

# libsbml

## An Overview and Brief Tutorial

Benjamin J. Bornstein<sup>1,2</sup>

<sup>1</sup>Jet Propulsion Laboratory, Machine Learning Systems

<sup>2</sup>California Institute of Technology

Presented at the 7<sup>th</sup> Forum on Software Platforms for Systems Biology

# libsbml :: Outline

- Overview
  - What is libsbml?
  - Motivations
  - Features Highlights
- Tutorial
  - Installation
  - SBML Classes in C
  - Primitive Types
  - Fields
  - Lists
  - Reading
  - Writing
  - Formulas

# libsbml :: Overview

- Library of functions to read, write, validate and manipulate SBML
- C data-binding for SBML
- Aims to take care of nuances of reading and writing SBML so you concentrate on more important things.

# libsbml :: Motivations

- Community needs (at least one) lightweight and lean SBML library
- Such a library could serve as a:
  - Reference implementation
  - Basis for future SBW services, e.g. new Validator, improved NOM, various translators, etc.
  - Basis for high-level language bindings

# libsbml :: Features

- Small Memory Footprint
  - Event-based (SAX), no intermediate DOM
  - C structs and enums mirror SBML specification
- Fast Runtime
  - Gepasi's 2Mb `100Yeast.xml` with 101 Compartments, 2K Reactions, etc...  
[1.18s, 1.4Mb memory]

# libsbml :: Features

- Portable
  - ANSI C
  - Autoconf for Linux, Solaris, Cygwin and MacOS X
  - Windows (precompiled) DLL and LIB
- Full SBML Support
  - Level 1, Level 2 (all but RDF and Writing)
  - Supports `<specie*>` or `<species*>`
  - Accepts `<annotation>` or `<annotations>`
  - Allows `<notes>` in top-level `<sbml>` in L1

# libsbml :: Features

- XML Schema Validation
  - Configurable schema location
  - Logs warnings, errors and fatal errors
  - Validates the schema itself (optional)
- Full Unicode Support
  - Writes ASCII, UTF-8, UTF-16, ISO 8859-1
  - Handles troublesome Windows 1252

# libsbml :: Features

- Well tested
  - Backed by 311 unit tests comprised of 1681 individual assertions
  - Regression test suite of 10K SBML files (KEGG, SBML Model repository, KGI repository, Gepasi export, etc.)
  - Custom memory tracing facility reports no leaks

# libsbml :: Installation

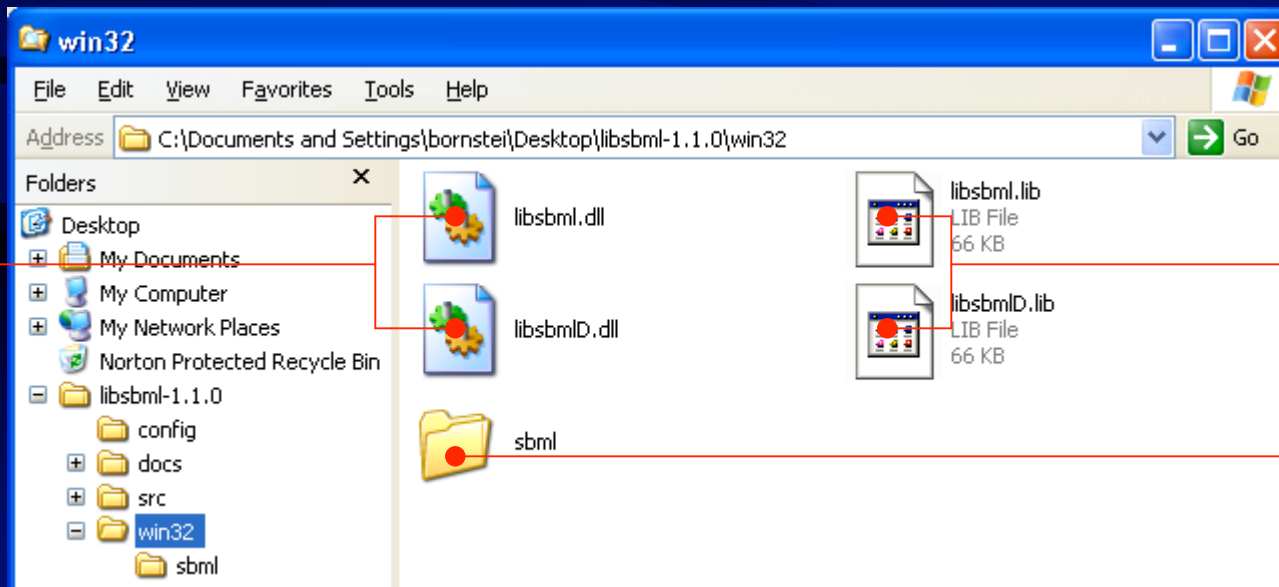
- Requires Xerces-C++ 2.1.0 or greater
  - <http://xml.apache.org/xerces-c/>
  - Debian and RedHat packages available
- Linux, Solaris, Cygwin, MacOS X

```
% tar zxvf libsbml-1.1.0.tar.gz
% cd libsbml-1.1.0/
% ./configure --prefix=/my/favorite/path
% make
% make check (optional)
% make install
```

# libsbml :: Installation

- Windows
  - Unzip `libsbml-1.1.0.zip`
  - Link as Multithreaded DLL

Release  
&  
Debug  
.DLL



Release &  
Debug .LIB

Header  
files

# libsbml :: SBML Classes in C

1. SBML classes map to C structs  
C “objects” are pointers to specific structs in memory.
2. Method (function) names begin with the SBML class, followed by an underscore
3. Methods take the object (pointer to struct) receiving the method as their first argument.
4. Constructors and destructors  
Similar to method names, but end in `create()` and `free()`, respectively.

# libsbml :: SBML Classes in C

---

SBML Class	C Class (typedef struct)
<i>SBase</i>	<code>SBase_t</code>
Model	<code>Model_t</code>
UnitDefinition	<code>UnitDefinition_t</code>
Unit	<code>Unit_t</code>
Compartment	<code>Compartment_t</code>
Parameter	<code>Parameter_t</code>
Species	<code>Species_t</code>
...	...

---

# libsbml :: SBML Classes in C

---

SBML Enumeration

C Enumeration  
(typedef enum)

---

UnitKind

UnitKind\_t

RuleType

RuleType\_t

---

# libsbml :: Primitive Types

1. UML attribute names are preserved
2. SName maps to a standard C string, `char *`
3. double and integer map to C `double` and `int`
4. boolean maps to C `int`
  - zero represents false and non-zero represents true.

# libsbml :: Primitive Types

```
typedef struct
{
    char    *name;
    char    *compartment;
    double  initialAmount;
    char    *units
    int     boundaryCondition;
    int     charge;
} Species_t;
```

## Species

```
name : SName
compartment : SName
initialAmount : double
units : SName {use="optional"}
boundaryCondition : boolean {use="optional" default="false"}
charge : integer {use="optional"}
```

# libsbml :: Object Creation and Destruction

```
Species_t *  
Species_create (void)
```

```
Species_t *  
Species_createWith (const char *name, ...)
```

```
Species_t *  
Species_free (Species_t *s)
```

# libsbml :: Fields

- Direct Access

```
s->initialAmount
```

- Getters

– General form: `XXX_getYYY (const XXX_t *x)`

```
double  
Species_getInitialAmount (const Species_t *s)
```

# libsbml :: Fields

- Setters
  - Always use
  - Guards against memory leaks for string fields
  - Tracks set vs. unset field state
  - General form: `XXX_setYYY(XXX_t *x, ...)`

```
void
Species_setInitialAmount (Species_t *s, double value)

void
Species_setName (Species_t *s, const char *sname)
```

# libsbml :: Fields

- Set vs. Unset Field State
  - Tracked for each optional field without a default value
  - General forms:
    - `XXX_isSetYYY(const XXX_t *x)`
    - `XXX_unsetYYY(XXX_t *x)`

```
int
Species_isSetInitialAmount (const Species_t *s)

int
Species_unsetUnits (Species_t *s)
```

# libsbml :: Lists

- Three Methods
  - General Forms:

```
XXX_addYYY(XXX_t *x, YYY_t *y)
```

```
XXX_getYYY(const XXX_t *x, unsigned int n)
```

```
XXX_getNumYYY(const XXX_t *x)
```

```
void  
Reaction_addReactant ( Reaction_t *r,  
                      SpeciesReference_t *sr )
```

```
/* And similarly for products */
```

# libsbml :: Reading SBML

- From File

```
SBMLDocument_t *d = readSBML("100Yeast.xml");
```

- From String

```
char *s;  
SBMLDocument_t *d;  
  
/* Let s point to some string containing SBML... */  
  
d = readSBMLFromString(s);
```

# libsbml :: Reading SBML

## XML Schema Validation

```
SBMLReader_t    *sr = SBMLReader_create();
SBMLDocument_t *d;

/* XML_SCHEMA_VALIDATION_BASIC == 1 */
SBMLReader_setSchemaValidationLevel(sr, 1);
SBMLReader_setSchemaFilename("sbml-11v1.xsd");

d = SBMLReader_readSBML(sr, "11v1-branch.xml");

SBMLDocument_printErrors(d, stderr);
```

# libsbml :: Reading SBML


```
typedef struct
{
    SBASE_FIELDS;

    unsigned int level;
    unsigned int version;

    List_t *error;
    List_t *fatal;
    List_t *warning;

    Model_t *model;
} SBMLDocument_t;
```

```
typedef struct
{
    char          *message;
    unsigned int  line;
    unsigned int  column;
} ParseMessage_t;
```



# libsbml :: Writing SBML

- To File

```
writeSBML(d, "my-model.xml");
```

- To String

```
/* This is a strange way to my-model.xml into  
a string. */
```

```
char *s;
```

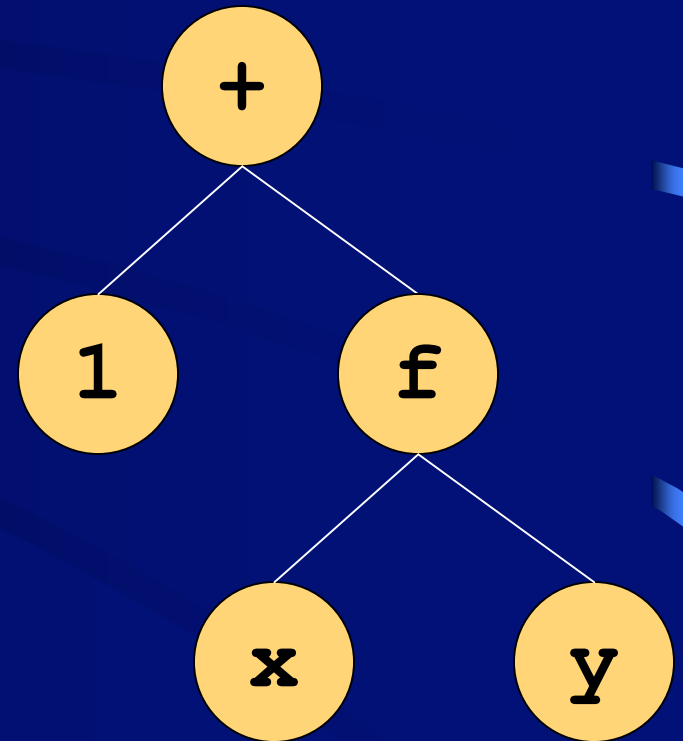
```
SBMLDocument_t *d = readSBML("my-model.xml");
```

```
s = writeSBMLToString(d);
```

# libsbml :: Formulas

- Abstract Syntax Trees (AST)

```
ASTNode_t *n =  
SBML_parseFormula("1+f(x,y)");  
  
ASTNode_getType(n);  
ASTNode_getName(n);  
ASTNode_getInteger(n);  
ASTNode_getReal(n);  
ASTNode_getLeftChild(n);  
ASTNode_getRightChild(n);  
ASTNode_getChild(n, 3);  
ASTNode_getNumChildren(n);
```



# libsbml :: Future Directions

- Full L2 Support
- Java and PHP Bindings
  - via Simple Wrapper Interface Generator (SWIG;  
<http://www.swig.org>)
  - Perhaps more...
- L1  $\leftrightarrow$  L2 Translator (and vice-versa where possible)
- Semantic Validator

# Acknowledgements

- Mike Hucka
- Andrew Finney
- Herbert Sauro
- Bruce Shapiro
- Ben Kovitz

# Questions / Feedback

<http://www.sbml.org/>

<http://www.sf.net/projects/sbml/>

# libsbml

## A C-based library for manipulating SBML

Benjamin J. Bornstein<sup>1,2</sup>

<sup>1</sup>Jet Propulsion Laboratory, Machine Learning Systems

<sup>2</sup>California Institute of Technology

Presented at the **6<sup>th</sup> Forum** on Software Platforms for Systems Biology

# libsbml

- Motivations
- Features
- API Overview
- Future Directions
- Questions / Feedback

# libsbml::Motivations

- Community needs (at least one) lightweight and lean SBML library
- Such a library could serve as a:
  - Reference implementation
  - Basis for future SBW services, e.g. new Validator, improved NOM, various translators, etc.
  - Basis for high-level language bindings

# libsbml::Features

- Memory Efficient and Fast
  - 100Yeast.xml (2M), 101 Compartments, 2K Reactions, 2K Rules, ... [ $\ll$  1 second, 1M RAM]
- Pure ANSI C
  - `gcc -ansi -pedantic-errors -Wall`
- Full Unicode Support

# libsbml::Features

- Can validate against XML Schema (sbml.xsd)
- Handles <notes> and <annotation> elements, including XML namespaces
- Well tested
  - Backed by 171 unit tests comprised of 964 individual assertions
  - Memory profiler shows no leaks

# libsbml::Features

- Runs on:
  - Every model in the SBML Model Repository
  - All exports from Gepasi posted on the website.
- Supports SBML:
  - Level 1 Version 1
  - Level 1 Version 2
  - Level 2 (in-progress...)

# libsbml::API Overview

- `Parameter_t *`  
`Parameter_create();`
- `void`  
`Parameter_free(Parameter_t *p);`
- `void`  
`Parameter_setName(Parameter_t *p,`  
`const char *sname);`

# libsbml::API Overview

- `int`

```
UnitKind_equals (UnitKind_t uk1,  
                 UnitKind_t uk2);
```

- `UnitKind_t`

```
UnitKind_forName (const char *sname);
```

- `const char *`

```
UnitKind_toString (UnitKind_t uk);
```

# libsbml::API Overview

- void  
`Model_addSpecies(Model_t *m, Species_t *s);`
- int  
`Model_getNumSpecies(Model_t *m);`
- Species\_t \*  
`Model_getSpecies(Model_t *m, unsigned int n);`
- Species\_t \*  
`Model_createSpecies(Species_t* s);`

# libsbml::API Overview

```
#include <SBMLReader.h>
#include <SBMLTypes.h>

int main(int argc, char *argv[])
{
    char *filename      = "l1v1-units.xml";
    SBMLDocument_t *d = readSBML(filename);
    /* or readSBMLFromString(...) */

    if (d == NULL) { /* Handle Error */ }
```

# libsbml::API Overview

```
/**  
 * <sbml level="1" version="1">  
 */  
assert(d->level == 1);  
assert(d->version == 1);  
  
Model_t *m = d->model;
```

# libsbml::API Overview

```
/**
 * <listOfUnitDefinitions>
 *   <unitDefinition name="substance"> ... </unitDefinition>
 *   <unitDefinition name="mls"> ... </unitDefinition>
 * </listOfUnitDefinitions>
 */
assert( Model_getNumUnitDefinitions(m) == 2 );

UnitDefinition_t *ud = Model_getUnitDefinition(m, 0);
assert( !strcmp(ud->name, "substance") );

ud = Model_getUnitDefinition(m, 1);
assert( !strcmp(ud->name, "mls"));
```

# libsbml::API Overview

```
/**
 * <unitDefinition name="substance">
 *   <listOfUnits>
 *     <unit kind="mole" scale="-3"/>
 *   </listOfUnits>
 * </unitDefinition>
 */
ud = Model_getUnitDefinition(m, 0);
assert( UnitDefinition_getNumUnits(ud) == 1 );

Unit_t *u = UnitDefinition_getUnit(ud, 0);
assert( u->kind == UNIT_KIND_MOLE );
assert( u->exponent == 0 );
assert( u->scale == -3 );
```

# libsbml::API Overview

```
/* And so on... */  
  
return 0;  
  
} /* main() */
```

# libsbml::Future Directions

- Formula Parse (Syntax)

Trees, e.g.:

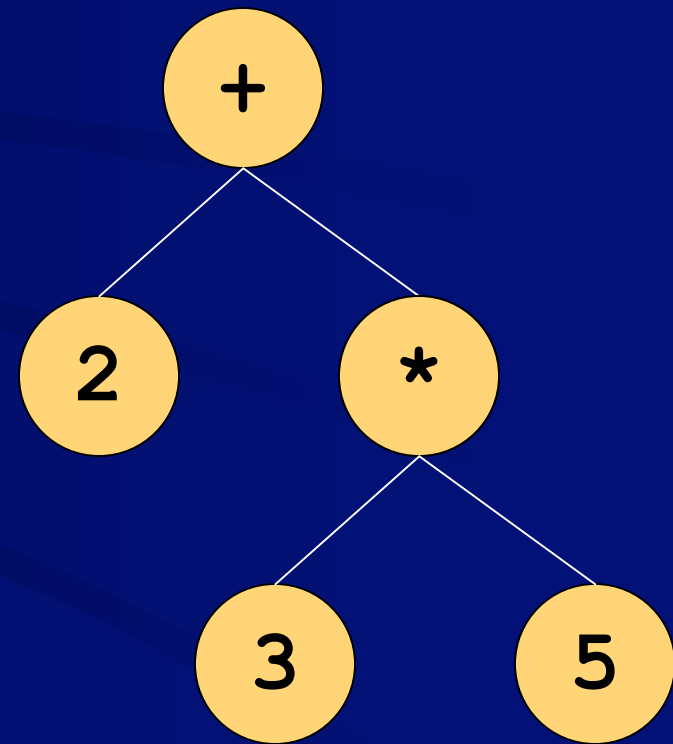
"2 + 3 \* 5"

- Preorder:

+ 2 \* 3 5

- Postorder:

2 3 5 \* +



# libsbml::Future Directions

- SBML Level 2, including:
  - MathML
  - RDF
- L1 to L2 translator and (when possible) vice-versa
- New Validator with pluggable rulesets
- Improved NOM

# Questions / Feedback