }`	Braces indicate values that must be denoted as a list. When used with vertical bars, braces enclose a set of values from which you must choose one or more. When you specify a list, the values must be enclosed by either quotation marks or braces. For example, `{val1 val2 val3}` and `"val1 val2 val3"` are legal lists.

Some argument are positional and must be used in the order they are shown. Any positional arguments that are used must be given after any arguments denoted with hyphens.

Customer Support

For assistance with Cadence products:

- Contact Cadence Customer Support

  Cadence is committed to keeping your design teams productive by providing answers to technical questions and to any queries about the latest software updates and training needs. For more information, visit: https://www.cadence.com/support

- Log on to Cadence Online Support

  Customers with a maintenance contract with Cadence can obtain the latest information about various tools at: https://support.cadence.com

Feedback about Documentation

You can contact Cadence Customer Support to open a service request if you:

- Find erroneous information in a product manual
- Cannot find in a product manual the information you are looking for
- Face an issue while accessing documentation by using Cadence Help

You can also submit feedback by using the following methods:

- In the Cadence Help window, click the *Feedback* button and follow instructions.
- On the Cadence Online Support Product Manuals page, select the required product and submit your feedback by using the *Provide Feedback* box.

1

Overview of Liberate DataBase eXplorer 2.0

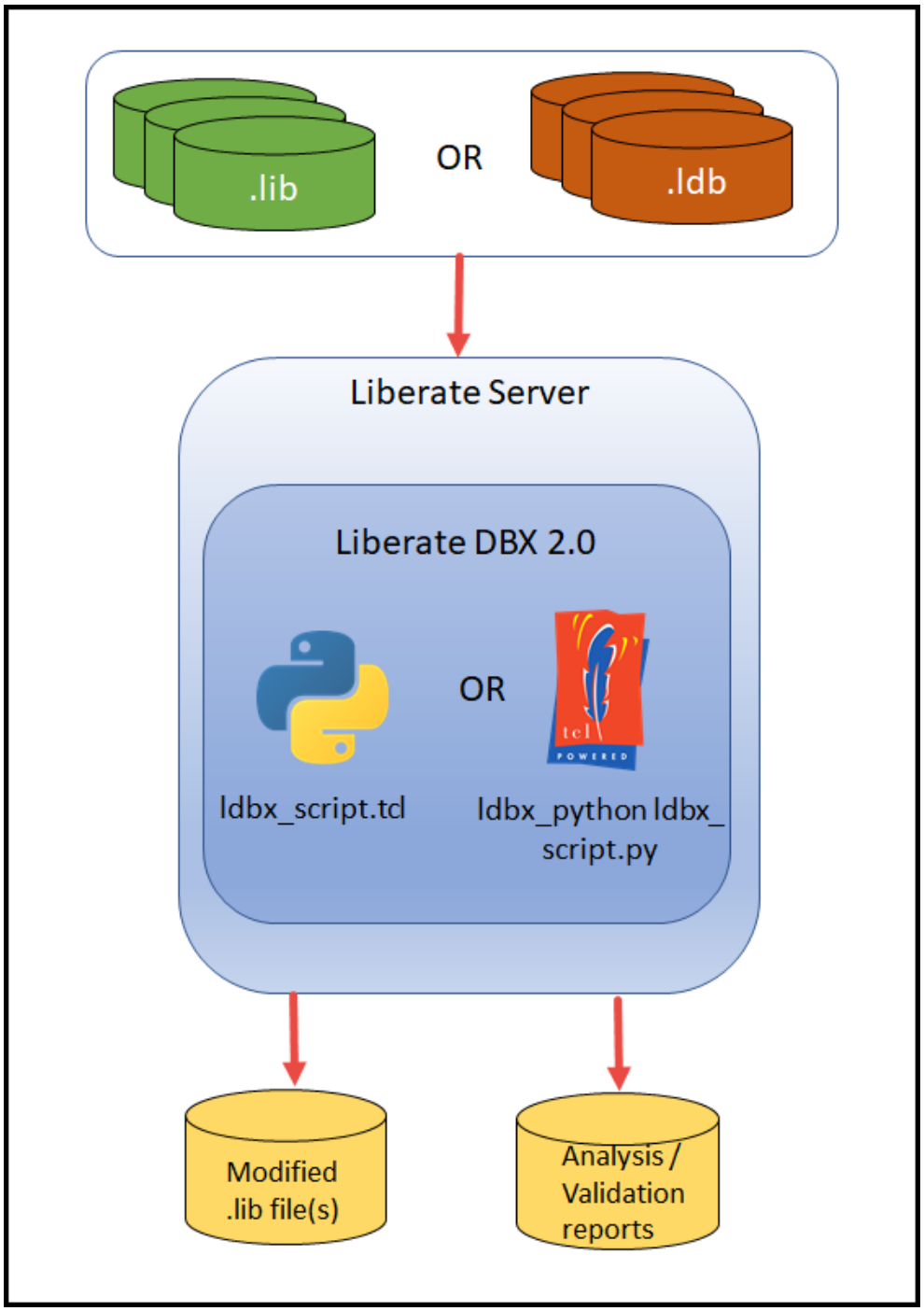
Liberate characterization portfolio provides solutions for performing characterization of libraries, memory, and AMS blocks. However, before the data gets into the final library, some post-processing of the data, such as adding a margin, data smoothening, and copying/ deleting arcs from a library, may be required. All of this post-processing can be done using the Liberate DataBase eXplorer 2.0 system, also known as LDBX 2.0.

LDBX 2.0 lets you:

- Load library files, which can be library database (`.ldb`) files or Liberty (`.lib`) files
- Read the contents of the (`.lib`) loaded files, and manipulate the data and write it into another `.lib` file

LDBX 2.0 is object-oriented and can be run using a Tcl or Python-based interface. You can use various complex functions and equations to perform complex calculations on the library data and write the required results into a new LDB or Library.

The figure below shows the typical flow of information in Liberate DBX:



For information about the licensing requirement to run Liberate DBX, refer to *LIBERATE Software Licensing and Configuration Guide*.

Note: The LDBX 2.0 is the new version of Liberate DataBase eXplorer (LDBX 1.0). LDBX 1.0 is still available and can be accessed like before. However, LDBX 1.0 has been deprecated and should no longer be used.

LDBX 2.0 Use Model

A LDBX session can be run using a Tcl or Python interface in one of the following modes:

- Interactive mode
- Batch mode

LDBX Tcl, Python, and Arc interface tutorial and example script packages are available in the `$ALTOSHOME/etc/tutorial/ldbx/` directory.

ldbx_python Tcl Command

This command is used to run LDBX 2.0 in a Python-based session.

Syntax

`ldbx_python <optional_script_name> <optional_list_of_arguments>`

Options

`<optional_script_name>`

Specifies the complete path to a LDBX Python script.

`<optional_list_of_arguments>`

Lists the arguments passed to the Python interpreter.

Interactive Mode – Starting and Exiting a Liberate DBX Session

Tcl-Based Session

The Tcl-based LDBX interface is built-in as an integral part of Liberate and is immediately available in batch or interactive mode. To enter interactive mode, call the tool without specifying any input file.

Python-Based Session

To start a LDBX session in interactive mode, use the following command on the command prompt of the tool from which you are running LDBX:

```
tools_cmd_prompt > ldbx_python
ldbxPython > #this is the Python interactive prompt.
```

To access the LDBX 2.0 commands, use the python `import` command in the following format:

```
ldbxPython > import ldbx
```

After running the `import` command, you can use the all the Tcl LDBX commands that are documented in the LDBX 2.0 Commands chapter.

For example:

```
ldbxPython > lib = ldbx.read_db("my_liberty_file.lib")
ldbxPython > lib.getName()
example.lib
```

To exit the session, use the following command:

```
ldbxPython > exit()
```

For example, the figure below illustrates how to start and exit a Liberate DBX session from Liberate MX:

```
liberate_mx > ldbx_python

>>>>>>>>>>>>>>> Liberate DataBase eXplorer (LDBX 2.0) <<<<<<<<<<<<<<<

Python 3.6.8 (default, Apr  8 2019, 00:08:27)
[GCC 6.3.0] on linux
Type "help", "copyright", "credits" or "license" for more information.
ldbxPython >
ldbxPython >
ldbxPython > exit ()
Peak memory usage:          455 MB
Peak virtual memory usage:  412 MB
Peak physical memory usage: 43 MB
Wall time      :    0.03 hours (1 minutes 38 seconds)
LIBERATE_MX exited on sj-abhar at Thu Aug 22 02:27:03 2019
```

When `ldbx_python` is run without any arguments, the Tcl shell will start Python in the interactive mode.

Note: Once the interactive mode is used, `ldbx_python` cannot be used again in the same Liberate session.

Running a LDBX Python script with and without arguments

Example 1: Running the python script `myscript.py`:

```
ldbx_python myscript.py
```

Example 2: Running the python script `myscript.py` with two arguments:

```
ldbx_python myscript.py [list  arg1 arg2]
```

The arguments appear in the `sys.argv` list with the sys module in Python. For this example, the arguments would be accessible in the `myscript.py` script as follows:

```
import sys
sys.argv
```

where, `sys.args` is a list of all the arguments.

The first argument is always the name of the script itself. So in this case:

```
sys.argv[0] is "myscript.py"
sys.argv[1] is "arg1"
sys.argv[2] is "arg2"
```

Note that only strings can be passed as arguments from Tcl.

The return code can be captured in Tcl as follows:

```
set return_code [ldbx_python myscript.py]
```

If the script exits with an exception, `return_code` will be set to `1`, otherwise `0`.

To force an exception in Python, you can use the Python raise command. For example, if you want your script to exit with a non-zero exit code, use the following command:

```
raise Exception('my exception')
```

This will exit the Python script and return a code of `1` for the `ldbx_python` return value in Tcl.

Batch Mode – Starting a Liberate DBX Session

Tcl-Based Script

To use batch mode for starting a Tcl-based LDBX session,

1. Create a Tcl script file containing the required LDBX commands.
2. Run the script file directly from the UNIX command prompt with a specific tool's executable using the following command:

```
% <tool_name> <filename>.tcl |& tee <filename>.log
```

Or, source your `ldbx.tcl` script in interactive mode at the Liberate prompt:

```
liberate>  source ldbx_script.tcl
```

Python-Based Script

To run a Python LDBX script,

1. Create a Python-based LDBX scrip
2. Create a Tcl script with `ldbx_python` Tcl command that loads the Python-based LDBX script.
3. Execute the Tcl script as described in step 2 of the Tcl-Based Script section.

Example

```
====myPythonScript.py====
import ldbx
lib = ldbx.read_db("my_liberty_file.lib")
lib.getName()
======================
====ldbx.tcl====
ldbx_python myPythonScript.py
=============
    =>liberate ldbx.tcl |& tee ldbx.log
```

Note: To access the LDBX API commands, your Python script must include the `import ldbx` command.

Running LDBX Python Script from Liberate Command Line

Use the `--python` option on the Liberate command line to start python shell or running python script.

Syntax

```
liberate --python [python_script] [arg1] [arg2] ...
```

Arguments

`[python_script]`

[Optional] Specifies the Python script to be run.

`[arg1] [arg2] ...`

[Optional] Lists the arguments to be passed to the Python interpreter.

If `--python` is run without a Python script and trailing arguments, it will start Python shell in the interactive mode. If a Python script and/or arguments are provided with `--python`, it will run the Python script and exit. This way, the tools avoids creation of another Tcl file to `ldbx_python` to run a Python script.

Examples

Example1

To start Python interactive mode from command line:

```
> liberate --python
```

Example2

To run a Python script from command line:

```
> liberate --python script.py
```

Example3

To run a Python script with trailing arguments from command line:

```
> liberate --python script.py example.lib
```

Sample script.py:

```
import ldbx
import sys
lib = ldbx.read_db(sys.argv[1])
print(lib.getHeader())
lib.writeDb("out.lib",False)
```

2

LDBX 2.0 Commands

LDBX 2.0 is based on an object oriented model, which supports both Tcl and Python extensions in separate scripts. The LDBX commands can be categorized as:

- Library-Level Commands
- Object-Oriented Commands

Differences in Tcl and Python Usage

Difference in Syntax

In Tcl, handles, commands, and arguments are space separated. The use model is "`handle/object command argument`".

Example

```
$group getChildren 'cell' 'INVX1'
```

In Python, dot (`.`) is used to call commands of a handle and arguments are put inside of parenthesis. The use model is "`handle.command(argument)`".

Example

```
group.getChildren('cell', 'INVX1')
```

Difference in Calling Library-Level APIs

`read_db`, `get_dbs`, `create_db`, and `del_db` are library-level APIs. Because a lib or ldb group does not have a parent group, they are not called using a group handle.

In Tcl, the use model is "`command <argument>`".

Example

```
read_db example.lib
```

In Python, use model is "`ldbx.command(argument)`".

Example

```
ldbx.read_db("example.lib")
```

Difference in Return type

`get_dbs`, `getChildren`, and, `getTableList` returns a list (tuple) of group handles or table handles. When the returned list contains only one element, the returned handle can be used directly in Tcl. But in Python, list indexing is still required to access the only element in tuple.

For example, to get the only cell group of `INVX1`:

In Tcl, cell is the only group handle:

```
set cellList [$lib getChildren cell INVX1]

$cellList getHeader

$cellList getChildren
```

In Python, cell is a tuple of handles containing one element:

```
cellList = lib.getChildren('cell', 'INVX1')

cellList[0].getHeader()

cellList[0].getChildren()
```

Library-Level Commands

Library-level commands are entrant to LDBX. They do not use handles, instead return handle(s).

Following is the list of supported library-level commands.

- create_db
- del_db

- read_db
- get_dbs

create_db

Creates a new LDB group and returns the handle to the new group.

Syntax

```
create_db libName
```

Options

<libName>	Name of the LDB group.

Example in Tcl

```
set new_lib [create_db test_library]
$new_lib getName
$new_lib addGroup cell INVX1
$new_lib getChildren
```

Example in Python

```
import ldbx
new_lib = ldbx.create_db("test_library")
new_lib.getName()
new_lib.addGroup("cell", "INVX1")
new_lib.getChildren()
```

del_db

Deletes an existing LDB group.

Syntax

```
del_db $ldbGroup
```

Options

<$ldbgroup>	Handle of the LDB group to be deleted.

Example in Tcl

```
import ldbx
new_lib = ldbx.create_db("test_library")
new_lib.getName()
new_lib.addGroup("cell", "INVX1")
new_lib.getChildren()
ldbx.del_db(new_lib)
```

Example in Python

```
set new_lib [create_db test_library]
```

read_db

Reads a `.ldb` or `.lib` file and populates the LDB database.

Syntax

`read_db` *`filename`*

Options

<filename>	Name of the file that is to be read.

Example in Tcl

```
set lib1 [read_db example.ldb]
puts "[$lib1 getHeader] [$lib1 getName]"
```

Example in Python

```
import ldbx
lib1 = ldbx.read_db("example.ldb")
print(lib1.getHeader() + " " + lib1.getName())
```

get_dbs

Returns a list of handles of all LDBs that were read using `read_db`.

Syntax

```
get_dbs
```

Options

None

Example in Tcl

```
read_db example.ldb
read_db example2.ldb
set ldbs [get_dbs]
foreach ldb $ldbs {
    puts "[$ldb getHeader] [$ldb getName]"
}
```

Example in Python

```
import ldbx
ldbx.read_db("example.ldb")
ldbx.read_db("example2.ldb")
ldbs = ldbx.get_dbs()
for ldb in ldbs:
    print(ldb.getHeader() + " " + ldb.getName())
```

Object-Oriented Commands

The new interface of LDBX uses an object-oriented model to access data in the database.

Syntax: Object (or handle) Command Argument(s)

Return value: Object

Example in Tcl:

```
$lib getChildren "cell" "INVX1"
```

Example in Python:

```
lib.getChildren('cell', 'INVX1')
```

Both, Tcl and Python APIs use an object-based model as shown in the above examples. Apart from the syntax of the language, the API usage is almost identical in the two languages.

The commands' syntax and examples shown in this chapter are Tcl based, however, Python usage can be easily derived from the Tcl usage.

The object-oriented commands can be further categorized as the following:

- Group Commands
  - addGroup
  - copyGroup
  - delGroup
  - equal
  - ==
  - getChildren
  - getParent
  - getHeader
  - getName
  - getNameList
  - setHeader
  - setName
  - writeDb

- Attribute Commands
  - addComment
  - delAttr
  - delCAttr
  - delComment
  - getAttr
  - getCAttr
  - getComment
  - setAttr
  - setCAttr
  - setOption
- Table Commands
  - addTable
  - delTable
  - hasTable
  - getTable
  - getTableList
  - isEmpty
  - getIndex
  - getIndices
  - getValue
  - setTable

addGroup

Adds a new subgroup under the current group (`$group`) and returns the handle of the added subgroup.

Syntax

```
$group addGroup grpHeader grpName {AttrNameValueList} [grpOptList]
```

Options

<grpHeader>	Header of the LDB group.
<grpName>	Name of the LDB group.
<AttrNameValueList>	[Optional] List of attributes in the following format: `{attr1 value1 attr2 value2 …}`
[grpOptList]	[Optional] Specifies the order of the new group to be added. Valid values: ■ `order_before` <*simple attribute name*>: Adds the new child group *before* the specified simple attribute. ■ `order_after` <*simple attribute name*>: Adds the new child group *after* the specified simple attribute. ■ `order_top { true | 1 }`: Adds the new child group at the top of the group. ■ `order_bottom { true | 1 }`: Adds the new child group at the bottom of the group. *simple attribute name* is the name of the existing simple attribute in the group.

Examples

Example 1

In Tcl:

```
set t [$pin getChildren timing]
```

```
set newgrp [$t addGroup dummy_rise delay_template {related_pin A timing_sense
positive_unit}]
puts "[$newgrp getHeader]  [$newgrp getAttr ]"
```

In Python:

```
t = pin.getChildren("timing")
newgrp = t[0].addGroup("dummy_rise", "delay_template", (("related_pin", "A"),
("timing_sense", "positive_unit")))
print(newgrp.getHeader() + " " + str(newgrp.getAttr()))
```

Example 2

In Tcl:

```
set t [$pin getChildren timing]
$t getChildren dummy_rise
set newgrp [$t addGroup dummy_rise delay_template  {related_pin A timing_sense
positive_unit}]
$t getChildren dummy_rise
$t getChildren rise_transition
```

In Python:

```
t = pin.getChildren("timing")
t[0].getChildren("dummy_rise")
newgrp = t[0].addGroup("dummy_rise", "delay_template", (("related_pin", "A"),
("timing_sense", "positive_unit")))
t[0].getChildren("dummy_rise")
t[0].getChildren("rise_transition")
```

Example 3

In Tcl:

```
set t [$pin getChildrentiming]
set newgrp [$t addGroup dummy_rise delay_template {related_pin A timing_sense
positive_unit} {order_after  my_attr}
```

This will add a new group after `my_attr` in group `$t`.

In Python:

```
t = pin.getChildren("timing")
newgrp = t[0].addGroup("dummy_rise", "delay_template", (("related_pin", "A"),
("timing_sense", "positive_unit")), (("order_after", "my_attr"),))
```

This will add a new group after `my_attr` in group `$t`.

Example 4

In Tcl:

```
set t [$pin getChildrentiming]
set newgrp [$t addGroup dummy_fall delay_template {} {order_before my_attr}
```

In Python:

```
t = pin.getChildren("timing")
newgrp = t[0].addGroup("dummy_fall", "delay_template", (), (("order_before",
"my_attr"),))
```

In this example `AttrNameValueList` is null but is specified as empty list { } so that `order_after` can be specified in the `grpOptList` option.

copyGroup

Hierarchically copies the source group (`$source`) and add it in the current group (`$group`). A handle to the newly copied subgroup is returned.

Synax

```
$group copyGroup $source [grpOptList]
```

Options

<group_handle>	Handle of the source group that is to be copied into the current group.

[grpOptList] [Optional] Specifies the order of the group to be copied. Valid values:

- `order_before <simple attribute name>`: Copies the new child group *before* the specified simple attribute.
- `order_after <simple attribute name>`: Copies the new child group *after* the specified simple attribute.
- `order_top { true | 1 }`: Copies the new child group at the top of the group.
- `order_bottom { true | 1 }`: Copies the new child group at the bottom of the group.

simple attribute name is the name of the existing simple attribute in the group.

Examples

Example 1

In Tcl:

```
set t [$pin getChildren timing]
set newgrp \
[$t addGroup dummy_rise delay_template {related_pin A timing_sense positive_unit}]
set copied [$t copyGroup $newgrp]
```

In Python:

```
t = pin.getChildren("timing")
newgrp = t[0].addGroup("dummy_rise", "delay_template", (("related_pin", "A"),
("timing_sense",  "positive_unit")))
copied = t[0].copyGroup(newgrp)
```

This example will copy `$newgrp` after `my_attr` in group `$t1`.

Example 2

In Tcl:

```
set t [$pin getChildren timing]
set rise [$t getChildren rise_transition]
set copied [$t copyGroup $rise]
$t getChildren rise_transition
```

In Python:

```
t = pin.getChildren("timing")
rise = t[0].getChildren("rise_transition")
copied = t[0].copyGroup(rise[0])
t[0].getChildren("rise_transition")
```

Example 3

In Tcl:

```
set t1 [$pin getChildren timing]
set t2 [$cell getChildren ff "IQ, IQN"]
set copied_grp [$t1 copyGroup $t2]
```

In Python:

```
t1 = pin.getChildren("timing")
t2 = cell.getChildren("ff", "IQ,IQN")
copied_grp = t1[0].copyGroup(t2[0])
```

In the above example the `grpOptList` argument is not specified, so the group `$t2` will be copied at the end in group `$t1`.

delGroup

Deletes one or more groups in the specified list (`$GroupHandleList`) in the current group (`$group`).

Syntax

```
$group_handle delGroup <GroupHandleList>
```

Options

<GroupHandleList> Handle to the list of groups to be deleted.

Example in Tcl

```
set t [$pin getChildren timing]
set rise [$t getChildren cell_rise]
$t delGroup $rise
$t getChildren cell_rise
```

Example in Python

```
t = pin.getChildren("timing")
rise = t[0].getChildren("cell_rise")
t[0].delGroup(rise)
t[0].getChildren("cell_rise")
```

equal

Compares the specified group handles and returns `true` if they are same.

Syntax

```
$group_handle equal <group_handle_to_compare>
```

Options

`<group_handle_to_compare>`

Handle of the group that is to be compared with the `group_hand`

Example

```
set cell1 [$lib getChildren cell INVX1]
set cell2 [$lib getChildren cell INVX1]
puts [$cell1 equal $cell2]
```

==

Python operator command to compare the specified group handles. The command returns `true` if the compared groups are the same.

Syntax

`group_handle == group_handle_to_compare`

Where, *`group_handle`* and *`group_handle_to_compare`* are the group IDs to be compared.

Example

```
cell1 = lib.getChildren("cell")[0]
cell2 = lib.getChildren("cell")[0]
print(cell1 == cell2)
```

getChildren

Returns a list of handles of all or part of the subgroups in the current group depending on the provided arguments.

Note: The search pattern will be based the option specified in the setOption command.

Syntax

```
$handle getChildren <grpHeader> <grpName> <AttrKVList>
```

Options

<grpHeader>	[Optional] Name of the group header.
<grpName>	[Optional] Name of the subgroup.
<AttrKVList>	[Optional] List of attribute Key-Values (KV).

Example in Tcl

```
set lib [read_db example.ldb]
set cells [$lib getChildren "cell" "INVX1" {area 0}]
foreach cell $cells {
puts "[$cell getHeader] [$cell getName]" }
```

Example in Python

```
lib = ldbx.read_db("example.ldb")
cells = lib.getChildren("cell", "INVX1", (("area", "0"),))
for cell in cells:
    print(cell.getHeader() + " " + cell.getName())
```

Legal command examples

```
$lib getChildren            ; # returns all subgroups under $lib group
$lib getChildren "*"        ; # returns all subgroups under $lib group
$lib getChildren "cell"     ; # returns all "cell" subgroups under $lib group
$lib getChildren "cell" "*" ; #returns all "cell" subgroups under $lib group
```

```
# returns "cell(INVX1)" subgroup(s) under the $lib group
$lib getChildren "cell" "INVX1" ;
# returns "cell(INVX1)" subgroups under $lib group with an attribute of "area : 0"
$lib getChildren "cell" "INVX1" {area 0} ;
# returns "cell(INVX1)" subgroups under $lib group with an attribute of "area",
regardless of the value of this attribute
$lib getChildren "cell" "INVX1" {area "*"} ;
#returns all "cell" subgroups under $lib group with an attribute of "area : 0"
$lib getChildren "cell" "*" {area 0} ;
# returns all subgroups under $lib group with an attribute of "area : 0"
$lib getChildren "*" "*" {area 0} ;
# returns all subgroups under $lib group that has empty group name
$lib getChildren "*" ""      ;
```

Notes

- The `grpheader` cannot be an empty string. Use "*" or omit this argument. For example, the following statement is illegal:

  ```
  $handle getChildren "" ;
  ```

- The `grpValue` can be an empty string when the group name is empty in the LDB or `.lib`.
- All the three optional arguments are order dependent, which means if one argument is omitted, all its following arguments should also be omitted. Use "*" as a placeholder for `grpHeader` or `grpName` if these fields are not required.
- `AttrKVList` must contain an even number of elements representing an `attrName` and associated `attrValue` pair. `attrName` cannot be an empty string. If not needed, `attrValue` can use "*" as a placeholder. For example, the following statement is illegal:

  ```
  $handle getChildren "cell" "*" {"" 0}
  ```

getParent

Returns the name of the parent group.

Syntax

```
$handle getParent
```

Options

None

Example in Tcl

```
set lib [read_db example.ldb]
set cell [$lib getChildren "cell" "INVX1"]
set parent [$cell getParent]
puts "[$parent getHeader] [$parent getName]"
```

Example in Python

```
lib = ldbx.read_db("example.ldb")
cell = lib.getChildren("cell", "INVX1")
parent = cell[0].getParent()
print(parent.getHeader() + " " + parent.getName())
```

getHeader

Returns the header of current group.

Syntax

```
$tableHandle getHeader
```

Options

None

Example in Tcl

```
set lib [read_db example.ldb]
puts "[$parent getHeader] [$parent getName]"
```

Example in Python

```
lib = ldbx.read_db("example.ldb")
print(parent.getHeader() + " " + parent.getName())
```

getName

Returns the name of the current group.

Syntax

```
$handle getName
```

Options

None

Example in Tcl

```
set lib [read_db example.ldb]
puts "[$parent getHeader] [$parent getName]"
```

Example in Python

```
lib = ldbx.read_db("example.ldb")
print(parent.getHeader() + " " + parent.getName())
```

getNameList

Returns the list of names in the current group.

Syntax

```
$handle getNameList
```

Options

None

Example in Tcl

```
set state_group [$group getChildren "statetable"]?
        foreach name [$state_group getNameList] {?
          puts $name?
        }?
```

Example in Python

```
state_group = group.getChildren("statetable")
for name in state_group[0].getNameList():
    print(name)
```

setHeader

Sets the specified header for the current group.

Syntax

```
$group setHeader <newHeader>
```

Options

newHeader	New header to be specified for the current group.

Example in Tcl

```
set cell [$lib getChildren DFFQX1]
$cell setHeader cellNew
puts "After setting, group is [$cell getHeader] [$cell getName]"
```

Example in Python

```
cell = lib.getChildren("cell","DFFQX1")[0]
cell.setHeader("cellNew")
print("After setting, group is " + cell.getHeader() + " " + cell.getName())
```

setName

Sets the specified name for the current group.

Syntax

```
$group setName <newName>
```

Options

newName	New name to be specified for the current group.

Example in Tcl

```
set cell [$lib getChildren DFFQX1]
$cell setName DFFQX2
```

```
puts "After setting, group is [$cell getHeader] [$cell getName]"
```

Example in Python

```
cell = lib.getChildren("cell","DFFQX1")[0]
cell.setName("DFFQX2")
print("After setting, group is " + cell.getHeader() + " " + cell.getName())
```

writeDb

Writes the specified group into a file in the Liberty format. Only a library group handle is in this command.

When using LDBX 2.0, formatting of the complex attribute values is based on the corresponding table template. Therefore, lu_table_template must be defined when creating a group, so that the table is written out in the Liberate standard multi-line format instead of a single line.

Syntax

```
$lib writeDb filename <gzip> <overwrite>
```

Options

filename	Name of the file that is to be updated.
<gzip>	[Optional] Compresses the output `.lib` file when set to `True`. Valid values are `True` (default) and `False`.
<overwrite>	[Optional] Overwrites the existing file with the same name when set to `True`. Valid values are `True` and `False` (default).

Example in Tcl

```
$lib writeDb out.lib False True
```

Example in Python

```
lib.writeDb("out.lib", False, True)
```

addComment

Adds a new comment in the current group. This command returns a *commentId* that is assigned to this new comment, if it is added successfully.

Syntax

```
$group addComment commentContent <commentPropsList>
```

Options

commentContent — The content for the comment to be added.

<commentPropsList> — [Optional] Location where the comment should be added. If *commentPropList* is not provided, the new comment will be added at top of current group.

- order_before <*simple attribute name*>: Forces the new comment to be before the specified simple attribute.
- order_after <*simple attribute name*>: Forces the new comment to be after the specified simple attribute.
- order_top { true | 1 }: Forces the new comment to be at the top of the group.
- order_bottom { true | 1 }: Forces the new comment to be at the bottom of the group.

simple attribute name is the name of the existing simple attribute in the group.

Examples

Consider that input .lib file originally has one comment under the cell group:

```
cell (INVX1) {
  area : 0.324;
  cell_leakage_power : 0.1833;
 /* original comment */
}
```

Example 1: To add a new comment, comment2, after line "cell (INVX1) {" in output file (default location)

In Tcl:

```
liberate> $cell addComment "a new comment"
liberate> $cell getComment
comment1 { original comment } comment2 { a new comment }
```

In Python:

```
ldbx_python> cell.addComment("a new comment")
ldbx_python> cell.getComment()
(('comment1', 'original comment'),('comment2', 'a new comment'))
```

Example 2: To add a new comment, comment3, after line "area : 0.324" in the output file

In Tcl:

```
liberate> set commentId [$cell addComment "another new comment" {order_after area}]
liberate> $cell getComment $commentId
comment3 {another new comment}
```

In Python:

```
ldbx_python> commentId = cell.addComment("another new comment", [("order_after",
"area")])
ldbx_python> cell.getComment(commentId)
(('comment3', 'another new comment'),)
```

delAttr

Deletes a list of simple attributes, if they exist in current group. This command deletes all simple attributes in the current group if *AttrNameList* is not provided.

Syntax

```
$group delAttr <AttrNameList>
```

Options

<AttrNameValueList>

[Optional] List of simple attributes in the format:

```
{attr1 value1 attr2 value2 …}
```

Example in Tcl

```
set t [$pin getChildren timing]
$t setAttr {dummy_attr 1.05 related_pin B}
$t delAttr {dummy_attr timing_type}
```

Example in Python

```
t = pin.getChildren("timing")
t[0].setAttr((("dummy_attr", "1.05"), ("related_pin", "B")))
t[0].delAttr(("dummy_attr", "timing_type"))
```

delCAttr

Deletes all complex attributes in a group. It deletes a list of complex attributes that are named `CAttrName` or that match `CAttrName` and `CAttrValue`.

Syntax

```
$group delCAttr <CAttrName> <CAttrValue>
```

Options

<CAttrName>	[Optional] Name of the complex attribute to be deleted.
<CAttrValue>	[Optional] Value of the specified complex attribute.

Example in Tcl

```
set t [$pin getChildren timing]
$t setCAttr  unit {cap 1 ff}
$t setCAttr unit {time 1 ps}
$t delCAttr unit time
$t getCAttr
=> unit {cap 1 ff}
```

Example in Python

```
t = pin.getChildren("timing")
t[0].setCAttr("unit", ("cap", "1", "ff"))
t[0].setCAttr("unit", ("time", "1", "ps"))
t[0].delCAttr("unit", "time")
```

```
t[0].getCAttr()
=> (('unit', ('cap', '1', 'ff')),)
```

delComment

Deletes the comment associated with the provided *commentId* in the current group. If the provided *commentId* does not exist in the current group, no comment is deleted.

Syntax

```
$group delComment commentId
```

Options

commentID	*commentID* of the comment to be deleted.

Examples

Consider that input `.lib` file has two comments under the `cell` group:

```
cell (INVX1) {
  area : 0.324;
  /* this is a comment */
  cell_leakage_power : 0.1833;
 /* this another comment */
}
```

Example 1

In Tcl:

```
liberate> $cell getComment
comment1 { this is a comment } comment2 { this is another comment }
liberate> $cell delComment comment1
liberate> $cell getComment
comment2 { this is another comment }
```

In Python:

```
ldbx_python> cell.getComment()
(('comment1', 'this is a comment'),('comment2', 'this is another comment'))
ldbx_python> cell.delComment("comment1")
```

```
ldbx_python> cell.getComment()
(('comment2', 'this is another comment'),)
```

Example 2

In Tcl:

```
liberate> $cell getComment
comment2 { this is another comment }
liberate> $cell delComment comment3
liberate> $cell getComment
liberate> comment2 { this is another comment }
```

In Python:

```
ldbx_python> cell.getComment()
(('comment2', 'this is another comment'),)
ldbx_python> cell.delComment("comment3)
ldbx_python> cell.getComment()
(('comment2', 'this is another comment'),)
```

getAttr

Returns the KV list of all or part of the simple attributes in the current group. The search is dependent on whether a list of attribute names is provided.

Note: The search pattern will be based the option specified in the setOption command.

Syntax

```
$group getAttr <AttrNameList>
```

Options

<AttrNameList>	[Optional] List of the attributes. **Note:** This option cannot be an empty string.

In Tcl, the return type of `getAttr` and argument type of `setAttr` is a flat list of attribute name and values.

For example: `{current_unit 1mA time_unit 1ns}`

In Python, the return type of `getAttr` and argument type of `setAttr` is tuple pairs of attribute name and attribute value.

For example: `(('current_unit','1mA'), ('time_unit','1ns'))`

Example in Tcl

```
set lib [read_db example.ldb]
puts [$lib getAttr]
puts [$lib getAttr {time_unit altos_compile_version}]
```

Example in Python

```
lib = ldbx.read_db("example.ldb")
print(lib.getAttr())
print(lib.getAttr(("time_unit", "altos_compile_version")))
```

getCAttr

Returns the KV list of all or part of the complex attributes in the current group. The search is dependent on whether a complex attribute names is provided.

Note: The search pattern will be based the option specified in the setOption command.

Syntax

```
$group getCAttr <CAttrName>
```

Options

<CAttrName>	[Optional] Name of the complex attribute. **Note:** This option cannot be an empty string.

In Tcl, the return type of `getCAttr` is a flat list of attribute name and values.

For example: `{current_unit {1mA} time_unit {1ns}}`

In Python, the return type of `getCAttr` is tuple pairs of attribute name and attribute value.

For example: `(('current_unit','1mA'), ('time_unit','1ns'))`

Example in Tcl

```
set lib [read_db example.ldb]
puts [$lib getCAttr]
puts [$lib getCAttr altos_set_var]
```

Example in Python

```
lib = ldbx.read_db("example.ldb")
print(lib.getCAttr())
print(lib.getCAttr(("altos_set_var")))
```

getComment

Queries and returns comments in the current group. If a *commentID* is not provided, this command will return all the comments in the current group (`$group`) as list of *commentId commentContent* pairs.

Note: Returned list of comments may not necessarily be in the same order as they are located inside the group. They are ordered by the *commentId* string.

Syntax

`$`*group* `getComment <`*commentId*`>`

Options

commentID	[Optional] *commentID* of the comment to be queried. If the provided *commentID* exists, `getcomment` will return the *commentId commentContent* pair.

Examples

Consider that input `.lib` file has two comments under the `cell` group:

```
cell (INVX1) {
  area : 0.324;
  /* this is a comment */
  cell_leakage_power : 0.1833;
 /* this another comment */
}
```

Example 1

In Tcl:

```
liberate> $cell getComment
comment1 { this is a comment } comment2 { this is another comment }
```

In Python:

```
ldbx_python> cell.getComment()
(('comment1', 'this is a comment'),('comment2', 'this is another comment'))
```

Example 2

In Tcl:

```
liberate> $cell getComment comment2
comment2 { this is another comment }
```

In Python:

```
ldbx_python> cell.getComment("comment2")
(('comment2', 'this is another comment'))
```

setAttr

Adds a list of new simple attributes or updates existing simple attributes in the current group.

Syntax

$*tableHandle* setAttr *AttrNameValueList* [*AttrPropsList*]

Options

<AttrNameValueList>

List of KVs in the following format:

{attr1 value1 attr2 value2…}.

Note: This option cannot be an empty string.

[AttrPropsList] [Optional] Specifies change in order of the attribute(s). Valid values:

- `order_before <simple attribute name>`: Forces the new order to be *before* the specified simple attribute.
- `order_after <simple attribute name>`: Forces the new order to be *after* the specified simple attribute.
- `order_top { true | 1 }`: Forces the new order to be at the top of the group.
- `order_bottom { true | 1 }`: Forces the new order to be at the bottom of the group.

simple attribute name is the name of the existing simple attribute in the group.

Example 1

In Tcl:

```
set t [$pin getChildren timing]
$t setAttr  { dummy_attr 1.05 related_pin B }
$t getAttr
=> dummy_attr 1.05 related_pin B timing_sense negative_unate
```

In Python:

```
t = pin.getChildren("timing")
t[0].setAttr((("dummy_attr", "1.05"), ("related_pin", "B")))
t[0].getAttr()
=> (('dummy_attr', '1.05'), ('related_pin', 'B'), ('timing_sense', '
negative_unate'))
```

Example 2

In Tcl:

```
set t [$pin getChildrentiming]
$t setAttr  {dummy_attr 1.05 related_pin B}  {order_before  my_attr}
```

In Python:

```
t = pin.getChildren("timing")
t[0].setAttr((("dummy_attr", "1.05"), ("related_pin", "B")), (("order_before",
"my_attr"),))
```

If attributes `dummy_attr` and `related_pin` specified in the first argument already exists, this command will edit them to update the specified value as well as change their order to place them before the simple attribute `my_attr` in the same group. Therefore, `[AttrOptList]` can also be used to change the order of an existing simple attribute.

However, if attributes `dummy_attr` and `related_pin` do not already exist, they will be added and placed before simple attribute `my_attr` in the same group.

setCAttr

Adds a new complex attribute in the current group. If an attribute specified in <*CAttrValueList*> already exists in the group, `setCAttr` will add an additional copy of that attribute. To update the value of an existing attribute in the complex attribute group, you must first delete that attribute and then add a new attribute.

Syntax

```
$tableHandle setCAttr CAttrName CAttrValueList [CAttrOptList]
```

Options

<CAttrName>	Name of the complex attribute.
<CAttrValueList>	List of values of the complex attribute.
[CAttrOptList]	[Optional] Specifies change in order of the new complex attribute to be added. Valid values: ■ `order_before` <*simple attribute name*>: Adds the new complex attribute *before* the specified simple attribute. ■ `order_after` <*simple attribute name*>: Adds the new complex attribute *after* the specified simple attribute. ■ `order_top { true | 1 }`: Adds the new complex attribute at the top of the group. ■ `order_bottom { true | 1 }`: Adds the new complex attribute at the bottom of the group. *simple attribute name* is the name of the existing simple attribute in the group.

Example 1

In Tcl:

```
set t [$pin getChildren timing]
$t setCAttr  unit {cap 1 ff}
$t setCAttr unit {time 1 ps}
$t getCAttr
=> unit {cap 1 ff} unit {time 1 ps}
```

In Python:

```
t = pin.getChildren("timing")
t[0].setCAttr("unit", ("cap", "1", "ff"))
t[0].setCAttr("unit", ("time", "1", "ps"))
t[0].getCAttr()
=> (('unit', ('cap', '1', 'ff')), ('unit', ('time', '1', 'ps')))
```

Example 2

In Tcl:

```
set t [$pin getChildrentiming]
$t setCAttr unit {cap 1 ff} {order_before  my_attr}
```

In Python:

```
t = pin.getChildren("timing")
t[0].setCAttr("unit", ("cap", "1", "ff"), (("order_before", "my_attr"),))
```

This example will add a new complex attribute *before* `my_attr` in group `$t`.

Example 3

In Tcl:

```
$pin delCAttr fall_capacitance_range
$pin setCAttr fall_capacitance_range {0.1 0.2}
```

In Python:

```
pin[0].delCAttr("fall_capacitance_range")
pin[0].setCAttr("fall_capacitance_range", ("0.1", "0.2"))
```

This will replace `pin`'s complex attribute `fall_capacitance_range`.

setOption

Specifies the pattern that should be used by the `getChildren`, `getAttr`, and `getCAttr` commands for searching the required return values.

Syntax

$lib setOption {*searchPattern*}

Options

<searchPattern>

Type of search pattern. Valid values are `search glob`, `search regex`, and `search exact`. Default value: `search exact`.

Example 1: Glob pattern search

In Tcl:

```
$lib  setOption {search glob} #Sets glob matching


$cell getChildren ff "IQ,IQN" #Returns ff group with name IQ,IQN


$cell getChildren ff "I\[A-Z\],IQN" #Returns ff group with name staring with I
followed by any letter from A to Z, followed by ",IQN"


$cell getChildren ff "*" #Returns everything which has group_iden == ff


set cell2 [$lib getChildren cell DFFQX1 {"altos_xtr_count" "2*"}] #Matches all the
cell groups having DFFQX1 as group_name, attribute  name -"altos_xtr_count" and
value - anything starting with 2.
set ff_group1 [$cell getChildren ff "IQ,IQN" {"c*" "C*"}] #Matches with all the ff
group with IQ,IQN as the group name and attribute name starting with c (anything
after c) and attribute value starting with C (anything after C)
```

In Python:

```
lib.setOption([("search","glob")]) #sets glob matching


ff_group1 = cell.getChildren("ff","IQ,IQN",[("c*","C*")])[0] #Matches with ff
group with "IQ, IQN" as the group name and attribute name starting with c (anything
after c) and attribute value starting with C (anything after C)
```

```
ff_group1.getAttr([("c[a-z]*")]) #Returns simple attribute of group ff starting
with c and anything after that from range[a-z]


cell3 = lib.getChildren("cell","DFFQX1",[("altos_extsim_cpu_time","4*\.*")])[0]
#Matches cell group with group name as DFFQX1 and having attribute
"altos_extsim_cpu_time" with value starting with 4 (anything after this) followed
by "." (anything after this).
```

Example 2: Regex pattern match

In Tcl:

```
$lib  setOption {search regex} #Sets regex matching


$ff_group getAttr clocked_on #Returns simple attribute 'clocked on' of ff group


$ $ff_group getAttr c\[a-z\]+_\[a-z\]+] #Returns return all attributes starting
with "c" followed by any number of letters in range [a-z] then "_" followed by any
letter in range[a-z].

$ff getAttr ".*" #Returns all simple attributes
```

In Python:

```
lib.setOption([("search","regex")]) #Sets regex matching


ff_group1 = cell.getChildren("ff","IQ,IQN",[("c.*","C.*")])[0] #Matches with ff
group with group name as "IQ, IQN" and attribute name starting with c (anything
after this) and attribute value starting with C (anything after this)


ff_group1.getAttr([("c[a-z].*")]) #Returns simple attribute of group ff starting
with c and anything after that from range[a-z]


cell3 = lib.getChildren("cell","DFFQX1",[("altos_extsim_cpu_time","4.*\..*")])[0]
#Matches with cell group with group name as DFFQX1 that has attribute
"altos_extsim_cpu_time" with value starting with 4 (anything after this) followed
by "." (anything after this).
```

Example 3: Exact pattern match

In Tcl:

```
$lib  setOption {search exact} #Sets exact matching


$lib getCAttr "altos_receiver_fall_cap_threshold" #Returns complex attribute
altos_receiver_fall_cap_threshold


$ff_group getCAttr "altos_receiver_" #Will give an error if user did not pass the
exact full name
```

In Python:

```
lib.setOption([("search","exact")]) #Sets exact matching


ff_group1 = cell.getChildren("ff","IQ,IQN",[("clocked_on","Ck")])[0] #Matches with
ff group with group name as "IQ,IQN" and attribute name "clocked_on" and attribute
value "CK"


ff_group1.getAttr([("clocked_on")]) #Matches with ff group having simple attribute
"clocked_on"


ff_group1.getAttr([("c[a-z].*")]) #Will give error


cell3 = lib.getChildren("cell","DFFQX1",[("altos_extsim_cpu_time","47.1988")])[0]
#Matches with cell group with group name as DFFQX1 and having attribute
"altos_extsim_cpu_time" with value 47.1988
```

addTable

Adds a new data table under the current group (`$group`) and returns the handle of the added table.

Syntax

```
$group addTable {{Index1_list} {Index_2_list}} {list_of_values}
     "value_name_str"
```

Options

<indices_list>	List of all the indices.
<value_list>	List of all the values.
<value_name_str>	Names of the values

Example in Tcl

```
set new_table [$group addTable {{0.1 0.2 0.3} {1 2 3}} {1 2 3 4 5 6 7 8 9} "values"]
puts "New Table:"
puts "Indices: [$new_table getIndices]"
puts "Values:  [$new_table getValue]"
```

Example in Python

```
new_table = group.addTable(((0.1, 0.2, 0.3), (1, 2, 3)), (1, 2, 3, 4, 5, 6, 7, 8, 9), "values")
print("New Table:")
print("Indices: ", new_table.getIndices())
print("Values:  ", new_table.getValue())
```

delTable

Deletes a table from the current group (*$group*) or table (*$table*).

Syntax

```
$group delTable
```

or

```
$table delTable
```

Options

None

Example in Tcl

```
set table [$r getTable]
$table getIndices
$table getValue
$table delTable
$table getIndices
```

Example in Python

```
table = r.getTable()
table.getIndices()
table.getValue()
table.delTable()
table.getIndices()
```

hasTable

Returns if group directly contains any table objects.

Syntax

```
$groupHandle hasTable
```

Options

None

Example in Tcl

```
set t [$pin getChildren timing]
set r [$t getChildren cell_rise]
if {[$r hasTable]==1} {
  set table [$r getTable]
  puts "Indices: [$table getIndices]"
  puts "Value:   [$table getValue]"
}
```

Example in Python

```
t = pin.getChildren("timing")
r = t[0].getChildren("cell_rise")
if r[0].hasTable():
    table = r[0].getTable()
    print("Indices: ", table.getIndices())
    print("Value:   ", table.getValue())
```

getTable

Returns the table object created from the `values` table. If the group has no table or the specified table name is not found, an empty table object is returned.

Syntax

```
$groupHandle getTable
```

Options

None

Example

```
A_group () {
 index_1 (…)
 index_2 (…)
 values (…..\
               ….\
 )
 values_energy (…\
                              … \
 )
```

In the above example, the tables will contain `index_1`, `index_2`, and values from `values`. `value_energy` will be ignored.

Python users with older Python script and not wanting to update the script can revert to the old behavior of `getTable` to return a list of tables by adding the following in python script after importing ldbx:

```
ldbx.option["table_as_list"] = 1
import ldbx_back_compatible
```

When the script updated, remove the above two lines from the script to adopt the new behavior.

getTableList

Returns a list of handles to the table object directly under the current group.

Syntax

`$`*`groupHandle`* `getTableList`

Options

None

Example in Tcl

```
set t [$pin getChildren timing]
set r [$t getChildren cell_rise]
set tables [$r getTableList]
foreach table $tables {
$table getIndices
$table getValue }
```

Example in Python

```
t = pin.getChildren("timing")
r = t[0].getChildren("cell_rise")
tables = r[0].getTableList()
for table in tables:
    print("Indices: ", table.getIndices())
    print("Value:   ", table.getValue())
```

isEmpty

Checks if the specified table object is empty or not.

Syntax

$*tableHandle* isEmpty

Options

None

getIndex

Returns the input number index of the table if the table is not empty.

Syntax

$*table* getIndex <*index_number*>

Options

Option	Description
<integer number>	The specified integer should be greater than `0` but less than `max_index_number`.

Example in Tcl

```
set t [$pin getChildren timing]
set r [$t getChildren cell_rise]
set table [$r getTable]
$table getIndex 2
```

Example in Python

```
t = pin.getChildren("timing")
r = t[0].getChildren("cell_rise")
table = r[0].getTable()
table.getIndex(2)
```

getIndices

Returns the indexes of the table if the table is not empty.

Syntax

```
$tableHandle getIndices
```

Options

None

Example in Tcl

```
set t [$pin getChildren timing]
set r [$t getChildren cell_rise]
set table [$r getTable]
set indices [$table getIndices]
set values [$table getValue]
```

Example in Python

```
t = pin.getChildren("timing")
r = t[0].getChildren("cell_rise")
table = r[0].getTable()
indices = table.getIndices()
values = table.getValue()
```

getValue

Returns the values of the table if the table is not empty.

Syntax

`$tableHandle getValue`

Options

None

Example in Tcl

```
set t [$pin getChildren timing]
set r [$t getChildren cell_rise]
set table [$r getTable]
set indices [$table getIndices]
set values [$table getValue]
```

Example in Python

```
t = pin.getChildren("timing")
r = t[0].getChildren("cell_rise")
table = r[0].getTable()
indices = table.getIndices()
values = table.getValue()
```

setTable

Updates the indices and/or values in an existing table under the current table (`$table`) and returns the handle of the updated table.

Syntax

$table setTable {list of list of indices} {list_of_values}

Options

<indices_list>	List of all the indices list.
<value_list>	List of all the values.

Example in Tcl

```
$table setTable $indices $new_val
puts "After adding margin:"
puts "  Indices: [$table getIndices]"
puts "  Values:  [$table getValue]"
```

Example in Python

```
table.setTable(indices, new_val)
print("After adding margin:")
print("Indices: ", table.getIndices())
print("Value:   ", table.getValue())
```

3

Data Structure Manipulation Scripts

This appendix covers the examples of the Tcl and Python scripts for LDBX:

- Tcl Interface Example Scripts
- Python Interface Examples

Tcl Interface Example Scripts

This section provides example scripts for Tcl interface:

- Adding a Cell Group
- Adding Margin to Timing/internal power Arc
- Checking Hold Values
- Compare Groups Recursively
- Deleting a Table and Adding a New One
- Deleting a Group Recursively
- Printing All Groups
- Printing All Groups Recursively
- Printing Timing Group Attributes
- Updating Group Attributes

Example Tcl Utilities File

The following is an example Tcl Utilities file, `utils.tcl`:

```
# Tcl untilities for LDBX 2.0

# print group header and name
```

```
proc pgrp {grp {level 0}} {
    return "[$grp getHeader] [$grp getName]"
}

# print group header and name with indent level
proc print_grp {grp level {cnt ""}} {
    set indent [expr ($level * 2)]
    puts -nonewline [format "%${indent}s" " "]
    puts "[$grp getHeader]${cnt}([$grp getName])"
}

# print attributes with indent level
proc print_attr {k v {level 0}} {
    set indent [expr ($level * 2)]
    puts -nonewline [format "%${indent}s" " "]
    puts "  attr: $k $v"
}

# print all attributes with indent level
proc print_attrs {kvlist {level 0}} {
    foreach {k v} $kvlist {
        set indent [expr ($level * 2)]
        puts -nonewline [format "%${indent}s" " "]
        puts "  attr: $k $v"
    }
}

# print arbitrary message with indent level
proc pretty_print {msg {level 0}} {
    set indent [expr ($level * 2)]
    puts -nonewline [format "%${indent}s" " "]
    puts "$msg"
}

# convert "values" or "index_*" attributes to Tcl list form without commas
proc str_to_list { inlist } {
    set outlist {}
    foreach v $inlist {
        regsub "," $v "" m
        lappend outlist $m
    }
```

```
    return $outlist
}

# get the hierachical path to the handle (recursive function)
proc get_hier { handle {res ""} } {
    set grp "[$handle getHeader]([$handle getName])"
    if { $res != "" } {
        set res "$grp-->$res"
    } else {
        set res "$grp"
    }
    if { [$handle getHeader] == "library" } {
        return $res
    }
    get_hier [$handle getParent] $res
}
```

You can perform the following functionalities on such data:

- Copy data from one library file to another
- Copy from one cell, pin, or group to another
- Copy to multiple destinations
- Modify data before or after the copy

Adding a Cell Group

You can add a new cell if it does not exist already as shown in the example code below.

```
#add_group.tcl

proc add_cell { lib cell attr} {
  set old_cell [$lib getChildren cell $cell]
  if {[llength $old_cell] == 0} {
    $lib addGroup "cell" $cell $attr
  } else {
    $lib delGroup $old_cell
    $lib addGroup "cell" $cell $attr
  }
}
```

```
set lib [read_db ./libs/example.ldb]
set attr {altos_pinlist "A Y" altos_vtn 0.35 altos_vtp 0.35 area 0  altos_xtr_count
4}
set buf_cell [add_cell $lib BUF1X $attr]
puts "[$buf_cell getHeader]
```

Adding Margin to Timing/internal_power Arc

You can add margin to the values in group as shown in the example code below.

```
#add_margin.tcl

proc add_margin { group margin } {
  set tables [$group getTable]
  if {[llength $tables] == 0} {
    puts "Group [$group getHeader] [$group getName] does't have table"
    return
  }
  foreach table $tables {
    set indices [$table getIndices]
    set values  [$table getValue]
    puts "Before adding margin:"
    puts "  Indices: $indices"
    puts "  Values:  $values"
    set new_val {}
    foreach v $values {
      lappend new_val [expr $v+$margin]
    }
    $table setTable $indices $new_val
    puts "After adding margin:"
    puts "  Indices: [$table getIndices]"
    puts "  Values:  [$table getValue]"
  }
}


source find_timing_power_arc.tcl
set lib [read_db ./libs/example.ldb]
set group [find_arc $lib AND2X1 Y {related_pin B} cell_fall delay_template]
add_margin $group 0.1
```

Checking Hold Values

```
#check_hold_values.tcl
source utils.tcl

#
# checks all hold_rising rise_constraint arcs
#
proc check_hold_values { lib } {
    foreach cell [$lib getChildren cell] {
        foreach pin [$cell getChildren pin] {
            #            foreach tmg [$pin getChildren timing * {timing_type
hold_rising}] {}
            foreach tmg [$pin getChildren timing] {
                foreach holdarc [concat [$tmg getChildren rise_constraint] [$tmg
getChildren fall_constraint]] {
                    puts "FOUND: [get_hier $holdarc]"
                    print_grp $holdarc 2
                    foreach {k v} [$holdarc getAttr] {
                        print_attr $k $v 3
                    }
                    foreach {k v} [$holdarc getCAttr] {
                        # ... insert checks here ...
                        print_attr $k [str_to_list $v] 4
                    }
                }
            }
        }
    }
}
```

Compare Groups Recursively

```
source utils.tcl

#
# compares simple attrbibutes between 2 groups
#
proc compare_attrs {grp1 grp2 {level 0}} {
    set attrs1 [$grp1 getAttr]
    array set attrs2 [$grp2 getAttr]
    foreach {k1 v1} $attrs1 {
```

```
        # check if attr exists
        if { ! [info exists attrs2($k1)] } {
            puts "[get_hier $grp1] : attr key mismatch : $k1 missing from lib2"
            continue
        }
        # check if attr val is same
        set v2 $attrs2($k1)
        if { $v1 != $v2 } {
           puts "[get_hier $grp1] : attr value mismatch : $k1 value mismatch: $v1
!= $v2"
            continue
        }
        #puts "matched $k1 $v1" $level
    }
}

#
# compares 2 libraries (or any 2 groups) hierarchicaly for groups and simple
attributes.
# Reports groups not present in lib2, and attributes that dont exist in lib2
# or dont have matching values
#
proc compare_groups_recursive {grp1 grp2 {level 0}} {
    # compare attrs
    compare_attrs $grp1 $grp2 $level

    incr level
    set children [$grp1 getChildren]
    if {[llength $children] == 0} {
        return
    }

    foreach subgrp $children {
        # make most groups unique
       set chkAttrs {related_pin timing_sense timing_type when} ; # attributes to
make groups unique
        set attrList [$subgrp getAttr $chkAttrs]
        set g2child [$grp2 getChildren [$subgrp getHeader] [$subgrp getName]
$attrList]
        if { [llength $g2child] > 1 } {
            # puts "script: group compare not unique. Add more attrs to uniquify
header/name of groups : [$subgrp getHeader]"
```

```
            continue
        }
        if { ! [llength $g2child] } {
            puts "[get_hier $subgrp] : group mismatch: missing group '[$subgrp
getHeader]([$subgrp getName])' in lib2"
            continue
        }
        # recurse IN (call cmp_grp func again) into next deeper level group
        compare_groups_recursive $subgrp $g2child $level
    }
}

################################################
# Run test
set lib1 [read_db ./libs/lib1.ldb]
set lib2 [read_db ./libs/lib2.ldb]
compare_groups_recursive $lib1 $lib2
#compare_groups_recursive $lib2 $lib1
################################################
```

Deleting a Table and Adding a New One

```
## add margin to the values in group ##
#####################################
proc add_margin { group margin } {
  set tables [$group getTable]
  if {[llength $tables] == 0} {
    puts "Group [$group getHeader] [$group getName] does't have table"
    return
  }
  foreach table $tables {
    set indices [$table getIndices]
    set values  [$table getValue]
    puts "Before adding margin:"
    puts "  Indices: $indices"
    puts "  Values:  $values"
    set new_val {}
    foreach v $values {
      lappend new_val [expr $v+$margin]
    }
    $table setTable $indices $new_val
```

```
    puts "After adding margin:"
    puts "  Indices: [$table getIndices]"
    puts "  Values:  [$table getValue]"
  }
}

source find_timing_power_arc.tcl
set lib [read_db ./libs/example.ldb]
set group [find_arc $lib AND2X1 Y {related_pin B} cell_fall delay_template]
add_margin $group 0.1
/home/doug/examples/liberate_LDBX/ldbx2_tutorial 2464> cat update_attribute.tcl
## Increase pin capacitance for DFFQX1, and add average_capacitance  ##
#########################################################

proc increase_pin_cap { lib cell value} {
  set cell_handle [$lib getChildren cell $cell]
  if {[llength $cell_handle] == 0} {
    puts "There is no cell $cell in this library [$lib getName]"
    return
  }
  set pins [$cell_handle getChildren pin]
  foreach pin $pins {
    puts "pin : [$pin getName]"
    puts "before increase: [$pin getAttr capacitance]"
    set cap [$pin getAttr capacitance]
    set old_val [lindex $cap 1]
    set new_val [expr $old_val+$value]
    lset cap 1 $new_val
    $pin setAttr $cap
    puts "after increase: [$pin getAttr capacitance]"
  }
}

## Add a simple attribute "average_capacitance" for each pin ##
## It's average of max and min capacitance if exist max/min, or equals to
capacitance ##
#########################################################################
#######
proc add_average_cap { lib cell } {
  set cell_handle [$lib getChildren cell $cell]
  if {[llength $cell_handle] == 0} {
```

```
    puts "There is no cell $cell in this library [$lib getName]"
    return
  }
  set pins [$cell_handle getChildren pin]
  foreach pin $pins {
    puts "pin : [$pin getName]"
    set max [$pin getAttr max_capacitance]
    set min [$pin getAttr min_capacitance]
    if {[llength $max] == 0 || [llength $min] == 0} {
      $pin getAttr capacitance
      set avg_val [lindex [$pin getAttr capacitance] 1]
    } else {
      set max_val [lindex $max 1]
      set min_val [lindex $min 1]
      set avg_val [expr ($max_val+$min_val)/2.0]
    }
    set avg_cap {average_capacitance}
    lappend avg_cap $avg_val
    $pin setAttr $avg_cap
    puts "Successfully add new attribute: [$pin getAttr average_capacitance]"
  }
}

## Add new or update old complex attribute ##
#############################################
proc update_altos_attribute { group attr_name value1 value2 } {
  set cattrs [$group getCAttr $attr_name]
  foreach {name vals} $cattrs {
    set target [lindex $vals 0]
    if {$target==$value1} {
      puts "Before update:"
      puts "$name $vals"
      $group delCAttr $attr_name $value1
      lappend new_vals $value1 $value2
      $group setCAttr $attr_name $new_vals
      puts "After update:"
      puts "$name $new_vals"
      return
    }
  }
  puts "There is no $attr_name for $value1. Adding new attribute:"
```

```
  lappend new_vals $value1 $value2
  $group setCAttr $attr_name $new_vals
  puts "$attr_name $new_vals"
}

set lib [read_db ./libs/example.ldb]

## increase pin cap for cell DFFQX1 ##
increase_pin_cap $lib DFFQX1 0.05

## add new simple attribute "average_capacitance" for cell AND2X1 for all pins ##
add_average_cap $lib AND2X1

## update altos_voltage_map VDD from 1 to 1.05 ##
#### add a new altos_vlotage_map VDDHG 1.27 #####
puts "Update VDD"
update_altos_attribute $lib altos_voltage_map VDD 1.05
puts "Add VCCHG"
update_altos_attribute $lib altos_voltage_map VCCHG 1.27
/home/doug/examples/liberate_LDBX/ldbx2_tutorial 2465> cat del
delete_add_table.tcl  delete_group.tcl
/home/doug/examples/liberate_LDBX/ldbx2_tutorial 2465> cat delete_add_table.tcl
proc delete_table { group } {
  set tables [$group getTable]
  if {[llength $tables] == 0} {'
    puts "There is not table under group [$group getHeader] [$group getName]"
    return
  }
  puts "Deleting the folling table in group [$group getHeader] [$group getName]:"
  foreach table $tables {
    puts " Table: "
    puts "Indices: [$table getIndices]"
    puts "Values:  [$table getValue]"
  }
  $group delTable
}

## delete cell_rise table from arc A->Y in cell INVX1 ##
#####################################################
source find_timing_power_arc.tcl
set lib [read_db ./libs/example.ldb]
```

```
set group [find_arc $lib INVX1 Y {related_pin A} cell_rise delay_template]
delete_table $group

## add new table in this group ##
################################
set new_table [$group addTable {{0.1 0.2 0.3} {1 2 3}} {1 2 3 4 5 6 7 8 9} "values"]
puts "New Table:"
puts "Indices: [$new_table getIndices]"
puts "Values:  [$new_table getValue]"
```

Deleting a Group Recursively

```
#delete_group.tcl
proc deleteGroup { group header name } {
  set list_to_delete  [$group getChildren $header $name]
  if {[llength $list_to_delete] > 0} {
    puts "deleting the following groups under [$group getHeader] [$group getName]"
  }
  foreach subgroup $list_to_delete {
    puts "    [$subgroup getHeader] [$subgroup getName]"
  }
  $group delGroup $list_to_delete
}


proc delete_DFS { group header name}  {
  set children [$group getChildren]
  if {[llength $children] == 0} {
    return
  }
  foreach child $children {
    delete_DFS $child $header $name
  }
  if {[$group getHeader] == "timing"} {
  }
  deleteGroup $group $header $name
 }

proc delete_BFS { group header name } {
  set children [$group getChildren]
  if {[llength $children] == 0} {
```

```
    return
  }
  foreach child $children {
    deleteGroup $group $header $name
    delete_BFS  $child $header $name
  }

}

# DFS traverse and delete
proc delete1 {} {
  set header receiver_capacitance7_rise
  set name receiver_cap_power_template
  set lib [read_db ./libs/example.ldb]
  delete_DFS $lib $header $name
  write_db $lib deleted1.ldb
}

# BFS traverse and delete
proc delete2 {} {
  set header cell_rise
  set name delay_template
  set lib [read_db ./libs/example.ldb]
  deleteGroup $lib $header $name
  delete_BFS $lib $header $name
  write_db $lib deleted2.ldb
}


################################################
# Run test
# delete receiver_capacitance7_rise receiver_cap_power_template group
delete1
# delete cell_rise delay_template group
delete2

################################################
# Run test
set lib [read_db ./libs/example_nldm.lib]
check_hold_values $lib
################################################
```

```
Find a timing or internal_power arc
## recursively traverse ldb to find timing or internal_power arc ##
#####################################################################
proc find_arc {group cell pin attr header name} {
  set children [$group getChildren]
  #foreach child $children {
  #  puts " children: [$child getHeader] [$child getName]"
  #}
  if {[llength $children] ==0 } {
    return
  }

  foreach child $children {
    if {[$child getHeader] == "cell" || [$child getHeader] == "pin"} {
      if {[$child getHeader] == "cell" && [$child getName] == $cell || [$child
getHeader] == "pin" && [$child getName] == $pin} {
        puts "Going to group: [$child getHeader] [$child getName]"
        return [find_arc $child $cell $pin $attr $header $name]
      } else { continue }
    }
   if {[$child getHeader] == "timing" || [$child getHeader] == "internal_power"} {
      set attr_name [lindex $attr 0]
      set attr_value [lindex $attr 1]
      if {[$child getAttr $attr_name] == $attr} {
        puts "Going to group: [$child getHeader] [$child getName]"
     return [find_arc $child $cell $pin $attr $header $name]
      } else { continue }
    }
    if {[$child getHeader] == $header && [$child getName] == $name} {
      puts "Found group: $header $name"
      return $child
    } else {
      find_arc $child $cell $pin $attr $header $name
    }
  }
}

#################################################
```

Printing All Groups

```
#print_all_groups.tcl
source utils.tcl

# prints all groups 6 levels deep
proc print_all_groups {lib} {
    print_grp $lib 0
    foreach g1 [$lib getChildren] {
        print_grp $g1 1
        foreach g2 [$g1 getChildren] {
            print_grp $g2 2
            foreach g3 [$g2 getChildren] {
                print_grp $g3 3
                foreach g4 [$g3 getChildren] {
                    print_grp $g4 4
                    foreach g5 [$g4 getChildren] {
                        print_grp $g5 5
                        foreach g6 [$g5 getChildren] {
                            print_grp $g6 6
                        }
                    }
                }
            }
        }
    }
}

# prints all groups and attributes 6 levels deep
proc print_all_groups_and_attrs {lib} {
    print_grp $lib 0
    print_attrs [$lib getAttr] 0
    foreach g1 [$lib getChildren] {
        print_grp $g1 1
        print_attrs [$g1 getAttr] 1
        foreach g2 [$g1 getChildren] {
            print_grp $g2 2
            print_attrs [$g2 getAttr] 2
            foreach g3 [$g2 getChildren] {
                print_grp $g3 3
                print_attrs [$g3 getAttr] 3
```

```
                foreach g4 [$g3 getChildren] {
                    print_grp $g4 4
                    print_attrs [$g4 getAttr] 4
                    foreach g5 [$g4 getChildren] {
                        print_grp $g5 5
                        print_attrs [$g5 getAttr] 5
                        foreach g6 [$g5 getChildren] {
                            print_grp $g6 6
                            print_attrs [$g6 getAttr] 6
                        }
                    }
                }
            }
        }
    }
}

################################################
# Run test
set lib [read_db ./libs/example_nldm.lib]
#print_all_groups $lib
print_all_groups_and_attrs $lib
################################################
```

Printing All Groups Recursively

```
#print_all_groups_recursive.tcl
source utils.tcl

proc print_all_groups_recursive {grp {level 0}} {
    print_grp $grp $level
    incr level
    set children [$grp getChildren]

    if {[llength $children] == 0} {
        return
    }
    foreach subgrp $children {
        # RECURSIVE CALL:
        print_all_groups_recursive $subgrp $level
    }
```

```
}

proc print_all_groups_and_attrs_recursive {grp {level 0}} {
    print_grp $grp $level
    incr level
    set children [$grp getChildren]

    # print attrs
    foreach {k v} [$grp getAttr] {
        print_attr $k $v 3
    }

    if {[llength $children] == 0} {
        return
    }
    foreach subgrp $children {
        # RECURSIVE CALL:
        print_all_groups_and_attrs_recursive $subgrp $level
    }
}

################################################
# Run test
set lib [read_db ./libs/example_nldm.lib]
#print_all_groups_recursive $lib
print_all_groups_and_attrs_recursive $lib
################################################
```

Printing Timing Group Attributes

```
#print_timing_group_attrs.tcl
source utils.tcl

proc print_timing_group_attrs {lib} {
    # get children of library group
    foreach cell [$lib getChildren cell] {
        # get its children pin groups
        foreach pin [$cell getChildren pin] {
            # get its children timing groups
            foreach tmg [$pin getChildren timing] {
                # get all the attributes and print them out
```

```
                pretty_print [get_hier $tmg] 1
                foreach {k v} [$tmg getAttr] {
                    print_attr $k $v 3
                }
            }
        }
    }
}

proc print_timing_group_attrs_FLAT {lib} {
    print_grp $lib 0
    foreach cell [$lib getChildren cell] {
        print_grp $cell 1
        foreach pin [$cell getChildren pin] {
            print_grp $pin 2
            foreach tmg [$pin getChildren timing] {
                print_grp $tmg 3
                foreach {k v} [$tmg getAttr] {
                    print_attr $k $v 4
                }
            }
        }
    }
}

################################################
# Run test
set lib [read_db ./libs/example_nldm.lib]
print_timing_group_attrs $lib
#print_timing_group_attrs_FLAT $lib
#
```

Updating Group Attributes

```
#update_attribute.tcl
## Increase pin capacitance for DFFQX1, and add average capacitance  ##
########################################################

proc increase_pin_cap { lib cell value} {
  set cell_handle [$lib getChildren cell $cell]
  if {[llength $cell_handle] == 0} {
```

```
    puts "There is no cell $cell in this library [$lib getName]"
    return
  }
  set pins [$cell_handle getChildren pin]
  foreach pin $pins {
    puts "pin : [$pin getName]"
    puts "before increase: [$pin getAttr capacitance]"
    set cap [$pin getAttr capacitance]
    set old_val [lindex $cap 1]
    set new_val [expr $old_val+$value]
    lset cap 1 $new_val
    $pin setAttr $cap
    puts "after increase: [$pin getAttr capacitance]"
  }
}


## Add a simple attribute "average_capacitance" for each pin ##
## It's average of max and min capacitance if exist max/min, or equals to
capacitance ##
##########################################################################
#######
proc add_average_cap { lib cell } {
  set cell_handle [$lib getChildren cell $cell]
  if {[llength $cell_handle] == 0} {
    puts "There is no cell $cell in this library [$lib getName]"
    return
  }
  set pins [$cell_handle getChildren pin]
  foreach pin $pins {
    puts "pin : [$pin getName]"
    set max [$pin getAttr max_capacitance]
    set min [$pin getAttr min_capacitance]
    if {[llength $max] == 0 || [llength $min] == 0} {
      $pin getAttr capacitance
      set avg_val [lindex [$pin getAttr capacitance] 1]
    } else {
      set max_val [lindex $max 1]
      set min_val [lindex $min 1]
      set avg_val [expr ($max_val+$min_val)/2.0]
    }
    set avg_cap {average_capacitance}
```

```
    lappend avg_cap $avg_val
    $pin setAttr $avg_cap
    puts "Successfully add new attribute: [$pin getAttr average_capacitance]"
  }
}

## Add new or update old complex attribute ##
###############################################
proc update_altos_attribute { group attr_name value1 value2 } {
  set cattrs [$group getCAttr $attr_name]
  foreach {name vals} $cattrs {
    set target [lindex $vals 0]
    if {$target==$value1} {
      puts "Before update:"
      puts "$name $vals"
      $group delCAttr $attr_name $value1
      lappend new_vals $value1 $value2
      $group setCAttr $attr_name $new_vals
      puts "After update:"
      puts "$name $new_vals"
      return
    }
  }
  puts "There is no $attr_name for $value1. Adding new attribute:"
  lappend new_vals $value1 $value2
  $group setCAttr $attr_name $new_vals
  puts "$attr_name $new_vals"
}

set lib [read_db ./libs/example.ldb]

## increase pin cap for cell DFFQX1 ##
increase_pin_cap $lib DFFQX1 0.05

## add new simple attribute "average_capacitance" for cell AND2X1 for all pins ##
add_average_cap $lib AND2X1

## update altos_voltage_map VDD from 1 to 1.05 ##
#### add a new altos_vlotage_map VDDHG 1.27 #####
puts "Update VDD"
update_altos_attribute $lib altos_voltage_map VDD 1.05
```

```
puts "Add VCCHG"
update_altos_attribute $lib altos_voltage_map VCCHG 1.27
```

Python Interface Examples

This section provides example scripts for Python interface:

- Comparing Groups Recursively
- Printing All groups
- Printing Timing Group Attributes
- Updating Atrributes

Example Python Utility Script

```
#utils.py
def print_grp(group, level, file):
    indent = level*2
    file.write(" "*indent + group.getHeader() + "(" +group.getName()+")\n")

def print_attrs(group, level, file):
    indent = level*2
    for attr in group.getAttr():
        file.write(" "*indent+"  "+"attr: "+attr[0]+" "+attr[1]+"\n")
def print_cattrs(group, level, file):
    indent = level*2
    for cattr in group.getCAttr():
        file.write(" "*indent+"  "+"attr: "+attr[0]+" "+str(attr[1])+"\n")

def pretty_print(msg,level,file):
    indent = level*2
    file.write(" "*indent+msg+"\n")

def print_hier(group,file,res=""):
    group_str = group.getHeader()+"("+group.getName()+")"
    if res!="":
        new_res = group_str+"-->"+res
    else:
        new_res = group_str
    if group.getHeader()=="library":
        file.write("  "+new_res)
        return
    print_hier(group.getParent(),file,new_res)
```

Comparing Groups Recursively

This script demonstrates comparison of simple attributes between two groups.

```
#compare_groups_recursive.py

def compare_attrs(grp1,grp2,file,level=0):
    attrs1 = grp1.getAttr()
    attrs2 = grp2.getAttr()
    for attr1 in attrs1:
        # check if attr exists
        attr2 = grp2.getAttr((attr1[0],))
        #print(str(attr1)+ "     " +str(attr2))
        if len(attr2)==0:
            print_hier(grp1,file)
            file.write(" : attr key mismatch : " + attr1[0] + " missing from
lib2\n")
            continue
        # check if attr val is same
        if attr1[1] != attr2[0][1]:
            print_hier(grp1,file)
            file.write(" : attr value mismatch : " + attr1[0] + " value mismatch:
"+ attr1[1] + " != "+attr2[0][1]+"\n")
            continue



#
# compares 2 libraries (or any 2 groups) hierarchicaly for groups and simple
attributes.
# Reports groups not present in lib2, and attributes that dont exist in lib2
# or dont have matching values
#

def compare_groups_recursive(grp1,grp2,file,level=0):
    if level==0:
        funcname = inspect.stack()[0][3]
        print("\n################## Running " + funcname + " ################")
        file.write("\n\n################## " + funcname + " ##################\n")
    # compare attrs
    compare_attrs(grp1,grp2,file,level)

    level=level+1
```

```
    children = grp1.getChildren()
    if len(children)==0:
        return


    for subgrp in children:
        # make most groups unique
        chkAttrs = ("related_pin","timing_sense","timing_type","when") #
attributes to make groups unique
        attrList = subgrp.getAttr(chkAttrs)
        g2child = grp2.getChildren(subgrp.getHeader(),subgrp.getName(),attrList)
        if len(g2child) > 0:
            continue
        if len(g2child) == 0:
            print_hier(subgrp,file)
            file.write(" : group mismatch: missing group
'"+subgrp.getHeader()+"("+subgrp.getName()+")' in lib2\n")
            continue
        # recurse IN (call cmp_grp func again) into next deeper level group
        compare_groups_recursive(subgrp,g2child[0],file,level)


check_hold_values.py
/home/doug/examples/liberate_dbx/Python/ldbx2_python_tutorial/DATA/tcl 2168> ls
check_hold_values.py     print_all_groups.py    print_timing_group_attrs.py
utils.py
compare_groups_recursive.py  print_all_groups_recursive.py  update_attribute.py
/home/doug/examples/liberate_dbx/Python/ldbx2_python_tutorial/DATA/tcl 2169> cat
check_hold_values.py
#
# checks all hold_rising rise_constraint arcs
#
def check_hold_values(lib,file):
    funcname = inspect.stack()[0][3]
    print("\n################## Running " + funcname + " ################")
    file.write("\n\n################## " + funcname + " ##################\n")
    for cell in lib.getChildren("cell"):
        for pin in cell.getChildren("pin"):
            for tmg in pin.getChildren("timing"):
                arcs =
tmg.getChildren("rise_constraint")+tmg.getChildren("fall_constraint")
                for holdarc in arcs:
                    file.write("FOUND: ")
                    print_hier(holdarc,file)
```

```
                    file.write("\n")
                    print_grp(holdarc,2,file)
                    print_attrs(holdarc,3,file)
                    print_cattrs(holdarc,3,file)
```

Printing All groups

This code will print all groups till six levels.

```
#print_all_groups.py

def print_all_groups (lib, file):
    funcname = inspect.stack()[0][3]
    print("\n################## Running " + funcname + " ################")
    file.write("\n\n################## " + funcname + " ##################\n")
    print_grp(lib,0,file)
    for g1 in  lib.getChildren():
        print_grp(g1,1,file)
        for g2 in  g1.getChildren():
            print_grp(g2,2,file)
            for g3 in g2.getChildren():
                print_grp(g3,3,file)
                for g4 in  g3.getChildren():
                    print_grp(g4,4,file)
                    for g5 in  g4.getChildren():
                        print_grp(g5,5,file)
                        for g6 in  g5.getChildren():
                            print_grp(g6,6,file)

# prints all groups and attributes 6 levels deep
def print_all_groups_and_attrs(lib,file):
    funcname = inspect.stack()[0][3]
    print("\n################## Running " + funcname + " ################")
    file.write("\n\n################## " + funcname + " ##################\n")
    print_grp(lib,0,file)
    print_attrs(lib,0,file)
    for g1 in  lib.getChildren():
        print_grp(g1,1,file)
        print_attrs(g1,1,file)
        for g2 in  g1.getChildren():
            print_grp(g2,2,file)
```

```
            print_attrs(g2,2,file)
            for g3 in g2.getChildren():
                print_grp(g3,3,file)
                print_attrs(g3,3,file)
                for g4 in  g3.getChildren():
                    print_grp(g4,4,file)
                    print_attrs(g4,4,file)
                    for g5 in  g4.getChildren():
                        print_grp(g5,5,file)
                        print_attrs(g5,5,file)
                        for g6 in  g5.getChildren():
                            print_grp(g6,6,file)
                            print_attrs(g6,6,file)

print_all_groups_recursive.py
def print_all_groups_recursive (grp, file, level=0):
    if level == 0:
        funcname = inspect.stack()[0][3]
        print("\n################## Running " + funcname + " ################")
        file.write("\n\n################## " + funcname + " ##################\n")
    print_grp(grp,level,file)
    level=level+1
    children = grp.getChildren()
    if len(children)==0:
        return
    for subgrp in children:
        # RECURSIVE CALL
        print_all_groups_recursive(subgrp,file,level)


def print_all_groups_and_attrs_recursive(grp,file,level=0):
    if level==0:
        funcname = inspect.stack()[0][3]
        print("\n################## Running " + funcname + " ################")
        file.write("\n\n################## " + funcname + " ##################\n")
    print_grp(grp,level,file)
    print_attrs(grp,level,file)
    level=level+1
    children = grp.getChildren()
    # print attribute
    if len(children)==0:
```

```
        return
    for subgrp in children:
        # RECURSIVE CALL
        print_all_groups_and_attrs_recursive(subgrp,file,level)
```

Printing Timing Group Attributes

```
#print_timing_group_attrs.py
def print_timing_group_attrs(lib,file):
    funcname = inspect.stack()[0][3]
    print("\n################## Running " + funcname + " ################")
    file.write("\n\n################## " + funcname + " ##################\n")
    # get children of library group
    for cell in lib.getChildren("cell"):
        # get its children pin groups
        for pin in cell.getChildren("pin"):
            # get its children timing groups
            for tmg in pin.getChildren("timing"):
                # get all the attributes and print them out
                print_hier(tmg,file)
                file.write("\n")
                print_attrs(tmg,3,file)

def print_timing_group_attrs_FLAT(lib,file):
    funcname = inspect.stack()[0][3]
    print("\n################## Running " + funcname + " ################")
    file.write("\n\n################## " + funcname + " ##################\n")
    print_grp(lib,0,file)
    for cell in lib.getChildren("cell"):
        print_grp(cell,1,file)
        for pin in cell.getChildren("pin"):
            print_grp(pin,2,file)
            for tmg in pin.getChildren("timing"):
                print_grp(tmg,3,file)
                print_attrs(tmg,4,file)
```

Updating Atrributes

```
#update_attribute.py
## Increase pin capacitance for DFFQX1, and add average_capacitance  ##
#####################################################
```

```
def increase_pin_cap(lib,cell,value,file):
    funcname = inspect.stack()[0][3]
    print("\n################## Running " + funcname + " ################")
    file.write("\n\n################## " + funcname + " ##################\n")
    cell_handle=lib.getChildren("cell",cell)
    if len(cell_handle)==0:
        file.write("There is no cell "+ cell + " in this library " + lib.getName()
+ "\n")
        return
    file.write("Increase caps by " + str(value) + " for cell " + cell + " in the
following pins:\n")
    pins = cell_handle[0].getChildren("pin")
    for pin in pins:
        file.write("pin : " + pin.getName() + "\n")
        cap = list(pin.getAttr(("capacitance",))[0])
        file.write("before increase: " + str(cap) + "\n")
        new_val = float(cap[1])+value
        cap[1] = str('%g'%(new_val))
        pin.setAttr((tuple(cap),))
        file.write("after increase: " +
str(pin.getAttr(("capacitance",))[0])+"\n")



def update_attribute(lib, file):
    funcname = inspect.stack()[0][3]
    print("\n################## Running " + funcname + " ################")
    increase_pin_cap(lib, "DFFQX1", 0.05, file)
```

4

Glossary

attribute

Structure in `.ldb` or `.lib` that do not use curly braces `{}`. There are two kinds of attributes, simple and complex.

simple attribute

The attribute name and value are separated by a colon (`:`).

Example: `variable_3 : time;`

Note: Simple attribute names cannot be duplicated in the library.

complex attribute

Attribute values are specified inside braces ().

Example 1: `index_1 ("0.006, 0.21") ;`

Example 2: `capacitive_load_unit (1,pf) ;`

Complex attributes allow duplicated attribute names but the values must be different.

group

A structure in a `.ldb` or `.lib` file with data that was put inside curly braces `{}`.

Example of a cell group:

```
cell (INVX1) {…} ;
```

Example of a timing group:

```
timing () {…} ;
```

group header

The head of a group.

Example:

`cell` is the group header:

```
cell (INVX1) {…} ;
```

group name

The name of a group.

Example:

`VSS` is the group name of `pg_pin`:

```
pg_pin (VSS) {…} ;
```

handle

A handle to a group object. Use this when calling group and attribute commands.

Example: `$handle getChildren`