Learn the OpenAccess API Using Python

Initial Contribution By

James Masters
Intel - 2013
Updates & Additions
by Silicon Integration Initiative - 2020

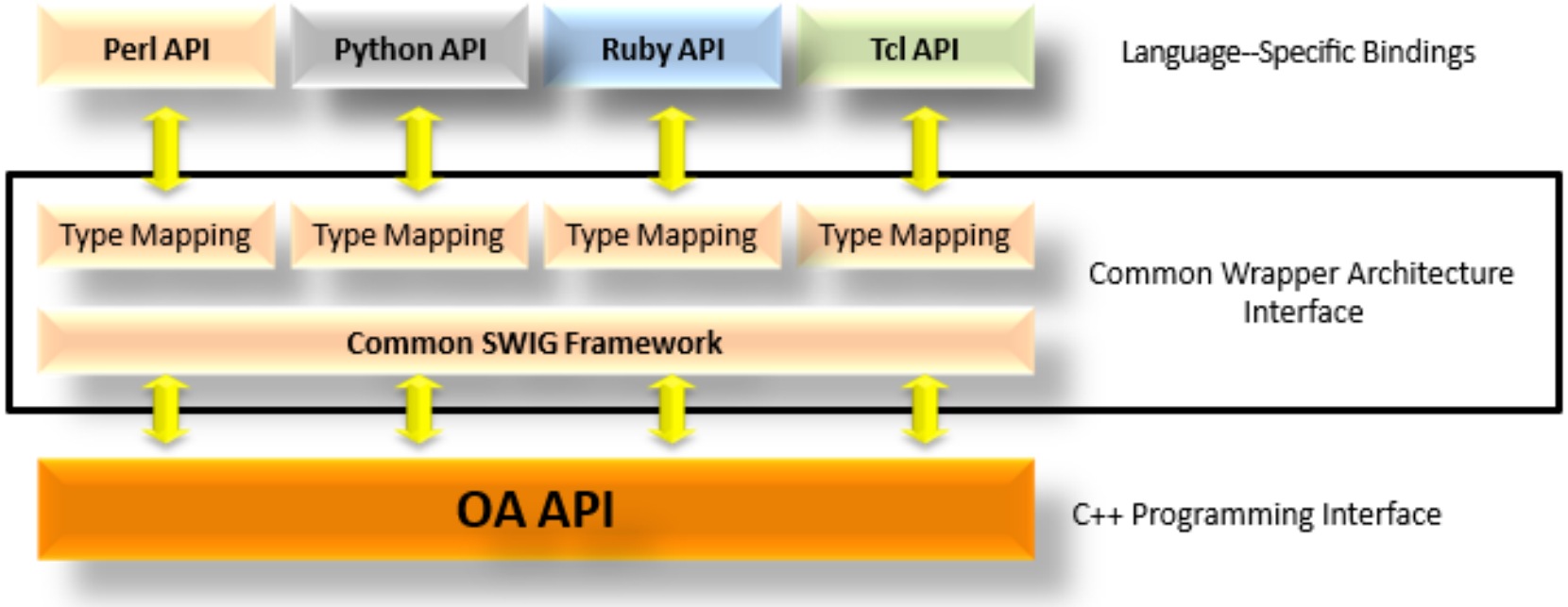
Lab 1.1

- This should Have been completed in the Training Set Up Module
- The lab simply looks for the libraries for the Python API to insure the installation is correct

see: labs/1.1/test.py

oaScript Overview

- The only access to OA data is through the C++ API
  - Write your own C++ application (longer formal development)
  - **Or…** oaScript is used to provide direct access through perl, python, ruby, or tcl (rapid prototyping and application development)
  - Interface to C++ API is handled using SWIG (www.swig.org)



Common Themes in Wrapped Languages

- Common structure/intent
  - Remain as close to the OA API as possible to allow reuse of the existing C++ OA documentation
  - Wrapped names may be slightly adjusted to meet language requirements (as needed)
  - Use type conversions where it makes sense

- Unique behaviors
  - Type conversion customizations for ease-of-use and match language data types where it makes sense
  - Matching idioms of a language

Section 2 Introduction

- The OpenAccess API has over 2000 classes
- The average user uses a small subset which we will cover in this course
- It is a good idea to know how to look up new things within the Exhaustive C++ API documentation

Class Listing

OpenAccess 2.2 C++ API

Topics | All Classes | Class Categories | Exceptions | Programmers Guide | Information Model | Index | Help

OpenAccess API List of Classes

1 | 2 | a | b | c | d | e | f | g | h | i | l | m | n | o | p | r | s | t | u | v | w

1
oa1DLookupTbl
2
oa2DLookupTbl
A
oaAbstractType
oaAlignmentType
oaAnalysisLib
oaAnalysisOpPoint
oaAnalysisPoint
oaAnalysisPointArray
oaAntennaArea
oaAntennaAreaArray
oaAntennaData
oaAntennaModel
oaAntennaRatioArrayValue
oaAntennaRatioValue
oaAppDef
oaAppObject
oaAppObjectDef
oaAppObjectDefCollection
oaAppProp
oaArc
oaAreaBlockage
oaAreaBoundary
oaAreaHalo
oaArray

oaInductor
oaInst
oaInstAttrDisplay
oaInstAttrType
oaInstHeader
oaInstPropDisplay
oaInstQuery
oaInstTerm
oaInstTermAttrType
oaInt1DTblValue
oaInt2DTblValue
oaIntAppDef
oaIntDualIntArrayTblValue
oaInterPointerAppDef
oaInterpolateType
oaIntFltTblValue
oaIntProp
oaIntRange
oaIntRangeArray
oaIntRangeArray1DTblValue

oaOccVectorInstDef
oaOpPoint
oaOpPointHeader
oaOrient
oaOSError
P
oaParam
oaParamArray
oaParamType
oaParasiticNetwork
oaPath
oaPathSeg
oaPathStyle
oaPcellDef
oaPcellLink
oaPcellObserver
oaPhysicalLayer
oaPiElmore
oaPin
oaPinConnectMethod

There are approximately 2000 OA classes… don't be overwhelmed! You will probably end up only using less than a hundred.

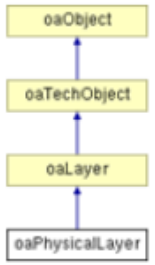
Lab 2.1: Using OA API Doc

- Use the API to determine how to create a polygon (oaPolygon)
  - How do you get the number of points in the polygon after it's created?
  - What is the return value for isOrthogonal()?
  - What is the return value for setPoints()?

The API Documentation

oaPhysicalLayer Class Reference

Inheritance diagram for oaPhysicalLayer:



Inheritance tree (clickable)

Lists all functions (including inherited)

Inherited Members | Schema Diagram

Public Methods

```
                oaMaterial getMaterial () const
            oaPrefRoutingDir getPrefRoutingDir () const
                  oaUInt4 getMaskNumber () const
                   oaDist getManufacturingGrid () const
          oaPhysicalLayer * getLayerAbove (oaMaterial material) const
          oaPhysicalLayer * getLayerBelow (oaMaterial material) const
oaCollection< oaLayer, oaTech > getLayerAbove (const oaTech *tech, oaMaterial material, oaBoolean local=false) const
oaCollection< oaLayer, oaTech > getLayerBelow (const oaTech *tech, oaMaterial material, oaBoolean local=false) const
                     void getExcludedLayers (oaLayerNameArray &excludedLayerNames) const
                oaBoolean hasExcludedLayers () const
                     void setMaterial (oaMaterial material)
                     void setPrefRoutingDir (oaPrefRoutingDir dir)
                     void setMaskNumber (oaUInt4 number)
                     void setManufacturingGrid (oaDist grid)
                     void resetManufacturingGrid ()
                     void setExcludedLayers (const oaLayerNameArray &excludedLayerNames)
                     void unsetExcludedLayers ()
```

Instance Methods (click for details)

Static Public Methods

```
oaPhysicalLayer * create (oaTech *tech, const oaString &name, oaLayerNum number, oaMaterial material=oacOtherMaterial, oaUInt4 maskNumber=oacUnsetMaskNumber)
oaPhysicalLayer * find (const oaTech *tech, const oaString &name)
oaPhysicalLayer * find (const oaTech *tech, oaLayerNum number)
oaPhysicalLayer * find (const oaTech *tech, const oaString &name, oaBoolean local)
oaPhysicalLayer * find (const oaTech *tech, oaLayerNum number, oaBoolean local)
```

Static/Class Methods (click for details)

Detailed Description

The oaPhysicalLayer class defines a physical layer that is referenced when creating physical design data.

Detailed Function Description further below

Mapping to OA API Documentation

- Only slight changes made to API where necessary in target languages
- 📖 Navigate to the “oaLib” class “create” method in the API document

Static Public Methods

oaLib * create (const oaScalarName &name, const oaString &libPath, oaLibMode mode=oacSharedLibMode, const oaString &dmSystem="oaDMSystem", const oaDMAttrArray *dmAttrList=NULL)

Perl	`new oa::oaLib::create(oaScalarName, string, oaLibMode, string, array|undef)`
Python	`oa.oaLib.create(oaScalarName, oaString, oaLibMode, oaString, array|None)`
Ruby	`Oa::OaLib.create(oaScalarName|String, String, oaLibMode|symbol, String, Array|nil)`
Tcl	`oa::oaLib_create oaScalarName string| oaLibMode string list|"NULL"`

Mapping to OA API Documentation

- Navigate to the “oaLib” class “getAccess” method in the API document

Public Methods

oaBoolean getAccess (oaLibAccess accessType, oaUInt4 ,meout=0)

Perl	`$lib->getAccess(new oa::oaLibAccess('read'));` `$lib->getAccess($oa::oacReadLibAccess);`
Python	`lib.getAccess(oa.oaLibAccess('read'))` `lib.getAccess(oa.oaLibAccess(oa.oacReadLibAccess))`

Ruby	`lib.getAccess(Oa::OaLibAccess.new('read'))`
	`lib.getAccess(Oa::OaLibAccess.new(Oa::OacReadLibAccess))`
	`lib.getAccess(:read)`

Tcl	`$lib getAccess [oa::oaLibAccess "read"]`
	`$lib getAccess [oa::oaLibAccess $oa::oacReadLibAccess]`
	`$lib getAccess $oa::oacReadLibAccess`

Lab 2.2

Goal - Learn to use the documentation and demonstrate ability to navigate the C++ API and translate into the native language format.

Write a script to:

1. Create an oaTimer object
2. Create some kind of delay
3. Get the elapsed number of seconds for the operation
4. print the value

Compare your script to labs/2.2/timer.py

Basic Type Mapping

(conversion from a basic OA type to a native language type)

	Perl	Python	Ruby	Tcl
void/NULL	undef	None	nil	“NULL”
oaBoolean	Integer (scalar)	bool	TrueClass/FalseClass	Integer
oa*Int	Integer (scalar)	int	Fixnum	Integer
oaFloat/Double	Float (scalar)	float	Float	Float
oaString	String (scalar)	oaString or str	String	String
oa*Array	oa*Array or native array	oa*Array or native array	Oa*Array or native array	oa*Array or native array
oaTime	oaTime	oaTime	Time	oaTime
oaTimestamp	oaTimeStamp	Int	Integer	oaTimeStamp
oaComplex	oaComplex	complex	OaComplex	oaComplex
oaPoint	oaPoint or (x,y) array	oaPoint or [x,y] array	OaPoint or [x,y] array	oaPoint or [x,y] array
oaBox	oaBox or [l,b,r,t] array	oaBox or [l,b,r,t] array	OaBox or [l,b,r,t] array	oaBox or [l,b,r,t] array
oaTransform	oaTransform or [x,y,o] array	oaTransform or [x,y,o] array	OaTransform or [x,y,o] array	oaTransform or [x,y,o] array

Enumeration Wrappers

- Enumerated wrappers represent a numerical value but can be retrieved using a string
- Navigate to “oaViewType” in the API

Perl	`oa::oaViewType::get('maskLayout')` `oa::oaViewType::get($oa::oacMaskLayout)`
Python	`oa.oaViewType.get('maskLayout')` `oa.oaViewType.get(oa.oacMaskLayout)`

Ruby	`Oa::OaViewType.get(:maskLayout)`
	`Oa::OaViewType.get('maskLayout')`
	`Oa::OaViewType.get(Oa::OacMaskLayout)`

Tcl	`oa::oaViewType_get $oa::oacMaskLayout` `oa::oaViewType_get maskLayout`

Output Arguments

- Some OA functions expect the return value to be passed through the argument list (argument is a pointer)
  - Function return value is C++ “void” (no return value)
  - You need to pre-allocate an empty object to be filled
  - Example: **`void oaName::get(oaString &out)`**
- As a convenience, oaScript does two things with these functions:

1. The output argument also becomes the return value instead of not returning anything
2. You do not need to pre-allocate the output argument

- Examples - both are equivalent in oaScript:
  - <u>Option 1:</u> as documented in the C++ API doc

```
name = oa.oaName("foo")
str = oa.oaString()
name.get(str) #=> returns "foo" and str="foo"
```

  - <u>Option 2:</u> without pre-allocation using oaScript:

```
name = oa.oaName("foo")
name.get() #=> returns "foo"
```

Note: Python is the only language which defines oaString() since the native Python string is immutable (cannot be changed in a function). Pre-allocate other strings in the other languages using just a regular string.

Lab 2.3

- Goal - Get more familiar with using enumeration wrappers and also not needing to pre-allocate a string on output arguments.

- Create a script to:

1. Create an oaViewType object from the "schematic“ oaReservedViewType enumeration wrapper
2. Print the view type object's name to the screen
   - From the object, not from the hard-coded "schematic" string

- Compare your script to labs/2.3/enum_view.py

  What are the enumerated values for oaViewType?

  Hint: Look at oaReservedViewType

Section 2 Summary

- Using the C++ Documentation
- Type Mapping
- Output arguments

Silicon Integration Initiative

www.si2.org

For details contact Marshall Tiner

Director of Production Standards

mtiner@si2.org