Hurricane Python Tutorial

Contents

1. Introduction

This tutorial is aimed at two goals :

- Presenting how to use Python scripts to control CORIOLIS .
- Make a basic introduction about the HURRICANE database and it's concepts.

While this tutorial is aimed at presenting the HURRICANE database, do not feel limited to it. You can use HURRICANE objects as attributes of PYTHON objects or use PYTHON containers to store them. The only limitation is that you may not use HURRICANE classes as base classes in PYTHON .

All HURRICANE objects implements the PYTHON `__str__()` function, they print the result of C++ `::getString()`.

1.1 Generalities

The C++ API has been exported in Python as closely as possible. Meaning that, save for a slight change in syntax, any code written in PYTHON could be easily transformed into C++ code. There is no specific documentation written for the PYTHON interface, you may directly use the C++ one.

Mostly:

- C++ namespaces are exported as PYTHON modules.
- The *scope resolution operator* `::` converts into `.` .
- C++ blocks (between braces `{}`) are replaced by indentations.
- In C++, names are manageds through a dedicated `Name` class. It has not been exported to the PYTHON interface, you only have to use `string`.
- Coordinates are expressed in `DbU` which are `long` with a special semantic (see ??).

In `hurricane/Session.h` header we have:

```
namespace Hurricane {

   class UpdateSession {
     public:
       static void  open  ();
       static void  close ();
   };

}
```

So we can use it the following way in C++:

```
#include "hurricane/Session.h"

using namespace Hurricane;

void  doSomething ()
{
  UpdateSession::open();
  // Something...
  UpdateSession::close();
}
```

The equivalent PYTHON code will be:

```
from Hurricane import *

def doSomething ():
     UpdateSession.open()
    # Something...
     UpdateSession.close()
```

1.2 Various Kinds of Constructors

Regarding the memory allocation, the HURRICANE database contains two kind of objects.

1. Objects that are linked to others in the database and whose creation or deletion implies coherency operations. This is the case for Net or Horizontal. They must be created using the static **create()** method of their class and destroyed with their **destroy()** method.

   And, of course, they cannot be copied (the copy constructor has been disabled).

   ```
   net = Net.create( cell, 'tmp' )  # Call the static Net.create() function.
                                    # Work with this net.
                                    # ...
   net.destroy()                    # Call the dynamic destroy() method.
   ```

2. Objects that are *standalone*, like Point or Box, uses the usual construction methods. They also use the PYTHON garbage collector mechanism and do not need to be explicitly deleted.

   ```
   def myfunc():
     bb = Box( DbU.fromLambda( 0.0)
             , DbU.fromLambda( 0.0)
             , DbU.fromLambda(15.0)
             , DbU.fromLambda(50.0) )
     return                           # bb will be freed at that point.
   ```

2. Setting up the Environment

2.1 Setting up the Pathes

To simplify the tedious task of configuring your environment, a helper is provided. It will setup or modify the **PATH**, **LD_LIBRARY_PATH** (or **DYLD_LIBRARY_PATH** under DARWIN), **PYTHONPATH** and **CORIOLIS_TOP** variables. It should automatically adapt to your kind of shell (`Bourne` like or `C-Shell` like).

```
<CORIOLIS_INSTALL>/etc/coriolis2/coriolisEnv.py
```

Use it like this (don't forget the `eval` **and** the backquotes):

```
dummy@lepka:~> eval `<CORIOLIS_INSTALL>/etc/coriolis2/coriolisEnv.py`
```

Note

Do not call that script in your environement initialisation. When used under RHEL6 or clones, it needs to be run in the **devtoolset2** environement. The script then launch a new shell, which may cause an infinite loop if it's called again in, say **~/.bashrc**.
Instead you may want to create an alias:

```
alias c2r='eval "`<CORIOLIS_INSTALL>/etc/coriolis2/coriolisEnv.py`"'
```

2.2 User's Configurations File

You may create, in the directory you are lanching CORIOLIS tools, a special sub-directory `.coriolis2/` that can contains two configuration files:

- `techno.py` tells which technology to use.
- `settings.py` can overrides almost any default configuration setting.

Those two files are *optional*, if they do not exists the default settings will be used and the technology is `symbolic/cmos` (i.e. purely symbolic).

Note

Those two files will by processed by the PYTHON interpreter, so they can contain any code in addition to the mandatory variables.

2.2.1 The `techno.py` File

Must provide one variable named **`technology`** which value the path towards the technology file. The available technologies are installed under `<CORIOLIS_INSTALL>/etc/coriolis2`. For example, to use the 45nm FreeDPK which is in:

```
<CORIOLIS_INSTALL>/etc/coriolis2/45/freepdk_45/
```

The `techno.py` file must contain:

```
technology = '45/freepdk_45'
```

2.2.2 The `settings.py` File

The entries of the `parametersTable` and their definitions are detailed in CGT - The Graphical Interface.

Example of file:

```
defaultStyle = 'Alliance.Classic [black]'

parametersTable = \
    ( ('misc.catchCore'           , TypeBool      , False    )
    , ('misc.info'                , TypeBool      , False    )
    , ('misc.paranoid'            , TypeBool      , False    )
    , ('misc.bug'                 , TypeBool      , False    )
    , ('misc.logMode'             , TypeBool      , False    )
    , ('misc.verboseLevel1'       , TypeBool      , False    )
    , ('misc.verboseLevel2'       , TypeBool      , True     )
    )
```

3. Creating Cell, Net and Component

In this part, we well show how to create and save a single Cell.

3.1 The AllianceFramework (CRL Core)

The HURRICANE database only manage objects in memory. To load or save something from the outside, we need to use a *framework*. As of today, only one is available : the Alliance framework. It allows CORIOLIS to handle ALLIANCE libraries and cells in the exact same way.

Note

To see how the AllianceFramework is configured for your installation, please have a look to `alliance.conf` in the `etc/coriolis2` directory. It must contains the same settings as the various `MBK_` variables used for ALLIANCE .

3.2 Session Mechanism (Hurricane)

In the HURRICANE database, all modifications must take place inside an UpdateSession. At the closing of a session, created or modificateds objects are fully inserted in the database. This is especially true for the visualisation, a created component will be visible *only* only after the session close.

Note

See `QuadTree` and `Query`.

3.3 Creating a new Cell (CRL Core)

The creation of a new Cell occurs through the AllianceFramework, and, as stated above, inside a UpdateSession. The AllianceFramework environment is provided by the CRL module.

```
from Hurricane import *
from CRL       import *

af = AllianceFramework.get()
UpdateSession.open()

cell = af.createCell( 'my_inv' )

# Build then save the Cell.

UpdateSession.close()
```

This is the simplest call to `createCell()`, and it that case, the newly created Cell will be saved in the *working library* (usually, the current directory). You may supply a second argument telling into which library you want the Cell to be created.

In the Hurricane Cell object, there is no concept of *view*, it contains completly fused logical (netlist) and physical (layout) views.

3.4 The DbU Measurement Unit

All coordinates in the Hurricane database are expressed in DbU (for *Database Unit*) which are integer numbers of foundry grid. To be more precise, they are fixed points numbers expressed in hundredth of foundry grid (to allow transient non-integer computation).

To work with symbolic layout, that is, using lambda based lengths, two conversion functions are provideds:

- `unit = DbU.fromLambda( lbd )` convert a lambda **`lbd`** into a `DbU`.
- `lbd = DbU.toLambda( unit )` convert a `DbU` into a lambda **`lbd`** .

In the weakly typed Python world, **`lbd`** is *float* while **`unit`** is *integer*.

3.5 Setting up the Abutment Box

To setup the abutment box, we use a Box which define a box from the coordinates of the lower left corner `(x1,y1)` and upper left corner `(x2,y2)`.

```
b = Box( DbU.fromLambda( 0.0)    # x1
       , DbU.fromLambda( 0.0)    # y1
       , DbU.fromLambda(15.0)    # x2
       , DbU.fromLambda(50.0) )  # y2
cell.setAbutmentBox( b )
```

Or more simply:

```
cell.setAbutmentBox( Box( DbU.fromLambda( 0.0)
                        , DbU.fromLambda( 0.0)
                        , DbU.fromLambda(15.0)
                        , DbU.fromLambda(50.0) ) )
```

3.6 Adding Nets and Components

In the Hurricane terminology, a **component** is any kind of physical object among:

- Contact

- Pad
- RoutingPad
- Horizontal
- Vertical
- Plug is the only exception and will be detailed later (see ??).

Components cannot be created *alone*. They must be part of a Net.

3.6.1 Getting a Layer

As physical elements, Components are created using a Layer. So prior to their creation, we must get one from the database. Layers are stored in the Technology, which in turn, is stored in the DataBase. So, to get a Layer:

```
layer = DataBase.getDB().getTechnology().getLayer( 'METAL1' )
```

Note

Convention for layer names. As the database can manage both real layers and symbolic ones we adopt the following convention:

- **Real layers** are named in lowercase (`metal1`, `nwell`).
- **Symbolic layers** are named in uppercase (`METAL1`, `NWELL`).

3.6.2 Creating a Net

As said above, prior to creating any Component, we must create the Net it will belongs to. In that example we also make it an *external* net, that is, a part of the interface. Do not mistake the name of the net given as a string argument `'i'` and the name of the *variable* `i` holding the Net object. For the sake of clarity we try to give the variable a close name, but this is not mandatory.

```
i = Net.create( cell, 'i' )
i.setExternal( True )
```

3.6.3 Creating a Component

Finally, we get ready to create a Component, we will make a Vertical segment of `METAL1`.

```
segment = Vertical.create( i                          # The owner Net.
                         , layer                      # The layer.
                         , DbU.fromLambda(  5.0 )     # The X coordinate.
                         , DbU.fromLambda(  2.0 )     # The width.
                         , DbU.fromLambda( 10.0 )     # The Y source coordinate.
                         , DbU.fromLambda( 40.0 ) )   # The Y target coordinate.
```

With this overload of the `Vertical.create()` function the segment is created at an absolute position. There is a second overload for creating a relatively placed segment, see *articulated layout*.

If the net is external, that is, part of the interface of the cell, you may have to declare some of it's components as physical connectors usable by the router. This is done by calling the NetExternalComponents class:

```
NetExternalComponents.setExternal( segment )
```

3.7 Saving to Disk (CRL Core)

Once you are finished building your cell, you have to save it on disk. Using the AllianceFramework you can save it as a pair of file:

View	Flag	File extension
Logical / Netlist	`Catalog.State.Logical`	`.vst`
Physical / Layout	`Catalog.State.Physical`	`.ap`

To save both views, use the `Catalog.State.Views` flag. The files will be written in the ALLIANCE `WORK_DIR`.

```
af.saveCell( cell, Catalog.State.Views )
```

4. Manipulating Cells, Nets and Components

In this part, we well show how to navigate through the Nets and Components of a Cell.

4.1 Hurricane Collections

In HURRICANE all kind of set of objects, whether organized in a real container like a `map<>` (dictionary / `dict`) or a `vector<>` (table / `list`) or an algorithmic walkthrough of the database can be accessed through a Collection.

C++ Collections object are exposed in PYTHON through the *iterable* protocol, allowing to simply write:

```
for net in cell.getNets():
  print 'Components of', net
  for component in net.getComponents():
    print '|', component
```

In C++ we would have written:

```
for ( Net* net : cell->getNets() ) {
  cout << "Components of " << net << endl;
  for ( Component* component : net->getComponents() ) {
    cout << "| " << component << endl,
  }
}
```

4.1.1 Restrictions about using Collections

Never delete or create an element while you are iterating over a Collection.

Results can be unpredictable, you may just end up with a core dump, but more subtly, some element of the Collection may be skippeds or processed twice. If you want to create or delete en element, do it outside of the collection loop. For example:

```
cellNets = []
for net in cell.getNets():
  cellNets.append( net )

# Remove all the anonymous nets.
for net in cellNets:
   if net.getName().endswith('nymous_'):
     print 'Destroy', net
     net.destroy()
```

4.2 Loading a Cell with AllianceFramework

As presented in 2.1 The Alliance Framework, the Cell that will be returned by the `getCell()` call wil be:

1. If a Cell of that name is already loaded into memory, it will be returned.

Note

It means that it shadows any modifications that could have been on disk since it was first loaded. Conversely, if the Cell has been modified in memory, you will get those modifications.

2. Search, in the ordered list of libraries, the first Cell that match the requested name.

Note

It means that if cells with the same name exists in different libraries, only the one in the first library will be ever used. Be also weary of cell files that may remains in the `WORK_LIB`, they may unexpectedly shadow cells from the libraries.

```
cell = af.getCell( 'inv_x1', Catalog.State.Views )
```

5. Make a script runnable through cgt

To use your you may run it directly like any other Python script. But, for debugging purpose it may be helpful to run it through the interactive layout viewer **cgt** .

For **cgt** to be able to run your script, you must add to your script file a function named **ScriptMain()** , which takes a dictionnary as sole argument (****kw**). The `kw` dictionnary contains, in particular, the CellViewer object we are running under with the keyword `editor`. You can then load your cell into the viewer using the menu:

- `Tools` → `Python Script` . The script file name must be given without the `.py` extension.

```
def buildInvertor ( editor ):
    UpdateSession.open()

    cell = AllianceFramework.get().createCell( 'invertor' )
    cell.setTerminal( True )

    cell.setAbutmentBox( Box( toDbU(0.0), toDbU(0.0), toDbU(15.0), toDbU(50.0) ) )

    if editor:
      UpdateSession.close()
      editor.setCell( cell )
      editor.fit()
      UpdateSession.open()

    # The rest of the script...

    return


def ScriptMain ( **kw ):
    editor = None
    if kw.has_key('editor') and kw['editor']:
      editor = kw['editor']

    buildInvertor( editor )
    return True
```

5.1 Using Breakpoints

It is possible to add breakpoints inside a script by calling the `Breakpoint.stop()` function. To be able to see exactly what has just been moficated, we must close the UpdateSession just before calling the breakpoint and reopen it just after. The `Breakpoint.stop()` function takes two arguments:

1. The `level` above witch it will be active.
2. An informative message about the purpose of the breakpoint.

We can create a little function to ease the work:

```
def doBreak ( level, message ):
    UpdateSession.close()
    Breakpoint.stop( level, message )
    UpdateSession.open()
```

6. The Complete Example File

The example files can be found in the `share/doc/coriolis2/examples/scripts/` directory (under the the root of the CORIOLIS installation).

```
#!/usr/bin/python

import sys
from   Hurricane import *
from   CRL       import *


def toDbU ( l ): return DbU.fromLambda(l)


def doBreak ( level, message ):
    UpdateSession.close()
    Breakpoint.stop( level, message )
    UpdateSession.open()


def buildInvertor ( editor ):
    UpdateSession.open()

    cell = AllianceFramework.get().createCell( 'invertor' )
    cell.setTerminal( True )

    cell.setAbutmentBox( Box( toDbU(0.0), toDbU(0.0), toDbU(15.0), toDbU(50.0) ) )

    if editor:
      UpdateSession.close()
      editor.setCell( cell )
      editor.fit()
      UpdateSession.open()

    technology = DataBase.getDB().getTechnology()
    metal1     = technology.getLayer( "METAL1"     )
    poly       = technology.getLayer( "POLY"       )
    ptrans     = technology.getLayer( "PTRANS"     )
    ntrans     = technology.getLayer( "NTRANS"     )
    pdif       = technology.getLayer( "PDIF"       )
    ndif       = technology.getLayer( "NDIF"       )
    contdifn   = technology.getLayer( "CONT_DIF_N" )
    contdifp   = technology.getLayer( "CONT_DIF_P" )
    nwell      = technology.getLayer( "NWELL"      )
    contpoly   = technology.getLayer( "CONT_POLY"  )
    ntie       = technology.getLayer( "NTIE"       )
    net = Net.create( cell, "nwell" )
    Vertical.create( net, nwell, toDbU(7.5), toDbU(15.0), toDbU(27.0), toDbU(51.0) )

    vdd = Net.create( cell, "vdd" )
    vdd.setExternal( True )
    vdd.setGlobal  ( True )
    h = Horizontal.create(vdd, metal1, toDbU(47.0), toDbU(6.0), toDbU(0.0), toDbU(15.0) )
    NetExternalComponents.setExternal( h )
    Contact.create ( vdd, contdifn, toDbU(10.0), toDbU(47.0), toDbU( 1.0), toDbU( 1.0) )
    Contact.create ( vdd, contdifp, toDbU( 4.0), toDbU(45.0), toDbU( 1.0), toDbU( 1.0) )
    Vertical.create( vdd, pdif    , toDbU( 3.5), toDbU( 4.0), toDbU(28.0), toDbU(46.0) )
    Vertical.create( vdd, ntie    , toDbU(10.0), toDbU( 3.0), toDbU(43.0), toDbU(48.0) )
    doBreak( 1, 'Done building vdd.' )

    vss = Net.create( cell, "vss" )
    vss.setExternal( True )
    vss.setGlobal  ( True )
    h = Horizontal.create(vss, metal1, toDbU(3.0), toDbU(6.0), toDbU(0.0), toDbU(15.0))
    NetExternalComponents.setExternal( h )
    Vertical.create( vss, ndif    , toDbU(3.5), toDbU(4.0), toDbU(4.0), toDbU(12.0) )
    Contact.create ( vss, contdifn, toDbU(4.0), toDbU(5.0), toDbU(1.0), toDbU( 1.0) )
    doBreak( 1, 'Done building vss.' )

    i = Net.create( cell, "i" )
    i.setExternal( True )
    v = Vertical.create ( i, metal1, toDbU(5.0), toDbU(2.0), toDbU(10.0), toDbU(40.0) )
    NetExternalComponents.setExternal( v )
    Vertical.create   ( i, ptrans  , toDbU( 7.0), toDbU( 1.0), toDbU(26.0), toDbU(39.0) )
    Vertical.create   ( i, ntrans  , toDbU( 7.0), toDbU( 1.0), toDbU( 6.0), toDbU(14.0) )
    Vertical.create   ( i, poly    , toDbU( 7.0), toDbU( 1.0), toDbU(14.0), toDbU(26.0) )
    Horizontal.create( i, poly    , toDbU(20.0), toDbU( 3.0), toDbU( 4.0), toDbU( 7.0) )
    Contact.create    ( i, contpoly, toDbU( 5.0), toDbU(20.0), toDbU( 1.0), toDbU( 1.0) )
    doBreak( 1, 'Done building i.' )

    nq = Net.create ( cell, "nq" )
    nq.setExternal( True )
    v = Vertical.create( nq, metal1, toDbU(10.0), toDbU(2.0), toDbU(10.0), toDbU(40.0) )
    NetExternalComponents.setExternal( v )
    Vertical.create( nq, pdif    , toDbU(10.0), toDbU( 3.0), toDbU(28.0), toDbU(37.0) )
    Vertical.create( nq, ndif    , toDbU(10.0), toDbU( 3.0), toDbU( 8.0), toDbU(12.0) )
    Contact.create ( nq, contdifp, toDbU(10.0), toDbU(35.0), toDbU( 1.0), toDbU( 1.0) )
    Contact.create ( nq, contdifp, toDbU(10.0), toDbU(30.5), toDbU( 1.0), toDbU( 1.0) )
    Contact.create ( nq, contdifn, toDbU(10.0), toDbU(10.0), toDbU( 1.0), toDbU( 1.0) )
    doBreak( 1, 'Done building q.' )

    UpdateSession.close()
    return


def ScriptMain ( **kw ):
    editor = None
    if kw.has_key('editor') and kw['editor']:
      editor = kw['editor']

    buildInvertor( editor )
    return True
```