Features in v2.3.0

- Read and write SBOL files
- Interface with online validation tool
- Interface with SynBioHub repository
- Parts-based design
- Hierarchical sequence assembly
- Workflow management and design-build-test-learn
- Extensible data model and custom annotations
- Support for combinatorial libraries
- Biosystem design (modules & interactions)

PySBOL Installation

PySBOL packages available in Python 2.7 and 3.6 on Windows, Mac OSX, and Linux

Installation:

$ pip install pysbol --user

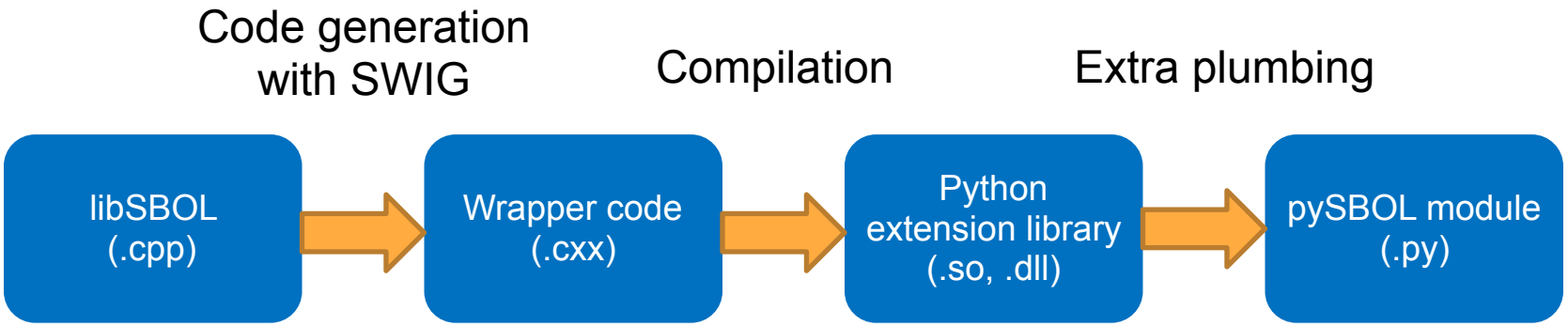
Documentation:

https://pysbol2.readthedocs.io/en/latest/

Repository:

https://github.com/SynBioDex/pySBOL

Python Code is Generated from C++



- LibSBOL can be translated into other languages implemented in C/C++ as well (eg, Matlab)
- Some consistency of the API across different languages
- Serialization is well-validated and predictable (ideal for a standard language such as SBOL!)

Guiding Philosophy

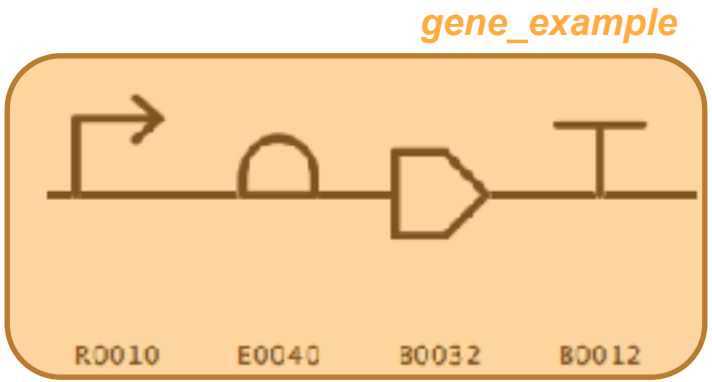
- User-experience: An object-oriented approach to synthetic biology
- Library implementation and specification diagrams are intuitively correlated
- Extensible data model

User Experience

Object-oriented Synthetic Biology

High-level Design Automation

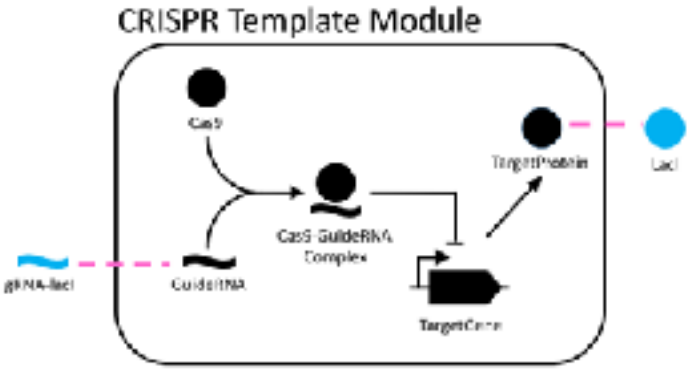
- **Assemble** hierarchies of parts



- **Compile** DNA sequences from different parts; replace cut-and-paste
- **Assemble** modules, eg, layered, regulatory gates and other modular systems

Other High-level Design Tasks

- Connecting Module Inputs and Outputs
- Mechanistic Modeling of Biochemical Interactions
- Overriding Components in a Template Design



Library Implementation and Specification Document are Closely Correlated

*The examples in the online documentation is the **first** point of entry for understanding the Python API*

Getting Started with SBOL

This beginner's guide introduces the basic principles of pySBOL for new users. Most of the examples discussed in this guide are excerpted from the example script. The objective of this documentation is to familiarize users with the basic patterns of the API. For more comprehensive documentation about the API, refer to documentation about specific classes and methods.

The class structure and data model for the API is based on the Synthetic Biology Open Language. For more detail about the SBOL standard, visit sbolstandard.org or refer to the specification document. This document provides diagrams and description of all the standard classes and properties that comprise SBOL.

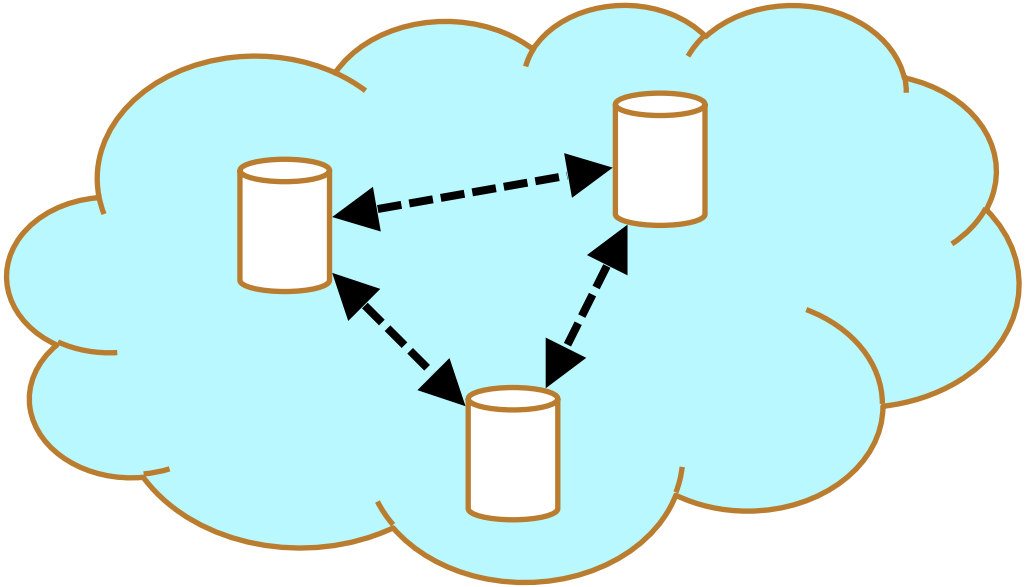
Creating an SBOL Document

In a previous era, engineers might sit at a drafting board and draft a design by hand. The engineer's drafting sheet in pySBOL is called a Document. The Document serves as a container, initially empty, for SBOL data objects which represent elements of a biological design. Usually the first step is to construct a Document in which to put your objects. All file I/O operations are performed on the Document. The `read` and `write` methods are used for reading and writing files in SBOL format.

```
>>> doc = Document()
>>> doc.read('crispr_example.xml')
>>> doc.write('crisp_example_out.xml')
```

Reading a Document will wipe any existing contents clean before import. However, you can import objects from multiple files into a single Document object using Document.append(). This can be advantageous when you want to integrate multiple objects from different files into a single design

SBOL Integrates Data Across the Semantic Web



Every SBOL data object has a uniform resource identifier (URI).

Every SBOL Objects has a Uniform Resource Identifier (URI).

"Compliant" URIs

http://sys-bio.org/my_design/1

scheme namespace displayId version

Typed URIs

http://sys-bio.org/SBOLClass/my_design/1

scheme namespace type displayId version

An Example Constructor

Sets default namespace for URI generation

```
>>> setHomespace('http://sys-bio.org')
>>> cd0 = ComponentDefinition('cd0’, BIOPAX_DNA)
>>> print cd0
http://sys-bio.org/ComponentDefinition/cd0/1
```

An Example Constructor

Every constructor takes an identifier as its first argument

```
>>> setHomespace('http://sys-bio.org')
>>> cd0 = ComponentDefinition('cd0’, BIOPAX_DNA)
>>> print cd0
http://sys-bio.org/ComponentDefinition/cd0/1
```

An Example Constructor

```
>>> setHomespace('http://sys-bio.org')
>>> cd0 = ComponentDefinition('cd0’, BIOPAX_DNA)
>>> print cd0
http://sys-bio.org/ComponentDefinition/cd0/1
```

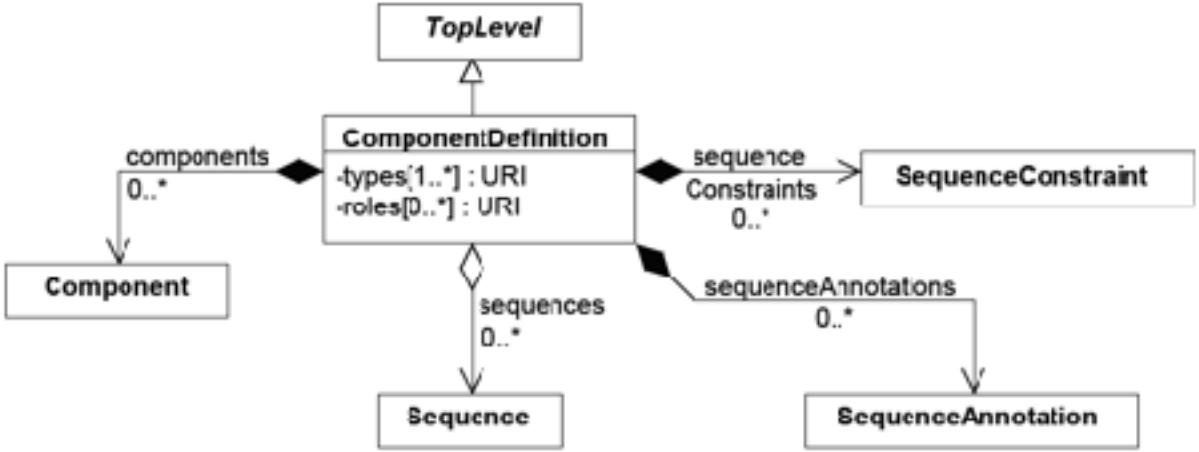
Note the full URI is constructed from the user specified ID

An Example Constructor

```
>>> setHomespace('http://sys-bio.org')
>>> cd0 = ComponentDefinition('cd0’, BIOPAX_DNA)
>>> print cd0
http://sys-bio.org/ComponentDefinition/cd0/1
>>> cd0.roles = [ SO_PROMOTER ]
```

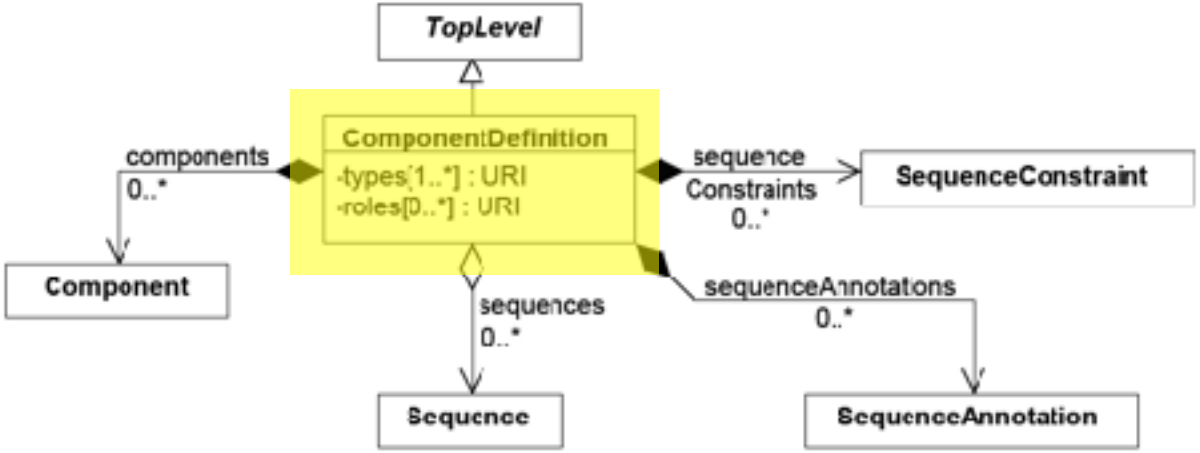
Optional fields can be set after an object is constructed

*The official specification documentation is the **second** point of entry for understanding the Python API*



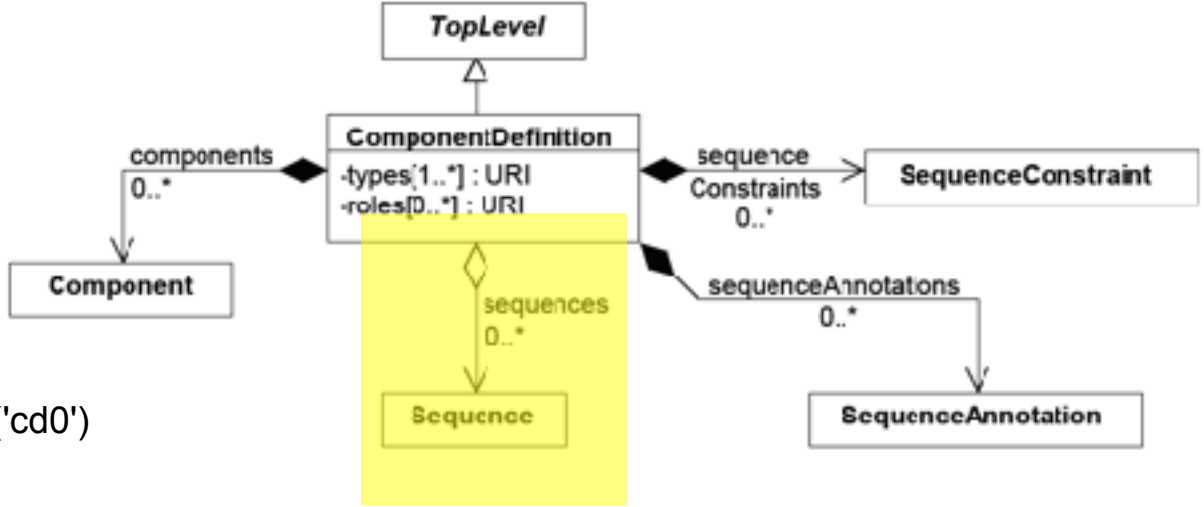
Cardinality dictates if property values are returned as a list versus a singleton value

Note: some properties are initialized with a default value



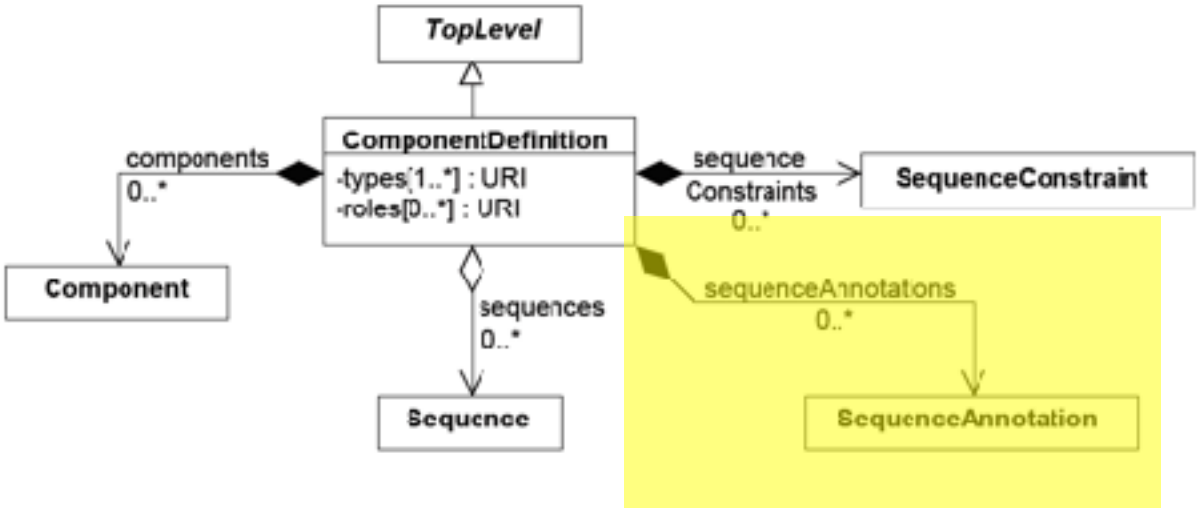

```
>>> from sbol import *
>>> cd0 = ComponentDefinition('cd0')
>>> print(cd0.types)
['http://www.biopax.org/release/biopax-level3.owl#DnaRegion']
>>> print(cd0.roles)
[]
```

An open diamond indicates the property contains URI(s)



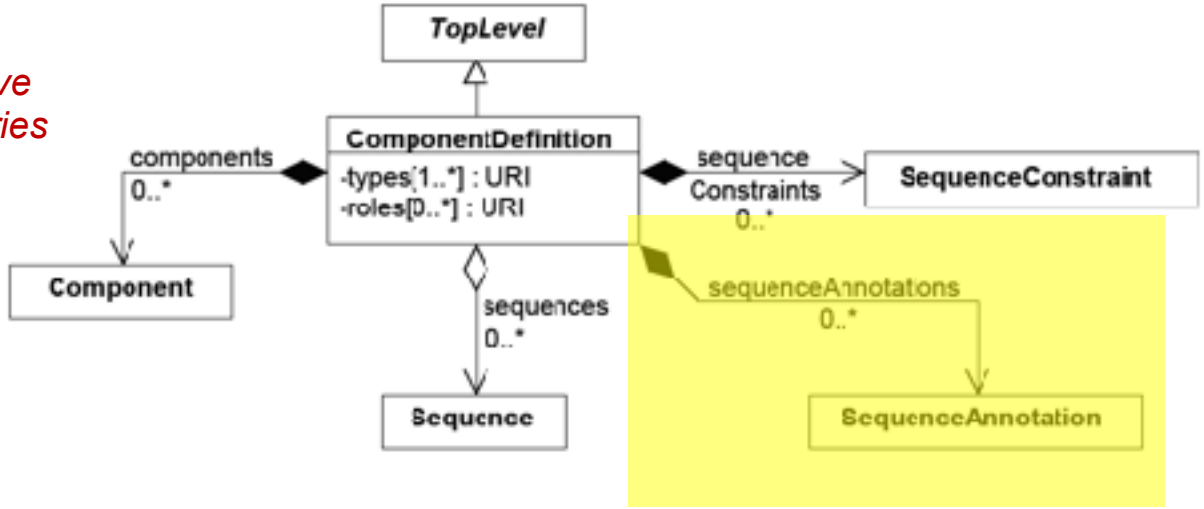

```
>>> cd0.sequences = Sequence('cd0')
>>> cd0.sequences
['http://examples.org/Sequence/cd0/1']
```

A closed diamond indicates c
ownership (i.e., composition)




```
>>> cd0 = ComponentDefinition('cd0')
>>> sa = cd0.sequenceAnnotations.create('sa')
```

Compositional properties behave somewhat like Python dictionaries

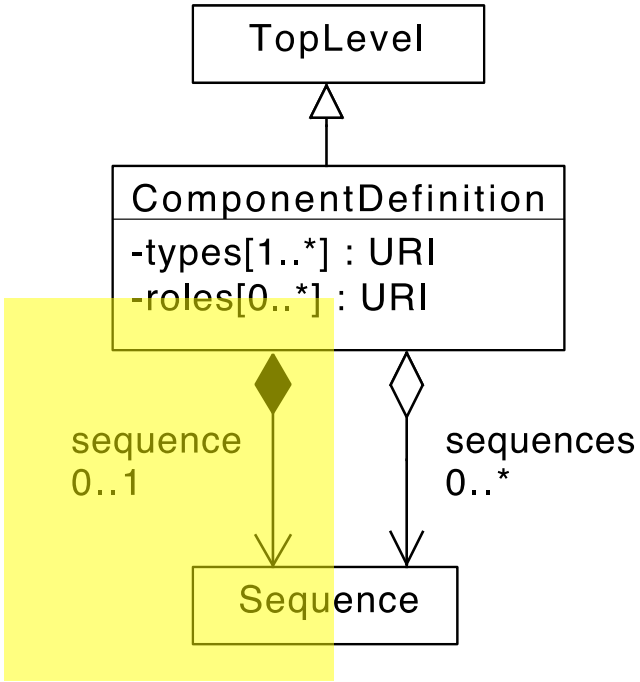



```
>>> cd0.sequenceAnnotations['sa'] = SequenceAnnotation('sa')   # same as create method
>>> cd0.sequenceAnnotations['sa']
SequenceAnnotation
>>> cd0.sequenceAnnotations['sa'].identity
'http://examples.org/ComponentDefinition/cd0/sa/1'
```

The pySBOL API includes shortcuts that are NOT specified by the data model.

For these cases, the examples in the pySBOL documentation are the best reference.

```
>>> cd.sequence = Sequence('seq')
>>> cd.sequences
['http://examples.org/Sequence/seq/1']
```



Secure | https://pysbol2.readthedocs.io/en/latest/API.html

pySBOL
latest

Search docs

Read the Docs v: latest

Docs » API

Edit on GitHub

The pySBOL API auto-documentation is not great…

API

class **Activity**(**args*) [source]

A generated Entity is linked through a wasGeneratedBy relationship to an Activity, which is used to describe how different Agents and other entities were used. An Activity is linked through a qualifiedAssociation to Associations, to describe the role of agents, and is linked through qualifiedUsage to Usages to describe the role of other entities used as part of the activity. Moreover, each Activity includes optional startedAtTime and endedAtTime properties. When using Activity to capture how an entity was derived, it is expected that any additional information needed will be attached as annotations. This may include software settings or textual notes. Activities can also be linked together using the wasInformedBy relationship to provide dependency without explicitly specifying start and end times.

- *startedAtTime : DateTimeProperty*
- *endedAtTime:DateTimeProperty*

  The endedAtTime property is OPTIONAL and contains a dateTime (see section Section 12.7) value, indicating when the activity ended.

- *wasInformedBy:ReferencedObject*

  The wasInformedBy property is OPTIONAL and contains a URI of another activity.

- *associations:OwnedObject< Association >*

  The qualifiedAssociation property is OPTIONAL and MAY contain a set of URIs that refers to Association

synbiodes.github.io/libSBOL/index.html

libSBOL 2.3.0

Introduction | Installation | Getting Started with SBOL | Sequence Assembly | Biosystem Design | Classes | Files

Search

libSBOL Documentation

libSBOL 2.3.0

libSBOL is a C++ library for reading, writing, and constructing genetic designs according to the standardized specification of the Synthetic Biology Open Language (SBOL).

INSTALLATION

To install go to Installation page.

PLATFORMS

libSBOL is available for Windows, Mac OSX, and Linux. The library is tested on Windows 7+, Mac OSX 10.9+, and Ubuntu 14.04+.

ACKNOWLEDGEMENTS

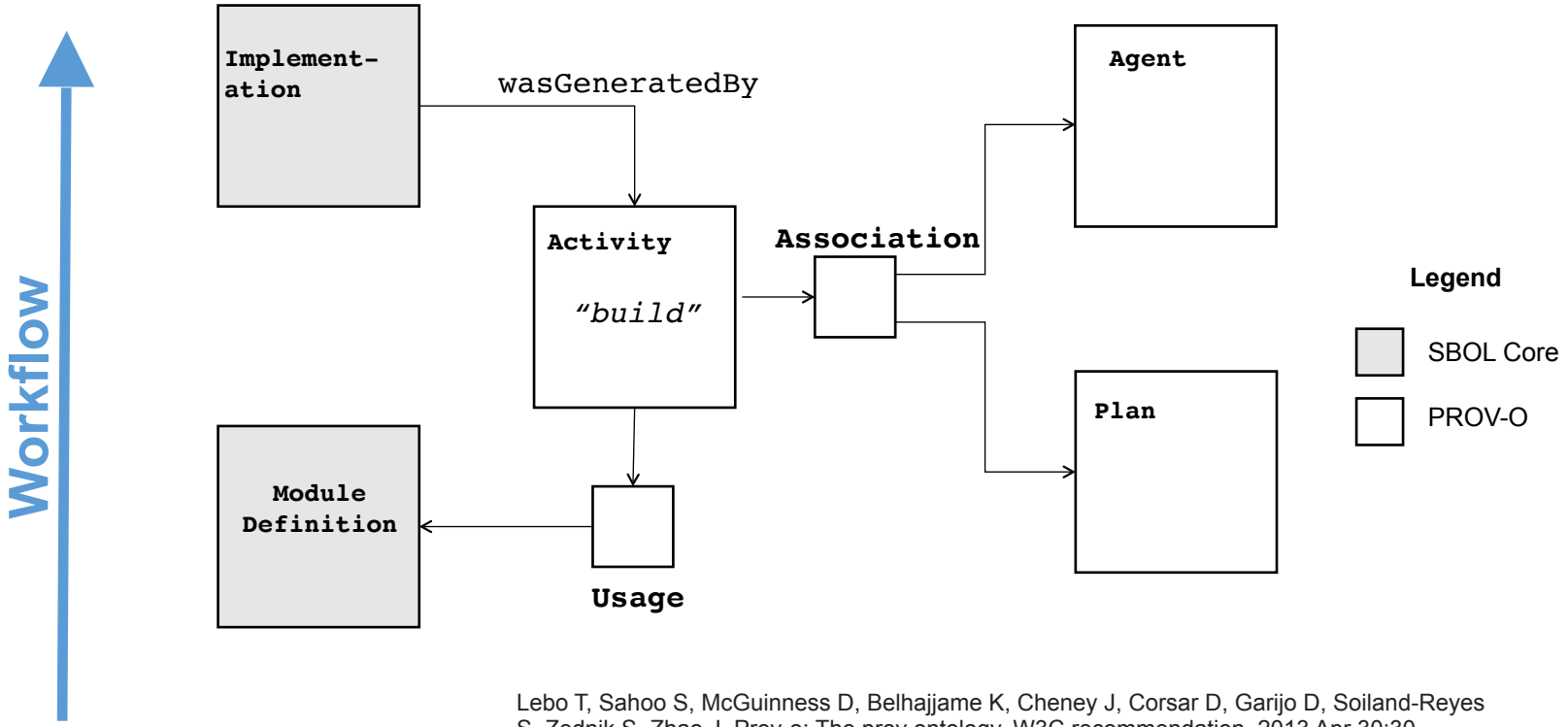
Current support for the development of libSBOL is generously provided by the NSF through the Synthetic Biology Open Language Resource collaborative award.

Generated on Wed Aug 1 2018 12:48:17 for libSBOL by doxygen 1.8.14

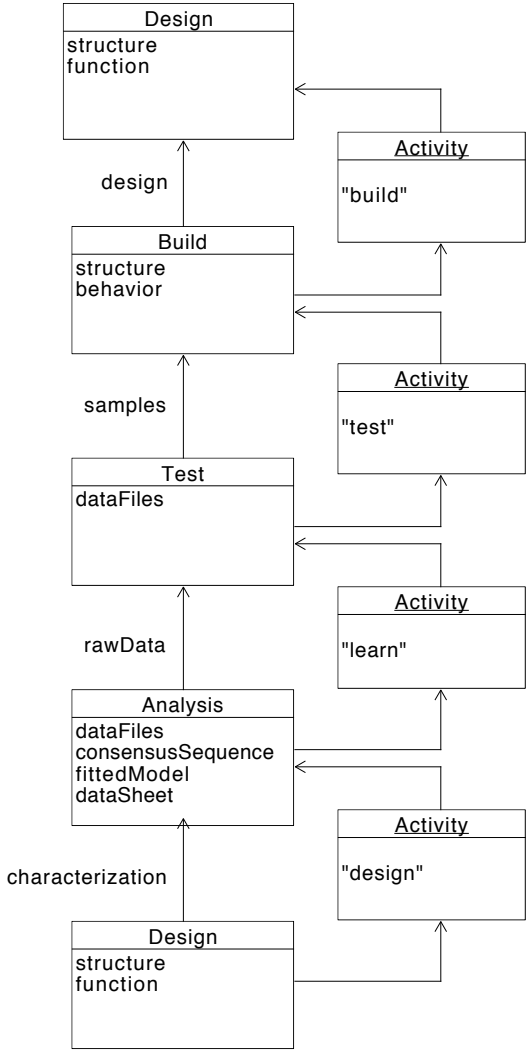
Refer back to the libSBOL API documentation for clarification

DBTL Workflows

SBOL Leverages the Provenance Ontology (PROV-O)



Lebo T, Sahoo S, McGuinness D, Belhajjame K, Cheney J, Corsar D, Garijo D, Soiland-Reyes S, Zednik S, Zhao J. Prov-o: The prov ontology. W3C recommendation. 2013 Apr 30;30.



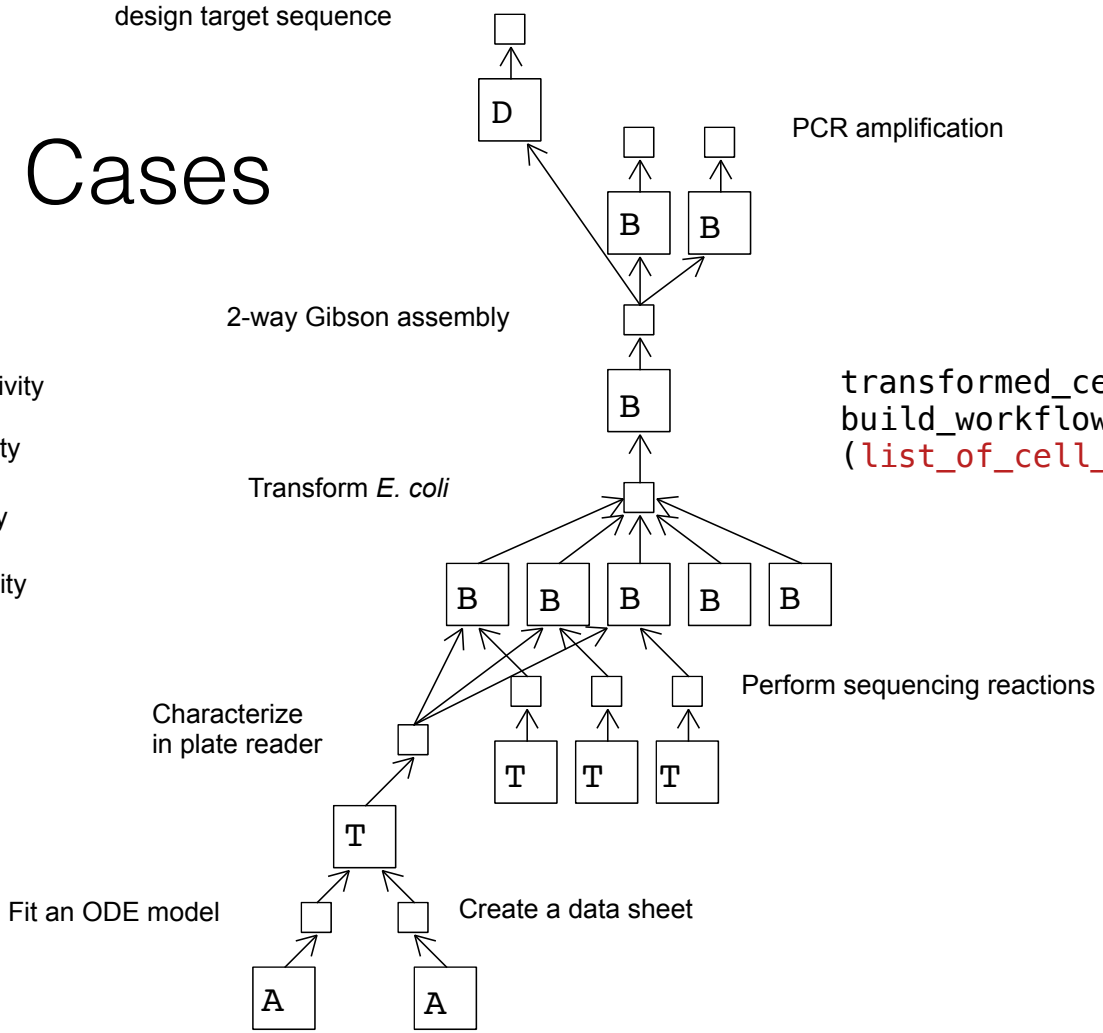

```
transformed_cells =
build_workflow_step.generateBuild('transformed_cells', design)
```

The pySBOL API provides helper classes to simplify understanding of SBOL's provenance rules.

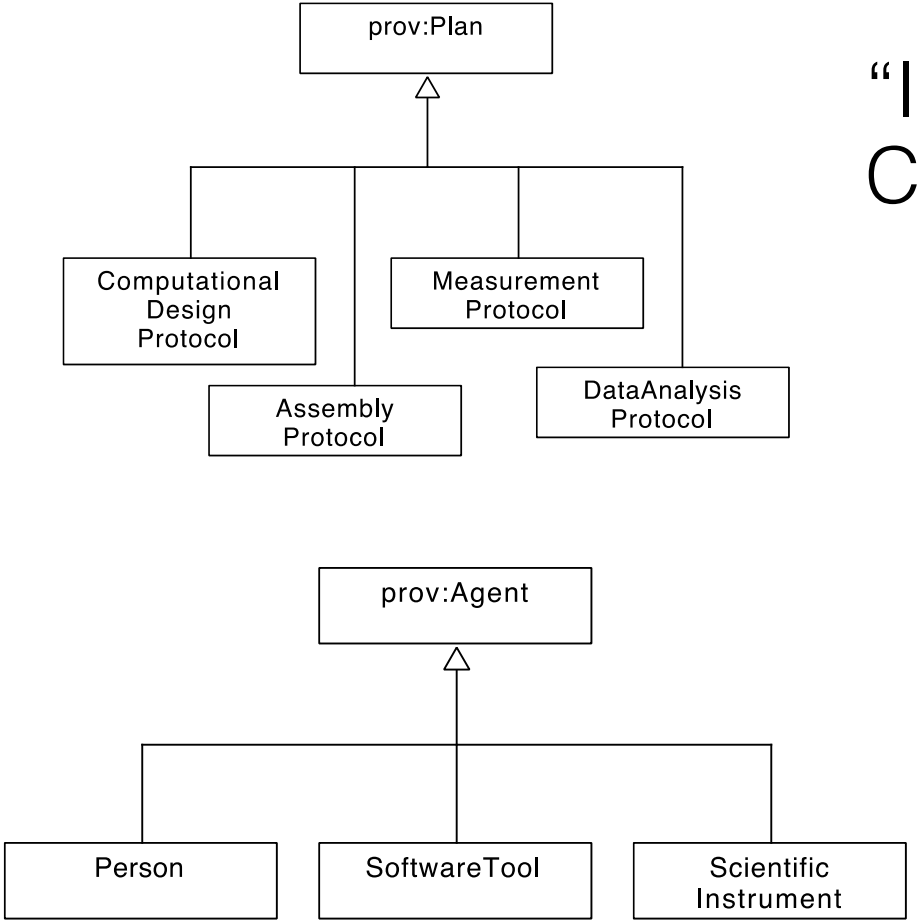
These classes are used in today's tutorial.

Use Cases

- *Design* Activity
- *Build* Activity
- *Test* Activity
- *Learn* Activity




```
transformed_cells =
build_workflow_step.generateBuild
(list_of_cell_ids, None, gibson_mix)
```



“In-house” Workflow Systems Can be Integrated with SBOL

- SBOL is an “extensible standard”
- SBOL libraries provide annotation and extension mechanisms
- Existing, “in-house” workflow systems can be linked to SBOL using these mechanisms

Extensibility

Custom Annotation Data

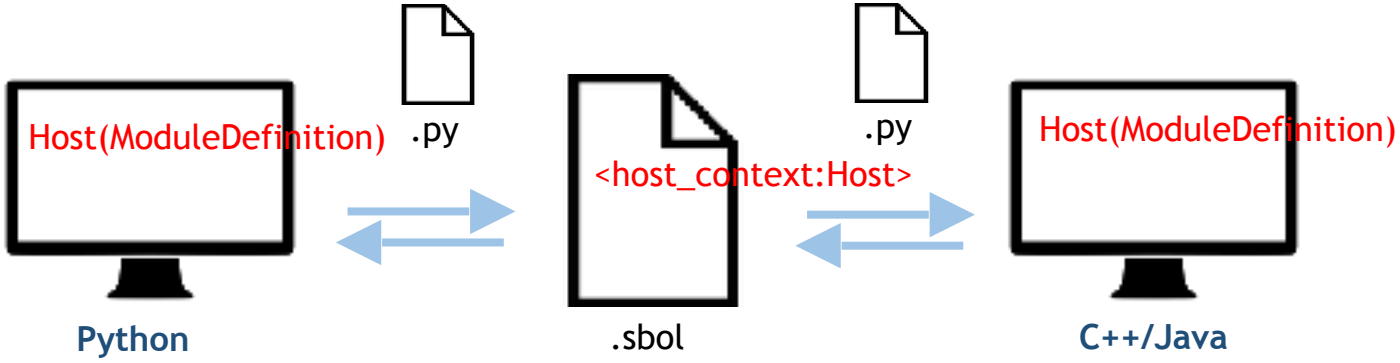
```
>>> cd = ComponentDefinition('cd0')
>>> annotation = TextProperty(cd, 'http://sys-bio.org#annotationProperty', '0', '1')
>>> annotation.set('This is a test property')
>>> annotation.get()
'This is a test property'
```


```
<?xml version="1.0" encoding="utf-8"?>
<rdf:RDF xmlns:dcterms="http://purl.org/dc/terms/"
   xmlns:prov="http://www.w3.org/ns/prov#"
   xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
   xmlns:sbol="http://sbols.org/v2#"
   xmlns:sys-bio="http://sys-bio.org#">
  <sbol:ComponentDefinition rdf:about="http://examples.org/
ComponentDefinition/cd0/1">
    <sbol:displayId>cd0</sbol:displayId>
    <sbol:persistentIdentity rdf:resource="http://examples.org/
ComponentDefinition/cd0"/>
    <sbol:type rdf:resource="http://www.biopax.org/release/biopax-
level3.owl#DnaRegion"/>
    <sbol:version>1</sbol:version>
    <sys-bio:annotationProperty>This is a test property</sys-
bio:annotationProperty>
  </sbol:ComponentDefinition>
</rdf:RDF>
```

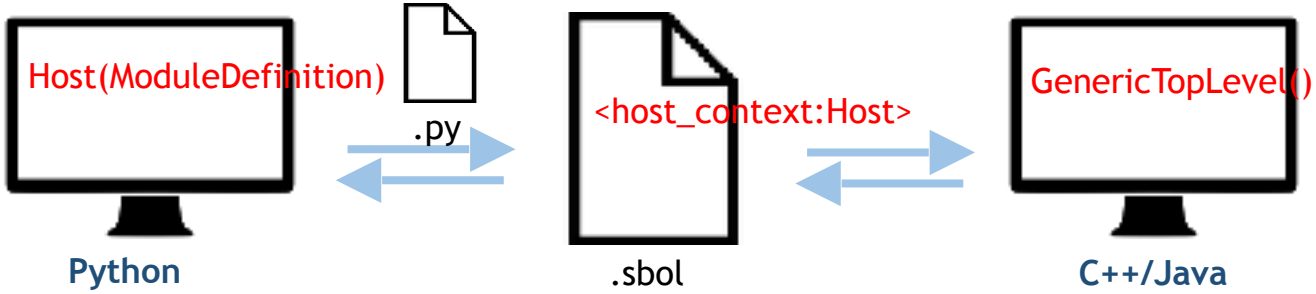
Extension Classes

```
DPL_NS = 'http://dnaplotlib.org#'

class ModuleDefinitionExtension(ModuleDefinition):
    def __init__(self, id = 'example'):
        ModuleDefinition.__init__(self, id)
        self.x_coordinate = TextProperty(self, DPL_NS + 'xCoordinate, '0', '1', '10')
        self.y_coordinate = IntProperty(self, DPL_NS + 'yCoordinate', '0', '1', 10)


doc = Document()
doc.addNamespace('http://dnaplotlib.org#', 'dnaplotlib')
md = ModuleDefinitionExtension('md_example')
print (md.x_coordinate)
md.y_coordinate = 5
print (md.y_coordinate)
```

1 Data exchange with pySBOL extension classes

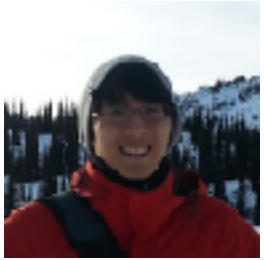


2 Data exchange with generic annotations



Concluding Remarks

Acknowledgements



Kiri Choi
PhD Student
UW

Kyle Medley
PhD Student
UW

Herbert Sauro,
Associate Professor of Bioengineering
UW

Anil Wipat and the Newcastle team

Chris Myers and the Utah team

The SBOL Editors

NSF award #1355909

Please try out!