Santana Technology File Reference Manual

Version 2022.06-SP2, December 2022

SYNOPSYS®

Copyright and Proprietary Information Notice

Destination Control Statement

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

Disclaimer

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

Trademarks

Synopsys and certain Synopsys product names are trademarks of Synopsys, as set forth at https://www.synopsys.com/company/legal/trademarks-brands.html.
All other product or company names may be trademarks of their respective owners.

Free and Open-Source Licensing Notices

If applicable, Free and Open-Source Software (FOSS) licensing notices are available in the product installation.

Third-Party Links

Any links to third-party websites included in this document are for your convenience only. Synopsys does not endorse and is not responsible for such websites and their practices, including privacy practices, availability, and content.

www.synopsys.com

Feedback

Contents

1. **Introduction** 5

2. **Summary Description** 6

3. **Header Group Structure** 10
   techId 10
   viewTypeUnits 10
   mfgGridResolution 11

4. **Layers Group Structure** 13
   layerMapping 13
   maskNumbers 14
   layerMaterials 15
   purposeMapping 16
   derivedLayers 16
   viaLayers 17
   connectivity 18

5. **Physical Rules Group Structure** 19
   spacingRules 20
   orderedSpacingRules 23
   deviceContext 25
   ruleset 26
   characterizationRules 29
   oxideDefinitions 29
   mosfetDefinitions 30

6. **Pre-Defined Layer and Purpose Names** 33

7. **Using Santana Technology Files with PyCell Studio** 38

Hybrid Technology File / OpenAccess (OA) Tech Database Mode 40

1

Introduction

This reference manual documents the format of the Santana technology file. This ASCII text file contains all of the process specific information concerning units, manufacturing grid sizes, layers, mask data, physical design rules and electrical design rules.

The Santana technology file can be created in a number of different ways:

- Use the information contained in this reference manual to create this technology file using a standard text editor.
- Convert an existing Cadence DF II technology file into the Santana technology file format, using a conversion utility which is supplied by Synopsys.
- Convert an existing DRC design rule deck into this Santana technology file format, using another conversion utility provided by Synopsys.
- Use a combination of these approaches, where some sections of this Santana technology file are created through conversion utilities, and other sections are created using a text editor.

Note that in some cases, it may be necessary to modify the data which is generated by these conversion utilities. Thus, a typical approach to creating the Santana technology file would involve using a combination of conversion utilities and text editing.

2

Summary Description

The Santana technology file is defined using the standard syntax of `Symbolic expressions` (`S-expressions`). Parentheses are used to group data into different sections; each section can then be viewed as a stand-alone element. The Santana technology file has the following pre-defined groups:

HEADER group

- techId
- viewTypeUnits
- mfgGridResolution

LAYERS group

- layerMapping
- purposeMapping
- maskNumbers
- layerMaterials
- derivedLayers
- viaLayers
- connectivity

PHYSICAL RULES group

- spacingRules
- orderedSpacingRules
- deviceContext
- ruleset

ELECTRICAL RULES group

- characterizationRules
- oxideDefinitions

mosfetDefinitions

These groups can be very briefly described as follows:

- HEADER group

  techId

  Defines name, version and revision information

  viewTypeUnits

  Defines user units and database units for different view types, such as mask layout; typical mask layout values are 1000 database units per 1 micron user unit.

  mfgGridResolution

  Defines default manufacturing grid resolution, as well as per-layer grid resolution

- LAYERS group

  layerMapping

  Defines all layers and layer numbers; layer number must be unique

  purposeMapping

  Defines purpose names and purpose numbers; purpose numbers must be unique

  maskNumbers

  Defines mask numbers for all layers which are mask layers

  layerMaterials

  Defines the type of material used by each mask layer

  derivedLayers

  Defines any derived layers for this technology

  viaLayers

  Defines all via layers, including the layers which are being connected

  Connectivity

  Defines connectivity which should exist between different layers

- PHYSICAL RULES group

spacingRules

Specifies design rule values for no layer, one layer and two layer design rules, where the order of the layers in two layer rules does not matter, such as in minimum spacing rules.

orderedSpacingRules

Specifies design rule values for two layer design rules, where the order of the layers in two layer rules does matter, such as minimum extension.

deviceContext

Specifies design rules to be used when constructing particular devices.

ruleset

Specifies sets of design rules which can be activated and used during PyCell construction, or when running the DRC program.

- ELECTRICAL RULES group

  characterizationRules

  Specifies electrical rule value for none, one and two layer electrical design rules

  oxideDefinitions

  Defines any electrical values which depend upon the type of oxide (thin or thick)

  mosfetDefinitions

  Defines electrical values for various MOSFET transistor model parameters

Note that each of these section names (such as `techId` or `layerMapping`) can be thought of as a keyword which defines the start of a new section. This simple `S expression` syntax makes it easy to analyze the structure of a Santana technology file. In fact, this `S expression` syntax can be thought of as a simple language, made up of pre-defined keywords (the section names listed above), matching left and right parentheses which are used to group data in each section, and comments, which are indicated by preceding any single line of text with a semicolon. Thus, the complete technology file can be constructed using this `S-expression` language.

Many of the numerical values present in this technology file are specified using floating-point numbers, which are defined using user units. These user unit values are internally converted into internal database units when the technology file is used with the PyCell Studio™ software product. In the case of numerical values used for specifying area values, square user units are assumed. In other cases, such as the specification of

electrical values (such as power), no units are assumed, so that no conversion between user units and database units will take place.

Once the Santana technology file has been completed, then it needs to be compiled from a text file into a technology library, using the `cngenlib` utility program. If there are any errors in the Santana text technology file, they will be identified when this `cngenlib` utility program is run. These errors should be corrected and the `cngenlib` program should then be rerun. Note that the `--force_binary_techfile` option can be used to generate a binary format technology file.

Note that in addition to compiling this technology text file into a technology library, this `cngenlib` utility program will also generate the different Contact library cells for this technology file. These Contact library cells will be constructed using the relevant design rule information stored in the technology file.

After the technology library has been generated, the Python parameterized cell programmer can then access all of this technology information through the Python API. The Tech class provides a large number of methods to access this technology information.

3

Header Group Structure

The header group for the Santana technology file contains all of the basic information about the technology file and the units which are used by the process technology described in this technology file. This header group is composed of three different sections: the **techId** section, the **viewTypeUnits** section and the **mfgGridResolution** section. Each of these sections is described as follows:

techId

Description: The **techId** section is used to provide a name which describes the technology being described in this technology file, along with a version number and a revision number. This information can be directly accessed through the Python API, using the name(), version(), revision() and id() methods for the Tech class. The name used to identify this technology file can be any string value, while the version number and revision number can be any integer value.

Example:

```
techId(
   ( name     "Fictional 0.13um" )
   ( version 4 )
   ( revision 0)
); techId
```

viewTypeUnits

Description: The **viewTypeUnits** section is used to specify the different user units and database units which can be used with this technology file. Note that all database unit information is stored using integer values; the conversion factor is also specified as an integer value. This conversion factor specifies the number of database units which should be used for each user unit. Each entry in this section consists of three items: the view type, the user units to be used with this view type, and the number of database units for each user unit, which is the database unit to user unit conversion factor which should be used. For example, there are typically 1000 database units per micron user unit for the maskLayout view type. However, these numbers should be defined as appropriate for the process technology being described in this technology file.

Feedback

The view type field must be one of the following pre-defined values: maskLayout, schematic, schematicSymbol or netlist. The user unit field must be one of the following pre-defined values: nanometer, micron, centimeter, meter, mil or inch. Finally, the database to user unit conversion factor field should be a positive integer value.

Example:

```
viewTypeUnits(
  ;( view-type       user-unit   DBU-per-UU )
  ;( --------------- ----------  ---------- )
   ( maskLayout       micron      2000       )
   ( schematic        inch        160        )
   ( schematicSymbol  inch        160        )
   ( netlist          inch        160        )
); viewTypeUnits
```

mfgGridResolution

Description: The **mfgGridResolution** section is used to specify the default manufacturing grid resolution which should be used to define the space between grid points when a chip is manufactured. This is the default grid value which should be used, unless it is overridden by a layer specific manufacturing grid resolution value. For some processes, layers such as nwell and pwell well layers use a coarser manufacturing grid than critical layers such as the contact layer, in order to control mask costs. In such cases, layer-specific manufacturing grid resolution values can be defined.

This manufacturing grid resolution field either consists of a single floating-point number, in user units, which will define the default manufacturing grid resolution value. Otherwise, this field can consist of a layer name and a grid resolution value. This layer name should be the name of a layer which is defined in this technology file, while the resolution value should be a floating point number defined in user units, which will define the manufacturing grid resolution value for the specified layer.

Examples:

; defines default manufacturing grid resolution

```
mfgGridResolution(
   ( 0.005 )
); mfgGridResolution
```

; defines default manufacturing grid resolution,

; along with individual layer grid resolutions

```
mfgGridResolution(
  ;( layer-name   grid )
  ;( -----------  -----)
   (              0.005)
```

```
        ( pwell         0.01 )
        ( nwell         0.01 )
    ); mfgGridResolution
```

4

Layers Group Structure

The layers group for the Santana technology file contains all of the basic information about the layers which are used by the process technology. This includes the following information:

- Layer names and corresponding layer numbers
- Mask layers and corresponding mask numbers
- Material used by each mask layer
- Purposes used to create Layer-Purpose Pairs
- Derived layers
- Via layers used for connections between two different mask layers
- Connectivity which should exist between different layers

This layers group is composed of seven different sections: the **layerMapping** section, the **maskNumbers** section, the **layerMaterials** section, the **purposeMapping** section, the **derivedLayers** section, the **viaLayers** section and the **connectivity** section. Each of these sections is described as follows:

layerMapping

Description: The **layerMapping** section is used to define the names and numbers for all of the layers which are defined in the process technology. This section defines all drawn layers which will be used, including process mask layers (such as the different metal layers), as well as the non-mask layers, such as flight line or text annotation layers.

This layer definition consists of a layer name and a layer number. The layer name can be any string value, while the layer number should be a positive integer. These layer names and layer numbers should be unique within this section. There is no requirement that these layer numbers should form an increasing sequence of integer values; they can be any positive integer, without any ordering requirements. Any positive integers can be used, and there can be gaps between any two different layer numbers.

Feedback

Example:

```
layerMapping(
  ;( name          number )
  ;( -----        --------
   ( pwell           1    )
   ( nwell           2    )
   ( diff            3    )
   ( od2             4    )
   ( pimp            7    )
   ( nimp            8    )
   ( poly1           9    )
   ( contact         10   )
   ( metal1          11   )
   ( via1            12   )
   ( metal2          13   )
   ( via2            14   )
   ( metal3          15   )
   ( via3            16   )
   ( metal4          17   )
   ( flightLine      100  )
   ( text            200  )
); layerMapping
```

maskNumbers

Description: The **maskNumbers** section is used to define the different process mask layers and corresponding mask numbers which are used in the process technology. Each of these mask layers should be defined as a layer in the **layerMapping** section.

The mask number layer definition consists of a layer name and a mask number. The layer name should be the name of a layer which is defined in this technology file, while the mask number should be a positive integer. Unlike the **layerMapping** section, these mask numbers are not required to be unique within this section, so that it is possible for different layers to use the same mask number. For example, some layout methodologies make use of several different diffusion layers, which are then logically OR-ed together to create the diffusion mask. Note that there is no requirement that these mask numbers be the same as the layer numbers which are defined for these layers. However, it is strongly suggested that these mask numbers correspond to the basic processing steps for each layer in the given process technology. That is, these mask numbers should correspond to process manufacturing steps.

Example:

```
maskNumbers(
  ;( layer         number )
  ;( -----        --------
   ( pwell           1    )
   ( nwell           2    )
```

Feedback

```
    ( diff              3     )
    ( od2               4     )
    ( poly1             5     )
    ( pimp              7     )
    ( nimp              8     )
    ( contact           10    )
    ( metal1            11    )
    ( via1              12    )
    ( metal2            13    )
    ( via2              14    )
    ( metal3            15    )
    ( via3              16    )
    ( metal4            17    )
); maskNumbers
```

layerMaterials

Description: The **layerMaterials** section is used to define the type of material which is used when a mask layer is created during the manufacturing process for the process technology. Each of these mask layers should have been defined as a mask layer in the **maskNumbers** section.

The layer material definition consists of a layer name and a mask material name. The layer name should be the name of a mask layer which is defined in this technology file, while the mask material name should be a pre-defined layer material name. These mask material names classify mask layers according to their fundamental electrical purpose. The pre-defined layer material names are the following: nWell, pWell, nDiff, pDiff, nImplant, pImplant, poly, cut, metal, contactlessMetal, diffusion, recognition or other. These pre-defined layer material names directly correspond to the layer material names used by the OpenAccess database (in the oaMaterial class).

Example:

```
layerMaterials(
  ;( layer         material    )
  ;( -----         ------------
   ( pwell         pWell       )
   ( nwell         nWell       )
   ( diff          diffusion   )
   ( od2           recognition )
   ( poly1         poly        )
   ( pimp          pImplant    )
   ( nimp          nImplant    )
   ( contact       cut         )
   ( metal1        metal       )
   ( via1          cut         )
   ( metal2        metal       )
   ( via2          cut         )
   ( metal3        metal       )
   ( via3          cut         )
```

Feedback

```
    ( metal4        metal        )
); layerMaterials
```

purposeMapping

Description: The **purposeMapping** section is used to define any purpose names and purpose numbers. Once any such purpose names and numbers have been defined, then they can be combined with one or more layer definitions to define a `Layer Purpose Pair` (abbreviated as `LPP`). The default purpose for any layer when it is constructed in the Python API is `drawing`; this default pre-defined purpose name is built-in to the Python API.

The layer purpose definition consists of a purpose name and a purpose number. The purpose name can be any string value, while the purpose number should be an integer.

The purpose name and purpose number should be unique within this section. Both the purpose name and purpose number are user-defined values, and can be used through the Python API to define any desired sets of Layer Purpose Pairs.

Example:

```
purposeMapping(
 ;( name     number )
 ;( ----     ------ )
  ( pin       251    )
  ( dummy     1      )
); purposeMapping
```

derivedLayers

Description: The **derivedLayers** section is an optional section, which is used to define layers which are derived from existing layers defined in the **layerMapping** section, or from other derived layers defined in this section. These derived layers are usually derived from existing layers through a combination of logical and/or sizing operations. For example, a derived layer can be defined as the logical `AND` or the logical `XOR` of two existing layers. Sizing operations can also be used to define derived layers. For example, a positive sizing number will increase the size of any polygons on the specified layer by the specified amount, while a negative number will decrease the size. Logical operations typically apply to two layers, while sizing operations only apply to a single layer. Derived layers can also be used in the definition of other derived layers. Thus, it is possible to define more complicated derived layers, based upon existing derived layers.

The derived layer definition consists of two required parts: the derived layer name, and the derivation sequence. The derived layer name can be any user-defined string value, which should be unique within this section. The derivation sequence is defined by combining layer (or derived layer) names, using pre-defined derivation operators, such as the following symbolic names: BBOX, SIZE, AND, NOT, OR, XOR, OUTSIDE, INSIDE_EDGE,

Feedback

INTERACT, ENCLOSE. (Note that a complete listing and brief description of all derivation operators can be found in the “DRC Operations” reference manual).

Note that the syntax for the derived layer name and the derivation sequence should not contain any blank spaces. That is, there should be no space between the derived layer name and the parenthesis, or between the derivation operator and the next parenthesis. For example, "(gate(AND(poly diff))" has correct syntax, while "(gate (AND(poly diff))" does not have correct syntax.

In many cases, this **derivedLayers** section will be generated by a supplied conversion utility, which will convert design rule files into the Santana technology file format.

Example:

```
derivedLayers(
 ;( name(derivation)            creation                    )
 ;( ------------------------   ------------------------- )
  ( GATE(AND(poly1 diff))                                  )
  ( FPOLY(NOT(poly1 diff))                                 )
  ( DCONT(OUTSIDE(contact poly1))                          )
 ;; NGATE and PGATE definitions use gate derived layer
  ( NGATE(AND(gate nimp))                                  )
  ( PGATE(AND(gate pimp))                                  )
); derivedLayers
```

viaLayers

Description: The **viaLayers** section is used to define the different via layers which are used to provide a direct connection between different mask layers, during the manufacturing process. Each of these via layer definitions consists of three parts: the via layer, the lower layer below this via layer, and the upper layer above this via layer. The via layer is used to provide a connection between the lower mask layer and the upper mask layer. Each of these mask layers should have been defined as a layer in the **maskNumbers** section.

This via layer definition consists of a lower layer name, a via layer name and an upper layer name. Note that the mask number for the upper layer should be greater than the mask number for the via layer, while the mask number for the via layer should be greater than the mask number for the lower layer.

Example:

```
viaLayers(
 ;( bottom layer   via layer    top layer )
 ;( ------------   ---------    --------- )
  ( poly1           contact      metal1   )
  ( diff            contact      metal1   )
  ( metal1          via1         metal2   )
  ( metal2          via2         metal3   )
```

Feedback

```
  ( metal3          via3          metal4    )
); viaLayers
```

connectivity

Description: The **connectivity** section is used to specify the connectivity which should exist between different layers. This connectivity information is used to build an internal connectivity database, which is then accessed by connectivity-related DRC operations. For example, there are options for the spacing DRC operations to restrict measurements to polygons which are connected (or are not connected) to the same net. As another example, the SELECT_CONNECTED DRC operation can be used to select only those polygons in one layer which are connected to another polygon in the second layer.

This connectivity section is defined by means of individual connectivity statements, which specify the layers which can be connected. There are three different types of connectivity statements: 1) `connect`, 2) `connectBy` and 3) `softConnect`. The `connect` connectivity statement is used to indicate that two different layers that can be connected. The `connectBy` connectivity statement is used to indicate that two different layers are connected through another layer; for example, two metal layers are typically connected through a via layer. The `softConnect` connectivity statement is used to indicate that a `soft` connectivity layer (such as a well layer or substrate) can be connected to a normal interconnect layer.

For example, the *nwell* layer can be `soft connected` to an active layer. Note that such a `soft` connectivity layer is considered to be too resistive to directly handle the transfer of signal or power to another layer, but it is also conductive.

Note that the information contained in this connectivity section is only used by the connectivity-related DRC operations which are defined in the technology file. There are no methods provided in the Python API to access this information, as are available to access different sections of the technology file (such as the deviceContext or ruleset sections), or to query individual design rules.

Example:

```
connectivity(
; connectivity-statement(layer-parameters)
; ---------------------------------------
connectBy(metal1  metal2 via1)
connectBy(metal2  metal3 via2)
connectBy(metal3  metal4 via3)
connectBy(metal4  metal5 via4)
connectBy(metal5  metal6 via5)
connectBy(metal6  metal7 via6)
softConnect(NTAP nwell)
softConnect(PTAP pwell)
); connectivity
```

5

Physical Rules Group Structure

The physical rules group for the Santana technology file contains all of the physical design rules used by the process technology. These physical design rules can be organized into two basic types: spacing rules and ordered spacing rules. Spacing rules are design rules specified for either no layer, one layer or two layers, in which the layers can be specified in any order for the two layer design rules. Typical spacing rules for single layers are design rules which specify the minimum width, minimum spacing or minimum area value for a single layer. Typical spacing rules for two layers are design rules which specify the minimum spacing or minimum overlap for two layers. By way of contrast, ordered spacing rules are design rules defined for two layers, in which the layers must be specified in a particular order. Typical ordered spacing rules are design rules which specify the minimum extension (or enclosure) of one layer over another layer. Note that the specification of the physical design rules in this physical rules group of the technology file is very general, and can accommodate a large number of different design rule formats.

For example, conditional design rules can be specified, where the specific value for a design rule depends upon the value of an associated parameter. In addition, design rules for Layer Purpose Pairs can be specified, where different purposes on the same layer can have different design rule values. The specification of design rule values is also very flexible; either a value can be specified for the design rule, or an optional set of one or more DRC operations can be provided.

The deviceContext section of the Santana technology file is used to define different design rules which can be used in the construction of particular devices. For example, design rules with different values can be conveniently accessed and used when high-voltage devices are created. The ruleset section of the Santana technology file is used to define different rule sets which can later be used to allow the PyCell author to access specific rule sets when PyCells are constructed or verified. For example, the PyCell author can make use of `recommended` rule sets when constructing the PyCell, or running the interactive DRC program to verify their PyCell design.

Feedback

This physical rules group is composed of four different sections: the **spacingRules** section the **orderedSpacingRules** section, the **deviceConetxt** section and the **ruleset** section. Each of these sections is described as follows:

spacingRules

Description: The **spacingRules** section is used to specify the physical design rule values for the process technology. These physical design rules can be specified for either no layer, one layer or two layers. In the case of two layer design rules, these layers can be specified in any order, without affecting the physical design rule value. Each of these layers should have been defined in this technology file, or can be a derived layer. Each physical design rule definition consists of four parts: the identifier for the design rule, the name for the design rule, the name of the layer (or layers) for the design rule, and the numerical value for the design rule. In addition, conditional design rules can be defined, where the parameter name and parameter range is specified right after the design rule value. An optional DRC command field is used to specify the individual DRC operations which can be used to calculate the value for the design rule. In addition, an optional comment field can be used to describe the functionality of the physical design rule.

This design rule definition consists of a user-defined design rule identifier and design rule name, along with the layer (or layers) used for the design rule, and the numerical value for the design rule. The design rule identifier can be any user-defined name; however, as a user convention, it is suggested that these identifiers be capitalized. The design rule name is used to specify the general function of the design rule, and can be any user-defined name. However, there are certain fundamental design rules which should use pre-defined names, as these names are used by the software to access design rule values for these fundamental design rules. For example, design rule names such as minArea, minSpacing, minWidth, minClearance and minOverlap are used. (Note that a complete listing of these pre-defined design rule names is contained in the last section of this document, “Using Santana Technology Files with PyCell Studio”). The layer should be a layer which is defined in the technology file. The value for the design rule is specified as a floating-point value in user units.

It is important to note that the user-defined names used for design rule identifiers must be unique within a particular technology file. That is because these design rule identifiers are used to refer to a particular design rule (and associated value) by name. Note that the design rule identifier refers to a specific design rule, while the design rule name only refers to the general function of the design rule (e.g., 'minSpacing'). These design rule identifiers are used by both the Python API and the Pyros™ viewer tool. The Pyros viewer uses this design rule identifier along with any optional design rule comment field when displaying DRC errors after the interactive design rule checking program has been run.

Note that the layer field can also specify a Layer Purpose Pair, which allows different design rule values to be specified for different purposes on the same layer. The layer for

Feedback

this Layer Purpose Pair should be defined in the **layerMapping** section, while the purpose should be defined in the **purposeMapping** section.

An optional field can be used to specify an optional conditional parameter and value, which would be used to specify conditional design rules. For example, the minimum spacing value for a certain layer could be defined using an optional `width` parameter, so that the minimum spacing value for this layer would depend upon the value of this `width` parameter. For example, this allows wide metal spacing rules to be easily defined.

In addition, an optional field can also be used to contain a list of design rule properties. For example, the minimum contact spacing rule for adjacent vias could be specified using properties such as `distance` and `numCuts`. Each of these properties would be specified using a name and value, just like the optional design rule parameter. In order to easily distinguish between design rule properties and parameters, these design rule property names are required to be preceded by the apostrophe symbol.

An optional DRC-command field allows the specification of individual DRC operations which can be used by the FG methods and for running interactive design rule checking in the Pyros viewer tool. . These DRC operations are defined using a pre-defined set of DRC operations, which are very similar to the DRC operations found in typical design rule decks used by DRC checking programs. These DRC operations include symbolic names such as the following: AND, OR, XOR, NOT, TOUCH, INSIDE, OUTSIDE, SIZE, GROW, GROW_EDGE, SLOPE, LENGTH, AREA, PERIMETER, WIDTH, OVERLAP, SPACING, ENCLOSURE.

As an example, the DRC operation for the minimum width `minWidth` design rule can be defined using the WIDTH operation: `WIDTH(metal3 <0.30)`. Note that this DRC command can also include an optional comment field, which can be used to describe the purpose or functionality of this DRC operation. The syntax for DRC operations requires that there not be any space between inequality/equality operations and values. For example, "metal3 <0.30" has correct syntax, while "metal3 < 0.30" does not.

Note that a design rule in this section can be specified using two different approaches. The first approach is to specify the design rule using the rule name, layer name(s) and design rule value, along with an optional DRC-commands field. The second approach is to simply define the design rule as a set of one or more DRC operations. In this second approach, the only required fields are the design rule identifier field, and the DRC-commands field; no layer names are required to be specified when the design rule is defined as a set of DRC operations.

The benefit of using the first approach is that a particular design rule can be queried for by name through the Python API (using the Tech class `getPhysicalRule()` method). For the second approach, there is no design rule name assigned, so that the design rule can not be queried for by name. However, for more complicated design rules, the second approach may be the preferred approach, since it may be difficult to specify a single numerical design rule value. The benefit of using both approaches is that it combines brief design rule descriptions with exact definitions of the design rule in terms of basic DRC

Feedback

operations, such as those found in DRC design rule decks. These DRC operations are then used by the FG methods and for running interactive design rule checking in the Pyros viewer tool. Thus, the preferred approach would be to use a combined syntax, where both approaches are used.

Example:

```
spacingRules(
;--------------------ID ( rule         layer1   layer2   value  )
;----------------------( --------     -------  -------  ------ )
              POLY.AREA ( minArea      poly1              0.09  )
              DIFF.AREA ( minArea      diff               0.10  )
            METAL1.AREA ( minArea      metal1             0.14  )
            METAL2.AREA ( minArea      metal2             0.16  )
            METAL3.AREA ( minArea      metal3             0.16  )
             DIFF.SPACE ( minSpacing   diff               0.22  )
             POLY.SPACE ( minSpacing   poly1              0.20  )
           METAL1.SPACE ( minSpacing   metal1             0.18  )
             VIA1.SPACE ( minSpacing   via1               0.18  )
           METAL2.SPACE ( minSpacing   metal2             0.20  )
             VIA2.SPACE ( minSpacing   via2               0.18  )
           METAL3.SPACE ( minSpacing   metal3             0.20  )
             POLY.WIDTH ( minWidth     poly1              0.13  )
           METAL1.WIDTH ( minWidth     metal1             0.18  )
             VIA1.WIDTH ( minWidth     via1               0.18  )
           METAL2.WIDTH ( minWidth     metal2             0.20  )
             VIA2.WIDTH ( minWidth     via2               0.18  )
           METAL3.WIDTH ( minWidth     metal3             0.20  )
        DIFF.CLEAR.POLY ( minClearance poly1    diff      0.24  )
); spacingRules
```

; fragment illustrating use of Layer Purpose Pairs;

; metal1 spacing is 0.25 for pin purpose, 0.12 otherwise

```
spacingRules(
;--------ID ( rule         layer1   layer2   value         )
;-----------( --------     -------  -------  ------        )

     M1.S.1 ( minSpacing   metal1             0.18         )
     M1.S.2 ( minSpacing   (metal1 pin)       0.25         )
); spacingRules
```

; fragment illustrating use of conditional rules;

; wide metal1 spacing value is 0.5, 0.18 otherwise

```
;--------ID ( rule         layer1   layer2   value         )
;-----------( --------     -------  -------  ------        )

     M1.S.1 ( minSpacing   metal1             0.18         )
```

Feedback

```
     M1.S.2 ( minSpacing   metal1             0.5 width>=10)
); spacingRules
```

; fragments illustrating optional DRC operations capability;

; DRC operations defined for some fundamental design rules

```
spacingRules(
;------ID ( rule       layer1    value   DRC-commands             )
;---------( --------   -------   ------  ---------------------   )
  M1.AREA ( minArea    metal1    0.14    AREA(metal1<0.14)       )
 M1.WIDTH ( minWidth   metal1    0.18    WIDTH(metal1<0.18)      )
 M1.SPACE ( minSpacing metal1    0.18    SPACING(metal1<0.18)    )
); spacingRules
spacingRules(
;------ID ( rule      lay1  lay2  value  DRC-commands           )
;---------( --------  ----- ----  ------ ----------------       )

DF.SPACE.P(minClearance poly1 diff 0.24  SPACING(diff poly<0.24))
DF.OVLAP.P(minOverlap poly1  diff  0.18  OVERLAP(diff poly<0.18))
); spacingRules
```

; define design rules only using DRC operations

```
spacingRules(
;------ID (  DRC-commands                  comment              )
;---------( -------------------------     ------------         )
  M1.AREA ( AREA(metal1<0.14)            "M1 area < 0.14"       )
 M1.WIDTH ( WIDTH(metal1<0.18)           "M1 width < 0.18"      )
 M1.SPACE ( SPACING(metal1<0.18)         "M1 spacing < 0.18"    )
DF.SPACE.P( SPACING(diff poly1<0.24) "diff-poly spacing < 0.24" )
DF.OVLAP.P( OVERLAP(diff poly1<0.18) "diff-poly overlap < 0.18" )
); spacingRules
```

---

orderedSpacingRules

Description: The **orderedSpacingRules** section is used to specify physical design rule values for the process technology. These ordered physical design rules are specified for two layers, where the order of the layers in the design rule definition affects the design rule value. For example, this would include minimum extension rules which specify the required extension of one layer over another layer. Each of these layers should have been defined in this technology file, or can be a derived layer. Each of these physical design rule definitions consists of four parts: the identifier for the design rule, the name for the design rule, the name of the layers for the design rule, and the value for the design rule.

As was described for the physical design rules defined in the **spacingRules** section, conditional design rules can be defined, where the parameter name and parameter range is specified right after the design rule value. In addition, design rule properties can also be provided. An optional DRC command field is used to specify the set of individual DRC

Feedback

operations which make up the design rule. In addition, an optional comment field can be used to describe the functionality of the physical design rule. These DRC commands have exactly the same syntax as defined in the **spacingRules** section.

This design rule definition consists of a user-defined design rule identifier and design rule name, along with the layers used for the design rule, and the actual value for the design rule. The design rule identifier can be any user-defined name; however, as a user convention, it is suggested that these identifiers be capitalized. The design rule name is used to specify the function of the design rule, and can be any user-defined name. However, there are certain fundamental design rules which should use pre-defined names, as these names are used by the software to access design rule values for these fundamental design rules. Design rule names for this section such as minExtension and minDualExtension minimum extension design rule names can be used. (Note that a complete listing of these pre-defined design rule names is contained in the last section of this document, `Using Santana Technology Files with PyCell Studio`). The two layers should be layers which are defined in the technology file. The value for the design rule is specified either as a floating-point value (for the minExtension rule name), or as a pair of floating-point values (for the minDualExtension rule name). These floating-point values should be specified in user units.

It is important to note that the user-defined names used for design rule identifiers must be unique within a particular technology file. That is because these design rule identifiers are used to refer to a particular design rule (and associated value) by name. Note that the design rule identifier refers to a specific design rule, while the design rule name only refers to the general function of the design rule (eg: 'minSpacing'). These design rule identifiers are used by both the Python API and the Pyros viewer tool. The Pyros viewer uses this design rule identifier along with any optional design rule comment field when displaying DRC errors after the interactive design rule checking program has been run.

Example:

```
orderedSpacingRules(
;--------------------ID ( rule       layer1  layer2   value     )
;----------------------( --------   ------- -------  --------- )
    DIFF.ENCLOSE.POLY ( minExtension diff    poly1     0.26     )
    POLY.ENCLOSE.DIFF ( minExtension poly1    diff     0.18     )
  M1.ENCLOSE.VIA1 ( minDualEnclosure metal1  via1   (0.02, 0.04))
  M2.ENCLOSE.VIA1 ( minDualEnclosure metal2  via1   (0.02, 0.04))
  M2.ENCLOSE.VIA2 ( minDualEnclosure metal2  via2   (0.02, 0.04))
  M3.ENCLOSE.VIA2 ( minDualEnclosure metal3  via2   (0.02, 0.04))  );
 orderedSpacingRules
```

; fragments illustrating optional DRC operations capability;

; DRC operations defined for minimum extension design rules

```
orderedSpacingRules(
;------ID ( rule              layer1  layer2 value   DRC-commands
    )
```

Feedback

```
;---------( ------------      ------- ------ -----   -------------  )
DIFF.ENC.POLY ( minExtension   diff   poly    0.26
                                    ENCLOSURE(diff poly <0.26)  )
POLY.ENC.DIFF ( minExtension   poly   diff    0.18
                                    ENCLOSURE(poly diff <0.18)  )
MTL1.ENC.VIA1 ( minDualExtension   metal1 via1  (0.02, 0.04)
                            CHECK_LINEEND(via1 metal1 0.02 0.04))
MTL2.ENC.VIA1 ( minDualExtension   metal2 via1  (0.02, 0.04)
                            CHECK_LINEEND(via1 metal2 0.02 0.04))
MTL2.ENC.VIA2 ( minDualExtension   metal2 via2  (0.02, 0.1)
                            CHECK_LINEEND(via2 metal2 0.02 0.04))
MTL3.ENC.VIA2 ( minDualExtension   metal3 via2  (0.02, 0.1)
                            CHECK_LINEEND(via2 metal3 0.02 0.04))
); orderedSpacingRules
```

deviceContext

Description: The **deviceContext** section is used to specify physical design rules which can be used to simplify the construction of devices which require the presence of a shape on one or more layers which cover all other shapes in the device. For example, a high-voltage recognition layer can be used for the construction of high-voltage devices. Once a device context section has been defined in the technology file, then it can be activated and used in the Python API to construct devices. The design rule values specified in the device context section would be used to replace the design rule values which would otherwise be used. For example, special design rules for high-voltage devices could be used in the construction of high-voltage devices.

This device context consists of a user-defined device context name, a list of the layers which are used by the device context, along with a set of design rule substitutions which define the design rule values for devices constructed using these design rules. The device context name can be any user-defined name. However, this name should be unique for the different device context sections defined in the technology file. That is because these device contexts are activated by name through the Python API. The layers in the list of layers must be drawn layers which are defined in this technology file. These layers will be the layers which are used to modify the behavior of the different FG methods in the Python API. The rule substitutions consist of two parts: the rule id for the design rule which is being substituted, and the rule id for the substituted design rule. These design rule identifiers must refer to physical design rules which are defined in this technology file. These design rule substitutions are then used to modify design rule queries. Using these unique design rule ids allows the proper design rule values to be obtained when this device context is activated and used to construct devices in the Python API.

Example:

```
deviceContext( high_voltage  pimp nwell od2
 (   rule          substitution    )
 ( --------      ----------------- )
```

```
    PO.S.1          PO.S.3          )  ; channel length
    OD.W.3          OD.W.2          )  ; channel width
    PO.S.3          PO.S.2          )  ; gate spacing
); deviceContext
```

ruleset

Description: The **ruleset** section is used to specify a set of physical design rules which can be activated by a PyCell author during the construction of a PyCell. Once a rule set has been defined in the technology file, then it can be activated and used in the Python API. The design rules specified in the rule set would be used in addition to the standard design rules, which would otherwise be used. For example, `recommended` (or DFM) design rules can be used in addition to standard design rules to improve yield.

This rule set consists of a named set of physical design rules. The rule set name can be any user-defined name. However, this name should be unique for the different rule sets which are defined in the technology file. That is because these rule sets are activated by name through the Python API. Note that the default rule set must be defined in the technology file. This default rule set is any rule set having the same name as the value of the Ruleset.defaultName class attribute, which is currently set to the name `default`.

The rule set can have a hierarchical structure, so that one rule set can be based upon another rule set. In this case, design rule IDs are used to identify and replace the corresponding design rules in the lower-level rule set. If a design rule does not exist in the lower-level rule set, it is simply appended to the list of design rules for the rule set. Note that there can be as many levels of hierarchy as desired; the only restriction is that multiple inheritance is not allowed.

In order to define a rule set, the name of the rule set is specified with the **physicalRules** section header command. In addition, the names of any rule sets upon which this rule set is based should also be specified, after the name for this rule set. These names then identify the following physical design rules as belonging to the named rule set. Note that if a name for a rule set is not specified in the technology file, then the name which will be used for this rule set is the name used for the default rule set, the value of the Ruleset.defaultName class attribute.

It is sometimes necessary to combine design rules from different rulesets into one combined ruleset. For example, a technology file may contain one ruleset named `recommended` which is used for DFM design rules, and one ruleset named `gridded` which is used for gridded placement of design objects. In order to satisfy both DFM and gridded placement rules, it is necessary to have a single ruleset which contains both sets of design rules. This combined ruleset can be defined using the optional **localRules** sub-section. The **localRules** sub-section specifies the name of a ruleset from which all `local` rules should be included into the ruleset which is being defined. These `local` rules are the design rules which are defined in the spacingRules and orderedSpacingRules sections of the ruleset being referenced by the **localRules** sub-section. The list of rules in the

combined ruleset is determined using the rule ids for the rules in the referenced rulesets, according to the following order:

- All rules from the ancestor ruleset are included
- Local rules from the ruleset referenced by the first **localRules** sub-section are included, according to their design rules id. If there is a design rule with the same rule id in the ancestor ruleset, then it is replaced by the local rule; otherwise, the local rule is simply added to the list of design rules.
- Local rules from subsequent rulesets referenced by additional **localRules** sub-sections are included, according to their design rule id. If there is a design rule with the same rule id in the ancestor ruleset or a ruleset referenced by a previous **localRules** sub-section, then it is replaced by the local rule; otherwise, the local rule is simply added to the list of design rules.
- Local rules from the defined ruleset itself are included, according to their rule ids. If there is a design rule with the same rule id in the ancestor ruleset or any ruleset referenced by a the **localRules** sub-section, then it is replaced by the local rule; otherwise, the local rule is simply added to the list of design rules.

Note that only the ancestor ruleset is included completely, and that only local rules are taken from all of the other rulesets which are referenced by the **localRules** sub-section. This approach was taken, in order to simplify the overall inheritance process for rulesets.

In order to illustrate this design rule inheritance with a simple example, consider the following basic ruleset definition:

```
physicalRules( "combined" "default"
    localRules("recommended")
    localRules("gridded")
  ; define spacing rules for this ruleset
  spacingRules(
       ...
  ); spacingRules
  ; define ordered spacing rules for this ruleset
  orderedSpacingRules(
       ...
  ); orderedSpacingRules
); "combined" ruleset
```

The inheritance process for rulesets can then be described as follows: First include all design rules from the `default` ruleset. Then include all local rules from the `recommended` ruleset; if there is a rule with the same design id in the `default` ancestor ruleset, then replace it by the local rule; otherwise, add this local rule to the list of design rules. Then include all local rules from the `gridded` ruleset; if there is a rule with the same design id in the `default` ancestor ruleset or the `recommended` ruleset, then replace it by the local rule; otherwise, add this local rule to the list of design rules. Finally, add the local rules defined for the `combined` ruleset itself; if there is a rule with the same design id in the `default`,

`recommended` or `gridded` rulesets, then replace it by the local rule; otherwise, add this local rule to the list of design rules. After these steps have been completed, then the list of design rules will be the complete set of design rules for the `combined` ruleset.

Example:

```
physicalRules( "recommended" "default"
; fragment of "recommended" rule set, based upon default
; "default" rule set (which must always be present)
spacingRules(
;--------------------ID ( rule         layer1   layer2   value  )
;-----------------------( --------     -------  -------  ------ )
              POLY.AREA ( minArea      poly1              0.09  )
              DIFF.AREA ( minArea      diff               0.10  )
           DIFF.SPACING ( minSpacing   diff               0.22  )
           POLY.SPACING ( minSpacing   poly1              0.20  )
             POLY.WIDTH ( minWidth     poly1              0.13  )
               M1.WIDTH ( minWidth     metal1             0.18  )
); spacingRules
orderedSpacingRules(
;--------------------ID ( rule          layer1  layer2   value  )
;-----------------------( --------      ------- -------  ------ )
  DIFF.ENCLOSE.POLY ( minExtension     diff     poly1     0.26  )
  POLY.ENCLOSE.DIFF ( minExtension     poly1    diff      0.18  )
  M1.ENCLOSE.VIA1 ( minDualExtension   metal1   via1 (0.02 0.04))
  M2.ENCLOSE.VIA1 ( minDualExtension   metal2   via1 (0.02 0.04))
  M2.ENCLOSE.VIA2 ( minDualExtension   metal2   via2 (0.02 0.04))
  M3.ENCLOSE.VIA2 ( minDualExtension   metal3   via2 (0.02 0.04))
); orderedSpacingRules
); "recommended" ruleset
```

Electrical Rules Group Structure

The electrical rules group for the Santana technology file contains all of the electrical design rules used by the process technology being described in this Santana technology file. These electrical design rules can be organized into three basic types: characterization rules, oxide definitions and MOSFET transistor definitions. The characterization rules are electrical rules which can specify electrical values for no layers, one layer or for two layers. Typical single layer electrical rules include area capacitance, current density or sheet resistance for the layer. Typical two layer electrical rules would specify the area capacitance or edge capacitance between these two layers. Typical oxide definitions include `thin` and `thick` oxide lengths, along with associated voltage supply values. The MOSFET transistor definitions are used to specify various electrical values for the basic MOSFET transistor model. These MOSFET transistor model parameters would then be used when constructing different types of MOSFET transistors or parameterized cells containing MOSFET transistors.

Feedback

This electrical rules group is composed of three different sections: the **characterizationRules** section, the **oxideDefinitions** section and the **mosfetDefinitions** section. Each of these sections is described as follows:

characterizationRules

Description: The **characterizationRules** section is used to specify basic electrical design rule values for no layers, single layers, or for two layers. Each of these electrical design rule definitions consists of three parts: the name for the electrical design rule, the names for any layers involved, and the value for this electrical design rule. The name field can be any user-defined name for this electrical design rule, while the layers should have been defined as mask layers in the **maskNumbers** section of this technology file. The actual electrical design rule value should be specified as an integer or a floating-point number. Note that there are no units assumed for this electrical design rule value; units are assumed to be handled as required by the application. Thus, there will be no conversion between user units and database units when these electrical design rule values are used by the PyCell Studio software product.

Example:

```
characterizationRules(
 ;( rule            layer1    layer2    value     )
 ;( --------------  -------   -------   --------- )
  ( areaCap          metal1              3.37e-05  )
  ( areaCap          metal2              1.487e-05 )
  ( areaCap          metal3              9.54e-06  )
  ( sheetRes         poly                350       )
  ( sheetRes         metal1              0.12      )
  ( sheetRes         metal2              0.12      )
  ( sheetRes         metal3              0.12      )
  ( areaCap          poly      metal1    6.5e-05   )
  ( areaCap          metal1    metal2    4e-05     )
  ( areaCap          metal1    metal3    4e-05     )
  ( areaCap          metal2    metal3    4e-05     )
  ( edgeCapacitance  poly      metal1    6.5e-12   )
  ( edgeCapacitance  metal1    metal2    4e-12     )
  ( edgeCapacitance  metal1    metal3    4e-12     )
  ( edgeCapacitance  metal2    metal3    4e-12     )
); characterizationRules
```

oxideDefinitions

Description: The **oxideDefinitions** section is used to specify the supply voltage values and any other electrical parameter values which should be used to define different oxide type values for the process technology. These different oxide types are user-specified; for the examples described here, `thin` and `thick` oxide values are used. For each oxide type (`thin` or `thick`), this section specifies the supply voltage which should be used, along with

Feedback

the values of any other user-defined parameters. These parameters and values should be used to specify the important electrical process values for these oxide types.

Each of these oxide definitions consists of two parts: the name for the oxide type, and the associated parameters and values for this oxide type. It is suggested that one of these parameters be the supply voltage value for this oxide type. The oxide type can be any user-defined string (typically `thin` or `thick`). The supply voltage values should be specified as floating-point numbers, where the supply voltage is measured in volts. Any other user-defined parameter values should have the proper units for their intended use.

Example:

; specify supply voltage and oxide thickness for each oxide type

```
oxideDefinitions(
thin (
;( parameter     value    )
;( ---------     -------  )
   ( supply      1.8      )
   ( tox         4.08e-09 )
); thin
thick (
  ;( parameter   value    )
  ;( ---------   -------  )
   ( supply      3.3      )
   ( tox         6.8e-09  )
); thick
```

mosfetDefinitions

Description: The **mosfetDefinitions** section is used to specify the various parameter values for a standard MOSFET transistor model, based upon the process technology. These MOSFET transistor model parameter values are specified, so that they can be used in the construction of transistors for different types of parameterized cells. There are several different possible parameters for this MOSFET transistor model, which can be briefly described as follows:

- type – transistor type (`nmos` or `pmos`)
- oxide – oxide type (defined in oxide section; typically defined to be `thin` or `thick`)
- minWidth – minimum gate oxide width value
- maxWidth – maximum gate oxide width value
- minLength – minimum poly length value
- maxLength – maximum poly length value

Feedback

Note that the parameter names used above for minimum and maximum width and length values are user-defined. In addition to the above parameters, there can be any number of user-defined parameters defined for each transistor type and oxide type. These additional user-defined parameters should be used to specify important electrical characteristics of the MOSFET transistor for the process technology. Each of these MOSFET definitions consists of three parts: the transistor type (typically ‘nmos’ or ‘pmos’), the oxide type (typically ‘thin’ or ‘thick’), and the specification of the values for the MOSFET transistor model parameters for this particular transistor type and oxide type. That is, each such MOSFET model parameter definition is a function of the transistor type and the oxide type. Note that the transistor type and the oxide type must be specified for each such MOSFET definition, or else an exception will be raised when the technology file is compiled.
Note that these MOSFET model parameter values are all either integer or floating-point numbers. Any user-defined parameter values should have proper units for their intended use.

Example:

```
mosfetDefinitions(
(
 ;( parameter    value       )
 ;( ---------    ------      )
   ( type         nmos        )
   ( oxide        thick       )
   ( maxLength    1           )
   ( maxWidth     16          )
   ( minLength    0.35        )
   ( minWidth     0.42        )
);
(
 ;( parameter    value       )
 ;( ---------    ------      )
   ( type         nmos        )
   ( oxide        thin        )
   ( maxLength    1           )
   ( maxWidth     16          )
   ( minLength    0.13        )
   ( minWidth     0.18        )
);
(
 ;( parameter    value       )
 ;( ---------    ------      )
   ( type         pmos        )
   ( oxide        thick       )
   ( maxLength    1           )
   ( maxWidth     16          )
   ( minLength    0.3         )
   ( minWidth     0.42        )
);
(
 ;( parameter    value       )
 ;( ---------    ------      )
```

Feedback

```
    ( type          pmos        )
    ( oxide         thin        )
    ( maxLength     1           )
    ( maxWidth      16          )
    ( minLength     0.13        )
    ( minWidth      0.18        )
  );
  ); mosfetDefinitions
```

6

Pre-Defined Layer and Purpose Names

As a convenience for PyCell development and the use of tools, there are a number of special layers and purposes which are pre-defined for the Santana technology file. Recall that Cadence tools such as Virtuoso make use of a number of pre-defined layers and purposes which are defined by default in the Cadence development environment. In addition, OpenAccess also provides eleven system-reserved purposes. Instead of requiring all of these layers and purposes to be defined in each Santana technology file, has taken the approach of pre-defining these system-reserved layers and purposes whenever the Tech object is accessed in the Santana development environment. This approach is both consistent with the Cadence development environment, and also makes it much more convenient for the PyCell developer.

The following table lists all of the Cadence layer names and layer numbers which are pre-defined in the Santana development environment. These pre-defined layers will be automatically generated for the Santana development environment, and do not need to be explicitly defined in the Santana technology file. However, if desired, these layers can be re-defined in the Santana technology file, by simply providing an explicit layer name and layer number definition.

Layer Name	Layer Number
Unrouted	200
Row	201
Group	202
Cannotoccupy	203
Canplace	204
hardFence	205
softFence	206
y0	207
y1	208
y2	209

Layer Name	Layer Number
y3	210
y4	211
y5	212
y6	213
y7	214
y8	215
y9	216
designFlow	217
stretch	218
edgeLayer	219
changedLayer	220
unset	221
unknown	222
spike	223
hiz	224
resist	225
drive	226
supply	227
wire	228
pin	229
text	230
device	231
border	232
snap	233
align	234
prBoundary	235

Layer Name	Layer Number
instance	236
annotate	237
marker	238
select	239
substrate	240
grid	251
axis	252
hilite	253
background	254

The following table lists all of the Cadence purpose names and purpose numbers which are pre-defined in the Santana development environment. These pre-defined purposes will be automatically generated for the Santana development environment, and do not need to be explicitly defined in the Santana technology file. However, if desired, these purposes can be re-defined in the Santana technology file, by simply providing an explicit purpose name and purpose number definition.

Purpose Name	Purpose Number
fatal	223
critical	224
soCritical	225
soError	226
ackWarn	227
info	228
track	229
blockage	230
grid	231
fillOPC	232
warning	234

Purpose Name	Purpose Number
tool1	235
tool0	236
label	237
flight	238
error	239
annotate	240
drawing1	241
drawing2	242
drawing3	243
drawing4	244
drawing5	245
drawing6	246
drawing7	247
drawing8	248
drawing9	249
boundary	250
pin	251
net	253
cell	254
all	255

In addition to these pre-defined Cadence purpose names and purpose numbers, there are also twelve system-reserved purposes which are defined by OpenAccess. These purposes are the following: `drawing`, `fill`, `slot`, OPCSerif", `OPCAntiSerif`, `annotation`, `gapFill`, `redundant`, `oaAny`, `oaNo`, `oaFillOPC` and `oaCustomFill`. Note that unlike the pre-defined Cadence layers and purposes, these OpenAccess system-reserved purposes can not be re-defined in the Santana technology file. These pre-defined OpenAccess purposes will also be automatically provided as part of the Santana development environment.

Note that although all of these pre-defined layers and purposes are available to the PyCell developer when using the Python API Tech class, these pre-defined layers and purposes will not be included in the list of layer names or purpose names which are provided by the `getSantanaLayerNames()` and `getSantanaPurposeNames()` methods. Instead, these two methods will only return the list of layer names and purpose names which are explicitly defined in the associated Santana technology file.

7

Using Santana Technology Files with PyCell Studio

Although the Santana technology file provides a very flexible syntax, there are some issues to be aware of when using these technology files with the PyCell Studio product. Although the designer is allowed to use any names for the names of design rules in this technology file, there are certain fundamental design rules which should be present in this technology file. These fundamental design rules are used internally by the PyCell Studio tools to construct `smart` DRC `Correct by Construction` layout objects, such as the Contact, DeviceContact, ContactRing, Bar and RoutePath class objects. PyCell Studio reads the design rule information contained in the technology file to construct these `smart` layout objects, but it uses specific names for these key design rules.

When the Santana technology file is created, it is important that the following design rules be described for the given process technology. In addition, the names used for these fundamental design rules should use the following names:

1. minimum width rules for single layer, named `minWidth`.
2. minimum spacing rules for single layer, named `minSpacing`.
3. minimum spacing rules for two layers, named "minClearance"
4. minimum area rules for each layer, named `minArea`.
5. minimum extension rules for pairs of layers, named `minExtension` and `minDualExtension`.
6. minimum overlap rules for pairs of layers, named `minOverlap`.

These fundamental design rules should use the above names, with the exact spelling and capitalization as indicated (that is, use `minWidth` and not `minwidth`). These are the exact design rule names which will be used to reference specific design rule values, when the `smart` layout objects (Contact, DeviceContact, ContactRing, Bar and RoutePath) are created by PyCell Studio.

In addition, any conditional design rules should be written using the same names for the relevant parameters. For example, conditional design rules can be used which query the width of the enclosing shape for the `minExtension` and `minDualExtension` design rules. In these cases, `width` should be used as the common parameter name for both of these

design rules. This approach helps to ensure that the PyCell Studio `smart` layout objects will incorporate and make use of all applicable design rule values.

There are a number of pre-defined design rule names which are used by the Santana technology file. Note that these design rule names follow the OpenAccess standards for design rule names. The following table lists the most commonly used standard design rule names, along with a brief description:

Design Rule Name	Description
minArea	Minimum area for shape defined on a layer
minEnclosedArea	Minimum empty area required for hole formed by a surrounding shape; used to check for `pinholes`.
minWidth	Minimum width for a shape defined on a layer
maxWidth	Maximum width for shape defined on a layer
minSpacing	Minimum horizontal or vertical spacing required between two shapes defined on a layer.
minSameNetSpacing	Minimum space required between two electrically equivalent shaped defined on the same layer
minAdjacentViaSpacing	Minimum distance required between Via cut shapes, when these Vias are adjacent to one another. Uses the 'distance and 'numCuts property values.
minLargeViaArrayCutSpacing	Minimum spacing between Via cuts in an array of Vias with a number of cuts which exceeds the 'minNumCuts property value.
minNumCuts	Minimum number of cuts required for Via
minClearance	Minimum separation required between the outside edges of two non-intersecting shapes, each of which is defined on a different layer.
minOverlap	Minimum distance required between the inside edge of layer2 shape to the inside edge of the layer1 shape.
minExtension	Minimum distance inside edge of layer1 shape extends beyond the outside edge of layer2 shape.
minDualExtension	Minimum distance inside edge of layer1 shape extends beyond the outside edge of layer2 shape, specified for both horizontal and vertical extension values.

Note that if the technology file does not specify DRC operations for the following fundamental design rules (minWidth, minSpacing, minOverlap, minExtension, minClearance and minArea), then the appropriate DRC operations will be generated by default, and will then be used by the `Geometry Engine` and the FG methods. These

default DRC operations will only be generated for non-conditional design rules; the user will need to define DRC operations for any conditional design rules. If design rules are specified in the technology file, which are not one of the six fundamental design rules listed above, and also have no DRC operations defined, then they will not be directly used by the PyCell Studio product. However, any such design rules will be available to the PyCell author through the Python API, and these design rules can then be accessed by name, using the Tech class `getPhysicalRule()`, `conditionalRuleExists()` and `physicalRuleExists()` methods. However, if DRC operations are defined for any design rule, then these DRC operations will be used by the `Geometry Engine` part of the PyCell Studio product. When more design rules are specified using DRC operations, the `Geometry Engine` has more design rule information available when it performs various layout operations. Thus, more optimal layout results will generally be obtained, if DRC operations are specified for all of the design rules defined in the technology file.

Another aspect of design rules and the PyCell Studio product has to do with the `Geometry Engine` FG methods, such as the `fgPlace() smart` placement method. This FG method will perform an optimal placement of two specified layout objects, based upon all relevant design rules for the specified layers. This is done by making use of all of the DRC Operations which have been specified in the technology file. Even if design rules are specified with numerical values, as well as DRC operations, only the DRC operations will be used, instead of the numerical values. Note that these DRC operations must report DRC errors in terms of SPACING operations; otherwise, the FG methods will not consider these DRC operations. However, these DRC operations will be considered when running the DRC program. Also note that these SPACING operation design rule violations must occur between the physical component being placed and the reference physical component. If the SPACING violation is only reported inside one of these two physical components, then this DRC operation will not be considered.

There are also separate design rule requirements which should be considered for the `fgEnclose()` and `fgAddEnclosingPolygon()` methods. It is required that ENCLOSURE DRC operations be used, where the first input layer and the second input layer for these ENCLOSURE operations can be any layer, including derived layers.

Hybrid Technology File / OpenAccess (OA) Tech Database Mode

If both the Santana techfile and the OpenAccess tech database are present in a tech lib, then tech info is read on per section basis:

If this section is present in the Santana techfile:	Then:
viewTypeUnits	Read the section from Santana techfile; otherwise read view type units from the OA tech DB.

Feedback

If this section is present in the Santana techfile:	Then:
mfgGridResolution	Read the section from Santana techfile; otherwise read manufacturing grid from the OA tech DB.
purposeMapping	Read the section from Santana techfile; otherwise read purpose definitions (purpose name to purpose number mapping) from the OA tech DB.
layerMapping	Read the section from Santana techfile; otherwise read physical layer definitions (layer name to layer number mapping) from the OA tech DB.
maskNumbers	Read the section from Santana techfile; otherwise read layer mask numbers from the OA tech DB.
layerMaterials	Read the section from Santana techfile; otherwise read layer materials from the OA tech DB.
viaLayers	Read the section from Santana techfile; otherwise read via layer information from the OA tech DB.
derivedLayers	Read the section from Santana techfile; otherwise read derived layer definitions from the OA tech DB.
physicalRules, spacingRules, or orderedSpacingRules	Read the section from Santana techfile; otherwise read design rules from the OA tech DB.

All other sections of the Santana techfile are specific to this format and are always loaded from the Santana techfile only.

NOTE: There is no merging of tech data within a techfile section. For example, either all design rules are read only from the Santana techfile or all design rules are read only from the OpenAccess tech database.

To supplement an existing OpenAccess tech lib with partial Santana techfile, the following command can be used:

```
cngenlib --update --techfile Santana.tech.partial --create_tech_db no …
```