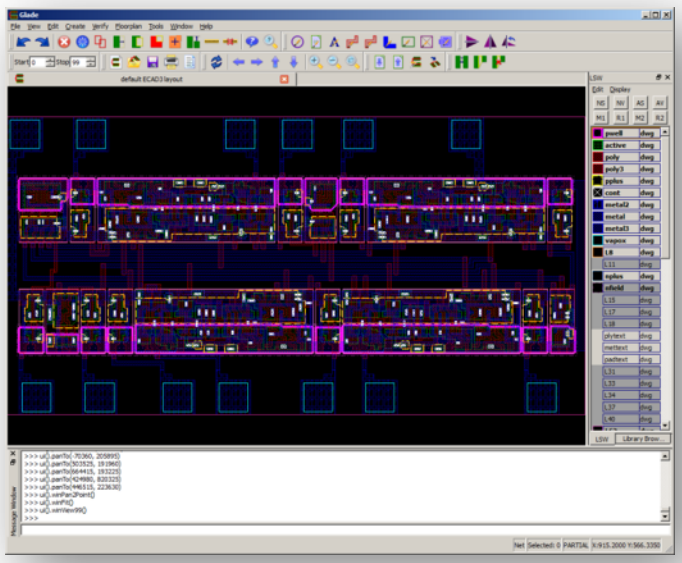
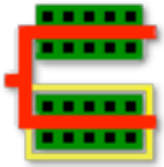

Glade
Peardrop Design Systems
10th June 2013

What is Glade?

- **G**DS **L**EF **a**nd **D**EF **E**ditor
  - Allows viewing and editing of physical data in a variety of formats
  - Multi Platform: runs on Windows, Linux, Mac (even Solaris!)
  - Simple yet powerful
    - Programmable in Python
    - Pcells (parameterised cells) in Python
    - DRC, extraction, LVS
    - Etc…

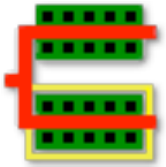


Contents

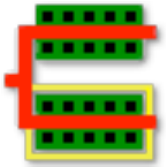


1. GUI
2. Techfile setup
3. Basic editing
4. Pcells
5. DRC / extraction / LVS
6. Python programming

GUI

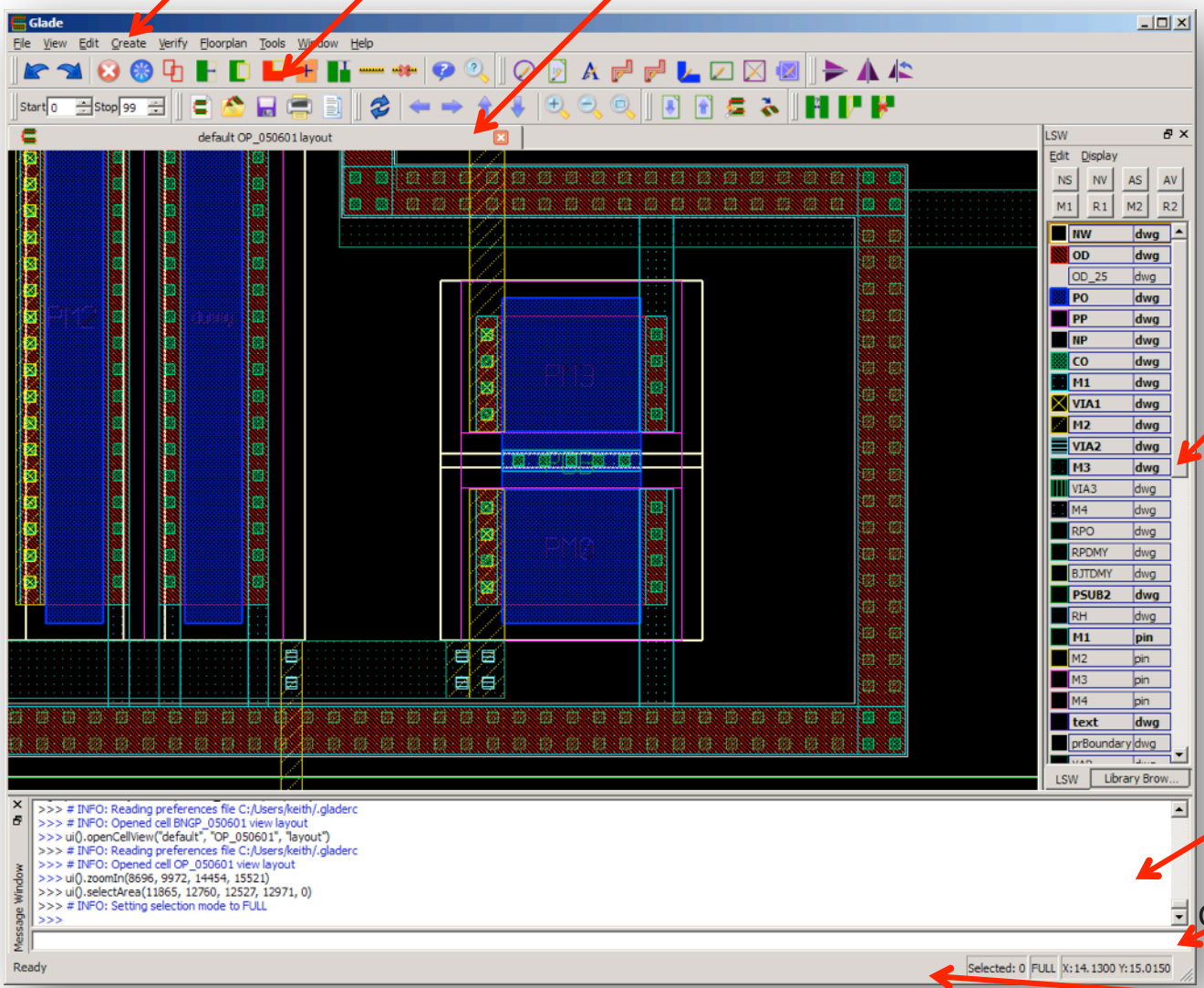


Starting Glade

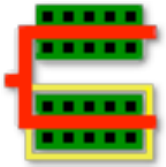


- From the icon
  - Set file association if you want to double click on e.g. a GDS file and have Glade open it
- From the command line
  - glade [options]
- On startup, Glade reads:
  - ~/.gladerc (Linux/Mac) or %HOME%\.gladerc (Windows)
    - Display and selection settings, window arrangement.
  - ~/.glade.py (Linux/Mac) or %HOME%\.glade.py (Windows)
    - Python script that is run on startup. Useful for loading default techfile, project library etc. Note that this is read **before** any command line options.

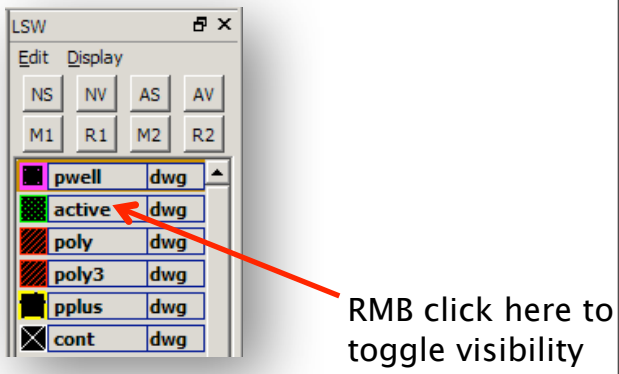
GUI – the basics

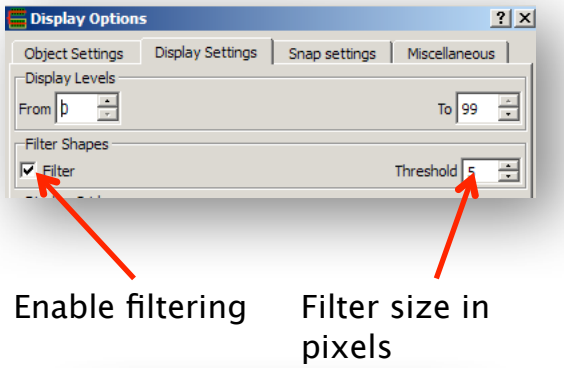


GUI – Visibility

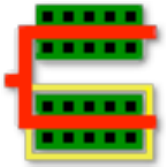


- Layer visibility controlled by LSW
  - Right mouse click on layer toggles visibility
- Small objects (less than filter size)
  - Not drawn to improve drawing speed on very large designs
    - Can be controlled by display options form
- Display of layout hierarchy
  - Controlled by start & stop depth
    - Toolbar / menu / bindkeys can change the display depth



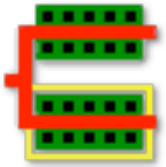


GUI – Selection

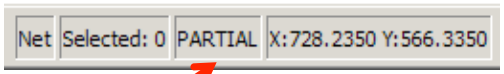


- Selection
  - Many commands work on the ‘selected set’ of objects
  - Use Left mouse button to select objects
    - Single click selects objects
    - Shift + click adds to selected set
    - Ctrl + click removes from selected set
    - Click and drag selects objects in the drag area
    - Double click selects and queries and object’s attributes
    - Number of items selected shown in the status bar
  - Selected objects are highlighted
  - Other unselected objects can be dimmed (set in Selection Options form)

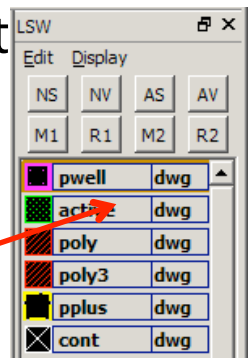
GUI – Selection



- Selection Mode
  - Full: whole objects are selected
  - Partial: Edges or Vertices of shapes are selected
  - Mode is toggled by F4 key
  - Current selection mode shown in status bar

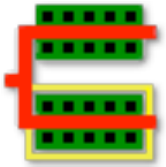


- Selection only possible on objects that are selectable
  - Controlled by LSW (Layer Select Window)
    - Use middle mouse button to toggle selectabilit
    - Unselectable layers greyed out



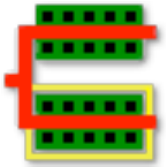
Middle mouse click here to toggle selectability

GUI – Zooming



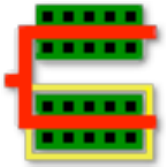
- Zoom by:
  - Bindkeys:
    - Z (zoom in by 2)
    - Shift-Z (zoom out by 2)
    - F (fit window)
    - X (zoom to selected set)
  - Use scroll wheel to zoom in/out
    - % zoomed in/out can be set in Pan/Zoom options form
  - Use right mouse button drag
    - Start at lower point and drag up – zooms in
    - Start at upper point and drag down – zooms out

GUI – Panning



- Pan (move around the viewport) by:
  - Tab (pans to the point under the cursor)
  - View->Pan To Point (pans to the X/Y coord specified)
  - Pan using left/right/up/down keys
    - Pans by default half the viewport width/height
    - Value can be changed in Pan/Zoom options form
  - Pan using middle mouse drag
    - Pans in real time

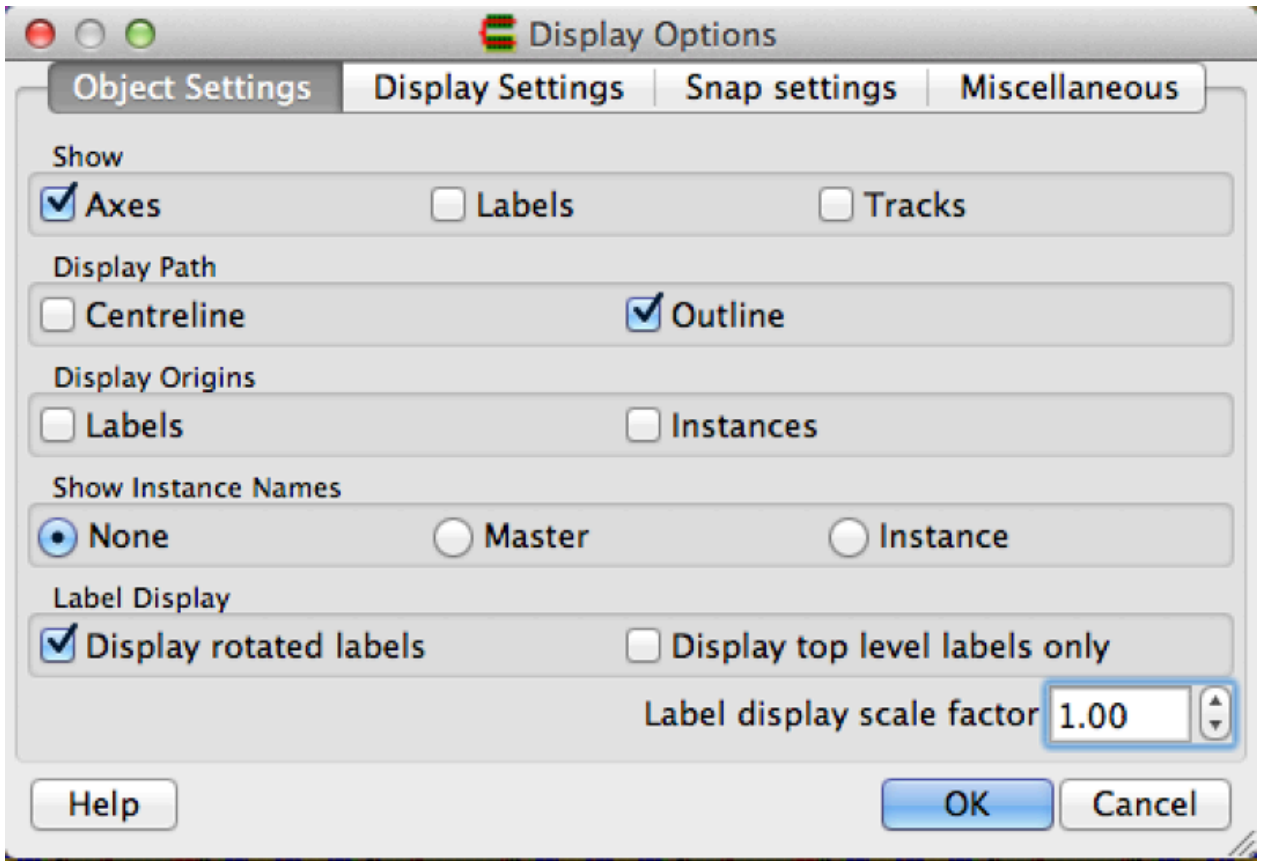
GUI – Options



- Lots of options!
  - Display options form (E bindkey)
    - How some objects are displayed
    - Grid, filtering, snapping etc
  - Selection options form (Shift-E bindkey)
    - Full/Partial select (F4 bindkey)
    - Selection type
    - Dimming
    - Gravity
    - Cursor style
    - Dynamic highlight

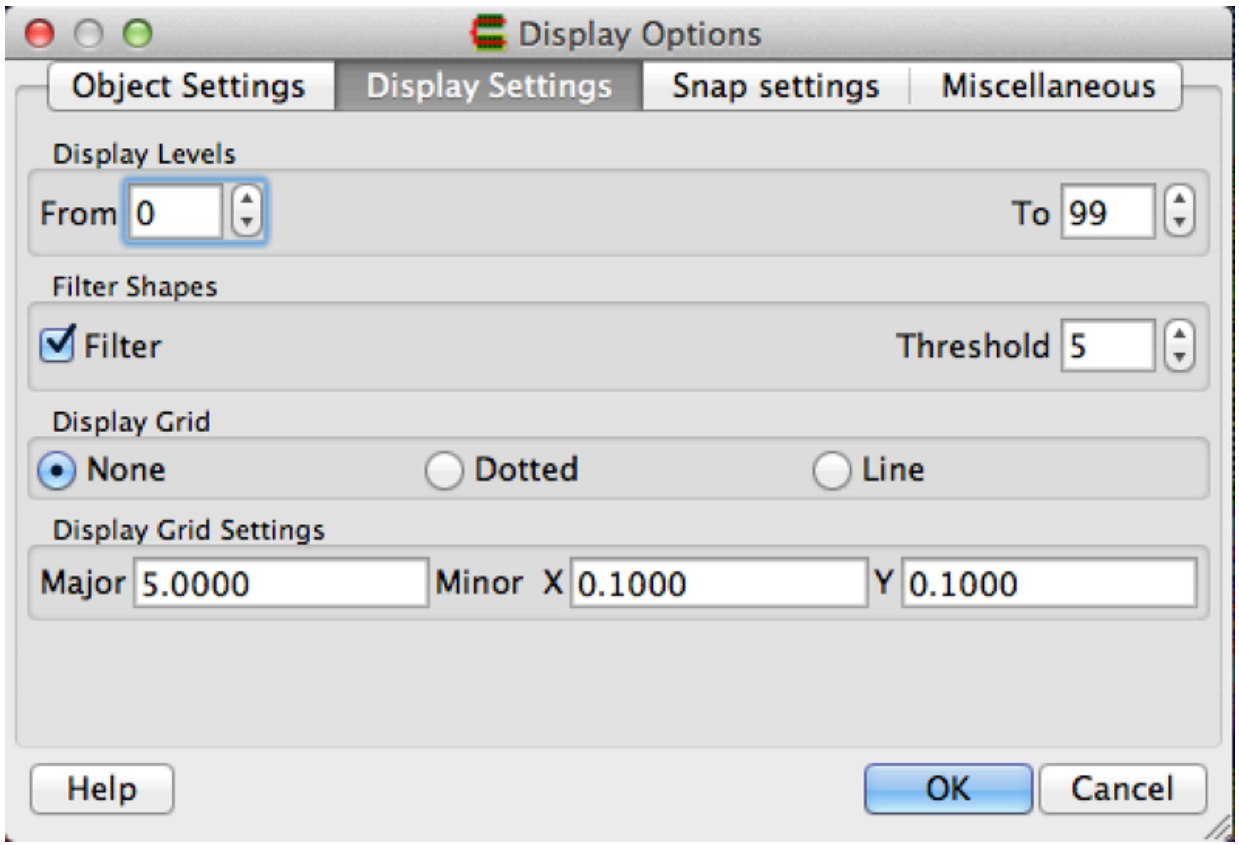
Display Options

- Object Setting control display of objects



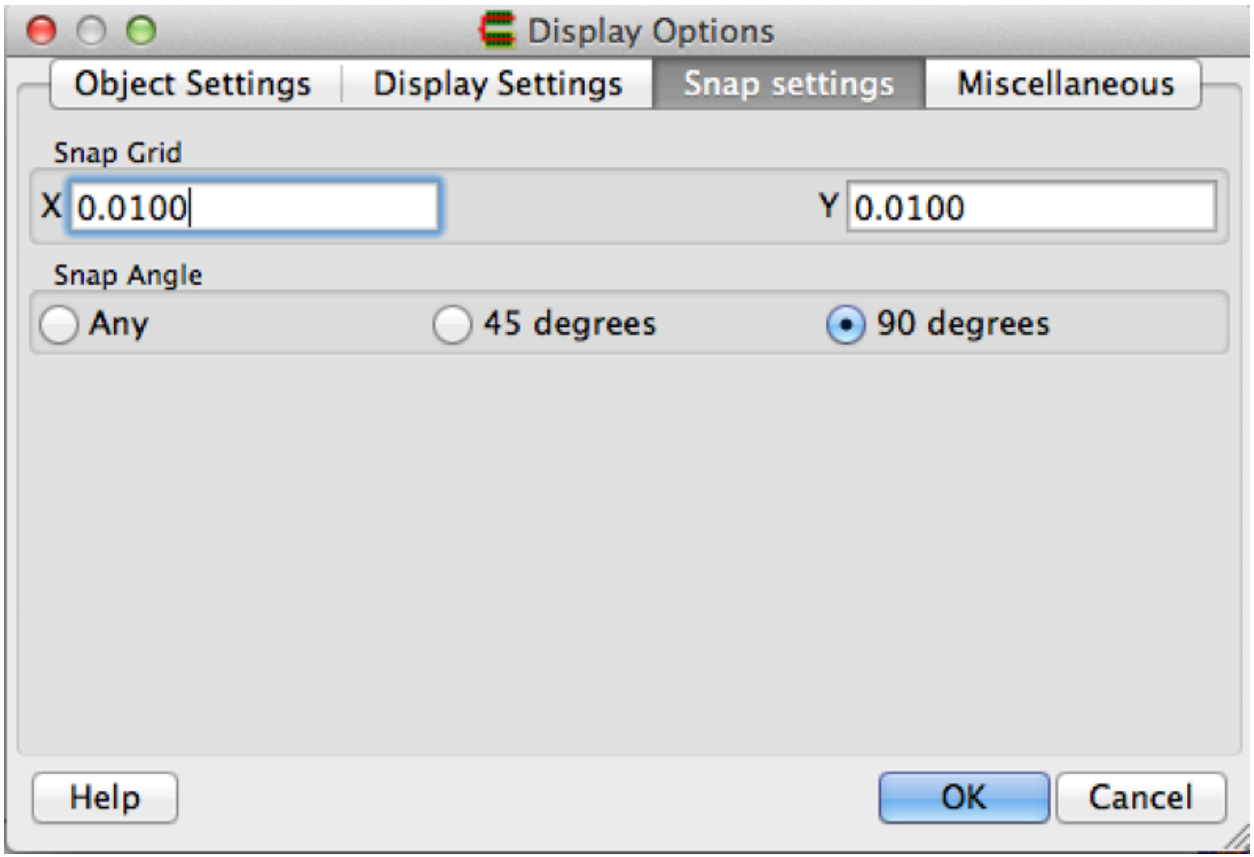
Display Options

- Display settings control drawing options



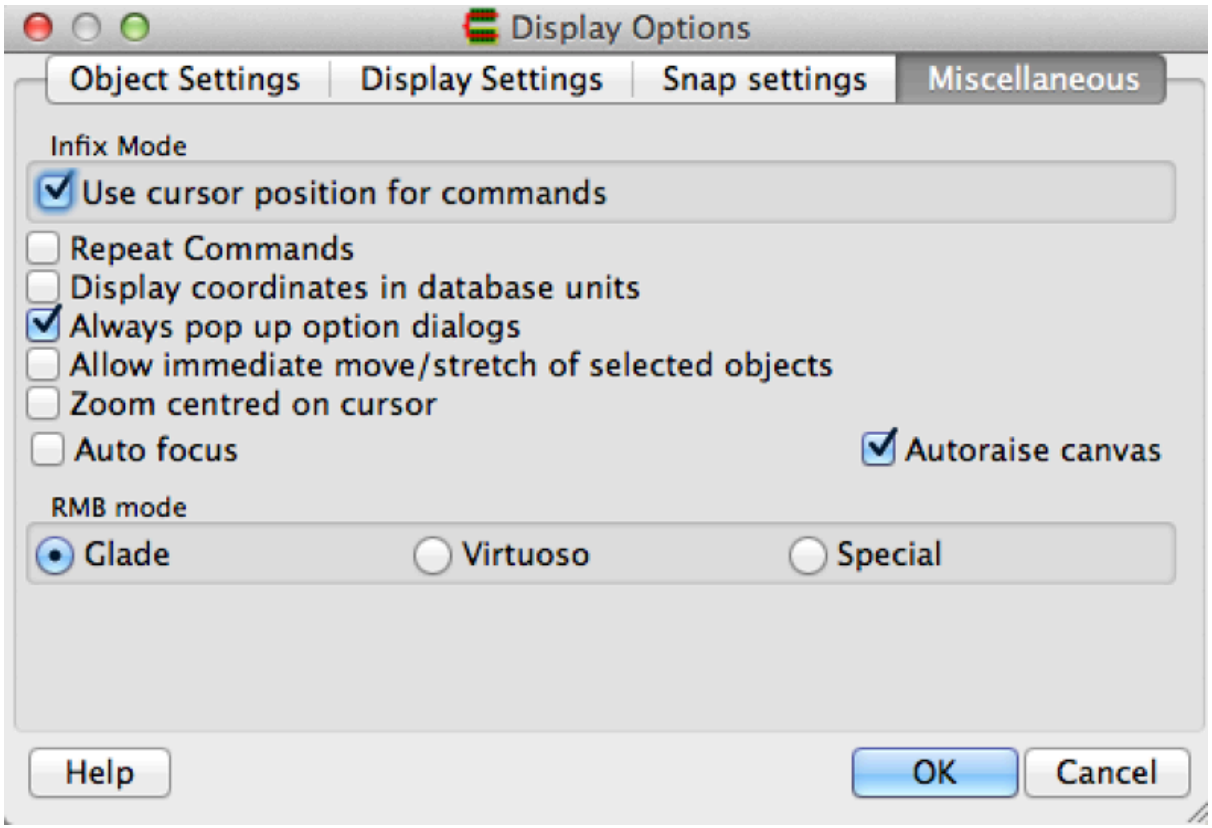
Display Options

- Snap settings set snap grid and angle



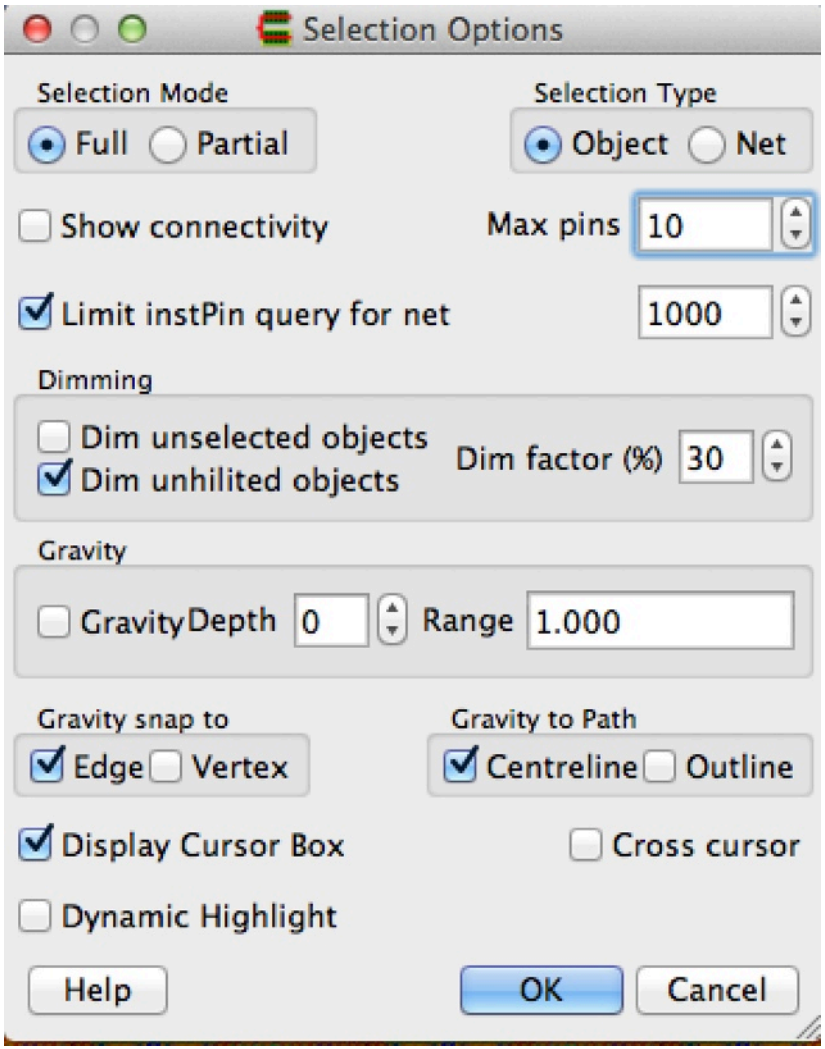
Display Options

- Miscellaneous options...

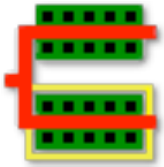


Selection Options

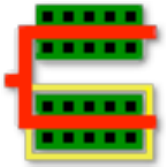
- Selection options dialog controls selectivity, gravity, dimming etc.



Techfile Setup

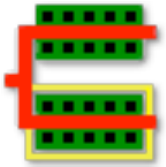


Techfile Setup



- The techfile is an ascii file containing information about the process you are working on.
- Defines
  - Layer names, colours, fill patterns, linestyles
  - Layer functions (routing / via etc)
  - Layer connections (used by net tracer)
  - Width/spacing rules
  - Via definitions
  - MPP (MultiPartPath) rules
- Glade can also read Cadence display.drf/ .tf or Laker .dsp/ .tf techfiles.

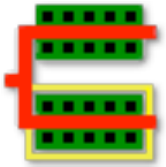
LAYER section



```
//  Name   Purpose  gds_num  gds_dtyp   RGBA    sel?  vis?  fillstyle linestyle
LAYER  psub            drawing   0       0       (217,150,150,128)       t            t
            empty       dashed2 ;
LAYER  nwell           drawing   1       0       (150,150,217,128)       t            t
            empty       dashed2 ;
LAYER  od  drawing   2       0       (217,204,0,128)          t            t            dots_rare    plain ;
LAYER  polyg           drawing   3       0       (255,0,0,128)                        t
            t           zagr1       plain ;
```

- All layers have a name and a purpose
- Layers have GDS number / datatype mappings
- Layer colour as RGBA (alpha)
- Visibility and Selectability
- Fill style (name of stipple pattern)
- Line style (name of line style pattern)

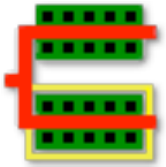
FUNCTION section



```
// Layer Function.
//
FUNCTION   polyg      drawing     ROUTING ;
FUNCTION   cont       drawing     CUT ;
FUNCTION   metal1     drawing     ROUTING ;
FUNCTION   via12      drawing     CUT ;
```

- FUNCTION of the layer/purpose pair can be one of:
  - CUT (a via or contact layer)
  - ROUTING (a layer that can have connectivity)
  - BLOCKAGE (as in LEF/DEF)
  - MASTERSLICE (as in LEF/DEF)
  - PIN (as in LEF/DEF)
  - OVERLAP (as in LEF/DEF)
  - WELL
  - DIFFUSION
  - POLY
  - IMPLANT
  - NONE
- Mainly used in LEF/DEF interface

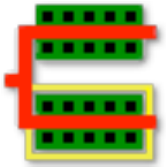
CONNECTIONS section



```
// Layer Connections.
//
CONNECT polyg drawing BY cont drawing TO metal1 drawing ;
CONNECT metal1 drawing BY via12 drawing TO metal2 drawing ;
CONNECT metal2 drawing BY via23 drawing TO metal3 drawing ;
```

- Used to define connectivity for e.g. the Net Tracer
- Can connect layer1 to layer2 by a via layer
- Can also directly connect two layers
  - CONNECT li drawing TO poly drawing ;

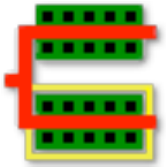
Spacing Rules section



```
// Spacing rules.
//
MINWIDTH nwell  drawing 1.80 ;
MINSPACE nwell  drawing 1.20 ;
MINAREA active drawing 0.45;
MINENC nwell drawing active drawing 0.20 ;
MINOVLP polyg drawing active drawing 0.18 ;
```

- Defines simple layer rules, used by Verify->Check… command

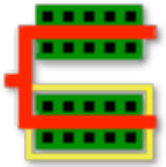
Via rules section



```
// Via rules.
//
VIA od_m1
        metal1 drawing -0.130 -0.130 0.130 0.130
        cont drawing -0.080 -0.080 0.080 0.080
        od drawing -0.150 -0.150 0.150 0.150
```

- Defines via definitions for Create->Via command
- Vias have an upper and lower layer normally of function ROUTING
- Vias have a layer of function CUT
- Can have multiple cut shapes in vias e.g. to create a 2x1 or 1x2 via.

MultiPartPath rules section



```
// MultiPartPath rules
//
MPP nguard LAYER nwell  drawing WIDTH 1.80 BEGEXT 0.90 ENDEXT 0.9 ;
MPP nguard LAYER od     drawing WIDTH 1.18 BEGEXT 0.59 ENDEXT 0.59 ;
MPP nguard LAYER nimp   drawing WIDTH 1.54 BEGEXT 0.77 ENDEXT 0.77 ;
MPP nguard LAYER cont   drawing WIDTH 0.16 BEGEXT -0.08 ENDEXT -0.08 SPACE 0.18 LENGTH 0.16 ;
MPP nguard LAYER metal1 drawing WIDTH 0.60 BEGEXT 0.30 ENDEXT 0.30 ;
```

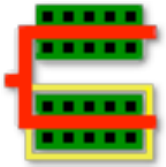
- MPP rules define how a MPP object is created
  - A MPP is like a normal path but with multiple layers and (optionally) contacts between the layers
  - Typically used for e.g. guard rings.
  - Can be edited and moved/stretched just like a normal path.

Stipple pattern section

```
//  Name   Type   Fill pattern
STIPPLE       zagl            STIPPLE
                                0 0 0 1 0 0 0 1 0 0 0 1 0 0 0 1
                                0 0 1 0 0 0 1 0 0 0 1 0 0 0 1 0
                                0 1 0 0 0 1 0 0 0 1 0 0 0 1 0 0
                                1 0 0 0 1 0 0 0 1 0 0 0 1 0 0 0
                                0 0 0 1 0 0 0 1 0 0 0 1 0 0 0 1
                                0 0 1 0 0 0 1 0 0 0 1 0 0 0 1 0
                                0 1 0 0 0 1 0 0 0 1 0 0 0 1 0 0
                                1 0 0 0 1 0 0 0 1 0 0 0 1 0 0 0
                                0 0 0 1 0 0 0 1 0 0 0 1 0 0 0 1
                                0 0 1 0 0 0 1 0 0 0 1 0 0 0 1 0
                                0 1 0 0 0 1 0 0 0 1 0 0 0 1 0 0
                                1 0 0 0 1 0 0 0 1 0 0 0 1 0 0 0
                                0 0 0 1 0 0 0 1 0 0 0 1 0 0 0 1
                                0 0 1 0 0 0 1 0 0 0 1 0 0 0 1 0
                                0 1 0 0 0 1 0 0 0 1 0 0 0 1 0 0
                                1 0 0 0 1 0 0 0 1 0 0 0 1 0 0 0
                                 ;
```

- Stipple patterns can be 4x4, 8x8, 16x16, 32x32
- Can create/edit with the stipple pattern editor

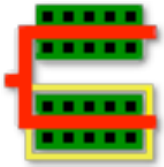
Line style section



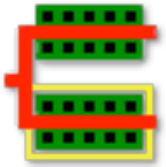
```
//   Name  Width  Style
LINE        plain        0          SOLID ;
LINE        thicksolid   4          SOLID ;
LINE        thick        2          SOLID ;
LINE        dashed2      2          DASH ;
LINE        dotted       0          DOT ;
LINE        dashdot      0          DASHDOT ;
LINE        dashdotdot   0          DASHDOTDOT ;
```

- Line styles used to define the line surround of a shape
- Defines the width of the lines (0 is default width)
- Also defines any pattern
  - – – – – – DASH
  - . . . . . DOT
  - – . – . – . DASHDOT etc.
- Can be set in the stipple pattern editor

Basic Editing

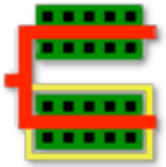


Basic Editing

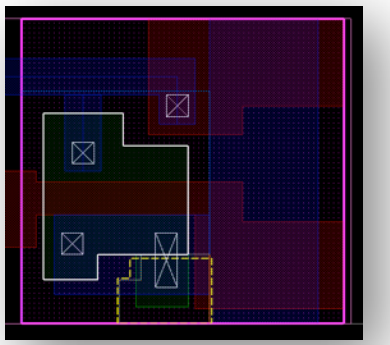


- Many commands work on the Selected Set.
  - Selected Set is a list of objects that have been selected.
  - Selected objects have their outline drawn in the ‘select’ layer (defaults to white).
  - Selection can be made by:
    - LMB (Left mouse button) click on an object
    - Shift+LMB adds to the selected set
    - Ctrl+LMB removes from the selected set
    - LMB drag will select objects wholly contained in the drag rectangle
    - LMB drag works with modifier keys Shift and Ctrl
  - Bindkey Ctrl-A selects all objects, Ctrl-D deselects all selected objects.

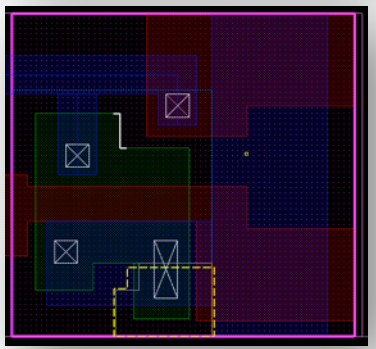
Selection – Continued



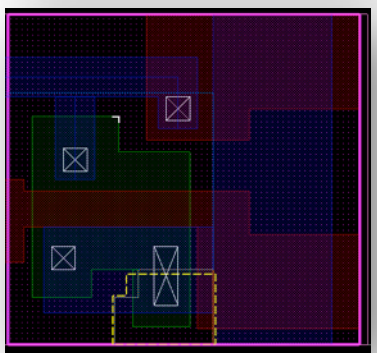
- Shapes can be selected according to the current selection mode
  - FULL : selects the whole shape
  - PARTIAL : selects either an edge of a shape or a vertex of a shape. Vertices will be selected if the cursor is within 10% of the nearest edge length to the vertex.
- Toggle between full/partial mode with the F4 bindkey
- Other objects can also be selected
  - Instances/arrays
  - Text labels
  - Etc.



Full

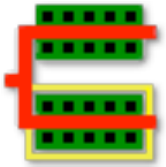


Partial (Edge)



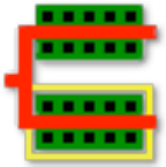
Partial (Vertex)

Selection – Continued



- Selection can be controlled by:
  - Making layers unselectable/invisible (can't select invisible layers)
  - Making the Instance layer unselectable prevents selecting instances or arrays
  - When there are multiple object under the cursor, if the cursor does not move and the left mouse button click repeated, objects with edges near the cursor are cycled through.
- Number of objects selected is shown in the status bar
- Query selected objects using the Q bindkey (Edit->Query)

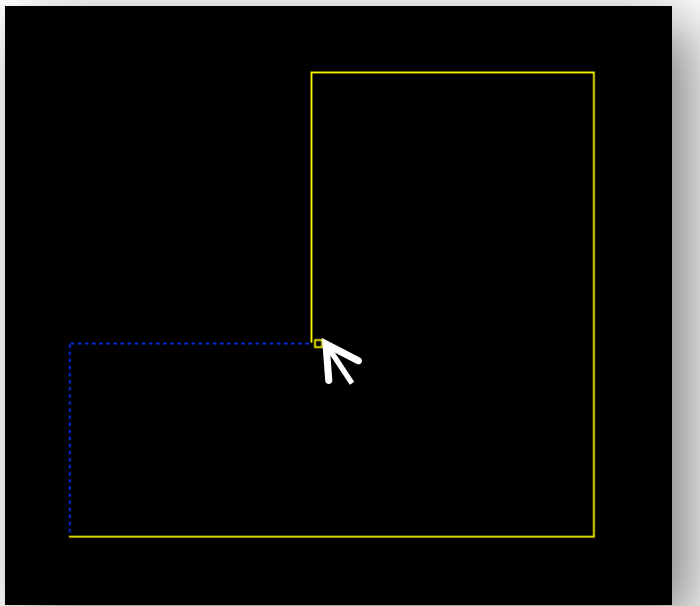
Basic Editing – Creating shapes

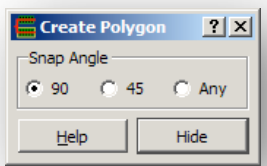


- Select the layer you want to draw on by LMB clicking on it in the LSW
- Use either menu, toolbar or shorcut key to create objects. Shortcut key has the advantage of being able to work in infix mode.
  - Infix mode uses current cursor position for the first coordinate. Set using Display Options form.
- During shape creation an options form can be displayed. It can be toggled between shown/hidden using the F3 key.
- For paths and polygons, finish by hitting Enter key, or by double clicking. There is no need to ‘close’ a polygon.

Create Polygon

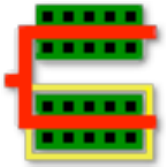
- Bindkey Shift+P starts a polygon
- Edges entered are drawn in yellow
- ‘Close edges’ are drawn dotted blue. These are edges that will be automatically added to close the polygon, if Enter is pressed or a double click at the current point.
- There is no need to enter the last point coincident with the start point as a result.





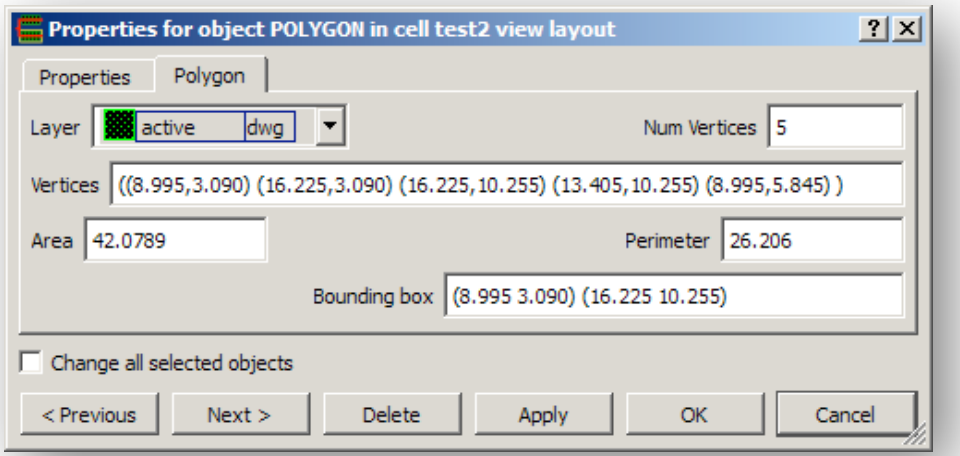
Create Polygon options form

Create Polygon – continued



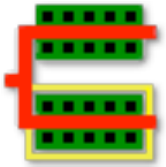
- Poygon points entered are snapped to the snap grid (defined in the Display Options form) and according to the snap angle (Manhattan / Diagonal / Any Angle).
- While entering a polygon, you can undo the last point entered by using the Backspace key.
- Colinear points entered (points with exactly the same X and Y values) are automatically removed
- Self-intersecting polygons are not allowed. An error will be given and the polygon will not be created.

Create Polygon – continued

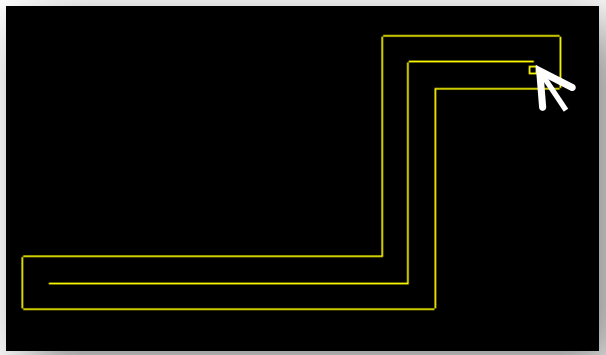


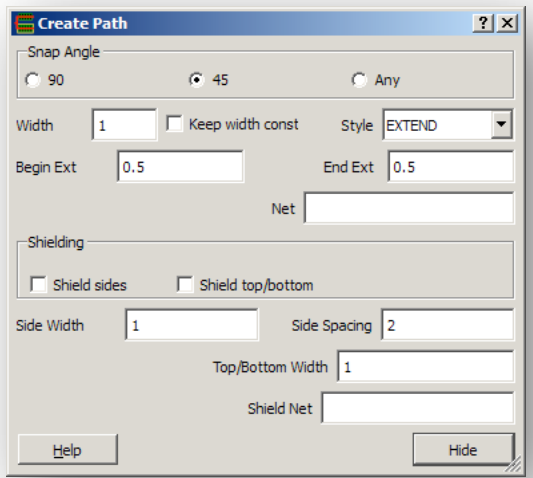
- Polygons can be selected and queried to make modifications
  - Change the layer
  - Edit vertices textually
- Query form shows other useful info like area, perimeter, bounding box.

Create Path

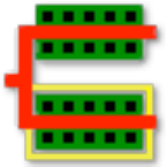


- Create Path
  - Creates a path. Width, extension, style and shielding can be defined in the options form
  - With layer function and vias defined in techfile, the U key will add a via and path creation continues on the next routing layer up. The D key will add a via and path creation continues on the next routing layer down.
- End a path with Enter or double click. Backup to undo last point entered.

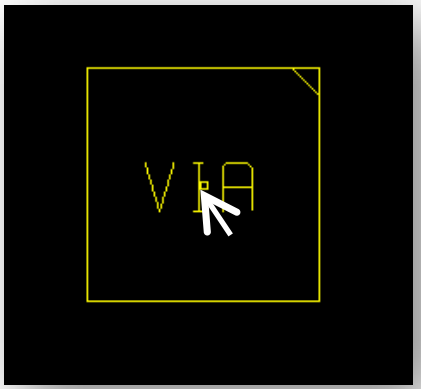


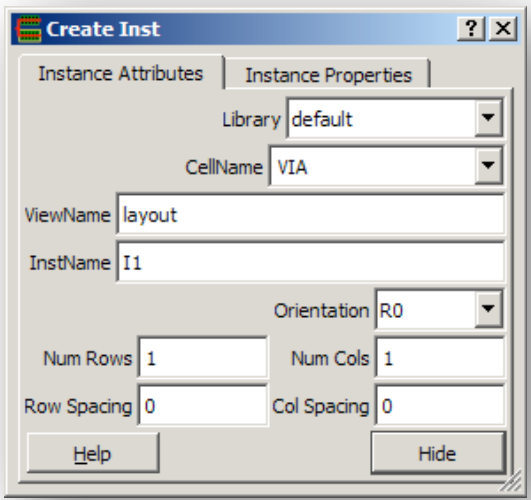


Create Instance



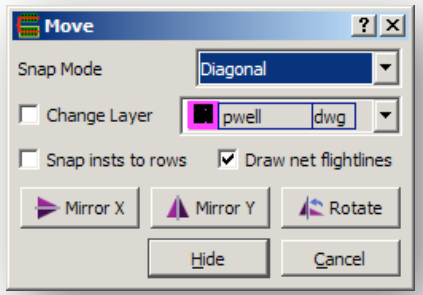
- Create Instance shows an outline of the instance, and its cell name.
- Use the option form to define innstance's library/ cell/view names.
- Arrays can be generated by setting number of rows/cols and spacing
- Instances current can have orientations R0, R90, R180, R270, MX, MXR90, MY, MYR90

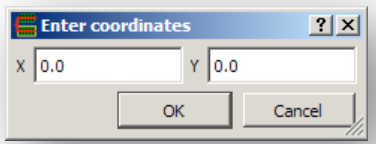




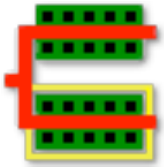
Editing Commands

- Move
  - Select object(s) and move. First point is the reference point, second point defines the delta for the move.
  - To move a fixed distance, Use Edit->Move, then F5. Enter X/Y coords as 0,0. Then F5 again and enter delta X,Y required.
  - Move can rotate/mirror shapes
    - Use r, x, y bindkeys during move
  - Move can change the layer of the shape

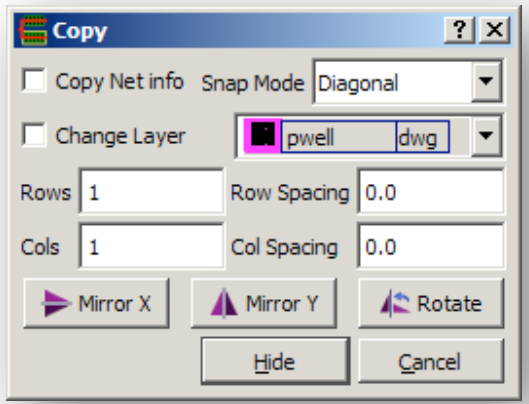




Editing Commands

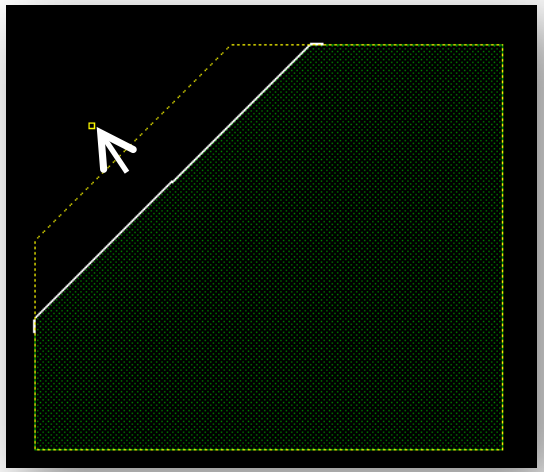


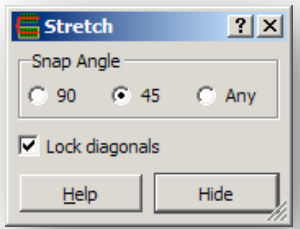
- Copy
  - Similar to move: select the object(s) to copy, then enter the reference coordinate and the destination coordinate.
  - Copy can copy the object(s) by array (not an instance array)
  - Copy can rotate/mirror during copy
    - Use r/x/y bindkeys or option form buttons
  - Copy can change layer of shape(s). All shapes will be changed to the target layer.



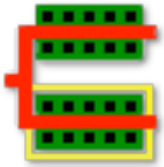
Editing Commands

- Stretch
  - Select edge in partial select mode
  - Stretch will show as yellow dotted line
  - Stretch options form allows ‘locking’ diagonals. Otherwise stretching a manhattan edge of the object may change the adjacent diagonal to any angle.
  - Stretch works on selected vertices as well
  - It’s not a good idea to try and stretch multiple edges/vertices at same time unless they belong to different shapes!
  - Stretch will move shapes that have been selected in full select mode

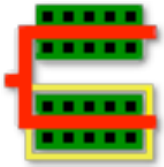




PCells

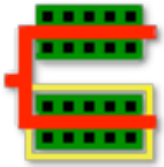


PCells



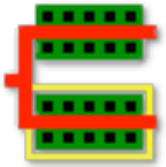
- PCells are Parameterised Cells
  - Instances of a PCell can be different depending on the properties of the instance. For example a MOS device can have W, L parameterisable.
- Benefits of PCells
  - Designed once – can be used in many variants
  - Reduce DRC errors
  - Faster layout times
- Glade PCells are NOT the same as Cadence Pcells or Synopsys PyCells!
  - Other vendor's PCells use proprietary languages (e.g. Skill) and/or plugins (PyCells)

PCells



- Glade PCells written in Python
  - Code can be debugged using print statements or using a Python debugger e.g. ActiveState Komodo.
  - PCell files must be kept in a directory in user's PYTHONPATH
  - Bytecode compiled PCells can be used (.pyc files) to distribute unreadable PCells.
- PCell code creates a cell called a SuperMaster. This cell is used to create instances of PCells. When an instance of the supermaster is created, a SubMaster cell is also created using the unique properties of the instance.
- SubMaster cells are not visible in the library browser – they are managed by the PCell subsystem.

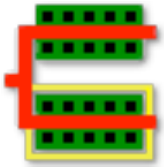
PCells



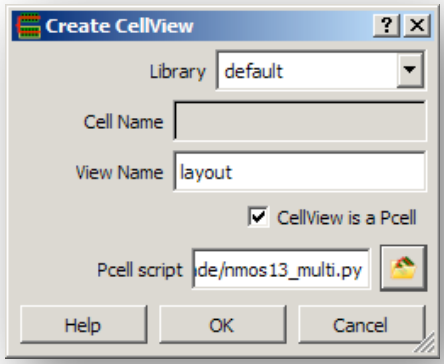
```
# Import the db wrappers
from ui import *

# The entry point. The function name *must* match the filename.
def nmos13_multi(cv, w=1.1, l=0.13, m=1) :
        lib = cv.lib()
        dbu = lib.dbuPerUU()
        width = int(w * dbu)
        length = int(l * dbu)
        fingers = int(m)
```

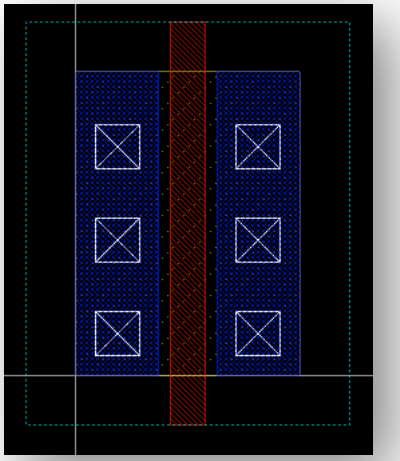
- An example: nmos13_multi.py (only the first few lines shown)
  - We define a function called nmos13_multi.
  - It has 4 arguments
    - 1st is always the cellView that the PCell instance is created in.
    - Remainder are the PCell parameters. They *must* have default values specified, so we can build the PCell if the properties have not yet been specified on the PCell instance

PCells



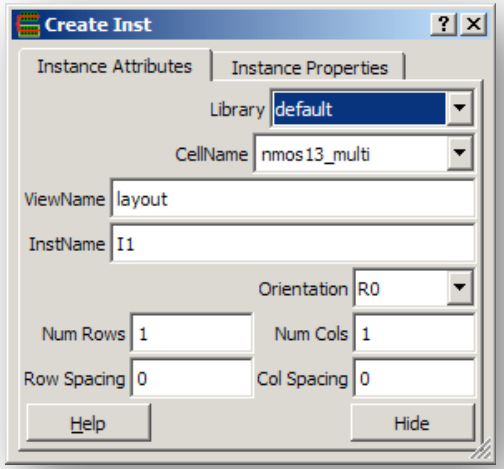
- Create a Pcell supermaster manually using the New Cell command.
- This creates the SuperMaster cell nmos13_multi
  - Do not edit this cell!
  - It is there for 2 reasons
    - To help you debug writing pcells
    - To allow the Create Instance command to reference it. An instance of its submaster is then created and used.

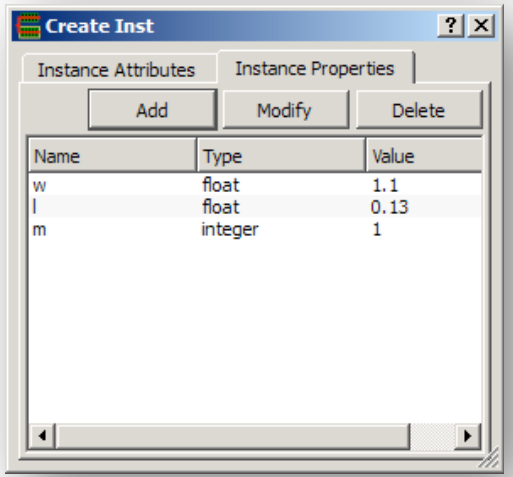




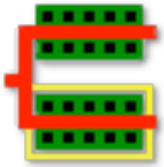
PCells

- To create a PCell instance in your layout:
  - Use the Create Instance command
  - Edit the Instance Properties tab on the options form to set the PCell parameters.
- The Properties tab shows the default Pcell parameters; you can modify these to the values you want.

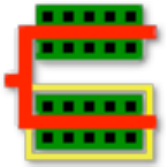




DRC / extraction / LVS

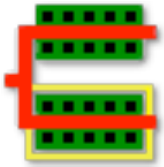


DRC



- Design Rule Checking (DRC) in Glade uses Python scripting
  - DRC rules files are scripts calling Python functions
- Boolean operations and DRC checks are performed by a scanline algorithm (Bentley-Ottman)
  - Not suited for very large designs as layers are processed flat
  - Maybe tiling in the future can address this
- DRC commands for common operations
  - Boolean ops to create derived layers e.g. gate = poly AND active
  - Connectivity extraction for samenet/diffnet rules
  - DRC commands to check e.g. width, spacing, overlap etc.

A simple DRC example

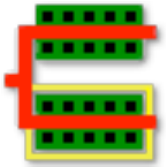


A simple DRC script might look like this:

```
from ui import *
cv = getEditCellView()
geomBegin(cv)
active = geomGetShapes("active", "drawing")
poly = geomGetShapes("poly", "drawing")
gate = geomAnd(active, poly)
geomWidth(gate, 0.18)
geomEnd()
```

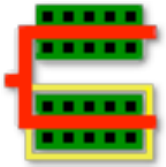
1. First we import the ui module so we can access the geom… Python functions
2. Next we get the cellView we want to check – in this case the current open cellView
3. We initialise the geometry engine with geomBegin, which takes a single arg – the cellView
4. We read in data on layers we wish to use
5. We perform some boolean operations to create derived layers
6. We perform a DRC check on the derived layer (in this case checking the width is not less than 0.18um)
7. Lastly we exit the geometry engine to free memory.

Importing layers



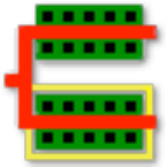
- geomGetShapes() is used to get all the shapes on a given layer
  - By default it flattens the hierarchy of the cellview
  - It creates an **edge file**. This is a temporary disk file in compact binary format that stores all shapes of a given layer as a set of edges for each polygon.
  - geomGetShapes() also merges shapes and orders polygon vertices as counterclockwise (internally the geometry engine stores polygons as counterclockwise, and holes as clockwise vertices).
  - The resulting edge file can be imagined as a '**layer**'. Layers generated from geomGetShapes() are known as '**original**' layers. Layers generated by subsequent boolean or selection functions are know as '**derived**' layers.

Boolean operations



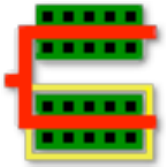
- There are many operations to process shapes on layers:
  - geomMerge() : performs single layer OR
  - geomOr() : performs two layer OR
  - geomAnd() : performs two layer AND
  - geomNot() : creates the inverse of the layer data
  - geomAndNot() : subtracts a layer from another layer (the inverse of geomAnd())
  - geomXor() : performs the XOR of two layers
  - geomSize() : up or down sizes a layer
  - geomTrapezoid(): converts a layer's polygons to trapezoids

Selection operations



- Similarly there are operations to select shapes based on some criteria:
  - geomTouching()
  - geomOverlapping()
  - geomInside()
  - geomOutside()
  - geomAvoiding()
  - geomButting()
  - geomCoincident()
  - geomHoles()
  - geomNoHoles()
  - geomGetTexted()

Labelling shapes

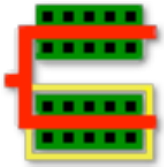


- geomLabel() will assign net names to shapes
  - Typical use is for extraction to label known nets

```
geomLabel(metal1, "m1txt", "drawing")
```

- The above uses text on the ‘m1txt drawing’ layer purpose pair to assign net names to shapes on metal1
  - Text origin must overlap the shape
- Labelled shapes will be used in connectivity extraction as starting points for connectivity tracing.

Connectivity extraction

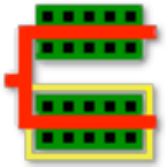


- The geometry engine can extract connectivity from shapes, using the geomConnect() function:

```
geomConnect( [
        [cont, active, poly, metal1],
        [via1, metal1, metal2]
    ])
```

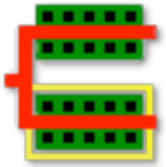
- In this example we connect the active, poly or metal1 shapes to each other via the ‘cont’ layer. Similarly for metal1/metal2 by the ‘via’ layer.
- geomConnect will use net names from e.g. geomLabel() else it will assign generated names (n1, n2….) to connected shapes as net names.
- geomConnect will warn of shorts e.g if a shape labelled ‘vdd’ is eventually connected to another shape with a different label e.g. ‘gnd’. It will report the coordinates and layers of the shorts.

DRC checking commands

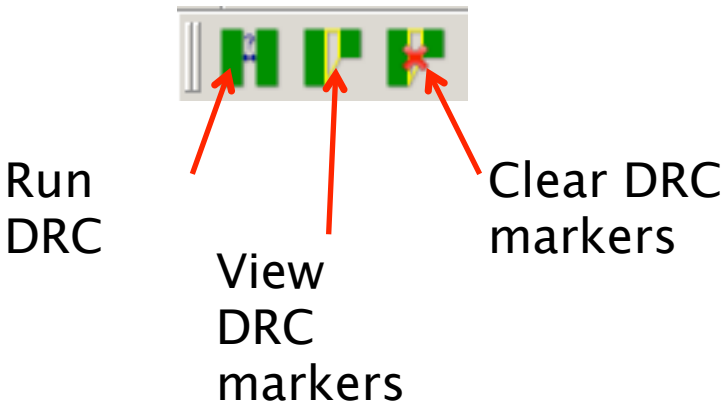


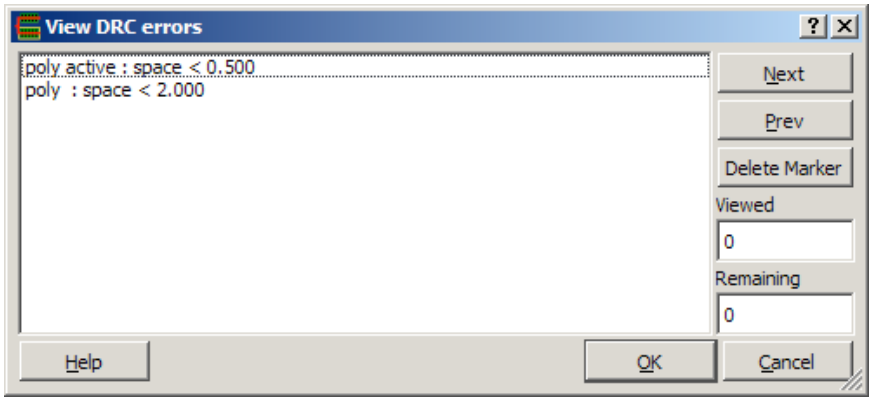
- DRC check commands check for specific rules:
  - geomWidth() : minimum width of a shape
  - geomSpace() : minimum space of shapes on 1 or 2 layers
  - geomNotch(): minimum space between edges of a shape
  - geomArea() : minimum / maximum area of a shape
  - geomEnclose() : enclosure of one shape by another
  - geomExtension() : extension of shape on one layer beyond shape on other layer's edge
  - geomOverlap() : minimum overlap of shape on one layer by shape on other layer.
- DRC commands generate markers on original layout that can be viewed by Verify->DRC->View Errors…
- DRC command also generate edge files that can be used:
  - errorLayer = geomWidth(metal1, 0.4)
    - Generates a derived layer 'errorLayer' with shapes that are the violations from the geomWidth() command.

Running DRC

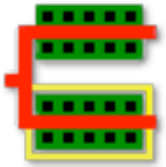


- Env var GLADE_DRC_FILE sets DRC rules file in 'Verify->DRC->Run' dialog
- Env var GLADE_DRC_WORK_DIR sets location of temporary files.
- Use Verify->DRC->View Errors… to display DRC marker viewer
  - Left click on rule will zoom in on first error for that rule
  - #Viewed / #Remaining show errors viewed and remaining to view.



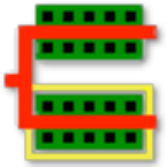


Extraction



- Extraction (Verify->Extract…) uses similar boolean processing as DRC.
  - Note that forming connectivity is optional for DRC, but mandatory for extraction!
- Extraction requires saveInterconnect() command to save connected shapes into extracted view.
  - This allows extraction of devices e.g. MOS, BJT, resistor etc.
- Devices are extracted using extract… commands.
  - Each needs a 'recognition region' i.e. a layer that uniquely identifies the type of device (e.g. gate = geomAnd(poly, active))
- Extraction uses PCells to form devices with a polygon outline created from the recognition region.
- Extracted view can be used for LVS or for netlisting (File->Export CDL…

saveInterconnect



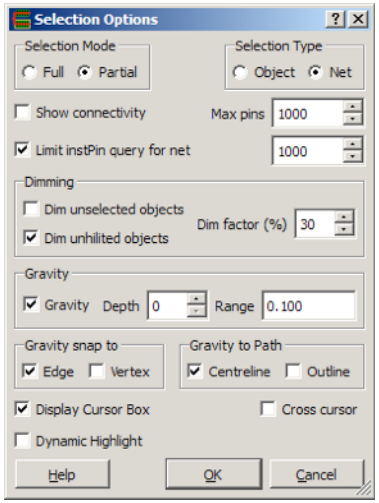
- saveInterconnect() is used to save shapes with connectivity to the extracted view when running extraction.

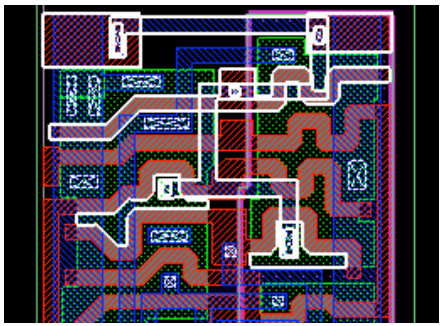
```
saveInterconnect([
        [psub, "psub"],
                nwell,
                [ndiff, "od"],
                [pdiff, "od"],
                [polyg, "polyg"],
                cont,
                metal1,
                via12,
                metal2])
```

- Save layers **must** be derived from geomConnect()
- Any derived layer **must** be saved to a named layer
  - The layer will be created if not defined by the techfile

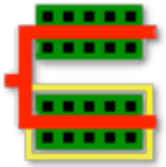
Running Extraction

- Env var GLADE_EXT_FILE sets extraction rules file in 'Verify->DRC->Run' dialog
- Env var GLADE_DRC_WORK_DIR sets location of temporary files
- Set 'Selection Type' to 'Net' to select all shapes on a net in the extracted view.
- Extraction will report any shorts found.

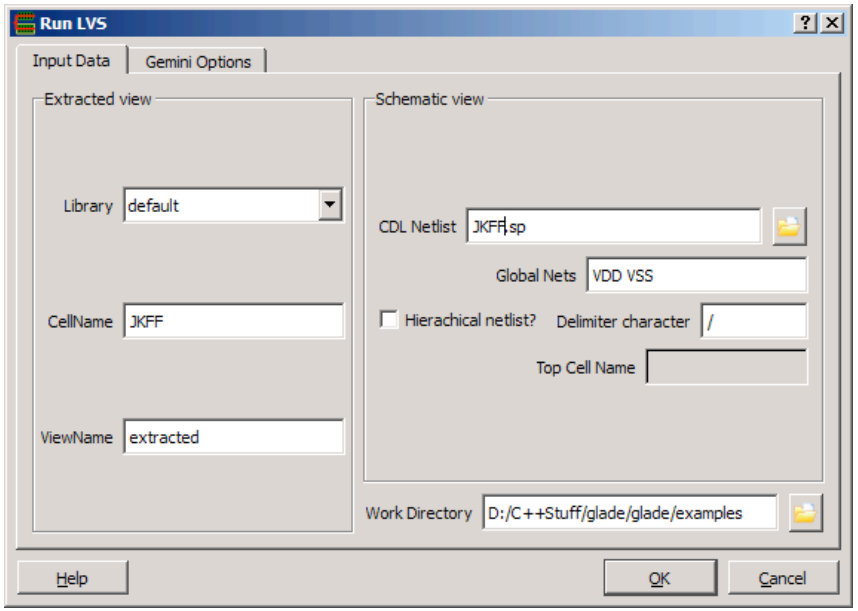




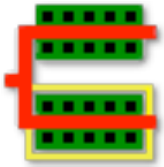
Running LVS



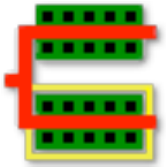
- Prerequisites:
  - Extraction must have completed successfully with no errors.
  - A flat or hierarchical CDL/Spice netlist must be available.
  - Optionally, layout should have labels for primary IOs and power/ground (helps Gemini) or an equivalence file (matches layout net names to CDL/Spice net names)
  - Env var GLADE_NETLIST_FILE can be set to CDL netlist file name to preset the LVS form
- For Gemini options, see Gemini documentation
  - (www-scf.usc.edu/~ee577/**manual**/**gemini**_man.ps)



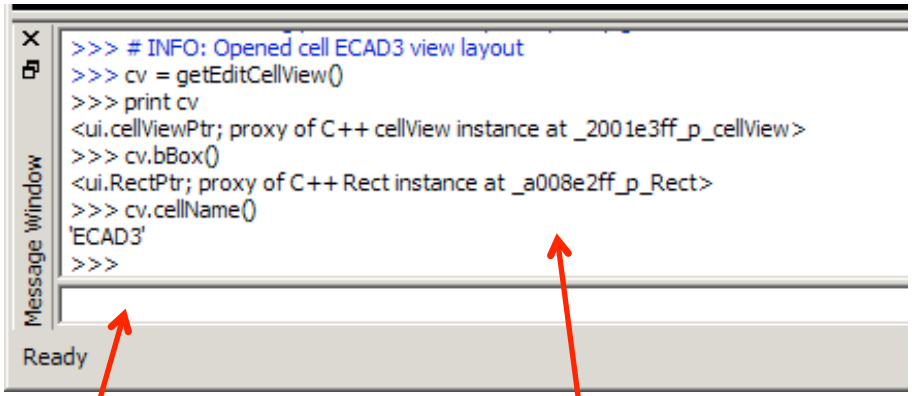
Python programming



Python Programming



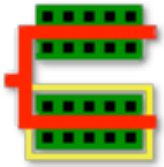
- Glade has an embedded Python interpreter.
- Database, GUI and geometry processing (DRC/Extract/LVS) C++ code is wrapped using SWIG to give Python callable functions.
- Python code can be executed using File->Run Script… or typed into the command line.
- Python cmd line supports history (use up/down arrows) and standard QLineEdit ctrl character sequences.



Python command line

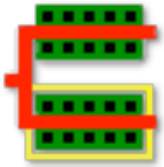
Python output to the message window

Python Programming



- Python looks for modules in PYTHONPATH env var!
  - Glade adds to this according to platform
    - %GLADE_HOME% (Windows & Mac)
    - $GLADE_HOME/bin (Linux)
  - You may want to add paths to e.g. PCell libraries.
- Python distribution libraries are at
  - %GLADE_HOME%/Python27 (WIN32/64 – contains libs and DLLs)
  - $PYTHONHOME (Linux/Mac) OR:
  - /usr/local/lib/python2.6 (Linux)
  - /usr/lib/python2.7 (Mac)

Example Python code



```
from ui import *                                        # Import the swig wrappers
ui = cvar.uiptr                                         # Get the pointer to the ui class
lib = library("fred")                                   # Create a library
cv = lib.dbOpenCellView("test", "layout", 'w')          # Create a new cellView in the library
tech = lib.tech()                                       # Get the tech class associated with the library
layer = tech.createLayer("layer1", "drawing")           # Create a layer we can draw on
ui.openCellView(lib.libName(), cv.cellName(), "layout") # Open the cellView in the gui so we can see it
nPoints=4
x = intarray(nPoints)                                   # Create an array of 4 points
y = intarray(nPoints)
x[0] = 1000 y[0] = 1000
x[1] = 6000 y[1] = 1000
x[2] = 6000 y[2] = 3000
x[3] = 1000 y[3] = 3000
poly = cv.dbCreatePolygon(x, y, nPoints, layer, 1)      # Create a polygon (default dbu/micron is 1000)
angle = 30.0
origin = Point(1000,1000)
trans = dbTransform64(angle, origin)                    # Create the transform
poly.transform(trans)
cv.update()                                             # Update the cellView (after object(s) are created)
ui.winRedraw()                                          # Redraw the gui
```