libsbml API Reference Manual

Ben Bornstein

bornstei@cds.caltech.edu

The SBML Team
Control and Dynamical Systems, MC 107-81
California Institute of Technology, Pasadena, CA 91125, USA
http://www.sbml.org/

DRAFT

10 August 2007

LIBSBML Version 2.3.5

Systems Biology Markup Language

Contents

1 Introduction

This manual is a reference for the LIBSBML application programming interface (API). LIBSBML provides C and C++ APIs for reading, writing and manipulating the Systems Biology Markup Language (SBML; Hucka et al., 2001, 2003; Finney and Hucka, 2003). Currently, the library supports SBML Level 1 Version 1 and Version 2, and nearly all of SBML Level 2 Version 1. (The still-unimplemented parts of Level 2 are: support for RDF, and support for MathML's `semantics`, `annotation` and `annotation-xml` elements. These will be implemented in the near future.) For more information about SBML, please see the references or visit `http://www.sbml.org/` on the Internet.

LIBSBML is entirely open-source and all specifications and source code are freely and publicly available. This document explains the library API in detail, but does not provide general information about LIBSBML, its use or its installation. For that, please consult the LIBSBML *Developer's Manual* (Bornstein, 2004).

2 API Reference

2.1 AlgebraicRule.h

AlgebraicRule_t * AlgebraicRule_create (void)

Creates a new AlgebraicRule and returns a pointer to it.

AlgebraicRule_t * AlgebraicRule_createWith (const char *formula)

Creates a new AlgebraicRule with the given formula and returns a pointer to it. This convenience function is functionally equivalent to:
`AlgebraicRule_t ar = AlgebraicRule_create(); Rule_setFormula((Rule_t ) ar, formula);`

AlgebraicRule_t * AlgebraicRule_createWithMath (ASTNode_t *math)

Creates a new AlgebraicRule with the given math and returns a pointer to it. This convenience function is functionally equivalent to:
`AlgebraicRule_t ar = AlgebraicRule_create(); Rule_setMath((Rule_t ) ar, math);`
The node **is not copied** and this AlgebraicRule **takes ownership** of it; i.e. subsequent calls to this function or a call to AlgebraicRule_free() will free the ASTNode (and any child nodes).

void AlgebraicRule_free (AlgebraicRule_t *ar)

Frees the given AlgebraicRule.

2.2 AssignmentRule.h

AssignmentRule_t * AssignmentRule_create (void)

Creates a new AssignmentRule and returns a pointer to it.
In L1 AssignmentRule is an abstract class. It exists soley to provide fields to its subclasess: CompartmentVolumeRule, ParameterRule and SpeciesConcentrationRule.
In L2 the three subclasses are gone and AssigmentRule is concrete; i.e. it may be created, used and destroyed directly.

AssignmentRule_t * AssignmentRule_createWith (const char *variable, ASTNode_t *math)

Creates a new AssignmentRule with the given variable and math and returns a pointer to it. This convenience function is functionally equivalent to:
`ar = AssignmentRule_create(); AssignmentRule_setVariable(ar, variable); Rule_setMath((Rule_t ) ar, math);`

void AssignmentRule_free (AssignmentRule_t *ar)

Frees the given AssignmentRule.

void AssignmentRule_initDefaults (AssignmentRule_t *ar)

The function is kept for backward compatibility with the SBML L1 API.
Initializes the fields of this AssignmentRule to their defaults:
- type = RULE_TYPE_SCALAR

RuleType_t AssignmentRule_getType (const AssignmentRule_t *ar)

Returns the type for this AssignmentRule.

const char * AssignmentRule_getVariable (const AssignmentRule_t *ar)

Returns the variable for this AssignmentRule.

int AssignmentRule_isSetVariable (const AssignmentRule_t *ar)

Returns 1 if the variable of this AssignmentRule has been set, 0 otherwise.

void AssignmentRule_setType (AssignmentRule_t *ar, RuleType_t rt)

Sets the type of this Rule to the given RuleType.

void AssignmentRule_setVariable (AssignmentRule_t *ar, const char *sid)

Sets the variable of this AssignmentRule to a copy of sid.

2.3 Compartment.h

Compartment_t * Compartment_create (void)

Creates a new Compartment and returns a pointer to it.

Compartment_t * Compartment_createWith (const char *sid, double size, const char *units, const char *outside)

Creates a new Compartment with the given id, size (volume in L1), units and outside and returns a pointer to it. This convenience function is functionally equivalent to:
`Compartment_t c = Compartment_create(); Compartment_setId(c, id); Compartment_setSize(c, size); ... ;`

void Compartment_free (Compartment_t *c)

Frees the given Compartment.

void Compartment_initDefaults (Compartment_t *c)

Initializes the fields of this Compartment to their defaults:
- volume = 1.0 (L1 only) - spatialDimensions = 3 (L2 only) - constant = 1 (true) (L2 only)

const char * Compartment_getId (const Compartment_t *c)

Returns the id of this Compartment.

const char * Compartment_getName (const Compartment_t *c)

Returns the name of this Compartment.

unsigned int Compartment_getSpatialDimensions (const Compartment_t *c)

Returns the spatialDimensions of this Compartment.

double Compartment_getSize (const Compartment_t *c)

Returns the size (volume in L1) of this Compartment.

double Compartment_getVolume (const Compartment_t *c)

Returns the volume (size in L2) of this Compartment.

const char * Compartment_getUnits (const Compartment_t *c)

Returns the units of this Compartment.

const char * Compartment_getOutside (const Compartment_t *c)

Returns the outside of this Compartment.

int Compartment_getConstant (const Compartment_t *c)

Returns true (non-zero) if this Compartment is constant, false (0) otherwise.

int Compartment_isSetId (const Compartment_t *c)

Returns 1 if the id of this Compartment has been set, 0 otherwise.

int Compartment_isSetName (const Compartment_t *c)

Returns 1 if the name of this Compartment has been set, 0 otherwise.
In SBML L1, a Compartment name is required and therefore **should always be set**. In L2, name is optional and as such may or may not be set.

int Compartment_isSetSize (const Compartment_t *c)

Returns 1 if the size (volume in L1) of this Compartment has been set, 0 otherwise.

int Compartment_isSetVolume (const Compartment_t *c)

Returns 1 if the volume (size in L2) of this Compartment has been set, 0 otherwise.
In SBML L1, a Compartment volume has a default value (1.0) and therefore **should always be set**. In L2, volume (size) is optional with no default value and as such may or may not be set.

int Compartment_isSetUnits (const Compartment_t *c)

Returns 1 if the units of this Compartment has been set, 0 otherwise.

int Compartment_isSetOutside (const Compartment_t *c)

Returns 1 if the outside of this Compartment has been set, 0 otherwise.

void Compartment_moveIdToName (Compartment_t *c)

Moves the id field of this Compartment to its name field (iff name is not already set). This method is used for converting from L2 to L1.

void Compartment_moveNameToId (Compartment_t *c)

Moves the name field of this Compartment to its id field (iff id is not already set). This method is used for converting from L1 to L2.

void Compartment_setId (Compartment_t *c, const char *sid)

Sets the id of this Compartment to a copy of sid.

void Compartment_setName (Compartment_t *c, const char *string)

Sets the name of this Compartment to a copy of string (SName in L1).

void Compartment_setSpatialDimensions (Compartment_t *c, unsigned int value)

Sets the spatialDimensions of this Compartment to value.
If value is not one of [0, 1, 2, 3] the function will have no effect (i.e. spatialDimensions will not be set).

void Compartment_setSize (Compartment_t *c, double value)

Sets the size (volume in L1) of this Compartment to value.

void Compartment_setVolume (Compartment_t *c, double value)

Sets the volume (size in L2) of this Compartment to value.

void Compartment_setUnits (Compartment_t *c, const char *sid)

Sets the units of this Compartment to a copy of sid.

void Compartment_setOutside (Compartment_t *c, const char *sid)

Sets the outside of this Compartment to a copy of sid.

void Compartment_setConstant (Compartment_t *c, int value)

Sets the constant field of this Compartment to value (boolean).

void Compartment_unsetName (Compartment_t *c)

Unsets the name of this Compartment. This is equivalent to: safe_free(c-¿name); c-¿name = NULL;

void Compartment_unsetSize (Compartment_t *c)

Unsets the size (volume in L1) of this Compartment.

void Compartment_unsetVolume (Compartment_t *c)

Unsets the volume (size in L2) of this Compartment.
In SBML L1, a Compartment volume has a default value (1.0) and therefore **should always be set**. In L2, volume (size) is optional with no default value and as such may or may not be set.

void Compartment_unsetUnits (Compartment_t *c)

Unsets the units of this Compartment. This is equivalent to: safe_free(c-¿units); c-¿units = NULL;

void Compartment_unsetOutside (Compartment_t *c)

Unsets the outside of this Compartment. This is equivalent to: safe_free(c-¿outside); c-¿outside = NULL;

int CompartmentIdCmp (const char *sid, const Compartment_t *c)

The CompartmentIdCmp function compares the string sid to c-¿id.
Returnss an integer less than, equal to, or greater than zero if sid is found to be, respectively, less than, to match or be greater than c-¿id. Returns -1 if either sid or c-¿id is NULL.

2.4 CompartmentVolumeRule.h

CompartmentVolumeRule_t * CompartmentVolumeRule_create (void)

Creates a new CompartmentVolumeRule and returns a pointer to it.

CompartmentVolumeRule_t * CompartmentVolumeRule_createWith (const char *formula, RuleType_t type, const char *compartment)

Creates a new CompartmentVolumeRule with the given formula, type and compartment and returns a pointer to it. This convenience function is functionally equivalent to:

```
 CompartmentVolumeRule_t cvr = CompartmentVolumeRule_create();
Rule_setFormula((Rule_t ) cvr, formula); AssignmentRule_setType((AssignmentRule_t
) cvr, type); ...;
```

void CompartmentVolumeRule_free (CompartmentVolumeRule_t *cvr)

Frees the given CompartmentVolumeRule.

const char * CompartmentVolumeRule_getCompartment (const CompartmentVolumeRule_t *cvr)

Returns the compartment of this CompartmentVolumeRule.

int CompartmentVolumeRule_isSetCompartment (const CompartmentVolumeRule_t *cvr)

Returns 1 if the compartment of this CompartmentVolumeRule has been set, 0 otherwise.

void CompartmentVolumeRule_setCompartment (CompartmentVolumeRule_t *cvr, const char *sname)

Sets the compartment of this CompartmentVolumeRule to a copy of sname.

2.5 EventAssignment.h

EventAssignment_t * EventAssignment_create (void)

Creates a new EventAssignment and returns a pointer to it.

EventAssignment_t * EventAssignment_createWith (const char *variable, ASTNode_t *math)

Creates a new EventAssignment with the given variable and math and returns a pointer to it. This convenience function is functionally equivalent to:

```
 ea = EventAssignment_create();  EventAssignment_setVariable(ea, variable);
EventAssignment_setMath(ea, math);
```

void EventAssignment_free (EventAssignment_t *ea)

Frees the given EventAssignment.

const char * EventAssignment_getVariable (const EventAssignment_t *ea)

Returns the variable of this EventAssignment.

const ASTNode_t * EventAssignment_getMath (const EventAssignment_t *ea)

Returns the math of this EventAssignment.

int EventAssignment_isSetVariable (const EventAssignment_t *ea)

Returns 1 if the variable of this EventAssignment has been set, 0 otherwise.

int EventAssignment_isSetMath (const EventAssignment_t *ea)

Returns 1 if the math of this EventAssignment has been set, 0 otherwise.

void EventAssignment_setVariable (EventAssignment_t *ea, const char *sid)

Sets the variable of this EventAssignment to a copy of sid.

void EventAssignment_setMath (EventAssignment_t *ea, ASTNode_t *math)

Sets the math of this EventAssignment to the given ASTNode.
The node **is not copied** and this EventAssignment **takes ownership** of it; i.e. subsequent calls to this function or a call to EventAssignment_free() will free the ASTNode (and any child nodes).

2.6 Event.h

Event_t * Event_create (void)

Creates a new Event and returns a pointer to it.

Event_t * Event_createWith (const char *sid, ASTNode_t *trigger)

Creates a new Event with the given id and trigger and returns a pointer to it. This convenience function is functionally equivalent to:

```
e = Event_create(); Event_setId(e, id); Event_setTrigger(e, trigger);
```

void Event_free (Event_t *e)

Frees the given Event.

const char * Event_getId (const Event_t *e)

Returns the id of this Event.

const char * Event_getName (const Event_t *e)

Returns the name of this Event.

const ASTNode_t * Event_getTrigger (const Event_t *e)

Returns the trigger of this Event.

const ASTNode_t * Event_getDelay (const Event_t *e)

Returns the delay of this Event.

const char * Event_getTimeUnits (const Event_t *e)

Returns the timeUnits of this Event

int Event_isSetId (const Event_t *e)

Returns 1 if the id of this Event has been set, 0 otherwise.

int Event_isSetName (const Event_t *e)

Returns 1 if the name of this Event has been set, 0 otherwise.

int Event_isSetTrigger (const Event_t *e)

Returns 1 if the trigger of this Event has been set, 0 otherwise.

int Event_isSetDelay (const Event_t *e)

Returns 1 if the delay of this Event has been set, 0 otherwise.

int Event_isSetTimeUnits (const Event_t *e)

Returns 1 if the timeUnits of this Event has been set, 0 otherwise.

void Event_setId (Event_t *e, const char *sid)

Sets the id of this Event to a copy of sid.

void Event_setName (Event_t *e, const char *string)

Sets the name of this Event to a copy of string.

void Event_setTrigger (Event_t *e, ASTNode_t *math)

Sets the trigger of this Event to the given ASTNode.
The node **is not copied** and this Event **takes ownership** of it; i.e. subsequent calls to this function or a call to Event_free() will free the ASTNode (and any child nodes).

void Event_setDelay (Event_t *e, ASTNode_t *math)

Sets the delay of this Event to the given ASTNode.
The node **is not copied** and this Event **takes ownership** of it; i.e. subsequent calls to this function or a call to Event_free() will free the ASTNode (and any child nodes).

void Event_setTimeUnits (Event_t *e, const char *sid)

Sets the timeUnits of this Event to a copy of sid.

void Event_unsetId (Event_t *e)

Unsets the id of this Event. This is equivalent to: safe_free(e-¿id); e-¿id = NULL;

void Event_unsetName (Event_t *e)

Unsets the name of this Event. This is equivalent to: safe_free(e-¿name); e-¿name = NULL;

void Event_unsetDelay (Event_t *e)

Unsets the delay of this Event. This is equivalent to: ASTNode_free(e-¿delay); e-¿delay = NULL;

void Event_unsetTimeUnits (Event_t *e)

Unsets the timeUnits of this Event. This is equivalent to: safe_free(e-¿timeUnits); e-¿timeUnits = NULL;

void Event_addEventAssignment (Event_t *e, EventAssignment_t *ea)

Appends the given EventAssignment to this Event.

ListOf_t * Event_getListOfEventAssignments (Event_t *e)

Returns the list of EventAssignments for this Event.

EventAssignment_t * Event_getEventAssignment (const Event_t *e, unsigned int n)

Returns the nth EventAssignment of this Event.

unsigned int Event_getNumEventAssignments (const Event_t *e)

Returns the number of EventAssignments in this Event.

int EventIdCmp (const char *sid, const Event_t *e)

The EventIdCmp function compares the string sid to e-¿id.
Returnss an integer less than, equal to, or greater than zero if sid is found to be, respectively, less than, to match or be greater than e-¿id. Returns -1 if either sid or e-¿id is NULL.

2.7 FunctionDefinition.h

FunctionDefinition_t * FunctionDefinition_create (void)

Creates a new FunctionDefinition and returns a pointer to it.

FunctionDefinition_t * FunctionDefinition_createWith (const char *sid, ASTNode_t *math)

Creates a new FunctionDefinition with the given id and math and returns a pointer to it. This convenience function is functionally equivalent to:

```
 fd = FunctionDefinition_create();        FunctionDefinition_setId(fd, id);
FunctionDefinition_setMath(fd, math);
```

void FunctionDefinition_free (FunctionDefinition_t *fd)

Frees the given FunctionDefinition.

const char * FunctionDefinition_getId (const FunctionDefinition_t *fd)

Returns the id of this FunctionDefinition.

const char * FunctionDefinition_getName (const FunctionDefinition_t *fd)

Returns the name of this FunctionDefinition.

const ASTNode_t * FunctionDefinition_getMath (const FunctionDefinition_t *fd)

Returns the math of this FunctionDefinition.

int FunctionDefinition_isSetId (const FunctionDefinition_t *fd)

Returns 1 if the id of this FunctionDefinition has been set, 0 otherwise.

int FunctionDefinition_isSetName (const FunctionDefinition_t *fd)

Returns 1 if the name of this FunctionDefinition has been set, 0 otherwise.

int FunctionDefinition_isSetMath (const FunctionDefinition_t *fd)

Returns 1 if the math of this FunctionDefinition has been set, 0 otherwise.

void FunctionDefinition_setId (FunctionDefinition_t *fd, const char *sid)

Sets the id of this FunctionDefinition to a copy of sid.

void FunctionDefinition_setName (FunctionDefinition_t *fd, const char *string)

Sets the name of this FunctionDefinition to a copy of string.

void FunctionDefinition_setMath (FunctionDefinition_t *fd, ASTNode_t *math)

Sets the math of this FunctionDefinition to the given ASTNode.
The node **is not copied** and this FunctionDefinition **takes ownership** of it; i.e. subsequent calls to this function or a call to FunctionDefinition_free() will free the ASTNode (and any child nodes).

void FunctionDefinition_unsetName (FunctionDefinition_t *fd)

Unsets the name of this FunctionDefinition. This is equivalent to: safe_free(fd-¿name); fd-¿name = NULL;

int FunctionDefinitionIdCmp (const char *sid, const FunctionDefinition_t *fd)

The FunctionDefinitionIdCmp function compares the string sid to fd-¿id.
Returnss an integer less than, equal to, or greater than zero if sid is found to be, respectively, less than, to match or be greater than fd-¿id. Returns -1 if either sid or fd-¿id is NULL.

2.8 KineticLaw.h

KineticLaw_t * KineticLaw_create (void)

Creates a new KineticLaw and returns a pointer to it.

KineticLaw_t * KineticLaw_createWith (const char *formula, const char *timeUnits, const char *substanceUnits)

Creates a new KineticLaw with the given formula, timeUnits and substanceUnits and returns a pointer to it. This convenience function is functionally equivalent to:
`KineticLaw_t kl = KineticLaw_create(); KineticLaw_setFormula(kl, formula); KineticLaw_setTimeUnits(kl, timeUnits); ...;`

void KineticLaw_free (KineticLaw_t *kl)

Frees the given KineticLaw.

const char * KineticLaw_getFormula (const KineticLaw_t *kl)

Returns the formula of this KineticLaw.

const ASTNode_t * KineticLaw_getMath (const KineticLaw_t *kl)

Returns the math of this KineticLaw.

ListOf_t * KineticLaw_getListOfParameters (KineticLaw_t *kl)

Returns the list of Parameters for this KineticLaw.

const char * KineticLaw_getTimeUnits (const KineticLaw_t *kl)

Returns the timeUnits of this KineticLaw.

const char * KineticLaw_getSubstanceUnits (const KineticLaw_t *kl)

Returns the substanceUnits of this KineticLaw.

int KineticLaw_isSetFormula (const KineticLaw_t *kl)

Returns true (non-zero) if the formula (or equivalently the math) of this KineticLaw has been set, false (0) otherwise.

int KineticLaw_isSetMath (const KineticLaw_t *kl)

Returns true if the math (or equivalently the formula) of this KineticLaw has been set, false otherwise.

int KineticLaw_isSetTimeUnits (const KineticLaw_t *kl)

Returns 1 if the timeUnits of this KineticLaw has been set, 0 otherwise.

int KineticLaw_isSetSubstanceUnits (const KineticLaw_t *kl)

Returns 1 if the substanceUnits of this KineticLaw has been set, 0 otherwise.

void KineticLaw_setFormula (KineticLaw_t *kl, const char *string)

Sets the formula of this KineticLaw to a copy of string.

void KineticLaw_setMath (KineticLaw_t *kl, ASTNode_t *math)

Sets the math of this KineticLaw to the given ASTNode.
The node **is not copied** and this KineticLaw **takes ownership** of it; i.e. subsequent calls to this function or a call to KineticLaw_free() will free the ASTNode (and any child nodes).

void KineticLaw_setFormulaFromMath (const KineticLaw_t *kl)

This function is no longer necessary. LibSBML now keeps formula strings and math ASTs synchronized automatically. The function is kept around for backward compatibility (and is used internally).

void KineticLaw_setMathFromFormula (const KineticLaw_t *kl)

This function is no longer necessary. LibSBML now keeps formula strings and math ASTs synchronized automatically. The function is kept around for backward compatibility (and is used internally).

void KineticLaw_setTimeUnits (KineticLaw_t *kl, const char *sname)

Sets the timeUnits of this KineticLaw to a copy of sname.

void KineticLaw_setSubstanceUnits (KineticLaw_t *kl, const char *sname)

Sets the substanceUnits of this KineticLaw to a copy of sname.

void KineticLaw_addParameter (KineticLaw_t *kl, Parameter_t *p)

Adds the given Parameter to this KineticLaw.

Parameter_t * KineticLaw_getParameter (const KineticLaw_t *kl, unsigned int n)

Returns the nth Parameter of this KineticLaw.

unsigned int KineticLaw_getNumParameters (const KineticLaw_t *kl)

Returns the number of Parameters in this KineticLaw.

void KineticLaw_unsetTimeUnits (KineticLaw_t *kl)

Unsets the timeUnits of this KineticLaw. This is equivalent to: safe_free(kl-¿timeUnits); kl-¿timeUnits = NULL;

void KineticLaw_unsetSubstanceUnits (KineticLaw_t *kl)

Unsets the substanceUnits of this KineticLaw. This is equivalent to: safe_free(kl-¿substanceUnits); kl-¿substanceUnits = NULL;

2.9 ListOf.h

ListOf_t * ListOf_create (void)

Creates a new ListOf and returns a pointer to it.

void ListOf_free (ListOf_t *lo)

Frees the given ListOf and its constituent items.
This function assumes each item in the list is derived from SBase.

void ListOf_append (ListOf_t *lo, void *item)

Adds item to the end of this List.

void * ListOf_get (const ListOf_t *lo, unsigned int n)

Returns the nth item in this List. If n ¿ ListOf_getNumItems(list) returns NULL.

unsigned int ListOf_getNumItems (const ListOf_t *lo)

Returns the number of items in this List.

void ListOf_prepend (ListOf_t *lo, void *item)

Adds item to the beginning of this ListOf.

void * ListOf_remove (ListOf_t *lo, unsigned int n)

Removes the nth item from this List and returns a pointer to it. If n ¿ ListOf_getNumItems(list) returns NULL.

2.10 Model.h

Model_t * Model_create (void)

Creates a new Model and returns a pointer to it.

Model_t * Model_createWith (const char *sid)

Creates a new Model with the given id and returns a pointer to it. This convenience function is functionally equivalent to:

```
Model_setId(Model_create(), sid);
```

Model_t * Model_createWithName (const char *string)

Creates a new Model with the given name and returns a pointer to it. This convenience function is functionally equivalent to:

```
Model_setName(Model_create(), string);
```

void Model_free (Model_t *m)

Frees the given Model.

const char * Model_getId (const Model_t *m)

Returns the id of this Model.

const char * Model_getName (const Model_t *m)

Returns the name of this Model.

int Model_isSetId (const Model_t *m)

Returns 1 if the id of this Model has been set, 0 otherwise.

int Model_isSetName (const Model_t *m)

Returns 1 if the name of this Model has been set, 0 otherwise.

void Model_moveAllIdsToNames (Model_t *m)

Moves the id field to the name field for this Model and all of its contituent UnitDefinitions, Compartments, Species, Parameters, and Reactions. This method is used for converting from L2 to L1.
NOTE: Any object with its name field already set will be skipped.
@see moveIdToName

void Model_moveAllNamesToIds (Model_t *m)

Moves the name field to the id field for this Model and all of its contituent UnitDefinitions, Compartments, Species, Parameters, and Reactions. This method is used for converting from L1 to L2.
NOTE: Any object with its id field already set will be skipped.
@see moveNameToId

void Model_moveIdToName (Model_t *m)

Moves the id field of this Model to its name field (iff name is not already set). This method is used for converting from L2 to L1.

void Model_moveNameToId (Model_t *m)

Moves the name field of this Model to its id field (iff id is not already set). This method is used for converting from L1 to L2.

void Model_setId (Model_t *m, const char *sid)

Sets the id of this Model to a copy of sid.

void Model_setName (Model_t *m, const char *string)

Sets the name of this Model to a copy of string (SName in L1).

void Model_unsetId (Model_t *m)

Unsets the id of this Model. This is equivalent to: safe_free(m-¿id); m-¿id = NULL;

void Model_unsetName (Model_t *m)

Unsets the name of this Model. This is equivalent to: safe_free(m-¿name); m-¿name = NULL;

FunctionDefinition_t * Model_createFunctionDefinition (Model_t *m)

Creates a new FunctionDefinition inside this Model and returns a pointer to it. This covenience function is functionally equivalent to:

```
Model_addFunctionDefinition(m, FunctionDefinition_create());
```

UnitDefinition_t * Model_createUnitDefinition (Model_t *m)

Creates a new UnitDefinition inside this Model and returns a pointer to it. This covenience function is functionally equivalent to:

```
Model_addUnitDefinition(m, UnitDefinition_create());
```

Unit_t * Model_createUnit (Model_t *m)

Creates a new Unit inside this Model and returns a pointer to it. The Unit is added to the last UnitDefinition created.
If a UnitDefinitions does not exist for this model, a new Unit is not created and NULL is returned.

Compartment_t * Model_createCompartment (Model_t *m)

Creates a new Compartment inside this Model and returns a pointer to it. This covenience function is functionally equivalent to:

```
Model_addCompartment(m, Compartment_create());
```

Species_t * Model_createSpecies (Model_t *m)

Creates a new Species inside this Model and returns a pointer to it. This covenience function is functionally equivalent to:

```
Model_addSpecies(m, Species_create());
```

Parameter_t * Model_createParameter (Model_t *m)

Creates a new Parameter inside this Model and returns a pointer to it. This covenience function is functionally equivalent to:

```
Model_addParameter(m, Parameter_create());
```

AssignmentRule_t * Model_createAssignmentRule (Model_t *m)

Creates a new AssignmentRule inside this Model and returns a pointer to it. This covenience function is functionally equivalent to:

```
Model_addRule(m, AssignmentRule_create());
```

(L2 only)

RateRule_t * Model_createRateRule (Model_t *m)

Creates a new RateRule inside this Model and returns a pointer to it. This covenience function is functionally equivalent to:

```
Model_addRule(m, RateRule_create());
```

(L2 only)

AlgebraicRule_t * Model_createAlgebraicRule (Model_t *m)

Creates a new AlgebraicRule inside this Model and returns a pointer to it. This covenience function is functionally equivalent to:

```
Model_addRule(m, AlgebraicRule_create());
```

CompartmentVolumeRule_t * Model_createCompartmentVolumeRule (Model_t *m)

Creates a new CompartmentVolumeRule inside this Model and returns a pointer to it. This covenience function is functionally equivalent to:

```
Model_addRule(m, CompartmentVolumeRule_create());
```

ParameterRule_t * Model_createParameterRule (Model_t *m)

Creates a new ParameterRule inside this Model and returns a pointer to it. This covenience function is functionally equivalent to:

```
Model_addRule(m, ParameterRule_create());
```

SpeciesConcentrationRule_t * Model_createSpeciesConcentrationRule (Model_t *m)

Creates a new SpeciesConcentrationRule inside this Model and returns a pointer to it. This covenience function is functionally equivalent to:

```
Model_addRule(m, SpeciesConcentrationRule_create());
```

Reaction_t * Model_createReaction (Model_t *m)

Creates a new Reaction inside this Model and returns a pointer to it. This covenience function is functionally equivalent to:

```
Model_addRule(m, Reaction_create());
```

SpeciesReference_t * Model_createReactant (Model_t *m)

Creates a new Reactant (i.e. SpeciesReference) inside this Model and returns a pointer to it. The SpeciesReference is added to the reactants of the last Reaction created.
If a Reaction does not exist for this model, a new SpeciesReference is not created and NULL is returned.

SpeciesReference_t * Model_createProduct (Model_t *m)

Creates a new Product (i.e. SpeciesReference) inside this Model and returns a pointer to it. The SpeciesReference is added to the products of the last Reaction created.
If a Reaction does not exist for this model, a new SpeciesReference is not created and NULL is returned.

ModifierSpeciesReference_t * Model_createModifier (Model_t *m)

Creates a new Modifer (i.e. ModifierSpeciesReference) inside this Model and returns a pointer to it. The ModifierSpeciesReference is added to the modifiers of the last Reaction created.
If a Reaction does not exist for this model, a new ModifierSpeciesReference is not created and NULL is returned.

KineticLaw_t * Model_createKineticLaw (Model_t *m)

Creates a new KineticLaw inside this Model and returns a pointer to it. The KineticLaw is associated with the last Reaction created.
If a Reaction does not exist for this model, or a Reaction does exist, but already has a KineticLaw, a new KineticLaw is not created and NULL is returned.

Parameter_t * Model_createKineticLawParameter (Model_t *m)

Creates a new Parameter (of a KineticLaw) inside this Model and returns a pointer to it. The Parameter is associated with the KineticLaw of the last Reaction created.
If a Reaction does not exist for this model, or a KineticLaw for the Reaction, a new Parameter is not created and NULL is returned.

Event_t * Model_createEvent (Model_t *m)

Creates a new Event inside this Model and returns a pointer to it. This covenience function is functionally equivalent to:

```
Model_addEvent(m, Event_create());
```

EventAssignment_t * Model_createEventAssignment (Model_t *m)

Creates a new EventAssignment inside this Model and returns a pointer to it. The EventAssignment is added to the the last Event created.
If an Event does not exist for this model, a new EventAssignment is not created and NULL is returned.

void Model_addFunctionDefinition (Model_t *m, FunctionDefinition_t *fd)

Adds the given FunctionDefinition to this Model.

void Model_addUnitDefinition (Model_t *m, UnitDefinition_t *ud)

Adds the given UnitDefinition to this Model.

void Model_addCompartment (Model_t *m, Compartment_t *c)

Adds the given Compartment to this Model.

void Model_addSpecies (Model_t *m, Species_t *s)

Adds the given Species to this Model.

void Model_addParameter (Model_t *m, Parameter_t *p)

Adds the given Parameter to this Model.

void Model_addRule (Model_t *m, Rule_t *r)

Adds the given Rule to this Model.

void Model_addReaction (Model_t *m, Reaction_t *r)

Adds the given Reaction to this Model.

void Model_addEvent (Model_t *m, Event_t *e)

Adds the given Event to this Model.

ListOf_t * Model_getListOfFunctionDefinitions (Model_t *m)

Returns the list of FunctionDefinitions for this Model.

ListOf_t * Model_getListOfUnitDefinitions (Model_t *m)

Returns the list of UnitDefinitions for this Model.

ListOf_t * Model_getListOfCompartments (Model_t *m)

Returns the list of Compartments for this Model.

ListOf_t * Model_getListOfSpecies (Model_t *m)

Returns the list of Species for this Model.

ListOf_t * Model_getListOfParameters (Model_t *m)

Returns the list of Parameters for this Model.

ListOf_t * Model_getListOfRules (Model_t *m)

Returns the list of Rules for this Model.

ListOf_t * Model_getListOfReactions (Model_t *m)

Returns the list of Rules for this Model.

ListOf_t * Model_getListOfEvents (Model_t *m)

Returns the list of Rules for this Model.

FunctionDefinition_t * Model_getFunctionDefinition (const Model_t *m, unsigned int n)

Returns the nth FunctionDefinition of this Model.

FunctionDefinition_t * Model_getFunctionDefinitionById (const Model_t *m, const char *sid)

Returns the FunctionDefinition in this Model with the given id or NULL if no such FunctionDefinition exists.

UnitDefinition_t * Model_getUnitDefinition (const Model_t *m, unsigned int n)

Returns the nth UnitDefinition of this Model.

UnitDefinition_t * Model_getUnitDefinitionById (const Model_t *m, const char *sid)

Returns the UnitDefinition in this Model with the given id or NULL if no such UnitDefinition exists.

Compartment_t * Model_getCompartment (const Model_t *m, unsigned int n)

Returns the nth Compartment of this Model.

Compartment_t * Model_getCompartmentById (const Model_t *m, const char *sid)

Returns the Compartment in this Model with the given id or NULL if no such Compartment exists.

Species_t * Model_getSpecies (const Model_t *m, unsigned int n)

Returns the nth Species of this Model.

Species_t * Model_getSpeciesById (const Model_t *m, const char *sid)

Returns the Species in this Model with the given id or NULL if no such Species exists.

Parameter_t * Model_getParameter (const Model_t *m, unsigned int n)

Returns the nth Parameter of this Model.

Parameter_t * Model_getParameterById (const Model_t *m, const char *sid)

Returns the Parameter in this Model with the given id or NULL if no such Parameter exists.

Rule_t * Model_getRule (const Model_t *m, unsigned int n)

Returns the nth Rule of this Model.

Reaction_t * Model_getReaction (const Model_t *m, unsigned int n)

Returns the nth Reaction of this Model.

Reaction_t * Model_getReactionById (const Model_t *m, const char *sid)

Returns the Reaction in this Model with the given id or NULL if no such Reaction exists.

Event_t * Model_getEvent (const Model_t *m, unsigned int n)

Returns the nth Event of this Model.

Event_t * Model_getEventById (const Model_t *m, const char *sid)

Returns the Event in this Model with the given id or NULL if no such Event exists.

unsigned int Model_getNumFunctionDefinitions (const Model_t *m)

Returns the number of FunctionDefinitions in this Model.

unsigned int Model_getNumUnitDefinitions (const Model_t *m)

Returns the number of UnitDefinitions in this Model.

unsigned int Model_getNumCompartments (const Model_t *m)

Returns the number of Compartments in this Model.

unsigned int Model_getNumSpecies (const Model_t *m)

Returns the number of Species in this Model.

unsigned int Model_getNumSpeciesWithBoundaryCondition (const Model_t *m)

Returns the number of Species in this Model with boundaryCondition set to true.

unsigned int Model_getNumParameters (const Model_t *m)

Returns the number of Parameters in this Model. Parameters defined in KineticLaws are not included.

unsigned int Model_getNumRules (const Model_t *m)

Returns the number of Rules in this Model.

unsigned int Model_getNumReactions (const Model_t *m)

Returns the number of Reactions in this Model.

unsigned int Model_getNumEvents (const Model_t *m)

Returns the number of Events in this Model.

ListOf_t * Model_getListOfLayouts (Model_t *m)

Returns a reference to the ListOf object that holds the layouts.

Layout_t * Model_getLayout (Model_t *m, unsigned int index)

Returns the layout object that belongs to the given index. If the index is invalid, NULL is returned.

void Model_addLayout (Model_t *m, Layout_t *layout)

Adds a copy of the layout object to the list of layouts.

Layout_t * Model_createLayout (Model_t *m)

Creates a new layout object and adds it to the list of layout objects. A reference to the newly created object is returned.

2.11 ModifierSpeciesReference.h

ModifierSpeciesReference_t * ModifierSpeciesReference_create (void)

Creates a new ModifierSpeciesReference and returns a pointer to it.

ModifierSpeciesReference_t * ModifierSpeciesReference_createWith (const char *species)

Creates a new ModifierSpeciesReference with the given species and returns a pointer to it. This convenience function is functionally equivalent to:

```
 ModifierSpeciesReference_t msr = ModiferSpeciesReference_create();
ModifierSpeciesReference_setSpecies(msr, species);
```

void ModifierSpeciesReference_free (ModifierSpeciesReference_t *msr)

Frees the given ModifierSpeciesReference.

const char * ModifierSpeciesReference_getSpecies (const ModifierSpeciesReference_t *msr)

Returns the species for this ModifierSpeciesReference.

int ModifierSpeciesReference_isSetSpecies (const ModifierSpeciesReference_t *msr)

Returns 1 if the species for this ModifierSpeciesReference has been set, 0 otherwise.

void ModifierSpeciesReference_setSpecies (ModifierSpeciesReference_t *msr, const char *sid)

Sets the species of this ModifierSpeciesReference to a copy of sid.

2.12 ParameterRule.h

ParameterRule_t * ParameterRule_create (void)

Creates a new ParameterRule and returns a pointer to it.

ParameterRule_t * ParameterRule_createWith (const char *formula, RuleType_t type, const char *name)

Creates a new ParameterRule with the given formula, type, and name and and returns a pointer to it. This convenience function is functionally equivalent to:
`ParameterRule_t pr = ParameterRule_create(); Rule_setFormula((Rule_t ) pr, formula); scr->type = type; ...;`

void ParameterRule_free (ParameterRule_t *pr)

Frees the given ParameterRule.

const char * ParameterRule_getName (const ParameterRule_t *pr)

Returns the (Parameter) name for this ParameterRule.

const char * ParameterRule_getUnits (const ParameterRule_t *pr)

Returns the units for this ParameterRule.

int ParameterRule_isSetName (const ParameterRule_t *pr)

Returns 1 if the (Parameter) name for this ParameterRule has been set, 0 otherwise.

int ParameterRule_isSetUnits (const ParameterRule_t *pr)

Returns 1 if the units for this ParameterRule has been set, 0 otherwise.

void ParameterRule_setName (ParameterRule_t *pr, const char *sname)

Sets the (Parameter) name for this ParameterRule to a copy of sname.

void ParameterRule_setUnits (ParameterRule_t *pr, const char *sname)

Sets the units for this ParameterRule to a copy of sname.

void ParameterRule_unsetUnits (ParameterRule_t *pr)

Unsets the units for this ParameterRule. This is equivalent to: safe_free(pr-¿units); pr-¿units = NULL;

2.13 Parameter.h

Parameter_t * Parameter_create (void)

Creates a new Parameter and returns a pointer to it.

Parameter_t * Parameter_createWith (const char *sid, double value, const char *units)

Creates a new Parameter with the given id, value and units and returns a pointer to it. This convenience function is functionally equivalent to:

```
 Parameter_t p = Parameter_create();                 Parameter_setId(p, id);
Parameter_setValue(p, value); ...  ;
```

void Parameter_free (Parameter_t *p)

Frees the given Parameter.

void Parameter_initDefaults (Parameter_t *p)

Initializes the fields of this Parameter to their defaults:
- constant = 1 (true) (L2 only)

const char * Parameter_getId (const Parameter_t *p)

Returns the id of this Parameter.

const char * Parameter_getName (const Parameter_t *p)

Returns the name of this Parameter.

double Parameter_getValue (const Parameter_t *p)

Returns the value of this Parameter.

const char * Parameter_getUnits (const Parameter_t *p)

Returns the units of this Parameter.

int Parameter_getConstant (const Parameter_t *p)

Returns true (non-zero) if this Parameter is constant, false (0) otherwise.

int Parameter_isSetId (const Parameter_t *p)

Returns 1 if the id of this Parameter has been set, 0 otherwise.

int Parameter_isSetName (const Parameter_t *p)

Returns 1 if the name of this Parameter has been set, 0 otherwise.
In SBML L1, a Parameter name is required and therefore **should always be set**. In L2, name is optional and as such may or may not be set.

int Parameter_isSetValue (const Parameter_t *p)

Returns 1 if the value of this Parameter has been set, 0 otherwise.
In SBML L1v1, a Parameter value is required and therefore **should always be set**. In L1v2 and beyond, a value is optional and as such may or may not be set.

int Parameter_isSetUnits (const Parameter_t *p)

Returns 1 if the units of this Parameter has been set, 0 otherwise.

void Parameter_moveIdToName (Parameter_t *p)

Moves the id field of this Parameter to its name field (iff name is not already set). This method is used for converting from L2 to L1.

void Parameter_moveNameToId (Parameter_t *p)

Moves the id field of this Parameter to its name field (iff name is not already set). This method is used for converting from L2 to L1.

void Parameter_setId (Parameter_t *p, const char *sid)

Sets the id of this Parameter to a copy of sid.

void Parameter_setName (Parameter_t *p, const char *string)

Sets the name of this Parameter to a copy of string (SName in L1).

void Parameter_setValue (Parameter_t *p, double value)

Sets the value of this Parameter to value and marks the field as set.

void Parameter_setUnits (Parameter_t *p, const char *sid)

Sets the units of this Parameter to a copy of sid.

void Parameter_setConstant (Parameter_t *p, int value)

Sets the constant of this Parameter to value (boolean).

void Parameter_unsetName (Parameter_t *p)

Unsets the name of this Parameter. This is equivalent to: safe_free(p-¿name); p-¿name = NULL;
In SBML L1, a Parameter name is required and therefore **should always be set**. In L2, name is optional and as such may or may not be set.

void Parameter_unsetValue (Parameter_t *p)

Unsets the value of this Parameter.
In SBML L1v1, a Parameter value is required and therefore **should always be set**. In L1v2 and beyond, a value is optional and as such may or may not be set.

void Parameter_unsetUnits (Parameter_t *p)

Unsets the units of this Parameter. This is equivalent to: safe_free(p-¿units); p-¿units = NULL;

int ParameterIdCmp (const char *sid, const Parameter_t *p)

The ParameterIdCmp function compares the string sid to p-¿id.
Returnss an integer less than, equal to, or greater than zero if sid is found to be, respectively, less than, to match or be greater than p-¿id. Returns -1 if either sid or p-¿id is NULL.

2.14 RateRule.h

RateRule_t * RateRule_create (void)

Creates a new RateRule and returns a pointer to it.

RateRule_t * RateRule_createWith (const char *variable, ASTNode_t *math)

Creates a new RateRule with the given variable and math and returns a pointer to it. This convenience function is functionally equivalent to:

```
 rr = RateRule_create();                 RateRule_setVariable(rr, variable);
Rule_setMath((Rule_t ) rr, math);
```

void RateRule_free (RateRule_t *rr)

Frees the given RateRule.

const char * RateRule_getVariable (const RateRule_t *rr)

Returns the variable for this RateRule.

int RateRule_isSetVariable (const RateRule_t *rr)

Returns 1 if the variable of this RateRule has been set, 0 otherwise.

void RateRule_setVariable (RateRule_t *rr, const char *sid)

Sets the variable of this RateRule to a copy of sid.

2.15 Reaction.h

Reaction_t * Reaction_create (void)

Creates a new Reaction and returns a pointer to it.

Reaction_t * Reaction_createWith (const char *sid, KineticLaw_t *kl, int reversible, int fast)

Creates a new Reaction with the given id, KineticLaw, reversible and fast and returns a pointer to it. This convenience function is functionally equivalent to:
`Reaction_t r = Reaction_create(); Reaction_setId(r, sid); Reaction_setKineticLaw(r, kl); ...;`

void Reaction_free (Reaction_t *r)

Frees the given Reaction.

void Reaction_initDefaults (Reaction_t *r)

Initializes the fields of this Reaction to their defaults:
- reversible = 1 (true) - fast = 0 (false) (L1 only)

const char * Reaction_getId (const Reaction_t *r)

Returns the id of this Reaction.

const char * Reaction_getName (const Reaction_t *r)

Returns the name of this Reaction.

KineticLaw_t * Reaction_getKineticLaw (const Reaction_t *r)

Returns the KineticLaw of this Reaction.

int Reaction_getReversible (const Reaction_t *r)

Returns the reversible status of this Reaction.

int Reaction_getFast (const Reaction_t *r)

Returns the fast status of this Reaction.

int Reaction_isSetId (const Reaction_t *r)

Returns 1 if the id of this Reaction has been set, 0 otherwise.

int Reaction_isSetName (const Reaction_t *r)

Returns 1 if the name of this Reaction has been set, 0 otherwise.
In SBML L1, a Reaction name is required and therefore **should always be set**. In L2, name is optional and as such may or may not be set.

int Reaction_isSetKineticLaw (const Reaction_t *r)

Returns 1 if the KineticLaw of this Reaction has been set, 0 otherwise.

int Reaction_isSetFast (const Reaction_t *r)

Returns 1 if the fast status of this Reation has been set, 0 otherwise.
In L1, fast is optional with a default of false, which means it is effectively always set. In L2, however, fast is optional with no default value, so it may or may not be set to a specific value.

void Reaction_moveIdToName (Reaction_t *r)

Moves the id field of this Reaction to its name field (iff name is not already set). This method is used for converting from L2 to L1.

void Reaction_moveNameToId (Reaction_t *r)

Moves the name field of this Reaction to its id field (iff id is not already set). This method is used for converting from L1 to L2.

void Reaction_setId (Reaction_t *r, const char *sid)

Sets the id of this Reaction to a copy of sid.

void Reaction_setName (Reaction_t *r, const char *string)

Sets the name of this Reaction to a copy of string (SName in L1).

void Reaction_setKineticLaw (Reaction_t *r, KineticLaw_t *kl)

Sets the KineticLaw of this Reaction to the given KineticLaw.

void Reaction_setReversible (Reaction_t *r, int value)

Sets the reversible status of this Reaction to value (boolean).

void Reaction_setFast (Reaction_t *r, int value)

Sets the fast status of this Reaction to value (boolean).

ListOf_t * Reaction_getListOfReactants (Reaction_t *r)

Returns the list of Reactants for this Reaction.

ListOf_t * Reaction_getListOfProducts (Reaction_t *r)

Returns the list of Products for this Reaction.

ListOf_t * Reaction_getListOfModifiers (Reaction_t *r)

Returns the list of Modifiers for this Reaction.

void Reaction_addReactant (Reaction_t *r, SpeciesReference_t *sr)

Adds the given reactant (SpeciesReference) to this Reaction.

void Reaction_addProduct (Reaction_t *r, SpeciesReference_t *sr)

Adds the given product (SpeciesReference) to this Reaction.

void Reaction_addModifier (Reaction_t *r, ModifierSpeciesReference_t *msr)

Adds the given modifier (ModifierSpeciesReference) to this Reaction.

SpeciesReference_t * Reaction_getReactant (const Reaction_t *r, unsigned int n)

Returns the nth reactant (SpeciesReference) of this Reaction.

SpeciesReference_t * Reaction_getReactantById (const Reaction_t *r, const char *sid)

Returns the reactant (SpeciesReference) in this Reaction with the given id or NULL if no such reactant exists.

SpeciesReference_t * Reaction_getProduct (const Reaction_t *r, unsigned int n)

Returns the nth product (SpeciesReference) of this Reaction.

SpeciesReference_t * Reaction_getProductById (const Reaction_t *r, const char *sid)

Returns the product (SpeciesReference) in this Reaction with the given id or NULL if no such product exists.

ModifierSpeciesReference_t * Reaction_getModifier (const Reaction_t *r, unsigned int n)

Returns the nth modifier (ModifierSpeciesReference) of this Reaction.

ModifierSpeciesReference_t * Reaction_getModifierById (const Reaction_t *r, const char *sid)

Returns the modifier (ModifierSpeciesReference) in this Reaction with the given id or NULL if no such modifier exists.

unsigned int Reaction_getNumReactants (const Reaction_t *r)

Returns the number of reactants (SpeciesReferences) in this Reaction.

unsigned int Reaction_getNumProducts (const Reaction_t *r)

Returns the number of products (SpeciesReferences) in this Reaction.

unsigned int Reaction_getNumModifiers (const Reaction_t *r)

Returns the number of modifiers (ModifierSpeciesReferences) in this Reaction.

void Reaction_unsetName (Reaction_t *r)

Unsets the name of this Reaction. This is equivalent to: safe_free(r-¿name); r-¿name = NULL;
In SBML L1, a Reaction name is required and therefore **should always be set**. In L2, name is optional and as such may or may not be set.

void Reaction_unsetKineticLaw (Reaction_t *r)

Unsets the KineticLaw of this Reaction. This is equivalent to: r-¿kineticLaw = NULL;

void Reaction_unsetFast (Reaction_t *r)

Unsets the fast status of this Reation.
In L1, fast is optional with a default of false, which means it is effectively always set. In L2, however, fast is optional with no default value, so it may or may not be set to a specific value.

int ReactionIdCmp (const char *sid, const Reaction_t *r)

The ReactionIdCmp function compares the string sid to r-¿id.
Returnss an integer less than, equal to, or greater than zero if sid is found to be, respectively, less than, to match or be greater than r-¿id. Returns -1 if either sid or r-¿id is NULL.

2.16 Rule.h

const char * Rule_getFormula (const Rule_t *r)

Returns the formula for this Rule.

const ASTNode_t * Rule_getMath (const Rule_t *r)

Returns the math for this Rule.

int Rule_isSetFormula (const Rule_t *r)

Returns true (non-zero) if the formula (or equivalently the math) for this Rule has been set, false (0) otherwise.

int Rule_isSetMath (const Rule_t *r)

Returns true (non-zero) if the formula (or equivalently the math) for this Rule has been set, false (0) otherwise.

void Rule_setFormula (Rule_t *r, const char *string)

Sets the formula of this Rule to a copy of string.

void Rule_setMath (Rule_t *r, ASTNode_t *math)

Sets the math of this Rule to the given ASTNode.
The node **is not copied** and this Rule **takes ownership** of it; i.e. subsequent calls to this function or a call to Rule_free() will free the ASTNode (and any child nodes).

void Rule_setFormulaFromMath (const Rule_t *r)

This function is no longer necessary. LibSBML now keeps formula strings and math ASTs synchronized automatically. The function is kept around for backward compatibility (and is used internally).

void Rule_setMathFromFormula (const Rule_t *r)

This function is no longer necessary. LibSBML now keeps formula strings and math ASTs synchronized automatically. The function is kept around for backward compatibility (and is used internally).

2.17 RuleType.h

RuleType_t RuleType_forName (const char *name)

Returns the RuleType with the given name (case-insensitive).

const char * RuleType_toString (RuleType_t rt)

Returns the name of the given RuleType. The caller does not own the returned string and is therefore not allowed to modify it.

2.18 SBase.h

void SBase_init (SBase_t *sb, SBMLTypeCode_t tc)

SBase "objects" are abstract, i.e., they are not created. Rather, specific "subclasses" are created (e.g., Model) and their SBASE_FIELDS are initialized with this function. The type of the specific "subclass" is indicated by the given SBMLTypeCode.

void SBase_clear (SBase_t *sb)

Clears (frees) only the SBASE_FIELDS of sb.

SBMLTypeCode_t SBase_getTypeCode (const SBase_t *sb)

Returns the type of this SBML object.

unsigned int SBase_getColumn (const SBase_t *sb)

Returns the column number for this SBML object.

unsigned int SBase_getLine (const SBase_t *sb)

Returns the line number for this SBML object.

const char * SBase_getMetaId (const SBase_t *sb)

Returns the metaid for this SBML object.

const char * SBase_getNotes (const SBase_t *sb)

Returns the notes for this SBML object.

const char * SBase_getAnnotation (const SBase_t *sb)

Returns the annotation for this SBML object.

int SBase_isSetMetaId (const SBase_t *sb)

Returns 1 if the metaid for this SBML object has been set, 0 otherwise.

int SBase_isSetNotes (const SBase_t *sb)

Returns 1 if the notes for this SBML object has been set, 0 otherwise.

int SBase_isSetAnnotation (const SBase_t *sb)

Returns 1 if the annotation for this SBML object has been set, 0 otherwise.

void SBase_setMetaId (SBase_t *sb, const char *metaid)

Sets the metaid field of the given SBML object to a copy of metaid. If object already has a metaid, the existing string is freed before the new one is copied.

void SBase_setNotes (SBase_t *sb, const char *notes)

Sets the notes field of the given SBML object to a copy of notes. If object already has notes, the existing string is freed before the new one is copied.

void SBase_setAnnotation (SBase_t *sb, const char *annotation)

Sets the annotation field of the given SBML object to a copy of annotations. If object already has an annotation, the existing string is freed before the new one is copied.

void SBase_unsetMetaId (SBase_t *sb)

Unsets the metaid for this SBML object. This is equivalent to: safe_free(sb-¿metaid); s-¿metaid = NULL;

void SBase_unsetNotes (SBase_t *sb)

Unsets the notes for this SBML object. This is equivalent to: safe_free(sb-¿notes); s-¿notes = NULL;

void SBase_unsetAnnotation (SBase_t *sb)

Unsets the annotation for this SBML object. This is equivalent to: safe_free(sb-¿annotation); s-¿annotation = NULL;

2.19 SBMLDocument.h

SBMLDocument_t * SBMLDocument_create (void)

Creates a new SBMLDocument and returns a pointer to it.
The SBML level defaults to 2 and version defaults to 1.

SBMLDocument_t * SBMLDocument_createWith (unsigned int level, unsigned int version)

Creates a new SBMLDocument with the given level and version.

Model_t * SBMLDocument_createModel (SBMLDocument_t *d)

Creates a new Model inside this SBMLDocument and returns a pointer to it. This convenience function is functionally equivalent to:

```
d->model = Model_create();
```

Model_t * SBMLDocument_createModelWith (SBMLDocument_t *d, const char *sid)

Creates a new Model inside this SBMLDocument and returns a pointer to it. The name field of this Model is set to a copy of sid.

void SBMLDocument_free (SBMLDocument_t *d)

Frees the given SBMLDocument.

unsigned int SBMLDocument_getLevel (const SBMLDocument_t *d)

Returns the level of this SBMLDocument.

unsigned int SBMLDocument_getVersion (const SBMLDocument_t *d)

Returns the version of this SBMLDocument.

ParseMessage_t * SBMLDocument_getWarning (SBMLDocument_t *d, unsigned int n)

Returns the nth warning encountered during the parse of this SBMLDocument or NULL if n ¿ getNumWarnings() - 1.

ParseMessage_t * SBMLDocument_getError (SBMLDocument_t *d, unsigned int n)

Returns the nth error encountered during the parse of this SBMLDocument or NULL if n ¿ getNumErrors() - 1.

ParseMessage_t * SBMLDocument_getFatal (SBMLDocument_t *d, unsigned int n)

Returns the nth fatal error encountered during the parse of this SBMLDocument or NULL if n ¿ getNumFatals() - 1.

Model_t * SBMLDocument_getModel (SBMLDocument_t *d)

Returns the Model associated with this SBMLDocument.

unsigned int SBMLDocument_getNumWarnings (const SBMLDocument_t *d)

Returns the number of warnings encountered during the parse of this SBMLDocument.

unsigned int SBMLDocument_getNumErrors (const SBMLDocument_t *d)

Returns the number of errors encountered during the parse of this SBMLDocument.

unsigned int SBMLDocument_getNumFatals (const SBMLDocument_t *d)

Returns the number of fatal errors encountered during the parse of this SBMLDocument.

void SBMLDocument_printWarnings (SBMLDocument_t *d, FILE *stream)

Prints all warnings encountered during the parse of this SBMLDocument to the given stream. If no warnings have occurred, i.e. SBMLDocument_getNumWarnings(d) == 0, no output will be sent to stream. The format of the output is:
N Warning(s): line: (id) message

void SBMLDocument_printErrors (SBMLDocument_t *d, FILE *stream)

Prints all errors encountered during the parse of this SBMLDocument to the given stream. If no errors have occurred, i.e. SBMLDocument_getNumErrors(d) == 0, no output will be sent to stream. The format of the output is:
N Error(s): line: (id) message

void SBMLDocument_printFatals (SBMLDocument_t *d, FILE *stream)

Prints all fatals encountered during the parse of this SBMLDocument to the given stream. If no fatals have occurred, i.e. SBMLDocument_getNumFatals(d) == 0, no output will be sent to stream. The format of the output is:
N Fatal(s): line: (id) message

void SBMLDocument_setLevel (SBMLDocument_t *d, unsigned int level)

Sets the level of this SBMLDocument to the given level number. Valid levels are currently 1 and 2.

void SBMLDocument_setVersion (SBMLDocument_t *d, unsigned int version)

Sets the version of this SBMLDocument to the given version number. Valid versions are currently 1 and 2 for SBML L1 and 1 for SBML L2.

void SBMLDocument_setModel (SBMLDocument_t *d, Model_t *m)

Sets the Model of this SBMLDocument to the given Model. Any previously defined model is unset and freed.

unsigned int SBMLDocument_checkConsistency (SBMLDocument_t *d)

Performs a set of semantic consistency checks on the document. Query the results by calling getWarning(), getNumError(),and getNumFatal().
Returns the number of failed checks (errors) encountered.

unsigned int SBMLDocument_validate (SBMLDocument_t *d)

@deprecated use SBMLDocument_checkConsistency() instead.

2.20 SBMLReader.h

SBMLReader_t * SBMLReader_create (void)

Creates a new SBMLReader and returns a pointer to it.
By default schema validation is off (XML_SCHEMA_VALIDATION_NONE) and schemaFilename is NULL.

void SBMLReader_free (SBMLReader_t *sr)

Frees the given SBMLReader.

const char * SBMLReader_getSchemaFilenameL1v1 (const SBMLReader_t *sr)

Returns the schema filename used by this SBMLReader to validate SBML Level 1 version 1 documents.

const char * SBMLReader_getSchemaFilenameL1v2 (const SBMLReader_t *sr)

Returns the schema filename used by this SBMLReader to validate SBML Level 1 version 2 documents.

const char * SBMLReader_getSchemaFilenameL2v1 (const SBMLReader_t *sr)

Returns the schema filename used by this SBMLReader to validate SBML Level 2 version 1 documents.

XMLSchemaValidation_t SBMLReader_getSchemaValidationLevel(const SBMLReader_t *sr)

Returns the schema validation level used by this SBMLReader.

SBMLDocument_t * SBMLReader_readSBML (SBMLReader_t *sr, const char *filename)

Reads an SBML document from the given file. If filename does not exist or is not an SBML file, a fatal error will be logged. Errors can be identified by their unique ids, e.g.:
¡code¿ SBMLReader sr; SBMLDocument_t d;
sr = SBMLReader_create(); SBMLReader_setSchemaValidationLevel(sr, XML_SCHEMA_VALIDATION_BASIC); SBMLReader_setSchemaFilenameL1v1("sbml-l1v1.xsd");
SBMLReader_setSchemaFilenameL1v2("sbml-l1v2.xsd");
SBMLReader_setSchemaFilenameL2v1("sbml-l2v1.xsd");
d = SBMLReader_readSBML(reader, filename);
if (SBMLDocument_getNumFatals(d) ¿ 0) ParseMessage_t pm = SBMLDocument_getFatal(d, 0); if (ParseMessage_getId(pm) == SBML_READ_ERROR_FILE_NOT_FOUND) if (ParseMessage_getId(pm) == SBML_READ_ERROR_NOT_SBML) ¡/code¿
Returns a pointer to the SBMLDocument read.

SBMLDocument_t * SBMLReader_readSBMLFromString (SBMLReader_t *sr, const char *xml)

Reads an SBML document from the given XML string.
The XML string must be complete and legal XML document. Among other things, it must start with an XML processing instruction. For e.g.,:
¡?xml version='1.0' encoding='UTF-8'?¿
This method will log a fatal error if the XML string is not SBML. See the function documentation for SBMLReader_readSBML(filename) for example error checking code.
Returns a pointer to the SBMLDocument read.

void SBMLReader_setSchemaFilenameL1v1 (SBMLReader_t *sr, const char *filename)

Sets the schema filename used by this SBMLReader to validate SBML Level 1 version 1 documents.
The filename should be either i) an absolute path or ii) relative to the directory contain the SBML file(s) to be read.

void SBMLReader_setSchemaFilenameL1v2 (SBMLReader_t *sr, const char *filename)

Sets the schema filename used by this SBMLReader to validate SBML Level 1 version 2 documents.
The filename should be either i) an absolute path or ii) relative to the directory contain the SBML file(s) to be read.

void SBMLReader_setSchemaFilenameL2v1 (SBMLReader_t *sr, const char *filename)

Sets the schema filename used by this SBMLReader to validate SBML Level 2 version 1 documents.
The filename should be either i) an absolute path or ii) relative to the directory contain the SBML file(s) to be read.

void SBMLReader_setSchemaValidationLevel (SBMLReader_t *sr, XMLSchemaValidation_t level)

Sets the schema validation level used by this SBMLReader.
The levels are:
XML_SCHEMA_VALIDATION_NONE (0) turns schema validation off.
XML_SCHEMA_VALIDATION_BASIC (1) validates an XML instance document against an XML Schema. Those who wish to perform schema checking on SBML documents should use this option.
XML_SCHEMA_VALIDATION_FULL (2) validates both the instance document itself and the XML Schema document. The XML Schema document is checked for violation of particle unique attribution constraints and particle derivation restrictions, which is both time-consuming and memory intensive.

SBMLDocument_t * readSBML (const char *filename)

Reads an SBML document from the given file. If filename does not exist or is not an SBML file, a fatal error will be logged. Errors can be identified by their unique ids, e.g.:
¡code¿ `SBMLDocument_t d = SBMLReader_readSBML(reader, filename);` if (SBMLDocument_getNumFatals(d) ¿ 0) `ParseMessage_t pm = SBMLDocument_getFatal(d, 0);` if (ParseMessage_getId(pm) == SBML_READ_ERROR_FILE_NOT_FOUND) if (ParseMessage_getId(pm) == SBML_READ_ERROR_NOT_SBML) ¡/code¿
Returns a pointer to the SBMLDocument read.

SBMLDocument_t * readSBMLFromString (const char *xml)

Reads an SBML document from the given XML string.
The XML string must be complete and legal XML document. Among other things, it must start with an XML processing instruction. For e.g.,:
¡?xml version='1.0' encoding='UTF-8'?¿
This method will log a fatal error if the XML string is not SBML. See the function documentation for readSBML(filename) for example error checking code.
Returns a pointer to the SBMLDocument read.

2.21 SBMLWriter.h

SBMLWriter_t * SBMLWriter_create (void)

Creates a new SBMLWriter and returns a pointer to it.
By default the character encoding is UTF-8 (CHARACTER_ENCODING_UTF_8).

void SBMLWriter_free (SBMLWriter_t *sw)

Frees the given SBMLWriter.

void SBMLWriter_setProgramName (SBMLWriter_t *sw, const char *name)

Sets the name of this program, i.e. the one about to write out the SBMLDocument. If the program name and version are set (setProgramVersion()), the following XML comment, intended for human consumption, will be written at the beginning of the document:
¡!– Created by ¡program name¿ version ¡program version¿ on yyyy-MM-dd HH:mm with libsbml version ¡libsbml version¿. –¿

void SBMLWriter_setProgramVersion (SBMLWriter_t *sw, const char *version)

Sets the version of this program, i.e. the one about to write out the SBMLDocument. If the program version and name are set (setProgramName()), the following XML comment, intended for human consumption, will be written at the beginning of the document:
¡!– Created by ¡program name¿ version ¡program version¿ on yyyy-MM-dd HH:mm with libsbml version ¡libsbml version¿. –¿

int SBMLWriter_writeSBML (SBMLWriter_t *sw, const SBMLDocument_t *d, const char *filename)

Writes the given SBML document to filename.
Returns non-zero on success and zero if the filename could not be opened for writing.)

char * SBMLWriter_writeSBMLToString (SBMLWriter_t *sw, const SBMLDocument_t *d)

Writes the given SBML document to an in-memory string and returns a pointer to it. The string is owned by the caller and should be freed (with free()) when no longer needed.
Returns the string on success and NULL if one of the underlying Xerces or Expat components fail (rare).

int writeSBML (const SBMLDocument_t *d, const char *filename)

Writes the given SBML document to filename. This convenience function is functionally equivalent to:

```
SBMLWriter_writeSBML(SBMLWriter_create(), d, filename);
```

Returns non-zero on success and zero if the filename could not be opened for writing.)

char * writeSBMLToString (const SBMLDocument_t *d)

Writes the given SBML document to an in-memory string and returns a pointer to it. The string is owned by the caller and should be freed (with free()) when no longer needed.
This convenience function is functionally equivalent to:

```
SBMLWriter_writeSBMLToString(SBMLWriter_create(), d);
```

Returns the string on success and NULL if one of the underlying Xerces or Expat components fail (rare).

2.22 SimpleSpeciesReference.h

const char * SimpleSpeciesReference_getSpecies (const SimpleSpeciesReference_t *ssr)

Returns the species for this SimpleSpeciesReference.

int SimpleSpeciesReference_isSetSpecies (const SimpleSpeciesReference_t *ssr)

Returns 1 if the species for this SimpleSpeciesReference has been set, 0 otherwise.

void SimpleSpeciesReference_setSpecies (SimpleSpeciesReference_t *ssr, const char *sid)

Sets the species of this SimpleSpeciesReference to a copy of sid.

const char * SimpleSpeciesReference_getId (const SimpleSpeciesReference_t *ssr)

Returns the id for the species reference.

void SimpleSpeciesReference_setId (SimpleSpeciesReference_t *ssr, const char *sid)

Sets the id for the given SpeciesReference.

int SimpleSpeciesReference_isSetId (const SimpleSpeciesReference_t *ssr)

Returns 0 if the id is not set.

void SimpleSpeciesReference_unsetId (SimpleSpeciesReference_t *ssr)

Sets the id to the empty string.

int SimpleSpeciesReferenceCmp (const char *sid, const SimpleSpeciesReference_t *ssr)

The SimpleSpeciesReferenceCmp function compares the string sid to ssr-¿species. Returnss an integer less than, equal to, or greater than zero if sid is found to be, respectively, less than, to match or be greater than ssr-¿species. Returns -1 if either sid or ssr-¿species is NULL.

2.23 SpeciesConcentrationRule.h

SpeciesConcentrationRule_t * SpeciesConcentrationRule_create (void)

Creates a new SpeciesConcentrationRule and returns a pointer to it.

SpeciesConcentrationRule_t * SpeciesConcentrationRule_createWith (const char *formula, RuleType_t type, const char *species)

Creates a new SpeciesConcentrationRule with the given formula, type and species and returns a pointer to it. This convenience function is functionally equivalent to:

```
 SpeciesConcentrationRule_t scr = SpeciesConcentrationRule_create();
Rule_setFormula((Rule_t ) scr, formula); AssignmentRule_setType((AssignmentRule_t
) scr, type); ...;
```

void SpeciesConcentrationRule_free (SpeciesConcentrationRule_t *scr)

Frees the given SpeciesConcentrationRule.

const char * SpeciesConcentrationRule_getSpecies (const SpeciesConcentrationRule_t *scr)

Returns the species of this SpeciesConcentrationRule.

int SpeciesConcentrationRule_isSetSpecies (const SpeciesConcentrationRule_t *scr)

Returns 1 if the species of this SpeciesConcentrationRule has been set, 0 otherwise.

void SpeciesConcentrationRule_setSpecies (SpeciesConcentrationRule_t *scr, const char *sname)

Sets the species of this SpeciesConcentrationRule to a copy of sname.

2.24 SpeciesReference.h

SpeciesReference_t * SpeciesReference_create (void)

Creates a new SpeciesReference and returns a pointer to it.

SpeciesReference_t * SpeciesReference_createWith (const char *species, double stoichiometry, int denominator)

Creates a new SpeciesReference with the given species, stoichiometry and denominator and returns a pointer to it. This convenience function is functionally equivalent to:

```
 SpeciesReference_t r = SpeciesReference_create();
SpeciesReference_setSpecies(r, species);                r->stoichiometry =
stoichiometry;  ...;
```

void SpeciesReference_free (SpeciesReference_t *sr)

Frees the given SpeciesReference.

void SpeciesReference_initDefaults (SpeciesReference_t *sr)

Initializes the fields of this SpeciesReference to their defaults:
- stoichiometry = 1 - denominator = 1

const char * SpeciesReference_getSpecies (const SpeciesReference_t *sr)

Returns the species of this SpeciesReference.

double SpeciesReference_getStoichiometry (const SpeciesReference_t *sr)

Returns the stoichiometry of this SpeciesReference.

const ASTNode_t * SpeciesReference_getStoichiometryMath (const SpeciesReference_t *sr)

Returns the stoichiometryMath of this SpeciesReference.

int SpeciesReference_getDenominator (const SpeciesReference_t *sr)

Returns the denominator of this SpeciesReference.

int SpeciesReference_isSetSpecies (const SpeciesReference_t *sr)

Returns 1 if the species of this SpeciesReference has been set, 0 otherwise.

int SpeciesReference_isSetStoichiometryMath (const SpeciesReference_t *sr)

Returns 1 if the stoichiometryMath of this SpeciesReference has been set, 0 otherwise.

void SpeciesReference_setSpecies (SpeciesReference_t *sr, const char *sname)

Sets the species of this SpeciesReference to a copy of sname.

void SpeciesReference_setStoichiometry (SpeciesReference_t *sr, double value)

Sets the stoichiometry of this SpeciesReference to value.

void SpeciesReference_setStoichiometryMath (SpeciesReference_t *sr, ASTNode_t *math)

Sets the stoichiometryMath of this SpeciesReference to the given ASTNode.
The node **is not copied** and this SpeciesReference **takes ownership** of it; i.e. subsequent calls to this function or a call to SpeciesReference_free() will free the ASTNode (and any child nodes).

void SpeciesReference_setDenominator (SpeciesReference_t *sr, int value)

Sets the denominator of this SpeciesReference to value.

2.25 Species.h

Species_t * Species_create (void)

Creates a new Species and returns a pointer to it.

Species_t * Species_createWith(const char *sid, const char *compartment, double initialAmount, const char *substanceUnits, int boundaryCondition, int charge)

Creates a new Species with the given id, compartment, initialAmount, substanceUnits, boundaryCondition and charge and returns a pointer to it. This convenience function is functionally equivalent to:

```
 Species_t s = Species_create();                    Species_setId(s, sid);
Species_setCompartment(s, compartment); ...;
```

void Species_free (Species_t *s)

Frees the given Species.

void Species_initDefaults (Species_t *s)

Initializes the fields of this Species to their defaults:
- boundaryCondition = 0 (false) - constant = 0 (false) (L2 only)

const char * Species_getId (const Species_t *s)

Returns the id of this Species

const char * Species_getName (const Species_t *s)

Returns the name of this Species.

const char * Species_getCompartment (const Species_t *s)

Returns the compartment of this Species.

double Species_getInitialAmount (const Species_t *s)

Returns the initialAmount of this Species.

double Species_getInitialConcentration (const Species_t *s)

Returns the initialConcentration of this Species.

const char * Species_getSubstanceUnits (const Species_t *s)

Returns the substanceUnits of this Species.

const char * Species_getSpatialSizeUnits (const Species_t *s)

Returns the spatialSizeUnits of this Species.

const char * Species_getUnits (const Species_t *s)

Returns the units of this Species (L1 only).

int Species_getHasOnlySubstanceUnits (const Species_t *s)

Returns true (non-zero) if this Species hasOnlySubstanceUnits, false (0) otherwise.

int Species_getBoundaryCondition (const Species_t *s)

Returns the boundaryCondition of this Species.

int Species_getCharge (const Species_t *s)

Returns the charge of this Species.

int Species_getConstant (const Species_t *s)

Returns true (non-zero) if this Species is constant, false (0) otherwise.

int Species_isSetId (const Species_t *s)

Returns 1 if the id of this Species has been set, 0 otherwise.

int Species_isSetName (const Species_t *s)

Returns 1 if the name of this Species has been set, 0 otherwise.
In SBML L1, a Species name is required and therefore **should always be set**. In L2, name is optional and as such may or may not be set.

int Species_isSetCompartment (const Species_t *s)

Returns 1 if the compartment of this Species has been set, 0 otherwise.

int Species_isSetInitialAmount (const Species_t *s)

Returns 1 if the initialAmount of this Species has been set, 0 otherwise.
In SBML L1, a Species initialAmount is required and therefore **should always be set**. In L2, initialAmount is optional and as such may or may not be set.

int Species_isSetInitialConcentration (const Species_t *s)

Returns 1 if the initialConcentration of this Species has been set, 0 otherwise.

int Species_isSetSubstanceUnits (const Species_t *s)

Returns 1 if the substanceUnits of this Species has been set, 0 otherwise.

int Species_isSetSpatialSizeUnits (const Species_t *s)

Returns 1 if the spatialSizeUnits of this Species has been set, 0 otherwise.

int Species_isSetUnits (const Species_t *s)

Returns 1 if the units of this Species has been set, 0 otherwise (L1 only).

int Species_isSetCharge (const Species_t *s)

Returns 1 if the charge of this Species has been set, 0 otherwise.

void Species_moveIdToName (Species_t *s)

Moves the id field of this Species to its name field (iff name is not already set). This method is used for converting from L2 to L1.

void Species_moveNameToId (Species_t *s)

Moves the name field of this Species to its id field (iff id is not already set). This method is used for converting from L1 to L2.

void Species_setId (Species_t *s, const char *sid)

Sets the id of this Species to a copy of sid.

void Species_setName (Species_t *s, const char *string)

Sets the name of this Species to a copy of string (SName in L1).

void Species_setCompartment (Species_t *s, const char *sid)

Sets the compartment of this Species to a copy of sid.

void Species_setInitialAmount (Species_t *s, double value)

Sets the initialAmount of this Species to value and marks the field as set. This method also unsets the initialConentration field.

void Species_setInitialConcentration (Species_t *s, double value)

Sets the initialConcentration of this Species to value and marks the field as set. This method also unsets the initialAmount field.

void Species_setSubstanceUnits (Species_t *s, const char *sid)

Sets the substanceUnits of this Species to a copy of sid.

void Species_setSpatialSizeUnits (Species_t *s, const char *sid)

Sets the spatialSizeUnits of this Species to a copy of sid.

void Species_setUnits (Species_t *s, const char *sname)

Sets the units of this Species to a copy of sname (L1 only).

void Species_setHasOnlySubstanceUnits (Species_t *s, int value)

Sets the hasOnlySubstanceUnits field of this Species to value (boolean).

void Species_setBoundaryCondition (Species_t *s, int value)

Sets the boundaryCondition of this Species to value (boolean).

void Species_setCharge (Species_t *s, int value)

Sets the charge of this Species to value and marks the field as set.

void Species_setConstant (Species_t *s, int value)

Sets the constant field of this Species to value (boolean).

void Species_unsetName (Species_t *s)

Unsets the name of this Species. This is equivalent to: safe_free(s-¿name); s-¿name = NULL;
In SBML L1, a Species name is required and therefore **should always be set**. In L2, name is optional and as such may or may not be set.

void Species_unsetInitialAmount (Species_t *s)

Unsets the initialAmount of this Species.
In SBML L1, a Species initialAmount is required and therefore **should always be set**. In L2, initialAmount is optional and as such may or may not be set.

void Species_unsetInitialConcentration (Species_t *s)

Unsets the initialConcentration of this Species.

void Species_unsetSubstanceUnits (Species_t *s)

Unsets the substanceUnits of this Species. This is equivalent to: safe_free(s-¿substanceUnits); s-¿substanceUnits = NULL;

void Species_unsetSpatialSizeUnits (Species_t *s)

Unsets the spatialSizeUnits of this Species. This is equivalent to: safe_free(s-¿spatialSizeUnits); s-¿spatialSizeUnits = NULL;

void Species_unsetUnits (Species_t *s)

Unsets the units of this Species (L1 only).

void Species_unsetCharge (Species_t *s)

Unsets the charge of this Species.

int SpeciesIdCmp (const char *sid, const Species_t *s)

The SpeciesIdCmp function compares the string sid to species-¿id.
Returnss an integer less than, equal to, or greater than zero if sid is found to be, respectively, less than, to match, or be greater than species-¿id. Returns -1 if either sid or species-¿id is NULL.

2.26 UnitDefinition.h

UnitDefinition_t * UnitDefinition_create (void)

Creates a new UnitDefinition and returns a pointer to it.

UnitDefinition_t * UnitDefinition_createWith (const char *sid)

Creates a new UnitDefinition with the given id and returns a pointer to it. This convenience function is functionally equivalent to:

```
UnitDefinition_setId(UnitDefinition_create(), sid);
```

UnitDefinition_t * UnitDefinition_createWithName (const char *string)

Creates a new UnitDefinition with the given name and returns a pointer to it. This convenience function is functionally equivalent to:

```
UnitDefinition_setName(UnitDefinition_create(), string);
```

void UnitDefinition_free (UnitDefinition_t *ud)

Frees the given UnitDefinition.

const char * UnitDefinition_getId (const UnitDefinition_t *ud)

Returns the id of this UnitDefinition.

const char * UnitDefinition_getName (const UnitDefinition_t *ud)

Returns the name of this UnitDefinition.

int UnitDefinition_isSetId (const UnitDefinition_t *ud)

Returns non-zero if the id of this UnitDefinition has been set, zero otherwise.

int UnitDefinition_isSetName (const UnitDefinition_t *ud)

Returns non-zero if the name of this UnitDefinition has been set, zero otherwise.
In SBML L1, a UnitDefinition name is required and therefore **should always be set**. In L2, name is optional and as such may or may not be set.

int UnitDefinition_isVariantOfArea (const UnitDefinition_t *ud)

Returns non-zero if this UnitDefinition is a variant of the builtin type area, i.e. square metres with only abritrary variations in scale, multiplier, or offset values, zero otherwise.

int UnitDefinition_isVariantOfLength (const UnitDefinition_t *ud)

Returns non-zero if this UnitDefinition is a variant of the builtin type length, i.e. metres with only abritrary variations in scale, multiplier, or offset values, zero otherwise.

int UnitDefinition_isVariantOfSubstance (const UnitDefinition_t *ud)

Returns non-zero if this UnitDefinition is a variant of the builtin type substance, i.e. moles or items with only abritrary variations in scale, multiplier, or offset values, zero otherwise.

int UnitDefinition_isVariantOfTime (const UnitDefinition_t *ud)

Returns non-zero if this UnitDefinition is a variant of the builtin type time, i.e. seconds with only abritrary variations in scale, multiplier, or offset values, zero otherwise.

int UnitDefinition_isVariantOfVolume (const UnitDefinition_t *ud)

Returns non-zero if this UnitDefinition is a variant of the builtin type volume, i.e. litre or cubic metre with only abritrary variations in scale, multiplier, or offset values, zero otherwise.

void UnitDefinition_moveIdToName (UnitDefinition_t *ud)

Moves the id field of this UnitDefinition to its name field (iff name is not already set). This method is used for converting from L2 to L1.

void UnitDefinition_moveNameToId (UnitDefinition_t *ud)

Moves the name field of this UnitDefinition to its id field (iff id is not already set). This method is used for converting from L1 to L2.

void UnitDefinition_setId (UnitDefinition_t *ud, const char *sid)

Sets the id of this UnitDefinition to a copy of sid.

void UnitDefinition_setName (UnitDefinition_t *ud, const char *string)

Sets the name of this UnitDefinition to a copy of string (SName in L1).

void UnitDefinition_unsetName (UnitDefinition_t *ud)

Unsets the name of this UnitDefinition. This is equivalent to: safe_free(ud-¿name); ud-¿name = NULL;
In SBML L1, a UnitDefinition name is required and therefore **should always be set**. In L2, name is optional and as such may or may not be set.

void UnitDefinition_addUnit (UnitDefinition_t *ud, Unit_t *u)

Adds the given Unit to this UnitDefinition.

ListOf_t * UnitDefinition_getListOfUnits (UnitDefinition_t *ud)

Returns the list of Units for this UnitDefinition.

Unit_t * UnitDefinition_getUnit (const UnitDefinition_t *ud, unsigned int n)

Returns the nth Unit of this UnitDefinition.

unsigned int UnitDefinition_getNumUnits (const UnitDefinition_t *ud)

Returns the number of Units in this UnitDefinition.

int UnitDefinitionIdCmp (const char *sid, const UnitDefinition_t *ud)

The UnitDefinitionIdCmp function compares the string sid to ud-¿id.
Returnss an integer less than, equal to, or greater than zero if sid is found to be, respectively, less than, to match or be greater than ud-¿id. Returns -1 if either sid or ud-¿id is NULL.

2.27 UnitKind.h

int UnitKind_equals (UnitKind_t uk1, UnitKind_t uk2)

Tests for logical equality between two UnitKinds. This function behaves exactly like C's == operator, except for the following two cases:
- UNIT_KIND_LITER == UNIT_KIND_LITRE - UNIT_KIND_METER == UNIT_KIND_METRE
where C would yield false (since each of the above is a distinct enumeration value), UnitKind_equals(...) yields true.
Returns true (!0) if uk1 is logically equivalent to uk2, false (0) otherwise.

UnitKind_t UnitKind_forName (const char *name)

Returns the UnitKind with the given name (case-insensitive).

const char * UnitKind_toString (UnitKind_t uk)

Returns the name of the given UnitKind. The caller does not own the returned string and is therefore not allowed to modify it.

int UnitKind_isValidUnitKindString (const char *string)

Returns nonzero if string is the name of a valid unitKind.

2.28 Unit.h

Unit_t * Unit_create (void)

Creates a new Unit and returns a pointer to it.

Unit_t * Unit_createWith (UnitKind_t kind, int exponent, int scale)

Creates a new Unit with the given kind, exponent and scale and returns a pointer to it. This convenience function is functionally equivalent to:

```
 Unit_t u = Unit_create();  Unit_setKind(kind); Unit_setExponent(exponent);
...;
```

void Unit_free (Unit_t *u)

Frees the given Unit.

void Unit_initDefaults (Unit_t *u)

Initializes the fields of this Unit to their defaults:
- exponent = 1 - scale = 0 - multiplier = 1.0 - offset = 0.0

UnitKind_t Unit_getKind (const Unit_t *u)

Returns the kind of this Unit.

int Unit_getExponent (const Unit_t *u)

Returns the exponent of this Unit.

int Unit_getScale (const Unit_t *u)

Returns the scale of this Unit.

double Unit_getMultiplier (const Unit_t *u)

Returns the multiplier of this Unit.

double Unit_getOffset (const Unit_t *u)

Returns the offset of this Unit.

int Unit_isAmpere (const Unit_t *u)

Returns non-zero if the kind of this Unit is 'ampere', zero otherwise.

int Unit_isBecquerel (const Unit_t *u)

Returns non-zero if the kind of this Unit is 'becquerel', zero otherwise.

int Unit_isCandela (const Unit_t *u)

Returns non-zero if the kind of this Unit is 'candela', zero otherwise.

int Unit_isCelsius (const Unit_t *u)

Returns non-zero if the kind of this Unit is 'Celsius', zero otherwise.

int Unit_isCoulomb (const Unit_t *u)

Returns non-zero if the kind of this Unit is 'coulomb', zero otherwise.

int Unit_isDimensionless (const Unit_t *u)

Returns non-zero if the kind of this Unit is 'dimensionless', zero otherwise.

int Unit_isFarad (const Unit_t *u)

Returns non-zero if the kind of this Unit is 'farad', zero otherwise.

int Unit_isGram (const Unit_t *u)

Returns non-zero if the kind of this Unit is 'gram', zero otherwise.

int Unit_isGray (const Unit_t *u)

Returns non-zero if the kind of this Unit is 'gray', zero otherwise.

int Unit_isHenry (const Unit_t *u)

Returns non-zero if the kind of this Unit is 'henry', zero otherwise.

int Unit_isHertz (const Unit_t *u)

Returns non-zero if the kind of this Unit is 'hertz', zero otherwise.

int Unit_isItem (const Unit_t *u)

Returns non-zero if the kind of this Unit is 'item', zero otherwise.

int Unit_isJoule (const Unit_t *u)

Returns non-zero if the kind of this Unit is 'joule', zero otherwise.

int Unit_isKatal (const Unit_t *u)

Returns non-zero if the kind of this Unit is 'katal', zero otherwise.

int Unit_isKelvin (const Unit_t *u)

Returns non-zero if the kind of this Unit is 'kelvin', zero otherwise.

int Unit_isKilogram (const Unit_t *u)

Returns non-zero if the kind of this Unit is 'kilogram', zero otherwise.

int Unit_isLitre (const Unit_t *u)

Returns non-zero if the kind of this Unit is 'litre' or 'liter', zero otherwise.

int Unit_isLumen (const Unit_t *u)

Returns non-zero if the kind of this Unit is 'lumen', zero otherwise.

int Unit_isLux (const Unit_t *u)

Returns non-zero if the kind of this Unit is 'lux', zero otherwise.

int Unit_isMetre (const Unit_t *u)

Returns non-zero if the kind of this Unit is 'metre' or 'meter', zero otherwise.

int Unit_isMole (const Unit_t *u)

Returns non-zero if the kind of this Unit is 'mole', zero otherwise.

int Unit_isNewton (const Unit_t *u)

Returns non-zero if the kind of this Unit is 'newton', zero otherwise.

int Unit_isOhm (const Unit_t *u)

Returns non-zero if the kind of this Unit is 'ohm', zero otherwise.

int Unit_isPascal (const Unit_t *u)

Returns non-zero if the kind of this Unit is 'pascal', zero otherwise.

int Unit_isRadian (const Unit_t *u)

Returns non-zero if the kind of this Unit is 'radian', zero otherwise.

int Unit_isSecond (const Unit_t *u)

Returns non-zero if the kind of this Unit is 'second', zero otherwise.

int Unit_isSiemens (const Unit_t *u)

Returns non-zero if the kind of this Unit is 'siemens', zero otherwise.

int Unit_isSievert (const Unit_t *u)

Returns non-zero if the kind of this Unit is 'sievert', zero otherwise.

int Unit_isSteradian (const Unit_t *u)

Returns non-zero if the kind of this Unit is 'steradian', zero otherwise.

int Unit_isTesla (const Unit_t *u)

Returns non-zero if the kind of this Unit is 'tesla', zero otherwise.

int Unit_isVolt (const Unit_t *u)

Returns non-zero if the kind of this Unit is 'volt', zero otherwise.

int Unit_isWatt (const Unit_t *u)

Returns non-zero if the kind of this Unit is 'watt', zero otherwise.

int Unit_isWeber (const Unit_t *u)

Returns non-zero if the kind of this Unit is 'weber', zero otherwise.

int Unit_isSetKind (const Unit_t *u)

Returns non-zero if the kind of this Unit has been set, zero otherwise.

void Unit_setKind (Unit_t *u, UnitKind_t kind)

Sets the kind of this Unit to the given UnitKind.

void Unit_setExponent (Unit_t *u, int value)

Sets the exponent of this Unit to the given value.

void Unit_setScale (Unit_t *u, int value)

Sets the scale of this Unit to the given value.

void Unit_setMultiplier (Unit_t *u, double value)

Sets the multiplier of this Unit to the given value.

void Unit_setOffset (Unit_t *u, double value)

Sets the offset of this Unit to the given value.

int Unit_isBuiltIn (const char *name)

Returns non-zero if name is one of the five SBML builtin Unit names ('substance', 'volume', 'area', 'length' or 'time'), zero otherwise.

References

Bornstein, B. J. (2004). LibSBML reference manual. Available on the Internet at http://www.sbml.org/software/libsbml.

Finney, A. M. and Hucka, M. (2003). Systems biology markup language: Level 2 and beyond. *Biochemical Society Transactions*, 31:1472–1473.

Hucka, M., Finney, A., Sauro, H. M., and Bolouri, H. (2001). Systems biology markup language (sbml) level 1: Structures and facilities for basic model definitions. Technical report. Available on the Internet at http://www.sbml.org/.

Hucka, M., Finney, A., Sauro, H. M., Bolouri, H., Doyle, J. C., Kitano, H., Arkin, A. P., Bornstein, B. J., Bray, D., Cornish-Bowden, A., Cuellar, A. A., Dronov, S., Gilles, E. D., Ginkel, M., Gor, V., Goryanin, I. I., Hedley, W. J., Hodgman, T. C., Hofmeyr, J.-H., Hunter, P. J., Juty, N. S., Kasberger, J. L., Kremling, A., Kummer, U., Le Novre, N., Loew, L. M., Lucio, D., Mendes, P., Minch, E., Mjolsness, E. D., Nakayama, Y., Nelson, M. R., Nielsen, P. F., Sakurada, T., Schaff, J. C., Shapiro, B. E., Shimizu, T. S., Spence, H. D., Stelling, J., Takahashi, K., Tomita, M., Wagner, J., and Wang, J. (2003). The systems biology markup language (sbml): A medium for representation and exchange of biochemical network models. *Bioinformatics*, 19(4):524–531.