libSBOL & pySBOL

Bryan Bartley
bartleyba@sbolstandard.org
The Sauro Systems & Synthetic Biology Lab
University of Washington, Seattle, WA

LibSBOL

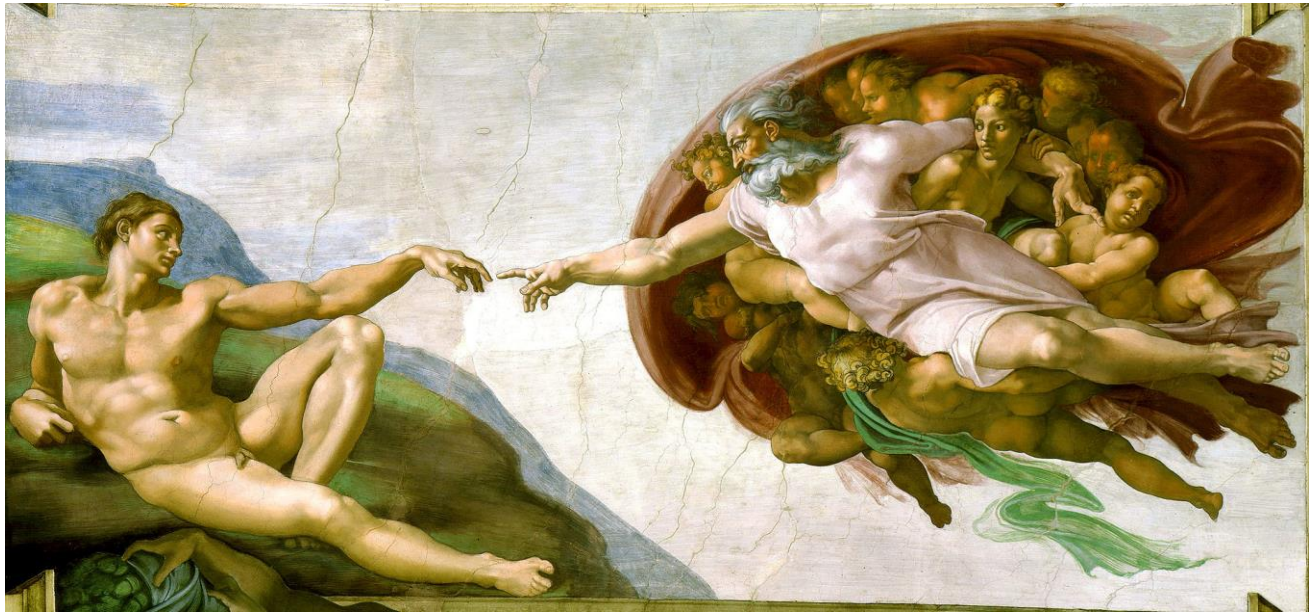
- C++ beta release on SynBioDex/libSBOL
- Cross-platform on MacOSX and Windows. Uses CMake meta-build tool
- SWIG-Python bindings on SynBioDex/pySBOL2 (uses setuptools for installation). Not yet available on PyPI
- Getting Started Tutorial & API reference
  http://synbiodex.github.io/libSBOL
- Sequence Assembly & Biosystem Design Tutorials
  http://synbiodex.github.io/libSBOL/sequences.html
  http://synbiodex.github.io/libSBOL/modular_design.html

Guiding Philosophy

- User-experience: An object-oriented approach to synthetic biology
- LibSBOL supports both open-world and SBOL-compliant object creation
- Library implementation and specification document are intuitively correlated
- Extensibility

User Experience

Building Designs with SBOL Should Be a Joyful Act of Creation



LibSBOL = Object-oriented Synthetic Biology

Synthetic biologists need to re-use and share biological components in the same way that programmers re-use and share software components. Synthetic biology needs an OOP framework.

LibSBOL's API uses a terse, imperative, noun-verb syntax. Think about the Data Model first, then act.

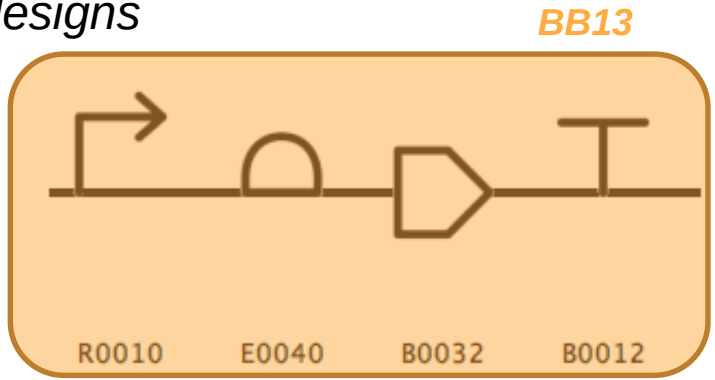
`EYFP_production.participations.create` **(C++)**

vs

`EYFP_production.createParticipation` **(Java)**

Automation of High-level Design Tasks

- ComponentDefinition::assemble *Assemble hierarchical DNA constructs; assemble template designs*

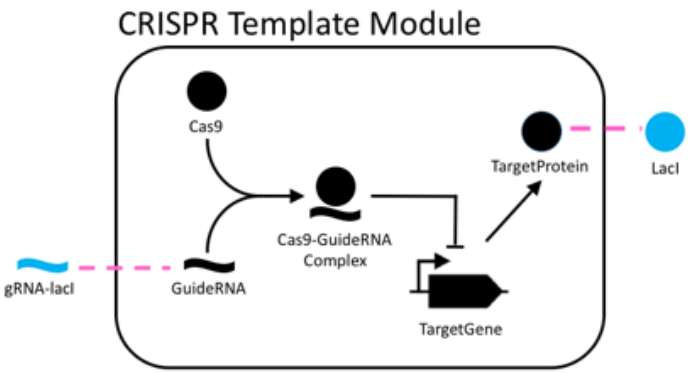


- Sequence::assemble *Stitch together DNA sequences from different parts; replace cut-and-paste*

- ModuleDefinition::assemble *Assembly of layered, regulatory gates and modular systems*

Other High-level Design Tasks in Today's Tutorial

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



LibSBOL Supports Both Open-world and SBOL-compliant Object Creation

SBOL Objects are Uniquely Identified by Uniform Resource Identifiers (URI).

For purposes of today's tutorial URIs consist of a scheme, a namespace, and an identifier.

A common scheme used is **http://**. An example namespace is **sys-bio.org**. And an example identifier might be **my_design**.

Thus, the complete URI is **http://sys-bio.org/my_design**

SBOL-Compliant Mode Simplifies URI Generation

- Automates and simplifies URI construction.
- Only top level objects (eg, ComponentDefinition, ModuleDefinition) use constructors; all other objects are constructed using create() methods.
- Object creation and manipulation is closely tied to an SBOL Document. Implicitly assumes that data exchange is primary motivation for using the Data Model
- **SBOL-compliance is enabled in libSBOL by default**

LibSBOL Also Supports a More “Open-world” Approach to Object Creation

- Few restrictions placed on URIs.
- Constructors are primary means of creating objects.
- Objects can be manipulated independent from an SBOL Document.
- Emphasizes other uses of Data Model for knowledge representation and computational biology. See my upcoming talk *Version and Variant Control for Synthetic Biology* at COMBINE 2016
- **For today’s tutorial, open-world approach should be avoided**!

An Example Constructor

Sets default namespace for URI generation

```
setHomespace("http://sys-bio.org");
ComponentDefinition& TargetPromoter = *new ComponentDefinition
    ("TargetPromoter", BIOPAX_DNA);
TargetPromoter.roles.set(SO_PROMOTER)
```

An Example Constructor

The identifier for the URI.
In SBOL-compliant mode, this also sets the object's displayId

```
setHomespace("http://sys-bio.org");
ComponentDefinition& TargetPromoter = *new ComponentDefinition
    ("TargetPromoter", BIOPAX_DNA)
TargetPromoter.roles.set(SO_PROMOTER)
```

An Example Constructor

After the identifier, comes 0 or more required fields. For ComponentDef's molecular type is required

```
setHomespace("http://sys-bio.org");
ComponentDefinition& TargetPromoter = *new ComponentDefinition
    ("TargetPromoter", BIOPAX_DNA);
TargetPromoter.roles.set(SO_PROMOTER)
```

An Example Constructor

```
setHomespace("http://sys-bio.org");
ComponentDefinition& TargetPromoter = *new ComponentDefinition
    ("TargetPromoter", BIOPAX_DNA);
TargetPromoter.roles.set(SO_PROMOTER)
```

Optional fields are set after construction

An Example Constructor

```
setHomespace("http://sys-bio.org");
ComponentDefinition& TargetPromoter = *new ComponentDefinition
    ("TargetPromoter", BIOPAX_DNA);
TargetPromoter.roles.set(SO_PROMOTER)
```

Some programmer's might regard this syntax as bad form.

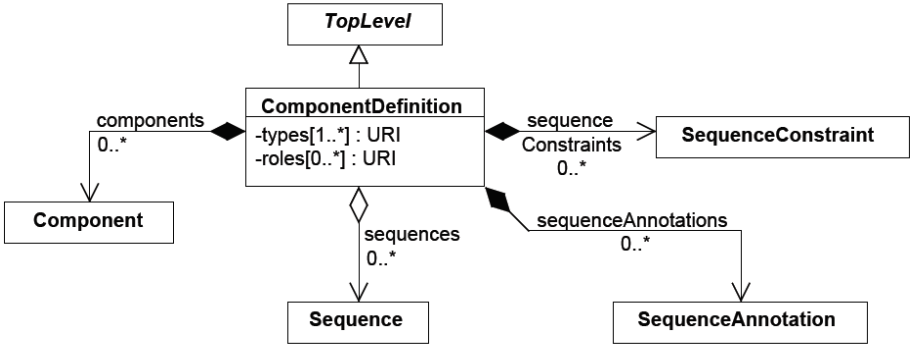
For a justification why I use this syntax, see:
http://synbiodex.github.io/libSBOL/getting_started.html#idiomatic_cpp

Then send hate-mail to sbol-editors@googlegroups.com!

Library Implementation and Specification Document are Intuitively Correlated

From Specification Diagram to Class Definition

LibSBOL's API is built around basic accessor methods and RDF triple serialization methods at the level of SBOL Properties, not SBOL Classes.




```
namespace sbol
{
class ComponentDefinition : public TopLevel
{
public:
  List<URIProperty> types;
  List<URIProperty> roles;
  ReferencedObject sequence;
  List<OwnedObject<SequenceAnnotation>> sequenceAnnotations;
  List<OwnedObject<Component>> components;
  List<OwnedObject<SequenceConstraint>> sequenceConstraints;

…
}
```

Validation Rules Cross-Reference to the Spec Document and are Easily Extensible

sbol-10202 ▲ The `identity` property of an `Identified` object MUST be globally unique.
Reference: Section 7.4 on page 16

```
Identified(sbol_type type = UNDEFINED, std::string uri_prefix = SBOL_URI,
std::string id = “Identified/example”) :

        identity(SBOL_IDENTITY, this, uri_prefix + "/" + id, ”,
            validation_rules = { sbol_rule_10202 })
        {
        };
```

Easy to Implement Future Enhancements in the SBOL Data Model

```
class Component : public ComponentInstance
{
public:
```

SBOL 2.0.0

```
    Component(std::string uri_prefix = SBOL_URI "/Component", std::string id = "example", std::string access
        Component(SBOL_COMPONENT, uri_prefix, id, access)
        {
        };
    ~Component() {};

protected:
    Component(sbol_type type, std::string uri_prefix, std::string id, std::string access) :
        ComponentInstance(type, uri_prefix, id, access)
        {
        };
};
```

Easy to Implement Future Enhancements in the SBOL Data Model

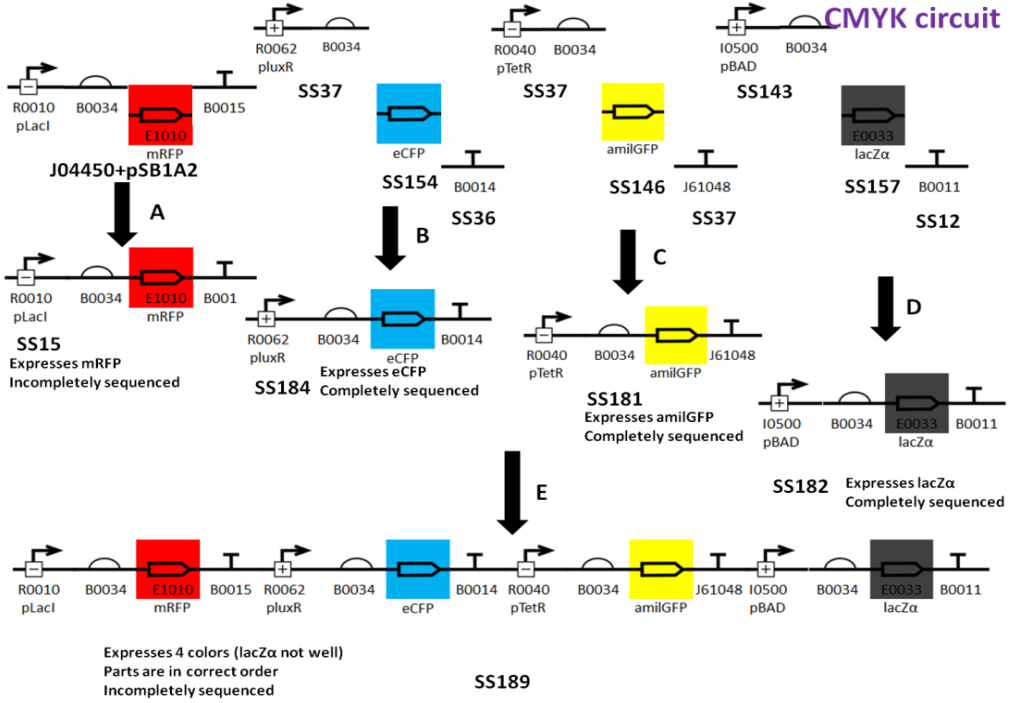
```
class Component : public ComponentInstance
{
public:
    URIProperty role;

    Component(std::string uri_prefix = SBOL_URI "/Component", std::string id = "example", std::string access
        Component(SBOL_COMPONENT, uri_prefix, id, access)
        {
        };
    ~Component() {};

protected:
    Component(sbol_type type, std::string uri_prefix, std::string id, std::string access) :
        ComponentInstance(type, uri_prefix, id, access)
        {
        };
};
```

SBOL 2.0.1

Easy to Write Application-Specific Annotations and Custom Extensions



Host Context Extension

```
#define EXTENSION_PREFIX "host_context"
#define EXTENSION_NS "sys-bio.org/HostContext#"
#define EXTENSION_CLASS "Host"
class Host : public ModuleDefinition
{
  Host(sbol_type type, std::string uri) :
    ModuleDefinition(type, uri),
    modules(EXTENSION_NS "modules", this),
    parents(EXTENSION_NS "parents", this),
    children(EXTENSION_NS "children", this),
    generation(EXTENSION_NS "generation", this, 1),
    medium(EXTENSION_NS "medium", this, "www.ebi.ac.uk/efo/EFO_0000579"),
    vendorId(EXTENSION_NS "vendorId", this, "sigmaaldrich.com/L2542")
  {
    register_extension < Host > (EXTENSION_PREFIX, EXTENSION_NS EXTENSION_CLASS);
  };
}
```

Host Context Serialization


```xml
<?xml version="1.0" encoding="utf-8"?>
<rdf:RDF xmlns:dcterms="http://purl.org/dc/terms/#"
  xmlns:host_context="sys-bio.org/HostContext#"
  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#">
 <host_context:Host rdf:about="sys-bio.org/BB1">
  <sbol:persistentIdentity rdf:resource="sys-bio.org/BB1"/>
  <host_context:generation>1</host_context:generation>
  <host_context:medium rdf:resource="www.ebi.ac.uk/efo/EFO_0000579"/>
  <host_context:vendorId>sigmaaldrich.com/L2542</host_context:vendorId>
 </host_context:Host>
 <sbol:ModuleDefinition rdf:about="sys-bio.org/CRISPRTemplate">
  <sbol:persistentIdentity rdf:resource="sys-bio.org/CRISPRTemplate"/>
 </sbol:ModuleDefinition>
 <sbol:ModuleDefinition rdf:about="sys-bio.org/CRPbCircuit">
  <sbol:persistentIdentity rdf:resource="sys-bio.org/CRPbCircuit"/>
 </sbol:ModuleDefinition>
</rdf:RDF>
```

Concluding Remarks

pySBOL API

- Generated from libSBOL using Simplified Wrapper and Interface Generator (SWIG)
- API closely resembles C++ API with a few exceptions:
  - No new operator required for object constructions
  - No templated methods, thus:

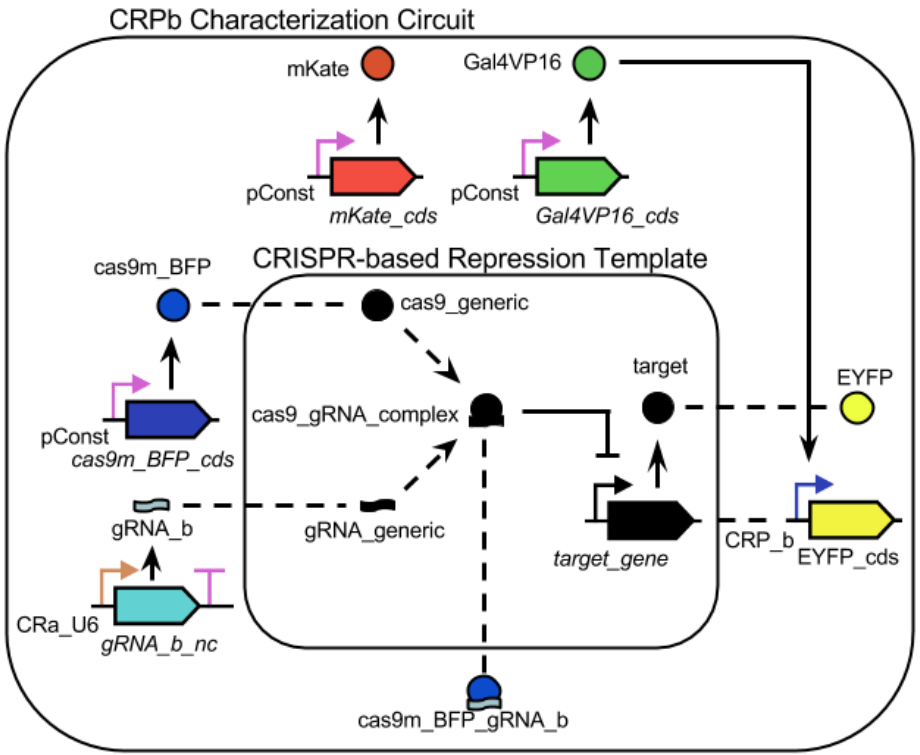
    doc.get<ComponentDefinition>() becomes doc.getComponentDefinition()

  - Use lists instead of vectors
- For today's tutorial, if working in Python, use the C++ API reference and methods

To Do

- `Pythonize' the pySBOL interface. To see what's possible, try pySBOL 1
- Implement validation rules
- Write tutorial for extension developers

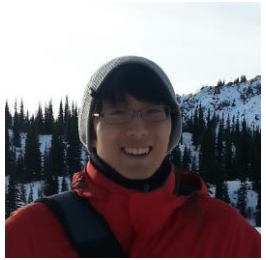
Tutorial Assignment:
CRISPR Characterization Circuit



Please try out!

Acknowledgements

Kyle Medley
PhD Student



Kiri Choi
PhD Student

Herbert Sauro,
Associate Professor of Bioengineering

Anil Wipat and the Newcastle team

Chris Myers and the Utah team

The SBOL Editors

NSF Collaborative awards #1355909