SBOL Tutorial

Bryan Bartley
bartleyba@sbolstandard.org
Raytheon-BBN Technologies
Cambridge, MA
Introduction to SBOL Data Exchange Standard
SBOL Integrates Data Across the Semantic Web

Every SBOL data object is considered a “resource” with a uniform resource identifier (URI).
Anatomy of SBOL

Human-readable

Synthetic Biology Object Model

XML Document Object Model

Resource Description Framework

Machine-readable

RDF
Python (pySBOL) & C++ (libSBOL) Libraries
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!)
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
The examples in the online documentation is the first point of entry for understanding the Python API.
Guiding Philosophy for Library Development

- 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

http://sbolstandard.org/data-model-specification/
Every SBOL Object has a Uniform Resource Identifier (URI).

“SBOL Compliant” URIs

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

Typed URIs

http://sys-bio.org/SBOLClass/my_design/1
An Example Constructor

Sets default namespace for URI generation

```python
>>> 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

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

```python
>>> 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

```python
>>> 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.

```python
>>> 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)

```python
>>> cd0.sequences = Sequence('cd0')
>>> cd0.sequences
['http://examples.org/Sequence/cd0/1']
```
A closed diamond indicates object ownership (i.e., composition)

```python
>>> cd0 = ComponentDefinition('cd0')
>>> sa = cd0.sequenceAnnotations.create('sa')
```
Compositional properties behave somewhat like Python dictionaries

```python
>>> 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.

```python
>>> cd.sequence = Sequence('seq')
>>> cd.sequences
['http://examples.org/Sequence/seq/1']
```
The pySBOL API auto-documentation is not great…

API

```python
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.
Refer back to the libSBOL API documentation for clarification
DBTL Workflows
SBOL Leverages the Provenance Ontology (PROV-O)

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
- Transform E. coli
- Characterize in plate reader

Build Activity
- 2-way Gibson assembly
- PCR amplification
- Transformed cells = build_workflow_step.generateBuild(list_of_cell_ids, None, gibson_mix)

Test Activity
- Perform sequencing reactions

Learn Activity
- Fit an ODE model
- Create a data sheet
“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

```python
>>> 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'
```
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)
print (md.y_coordinate = 5)
print (md.y_coordinate)
1. Data exchange with pySBOL extension classes

- Python
  - Host(ModuleDefinition).py
  - .sbol

- C++/Java
  - Host(ModuleDefinition).py
  - .sbol

2. Data exchange with generic annotations

- Python
  - Host(ModuleDefinition).py
  - .sbol

- C++/Java
  - GenericTopLevel().py
  - <host_context:Host>
Concluding Remarks
Acknowledgements

Kiri Choi
PhD Student
UW

Kyle Medley
PhD Student
UW

Anil Wipat and the Newcastle team

Chris Myers and the Utah team

Herbert Sauro,
Associate Professor of Bioengineering
UW

The SBOL Editors

NSF award #1355909

NSF
Please try out!