

SBML Level 3 Package: Dynamic Structures (dyn)

NEED AUTHORS HERE

Version 1 (Draft)

August 8, 2014 (0.1 α)

This is a draft specification for the package 'dyn' and not a normative document. Please send feedback to the Package Working Group mailing list at sbml-dynamic@lists.sourceforge.net

The latest release, past releases, and other materials related to this specification are available at
http://sbml.org/Documents/Specifications/SBML_Level_3/Packages/dyn

This release of the specification is available at
[ToBeDecided](#)



Contents

1 Introduction	3
1.1 Proposal corresponding to this package specification	3
1.2 Package dependencies	3
1.3 Document conventions	3
2 Background	5
2.1 Problems with current SBML approaches	5
2.2 Past work on this problem or similar topics	5
3 Proposed syntax and semantics	6
3.1 Overview of dyn extension	6
3.2 Namespace URI and other declarations necessary for using this package	6
3.3 Primitive data types	6
3.3.1 <i>Type</i> CBOTerm	6
3.3.2 <i>Type</i> CoordinateKind	7
3.4 The extended Event object	7
3.4.1 The cboTerm attribute	7
3.4.2 The applyToAll attribute	8
3.4.3 The ListOfElements class	8
3.5 The Element class	8
3.5.1 The element attribute	8
3.6 The extended Compartment object	8
3.6.1 The ListOfCoordinateComponents class	8
3.7 The CoordinateComponent class	9
3.7.1 The coordinateIndex attribute	9
3.7.2 The variable attribute	9
4 The Cell Behavior Ontology and the cboTerm attribute	10
4.1 Cell Behavior Ontology (CBO)	10
4.1.1 Structure of the Cell Behavior Ontology	10
4.2 Using CBO and cboTerm	11
4.2.1 Supported CBO terms in Event Components	11
4.2.2 Tradeoffs in using CBO terms	12
4.2.3 Relationships to the SBML annotation element	12
5 Examples	13
5.1 Example of using dynamic events	13
5.2 Example for using dyn in a lattice-based model	14
5.3 Example for using dyn with the SBML Groups package	16
6 Best practices	18
A Validation Rules	19
Acknowledgments	23
References	24

1 Introduction


Intrinsic dynamic cellular behaviors characteristic of multicellular systems (e.g., proliferation, differentiation, endocytosis, exocytosis, and cell death) may be modeled using a variety of mathematical and spatial frameworks. However, fully representing systems that undergo discrete structural changes during simulation cannot currently be accomplished using either SBML Level 3 Version 2 Core's structural constructs or available language extensions. While mathematical encoding of dynamic behaviors can be approximated, common constructs and behavior-specific semantics are still needed to signal how static constructs interact to emulate dynamic cellular behavior.

This package proposal is completely independent of all other package proposals. Though initially envisioned as being dependent on the Arrays package since dynamic structures are usually associated with compound structures such as lists, sets, and arrays, this proposal does not use constructs from any other language extension and can be used independently. Nevertheless, to facilitate the detailed encoding multicellular systems displaying dynamic cellular events, this extension is designed to work with already existing SBML packages. For example, the Hierarchical Model Composition package could be used to encapsulate portions of a model that need to change dynamically in response to a particular cellular process. The Spatial package could also be used to either describe time-varying species concentration fields as partial differential equations to match native modeling paradigms, or to specify the geometric representation of specific cellular components.

There are a sufficiently large number of multicellular simulators modeling dynamic cellular behaviors to justify the effort of creating a dynamic structure modeling extension. It is the purpose of this SBML Level 3 Version 2 proposal to define a common representation that enables modelers to encode the initial conditions and dynamics of models that include dynamic processes regardless of the spatial or mathematical methods used for simulation.

1.1 Proposal corresponding to this package specification

NEED THE URL TO UPLOADED PROPOSAL

 **Harold:** After proposal acceptance, all dev efforts need a current editor to be part of the package working group (dev requirement). Any ideas as to who would be most suitable /willing to take on the job?

1.2 Package dependencies

The dynamic structures package has no dependencies on other SBML Level 3 packages. It is also designed with the goal of being able to work seamlessly with other SBML Level 3 packages. For instance, one can describe dynamic cellular events in a model that incorporates features from the Hierarchical Model Composition, Groups, or Spatial package. (If any incompatibilities are discovered, please report them to the dyn Package Working Group at sbml-dynamic@lists.sourceforge.net)

1.3 Document conventions

Following the precedent set by the SBML Level 3 Core specification document, we use UML 1.0 (Unified Modeling Language; Eriksson and Penker 1998; Oestereich 1999) class diagram notation to define the constructs provided by this package. We also use color in the diagrams to carry additional information for the benefit of those viewing the document on media that can display color. The following are the colors we use and what they represent:

- **Black:** Items colored black in the UML diagrams are components taken unchanged from their definition in the SBML Level 3 Core specification document.
- **Green:** Items colored green are components that exist in SBML Level 3 Core, but are extended by this package. Class boxes are also drawn with dashed lines to further distinguish them.
- **Blue:** Items colored blue are new components introduced in this package specification. They have no equivalent in the SBML Level 3 Core specification.

We also use the following typographical conventions to distinguish the names of objects and data types from other entities; these conventions are identical to the conventions used in the SBML Level 3 Core specification document:

AbstractClass: Abstract classes are classes that are never instantiated directly, but rather serve as parents of other object classes. Their names begin with a capital letter and they are printed in a slanted, bold, sans-serif typeface. In electronic document formats, the class names defined within this document are also hyperlinked to their definitions; clicking on these items will, given appropriate software, switch the view to the section in this document containing the definition of that class. (However, for classes that are unchanged from their definitions in SBML Level 3 Core, the class names are not hyperlinked because they are not defined within this document.)

Class: Names of ordinary (concrete) classes begin with a capital letter and are printed in an upright, bold, sans-serif typeface. In electronic document formats, the class names are also hyperlinked to their definitions in this specification document. (However, as in the previous case, class names are not hyperlinked if they are for classes that are unchanged from their definitions in the SBML Level 3 Core specification.)

Something, otherThing: Attributes of classes, data type names, literal XML, and generally all tokens *other* than SBML UML class names, are printed in an upright typewriter typeface. Primitive types defined by SBML begin with a capital letter; SBML also makes use of primitive types defined by XML Schema 1.0 ([Biron and Malhotra, 2000](#); [Fallside, 2000](#); [Thompson et al., 2000](#)), but unfortunately, XML Schema does not follow any capitalization convention and primitive types drawn from the XML Schema language may or may not start with a capital letter.

For other matters involving the use of UML and XML, we follow the conventions used in the SBML Level 3 Core specification document.

2 Background

2.1 Problems with current SBML approaches

Representation of dynamical systems that undergo discrete structural changes during simulation cannot currently be accomplished mainly because SBML's structural constructs are fixed. This means that it is not possible to define the creation or removal of species, compartments, reactions, or other components from within a model. To simulate the creation or destruction of compartments, one has to use tricks. For example, a model could define all the compartments it could ever need and use variables to indicate which compartments are actually "active" at any given time. However, this would only work if the total number of compartments needed is known at the beginning of a simulation. This approach hard-codes the anatomical description of a model, which limits portability and becomes impractical beyond a few compartments.

Another limitation is the lack of appropriate constructs to represent the spatial location rearrangement that follows dynamic cellular processes. It can be argued that though unavailable in Core, SBML Level 3 Version 1 extensions such as Layout and Spatial provide the necessary constructs to represent the spatial positioning of modeling elements. However, location as implemented in the Layout package only indicates how components are to be positioned only for user visualization, which may be different from where elements are during simulation. Similarly, though the Spatial package enables the spatial mapping of modeling elements, the position of mapped components is not readily accessible, so it can not be updated as cells move, die and proliferate throughout simulation.

In defense of SBML's limitation in this area, it should be pointed out that the advent of software tools supporting dynamic cellular processes was only a handful until a few years ago. It is now clear that a variety modeling problems and platforms would benefit from this capability.

2.2 Past work on this problem or similar topics


A previous proposal for this language extension outlined the need to use compound data structures available in the Arrays and Sets package to accommodate for dynamic behavior of static SBML components. In this way, cell proliferation could be represented by adding a new cell to an array of cells, or increasing the dimension of an array of compartments. Similarly, cell death could be represented by the removal of a cell from a set. Nonetheless, dynamic creation/destruction of components using arrays proved to be unpractical for large dynamic simulations. The current approach involves extending already existing SBML components and introducing new ones to emulate dynamic cellular processes as opposed to using constructs from other SBML extensions.

3 Proposed syntax and semantics

In this section, we define the syntax and semantics of the Dynamic Structures package for SBML Level 3 Version 2. We delineate the data types and constructs defined in this package, then in [Section 5 on page 13](#), we provide complete examples of using the constructs in example SBML models.

3.1 Overview of dyn extension

The primary mechanism this package uses to support dynamic cellular behavior are the **Event** constructs. The Dynamic Structures package extends this component not only to allow modelers to indicate at which point during simulation (i.e., under what cellular conditions), but which modeling components are affected in response to a particular cellular process. To fully enable this, dyn also extends the **Compartment** object class to facilitate the representation of spatial rearrangements that follow dynamic cellular processes. Though such representations vary across tools based on the chosen modeling paradigm, this language extension defines the necessary constructs to enable tools to encode dynamic cellular behavior regardless of mathematical method or spatial representation framework.

 **Harold:** Out of the types of off-lattice frameworks, our proposed approach only covers center-based (objects spatially identified by the coordinates of their center of mass) and doesn't support vertex-based (where each object has several vertices that together uniquely describe its position)

3.2 Namespace URI and other declarations necessary for using this package

Every SBML Level 3 package is identified uniquely by an XML namespace URI. For an SBML document to be able to use a given SBML Level 3 package, it must declare the use of that package by referencing its URI. The following is the namespace URI for this version of the Dynamic Structures package for SBML Level 3 Version 2:

`"http://www.sbml.org/sbml/level3/version2/dyn/version1"`

In addition, SBML documents using a given package must indicate whether understanding the package is required for complete mathematical interpretation of a model, or whether the package is optional. This is done using the attribute **required** on the `<sbml>` element in the SBML document. For the Dynamic Structure package, the value of this attribute must be set to `"true"`. The following fragment illustrates the beginning of a typical SBML model using SBML Level 3 Version 2 Core and this version of the dyn package:

```
<?xml version="1.0" encoding="UTF-8"?>
<sbml xmlns="http://www.sbml.org/sbml/level3/version2/core" level="3" version="2"
  xmlns:dyn="http://www.sbml.org/sbml/level3/version2/dyn/version1" dyn:required="true">
```

3.3 Primitive data types

The Dynamic Structure package uses a number of the primitive data types described in Section 3.1 of the SBML Level 3 Version 2 Core specification as well as a number of XML Schema 1.0 data types ([Biron and Malhotra, 2000](#)). More specifically, we make use of **boolean** and **SidRef**. This package also adds additional primitive data types described below.

3.3.1 Type CBOTerm

The type **CBOTerm** is used as the data type of the attribute **cboTerm** on the **Event** object class. **CBOTerm** follows the syntax defined for the **anyURI** data type, which considers it a character string data type whose values are interpretable as URIs (Universal Resource Identifiers; ([Harold and Means, 2001](#)); ([W3C, 2000](#))) as described by the W3C document RFC 3986 ([Berners-Lee et al., 2005](#)). Examples of valid string values of type **CBOTerm** are:

http://cbo.biocomplexity.indiana.edu/svn/cbo/trunk/CBO_1_0.owl#CellDeath

http://cbo.biocomplexity.indiana.edu/svn/cbo/trunk/CBO_1_0.owl#Movement

These values are meant to be the identifiers of terms from the Cell Behavior Ontology (CBO) whose curated vocabulary describes cellular entities and processes in computational models. [Section 4 on page 10](#) provides more information about the ontology, supported ontological terms and general principles for their use in SBML models.

3.3.2 Type **CoordinateKind**

The **CoordinateKind** primitive data type is used in the definition of the **CoordinateComponent** class. **CoordinateKind** is derived from the basic XML type **string** though it restricts possible values attributes of this data type. Supported values for attributes of this type are bound to a Cartesian coordinate system and include: “**cartesianX**”, “**cartesianY**”, “**cartesianZ**”. Attributes of **CoordinateKind** type are discussed in the context of the extended **Compartment** object in [Section 3.6 on the following page](#).

3.4 The extended **Event** object

The **Event** class is extended in the Dynamic Structure package. The addition of a **cboTerm** attribute is designed to allow a modeler or a software package to attach semantical information to events triggered to model dynamic cellular processes. The **applyToAll** attribute and subobjects **ListOfElements** and **Element** allow modelers to indicate which SBML components are involved in an **Event** triggered to model dynamic cellular behavior. [Figure 1](#) provides the corresponding UML diagram.

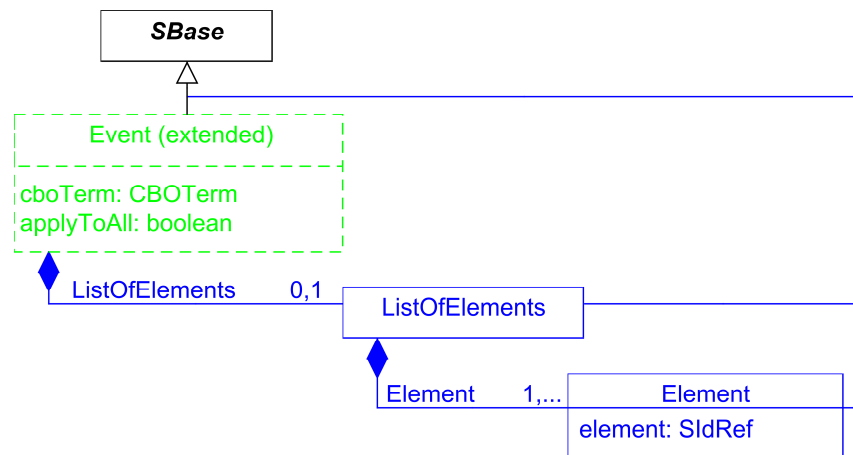


Figure 1: A UML representation of the extended **Event** class and newly defined **ListOfElements** and **Element** objects. See [Section 1.3](#) for conventions related to this figure.

3.4.1 The **cboTerm** attribute

The attribute called **cboTerm** on the **Event** class supports the use of Cell Behavior Ontology (CBO) terms to provide semantic encoding of dynamic cellular events in SBML models. By adding CBO terms, modelers can semantically annotate event constructs to explicitly indicate which cellular process is being modeled by each **Event** instance. The relationship is of the form “the Event is-a X”, where X is the CBO term. The term chosen should be the most precise one that captures the role of the Event in the model. The value of **cboTerm** must conform to the syntax permitted by the **CBOTerm** data type previously described in [Section 3.3.1 on the previous page](#) of the current specification. The scope of the **cboTerm** attribute is local to the enclosing object definition and is not visible outside the object definition. For a discussion on supported values for **cboTerm** and CBO in general see [Section 4 on page 10](#).

3.4.2 The **applyToAll** attribute

The **applyToAll** attribute is used as a mechanism for indicating which SBML model components are impacted by the execution of the current dynamic **Event**. If the value of this attribute is “**true**”, then all SBML elements in the model are involved in the **Event**. If “**false**”, modelers must list the specific components that are affected by means of the **ListOfElements** class, which must not be empty if present. For instance, if the value of the **cboTerm** attribute semantically annotates the current **Event** as modeling cell death, the **applyToAll** attribute allows modelers to chose whether only some or all of the model components are to be removed from the simulation.

3.4.3 The **ListOfElements** class

An extended **Event** component with “**false**” as value for its **applyToAll** attribute must define a non-empty object of class **ListOfElements**. This object contains a list of **Element** references that uniquely identify the set of SBML components involved in the execution of a given dynamic **Event**.

3.5 The **Element** class

An **Element** object contains unique references to SBML components defined within a model. These references are pooled by a **ListOfElements** construct and modified characteristically depending on the cellular process that is modeled. This construct defines an **element** attribute which stores a single reference to a given SBML model component.

3.5.1 The **element** attribute

Element is a mandatory attribute of the type **SIdRef** that is used to reference single anatomical structures (compartments or submodels) or additional SBML components that are altered following the execution of a given dynamic event. Only model elements having identifiers of type **SId** can be referenced by this attribute.

When **element** points to **Compartment** element, it is advisable to define a **ListOfCoordinateComponents** as seen in [Section 3.6](#). This ensures that when an extended **Event** is executed, referenced cellular compartments will have an assigned spatial location, which is important in representing dynamic behavior. On the other hand, if this attribute points to the identifier of a group from the SBML Groups extension, an **Element** references not a single component, but a set of SBML objects instead. For a complete description of how **Element** class instances are affected by different behaviors modeled in an extended **Event**, refer to [Section 4.2.1 on page 11](#).

 **Harold:** Are there any SBML components that won't be able to be referenced given the **SIdRef** type restriction?

3.6 The extended **Compartment** object

The **Compartment** class may be extended to represent the spatial location and subsequent rearrangement that follows dynamic cellular processes emulated by extended **Event** constructs. Refer to [Section 4 on page 10](#) for a list of supported dynamic cellular processes and associated ontological terms. This SBML extension adds to **Compartment** the subobjects **ListOfCoordinateComponents** and **CoordinateComponent** to describe the spatial location of cellular components. [Figure 2](#) provides the corresponding UML diagram of the various features of the extended **Compartment** class.

 **Harold:** If we add a rotation attribute here to support representations (bacterial growth) where rotation is important in spatially describing objects, it should be as 3 optional params of type **SIdRef** corresponding to rotations along the x, y, and z axes.

 **Harold:** Do we expect to model something of higher dimensions than 3D cartesian coordinate system?

3.6.1 The **ListOfCoordinateComponents** class

Compartments mapped by an **Element** construct may contain a **ListOfCoordinateComponents** element, of class **ListOfCoordinateComponents**. Each list contains **CoordinateComponent** objects that together describe the spa-

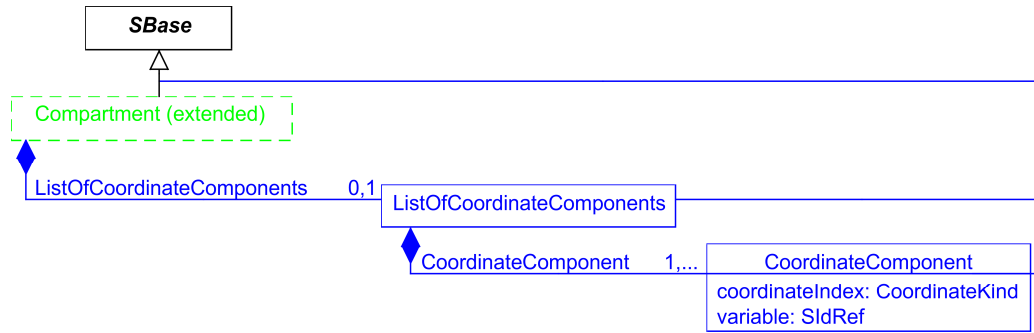


Figure 2: UML diagram for the extension of the **Compartment** object and the definition of **ListOfCoordinateComponents** and **CoordinateComponent** classes. See [Section 1.3](#) for conventions related to this figure.

tial location (center of mass along a Cartesian coordinate system) of an individual **Compartment**. If mapped, a **Compartment** cannot have an empty **ListOfCoordinateComponents** list.

3.7 The **CoordinateComponent** class

Instances of the **CoordinateComponent** class represent individual coordinate components of the spatial location of mapped **Compartment** structures that are involved in dynamic events. This construct defines two attributes: **coordinateIndex** and **variable**. Being derived from **SBase**, this class also has all the usual attributes and elements of its parent class.

Harold: So far we've defined the coordinate system and components to each x,y,z triplet for all compartments involved but nowhere have we defined the minimum and maximum values of the coordinate axis along which objects are modeled.

3.7.1 The **coordinateIndex** attribute

The attribute **coordinateIndex** of type **CoordinateKind** is added to uniquely identify a coordinate axes in the defined **coordinateSystem**. **CoordinateIndex** may take one of all the possible **CoordinateKind** values specified in [Section 3.3.2 on page 7](#). A single "cartesianX" **CoordinateComponent** can be used to define one-dimensional systems; two-dimensional systems are in turn characterized by having two **CoordinateComponent** children with **coordinateIndex** values of "cartesianX" and "cartesianY"; and three-dimensional ones can be defined by having three **CoordinateComponent** elements with **coordinateIndex** values of "cartesianX", "cartesianY", and "cartesianZ".

3.7.2 The **variable** attribute

The **variable** attribute of type **SIdRef** contains the identifier of a **Parameter** defined in the model to explicitly specify an object's position. The scope of the referenced **Parameter** component must be global to the whole model and its **constant** attribute must be set to "false" as spatial location of the associated **Compartment** may change in response to dynamic processes.

4 The Cell Behavior Ontology and the **cboTerm** attribute

It is difficult to determine the semantics of **Event** constructs used to model intrinsic cellular behavior from SBML attributes alone. The **id** attribute on **Event** objects allows for unique identification and cross-referencing while the **name** attribute allows the assignment of human readable labels to Events. Possible values for these attributes are unrestricted so that modelers can choose whichever fits their modeling framework and preference best. However, this means that without any additional human intervention, software tools are unable to discern the semantics of an extended **Event** element modeling dynamic behavior. For instance, it would be inadvisable to interpret that an **Event** is modeling the process of cellular death even if the **id** and **name** of such **Event** have the string “Cell Death” as value. Additionally, as one may need to convert a dynamic **Event** between different representations (e.g., Cellular Potts Model vs. Center-based off-lattice Model), there is a need to provide a standard, framework-independent, way of associating **Event** components with given cellular processes.

A solution inspired by SBML Level 3 Version 2 Core is to associate model components with terms from carefully curated controlled vocabularies (CVs). This is the purpose of the **cboTerm** provided in the extended **Event** class in [Section 3.4 on page 7](#). The **cboTerm** facilitates the annotation of **Event** components with terms belonging to the Cell Behavior Ontology (CBO) ([Sluka et al., 2014](#)). In this section, we discuss CBO, its usage in SBML models via the **cboTerm** attribute and relevant modeling implications.

4.1 Cell Behavior Ontology (CBO)

The development and use of bio-ontologies stems from the need to characterize and describe domains of biology in a standard way. The Cell Behavior Ontology (CBO) provides a carefully curated, controlled vocabulary that can be used to describe the behavior of a cell over time (dynamics) in a framework-independent manner, which enables the reliable exchange of biological descriptions. The Dynamic Structures SBML extension allows modelers to use a subset of the available identifiers to tag SBML **Event** components via its attribute **cboTerm** to make the underlying biology and spatiality of the cellular process being modeled more explicit. The relationship between a **cboTerm** term describing an extended **Event** and the CBO term being used is of the form “the Event is-A X”, where X is the CBO term. Though CBO support provides an important source of information to understand the meaning of an **Event**, software does not need to support **cboTerms** to be considered SBML-compliant.

The presence of a **cboTerm** attribute in an extended **Event** is understood to change the way the **Event** is interpreted and simulated. Annotating SBML **Event** elements with CBO terms adds necessary semantic information that may also be used to convert such **Event** from one framework to another when shared; it enables software tools to recognize precisely what cellular behavior this component is meant to model. For example, if the **cboTerm** has the value of http://cbo.biocomplexity.indiana.edu/svn/cbo/trunk/CBO_1_0.owl#CellDeath for a given **Event**, regardless of the value that the **id** or **name** attributes are given or the modeling method used, the **Event** labeled with this ontological term will be understood to model the dynamics of cellular death.

4.1.1 Structure of the Cell Behavior Ontology

The purpose of CBO is to standardize the description of the intrinsic physical and biological characteristics of cells and tissues, which provides a basis for describing the spatial and observable dynamic behavior of cells in SBML models. To achieve this, CBO is split into controlled vocabularies for **CBO _Objects**, which describe the physical entities of a biological model and **CBO _Processes**, which describe the processes the aforementioned objects participate in. [Figure 3](#) illustrates the taxonomy of CBO at the highest level.

As this SBML extension uses CBO only as a reference ontology for the description of dynamic processes described as events, all of the supported vocabulary is taken from the **CBO _Process** branch. Though **CBO _Process** terms encompass length scales that range from subcellular to cell aggregates and time scales that range from seconds to decades, only the subset in [Section 4.2.1 on the next page](#) is allowed.

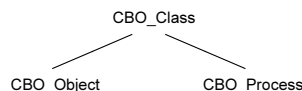



Figure 3: The controlled vocabularies that make up the main branches of CBO.

4.2 Using CBO and cboTerm

The `cboTerm` attribute for extended **Event** constructs is always of `CBOTerm` data type, as defined in [Section 3.4.1 on page 7](#). When present, the attribute's value must be the full identifier of a single term taken from the Cell Behavior Ontology (<http://biportal.bioontology.org/ontologies/CBO>). The term chosen should be the most precise one that best summarizes the phenomenon represented by the extended **Event** object. The relationship indicated by the presence of a non-empty `cboTerm` attribute in an **Event** is of the form "the Event is-A X", where X is the CBO term.

4.2.1 Supported CBO terms in Event Components

One of the mechanisms the Dynamic Structures package uses to support the modeling of dynamic cellular behavior is the extension of the already existing SBML **Event** construct. Under this extension, **Event** objects carry a `cboTerm` attribute whose value must be a full term identifier, taken from the `CBO_Process` vocabulary branch, which describes the behavior modeled by said **Event**. Given substantial community input, the initial version of this package only supports a handful of dynamic cellular behaviors. [Table 1](#) displays supported dynamic processes and their corresponding CBO terms.

 **Harold:** I am being very conservative here. In consideration are growth and differentiation. Any other processes we wish this package to model?

Cell Behaviors	CBO Terms
Cell Division	http://cbo.biocomplexity.indiana.edu/svn/cbo/trunk/CBO_1_0.owl#CellDivision
Cell Death	http://cbo.biocomplexity.indiana.edu/svn/cbo/trunk/CBO_1_0.owl#CellDeath
Cell Movement	http://cbo.biocomplexity.indiana.edu/svn/cbo/trunk/CBO_1_0.owl#Movement

Table 1: Supported cellular behaviors and corresponding CBO terms

We provide a description of each supported CBO term and of the expected simulation semantics given the constructs previously defined in [Section 3 on page 6](#):

- A **Cell Division** term as value for the `cboTerm` attribute indicates that the **Event** defines, by means of a **Trigger** subobject, the mathematical conditions under which cellular division is to take place. When the `applyToAll` **Event** attribute is set to "false", the presence of this CBO term also dictates that model components whose `id` is referenced by an **Element** in the **ListOfElements** subobject have to be duplicated into a daughter cell. Quantities of the respective species and size of the compartments are set to half of the original in the case of symmetric cell division. When the value of `applyToAll` is `valtrue`, all model elements are duplicated.
- A CBO term for **Cell Death** as value for a `cboTerm` attribute indicates that the **Event** defines, by means of a **Trigger** subobject, the mathematical conditions under which cell death is to take place. When the `applyToAll` **Event** attribute is set to "false", the presence of this CBO term also dictates that SBML model components whose `id` is referenced by an **Element** in the **ListOfElements** subobject have to be removed from the model. When the value of `applyToAll` is `valtrue`, all model elements are removed.
- A CBO term for **Movement** as value for a `cboTerm` attribute indicates that the **Event** defines, by means of its **Trigger** subobject, the mathematical conditions under which cell movement is to take place. When the `applyToAll` **Event** attribute is set to "false", the presence of this CBO term also dictates that SBML model

Compartments whose **id** is referenced by an **Element** in the **ListOfElements** subobject will have their position updated. When the value of **applyToAll** is **valtrue**, the spatial location of all model **Compartments** is updated.

Should we include specific terms to support different kinds of cell division and cell death?

4.2.2 Tradeoffs in using CBO terms

The presented CBO-based approach to annotating SBML **Event** components with controlled terms has, just like the SBO-based approach presented in SBML Level 3 Version 2 Core, the following strengths:

1. The syntax required is very straight-forward and requires a single **string** containing the **id** of a supported CBO term.
2. Supported CBO terms cover a relevant portion of the cellular behaviors required by the community.
3. It does not interfere with already-existing annotation schemes implemented by either Core and SBML extensions.

The following list illustrates some of the weaknesses of the proposed approach:

1. The Cell Behavior Ontology is a recent and evolving ontology. As such, it is susceptible to minor changes in its hierarchical taxonomy. These however, should not affect the **ids** of the terms themselves but rather the structure of the ontology itself.

 **Harold:** Can we think of any other benefits or weaknesses?

4.2.3 Relationships to the SBML annotation element

A common mechanism used to provide information regarding modeled dynamic cellular behaviors is the **sBase Annotation** component. Annotations are commonly used by software tools, which generally have their own vocabulary for supporting similar cellular behaviors. However, the best-practice recommendation for interoperability is to use the **cboTerm** attribute in the **Event** object rather than an **Annotation** component. Software tools are encouraged to translate their tool-specific **Annotation** schemes to the proposed **CBO-based** approach when writing SBML models that include dynamic cellular behaviors.

5 Examples

This section contains a variety of examples on how to employ the constructs and extensions provided by the SBML Level 3 Package Specification for Dynamic Structures, Version 1.

5.1 Example of using dynamic events

This example depicts a model of single-cell population dynamics using the extended **Event** construct introduced in [Section 3.4 on page 7](#).

```
<?xml version="1.0" encoding="UTF-8"?>
<sbml xmlns="http://www.sbml.org/sbml/level3/version2/core" level="3" version="2"
  xmlns:dyn="http://www.sbml.org/sbml/level3/version2/dyn/version1" dyn:required="true">

  <model id="singleCell">
    <listOfCompartments>
      <compartment id="Extracellular" spatialDimensions="3" size= "8000000" constant= "true" />
      <compartment id="PlasmaMembrane" spatialDimensions="2" size= "314" constant= "true"/>
      <compartment id="Cytosol" spatialDimensions="3" size= "523" constant= "true"/>
    </listOfCompartments>
    <listOfSpecies>
      <species id="C_EC" compartment="Extracellular" hasOnlySubstanceUnits="false"
        boundaryCondition="false" constant="false"/>
      <species id="RTR_M" compartment="PlasmaMembrane" hasOnlySubstanceUnits="false"
        boundaryCondition="false" constant="false"/>
      <species id="RCC_M" compartment="PlasmaMembrane" hasOnlySubstanceUnits="false"
        boundaryCondition="false" constant="false"/>
      <species id="A_C" compartment="Cytosol" hasOnlySubstanceUnits="false"
        boundaryCondition="false" constant="false"/>
      <species id="AA_C" compartment="Cytosol" hasOnlySubstanceUnits="false"
        boundaryCondition="false" constant="false"/>
    </listOfSpecies>
    <listOfReactions>
      <reaction id="r1" reversible="true" fast="false" compartment="Extracellular">
        <listOfReactants>
          <speciesReference species="RTR_M" stoichiometry="1" constant="true"/>
          <speciesReference species="C_EC" stoichiometry="1" constant="true"/>
        </listOfReactants>
        <listOfProducts>
          <speciesReference species="RCC_M" stoichiometry="1" constant="true"/>
        </listOfProducts>
      </reaction>
      <reaction id="r2" reversible="false" fast="false" compartment="Cytosol">
        <listOfReactants>
          <speciesReference species="A_C" stoichiometry="1" constant="true"/>
        </listOfReactants>
        <listOfProducts>
          <speciesReference species="AA_C" stoichiometry="1" constant="true"/>
        </listOfProducts>
        <listOfModifiers>
          <modifierSpeciesReference species="RCC_M"/>
        </listOfModifiers>
      </reaction>
      <reaction id="r3" reversible="true" fast="false" compartment="Cytosol">
        <listOfReactants>
          <speciesReference species="AA_C" stoichiometry="1" constant="true"/>
        </listOfReactants>
        <listOfProducts>
          <speciesReference species="A_C" stoichiometry="1" constant="true"/>
        </listOfProducts>
      </reaction>
    </listOfReactions>
    <listOfEvents>
```

```

<event useValuesFromTriggerTime="true" applyToAll="true"
  cboTerm="http://cbo.biocomplexity.indiana.edu/svn/cbo/trunk/CBO_1_0.owl#CellDeath">
  <trigger initialValue="false" persistent="true">
    <math xmlns="http://www.w3.org/1998/Math/MathML">
      <apply> <lt;/> <ci> AA_C </ci> <ci> T </ci> </apply>
    </math>
  </trigger>
</event>
<event useValuesFromTriggerTime="true" applyToAll="true"
  cboTerm="http://cbo.biocomplexity.indiana.edu/svn/cbo/trunk/CBO_1_0.owl#CellDivision">
  <trigger initialValue="false" persistent="true">
    <math xmlns="http://www.w3.org/1998/Math/MathML">
      <apply> <lt;/> <ci> AA_C </ci> <ci> S </ci> </apply>
    </math>
  </trigger>
</event>
</listOfEvents>
</model>
</sbml>

```

In the model above, a single cell was defined with species “C_EC” and “RTR_M” forming the complex “RCC_M” on the membrane. The intracellular domain of this complex then catalyzes the activation of the “A_C” into “AA_C”, which is later returns to “A_C”. Each of the extended **Event** structures supports the modeling of a dynamic behavior and is triggered when the concentration of “AA_C” goes below a given threshold value. When executed, each **Event** will impact all model components as the **applyToAll** attribute is set to “true”.

5.2 Example for using **dyn** in a lattice-based model

This example illustrates a multicellular grid-based model using the extended **Event** and **Compartment** constructs introduced in [Section 3 on page 6](#). To accurately encode the model aggregation shown below, several elements from the Hierarchical Model Composition package were also used.

```

<?xml version="1.0" encoding="UTF-8"?>
<sbml xmlns="http://www.sbml.org/sbml/level3/version2/core"
  xmlns:comp="http://www.sbml.org/sbml/level3/version1/comp/version1"
  xmlns:dyn="http://www.sbml.org/sbml/level3/version2/dyn/version1"
  comp:required="true" dyn:required="true" level="3" version="2">

<model id="grid2x2">
  <listOfCompartments>
    <compartment id="Loc1" spatialDimensions="2" size="1" constant="false">
      <dyn:listOfCoordinateComponents>
        <dyn:coordinateComponent coordinateIndex="cartesianX" variable="q1_X" />
        <dyn:coordinateComponent coordinateIndex="cartesianY" variable="q1_Y" />
      </dyn:listOfCoordinateComponents>
      <comp:listOfReplacedElements>
        <comp:replacedElement comp:idRef="C" comp:submodelRef="GRID_1_1_cell"/>
      </comp:listOfReplacedElements>
    </compartment>
    <compartment id="Loc2" spatialDimensions="2" size="1" constant="false">
      <dyn:listOfCoordinateComponents>
        <dyn:coordinateComponent coordinateIndex="cartesianX" variable="q2_X" />
        <dyn:coordinateComponent coordinateIndex="cartesianY" variable="q2_Y" />
      </dyn:listOfCoordinateComponents>
      <comp:listOfReplacedElements>
        <comp:replacedElement comp:idRef="C" comp:submodelRef="GRID_1_2_cell"/>
      </comp:listOfReplacedElements>
    </compartment>
    <compartment id="Loc3" spatialDimensions="2" size="1" constant="false">
      <dyn:listOfCoordinateComponents>
        <dyn:coordinateComponent coordinateIndex="cartesianX" variable="q3_X" />
        <dyn:coordinateComponent coordinateIndex="cartesianY" variable="q3_Y" />
      </dyn:listOfCoordinateComponents>
    </compartment>
  </listOfCompartments>

```

```

    <comp:listOfReplacedElements>
      <comp:replacedElement comp:idRef="C" comp:submodelRef="GRID_2_1_cell"/>
    </comp:listOfReplacedElements>
  </compartment>
  <compartment id="Loc4" spatialDimensions="2" size="1" constant="false">
    <dyn:listOfCoordinateComponents>
      <dyn:coordinateComponent coordinateIndex="cartesianX" variable="q4_X" />
      <dyn:coordinateComponent coordinateIndex="cartesianY" variable="q4_Y" />
    </dyn:listOfCoordinateComponents>
    <comp:listOfReplacedElements>
      <comp:replacedElement comp:idRef="C" comp:submodelRef="GRID_2_2_cell"/>
    </comp:listOfReplacedElements>
  </compartment>
</listOfCompartments>
<listOfParameters>
  <parameter id="q1_X" value="1" constant="false"/>
  <parameter id="q1_Y" value="1" constant="false"/>
  <parameter id="q2_X" value="2" constant="false"/>
  <parameter id="q2_Y" value="1" constant="false"/>
  <parameter id="q3_X" value="1" constant="false"/>
  <parameter id="q3_Y" value="2" constant="false"/>
  <parameter id="q4_X" value="2" constant="false"/>
  <parameter id="q4_Y" value="2" constant="false"/>
</listOfParameters>
<comp:listOfSubmodels>
  <comp:submodel comp:id="GRID_1_1_cell" comp:modelRef="Cell"/>
  <comp:submodel comp:id="GRID_1_2_cell" comp:modelRef="Cell"/>
  <comp:submodel comp:id="GRID_2_1_cell" comp:modelRef="Cell"/>
  <comp:submodel comp:id="GRID_2_2_cell" comp:modelRef="Cell"/>
</comp:listOfSubmodels>
</model>

<comp:listOfModelDefinitions>
  <comp:modelDefinition id="Cell">
    <listOfCompartments>
      <compartment id="C" spatialDimensions="2" size="1" constant="false"/>
    </listOfCompartments>
    <listOfSpecies>
      <species id="R" compartment="C" hasOnlySubstanceUnits="false"
        boundaryCondition="false" constant="false"/>
      <species id="S" compartment="C" hasOnlySubstanceUnits="false"
        boundaryCondition="false" constant="false"/>
    </listOfSpecies>
    <listOfReactions>
      <reaction id="Degradation_R" reversible="false" fast="false" compartment="C">
        <listOfReactants>
          <speciesReference species="R" stoichiometry="1" constant="true"/>
        </listOfReactants>
      </reaction>
      <reaction id="Degradation_S" reversible="false" fast="false" compartment="C">
        <listOfReactants>
          <speciesReference species="S" stoichiometry="1" constant="true"/>
        </listOfReactants>
      </reaction>
    </listOfReactions>
    <listOfEvents>
      <event id="event0" useValuesFromTriggerTime="false" applyToAll="true"
        cboTerm="http://cbo.biocomplexity.indiana.edu/svn/cbo/trunk/CBO_1_0.owl#CellDivision">
        <trigger initialValue="false" persistent="false">
          <math xmlns="http://www.w3.org/1998/Math/MathML">
            <true/>
          </math>
        </trigger>
      </event>
    </listOfEvents>
  </comp:modelDefinition>

```

```
</comp:listOfModelDefinitions>
</sbml>
```

In the model above, we defined a cell with a degradation reaction for species “R” and species “S”. This model also contains an extended **Event** structure to indicate when the cell divides. Given that the **applyToAll** attribute is set to “true”, all of the cell components will be affected as a result of this **Event**. The “Cell” model is instantiated 4 times within the “grid2x2” model to define a grid of 2 by 2. This model contains 4 **Compartment** elements each extending a **ListOfCoordinateComponents** which positions them within the grid.

5.3 Example for using **dyn** with the SBML **Groups** package

This example shows a multicellular lattice-free model using the extended **Event** construct introduced in [Section 3 on page 6](#) and provides a use case for data objects defined in the **Groups** package.

```
<?xml version="1.0" encoding="UTF-8"?>
<sbml xmlns="http://www.sbml.org/sbml/level3/version2/core"
      xmlns:dyn="http://www.sbml.org/sbml/level3/version2/dyn/version1"
      xmlns:groups="http://www.sbml.org/sbml/level3/version1/groups/version1"
      level="3" version="2" groups:required="false" dyn:required="true" >

  <model id="Cell">
    <listOfCompartments>
      <compartment id="C" spatialDimensions="3" constant="false" />
    </listOfCompartments>
    <listOfSpecies>
      <species id="E" compartment="C" hasOnlySubstanceUnits="false"
        boundaryCondition="false" constant="false"/>
      <species id="P" compartment="C" hasOnlySubstanceUnits="false"
        boundaryCondition="false" constant="false"/>
      <species id="EP" compartment="C" hasOnlySubstanceUnits="false"
        boundaryCondition="false" constant="false"/>
    </listOfSpecies>
    <listOfReactions>
      <reaction id="Association" compartment="C" reversible="false" fast="false">
        <listOfReactants>
          <speciesReference species="E" stoichiometry="1" constant="true"/>
          <speciesReference species="P" stoichiometry="1" constant="true"/>
        </listOfReactants>
        <listOfProducts>
          <speciesReference species="EP" stoichiometry="1" constant="true"/>
        </listOfProducts>
      </reaction>
      <reaction id="Dissociation" compartment="C" reversible="false" fast="false">
        <listOfReactants>
          <speciesReference species="EP" stoichiometry="1" constant="true"/>
        </listOfReactants>
        <listOfProducts>
          <speciesReference species="E" stoichiometry="1" constant="true"/>
          <speciesReference species="P" stoichiometry="1" constant="true"/>
        </listOfProducts>
      </reaction>
    </listOfReactions>
    <listOfGroups>
      <group id="cellGroup" kind="partonomy">
        <listOfMembers>
          <member idRef="C"/>
          <member idRef="E"/>
          <member idRef="P"/>
          <member idRef="EP"/>
          <member idRef="Association"/>
          <member idRef="Dissociation"/>
        </listOfMembers>
      </group>
```



```

</listOfGroups>
<listOfEvents>
  <event useValuesFromTriggerTime="true" applyToAll= "false"
    cboTerm="http://cbo.biocomplexity.indiana.edu/svn/cbo/trunk/CBO_1_0.owl#CellDeath">
    <trigger initialValue="false" persistent="true">
      <math xmlns="http://www.w3.org/1998/Math/MathML">
        <apply> <lt/> <ci> EP </ci> <ci> T </ci> </apply>
      </math>
    </trigger>
    <listOfElements>
      <element element="cellGroup"/>
    </listOfElements>
  </event>
</listOfEvents>
</model>
</sbml>

```

The previous example portrays a simple model containing both an association reaction that produces the complex “EP” from “E” and “P” and a reverse dissociation reaction. This cell model also contains an extended **Event** structure to signal that death only occurs when the concentration of “EP” is less than a given threshold “T”. This example also illustrates a different way to indicate which SBML components are affected during a cellular event. While the **applyToAll** attribute provides the means to specify whether all or some SBML elements are impacted, modelers can also use the **ListofGroups** construct from the **Groups** package to do so.

6 Best practices

In this section, we recommend a number of practices for using and interpreting the various constructs in the **dyn** package. Neglecting these recommendations may not render a model invalid, but may reduce the degree of interoperability that can be achieved when sharing it. Hence, we strongly advocate the following practices when using the SBML Level 3 Package Specification for Dynamic Structures, Version 1:

A Validation Rules

This section summarizes all the conditions that must (or in some cases, at least *should*) be true of an SBML Level 3 Version 2 model that uses the Dynamic Structures package. We use the same conventions as are used in the SBML Level 3 Version 2 Core specification document. In particular, there are different degrees of rule strictness. Formally, the differences are expressed in the statement of a rule: either a rule states that a condition *must* be true, or a rule states that it *should* be true. Rules of the former kind are strict SBML validation rules—a model encoded in SBML must conform to all of them in order to be considered valid. Rules of the latter kind are consistency rules. To help highlight these differences, we use the following three symbols next to the rule numbers:

- ☑ A checked box indicates a *requirement* for SBML conformance. If a model does not follow this rule, it does not conform to the Dynamic Structures specification. (Mnemonic intention behind the choice of symbol: “This must be checked.”)
- ▲ A triangle indicates a *recommendation* for model consistency. If a model does not follow this rule, it is not considered strictly invalid as far as the Dynamic Structures specification is concerned; however, it indicates that the model contains a physical or conceptual inconsistency. (Mnemonic intention behind the choice of symbol: “This is a cause for warning.”)
- ★ A star indicates a strong recommendation for good modeling practice. This rule is not strictly a matter of SBML encoding, but the recommendation comes from logical reasoning. As in the previous case, if a model does not follow this rule, it is not considered an invalid SBML encoding. (Mnemonic intention behind the choice of symbol: “You’re a star if you heed this.”)

The validation rules listed in the following subsections are all stated or implied in the rest of this specification document. They are enumerated here for convenience. Unless explicitly stated, all validation rules concern objects and attributes specifically defined in the Dynamic Structures package.

- 🗨 For convenience and brevity, we use the shorthand “**dyn:x**” to stand for an attribute or element name **x** in the namespace for the Dynamic Structures package, using the namespace prefix **dyn**. In reality, the prefix string may be different from the literal “**dyn**” used here (and indeed, it can be any valid XML namespace prefix that the modeler or software chooses). We use “**dyn:x**” because it is shorter than to write a full explanation everywhere we refer to an attribute or element in the Dynamic Structures package namespace.

General rules about the Dynamic Structures package

- dyn-10101** ☑ To conform to Version 1 of the Dynamic Structures package specification for SBML Level 3, an SBML document must declare the use of the following XML Namespace: “<http://www.sbml.org/sbml/level3/version2/dyn/version1>”. (Reference: SBML Level 3 Package Specification for Dynamic Structures, Version 1, [Section 3.2 on page 6](#).)
- dyn-10102** ☑ Wherever they appear in an SBML document, elements and attributes from the Dynamic Structures package must be declared either implicitly or explicitly to be in the XML namespace: “<http://www.sbml.org/sbml/level3/version2/dyn/version1>”. (Reference: SBML Level 3 Package Specification for Dynamic Structures, Version 1, [Section 3.2 on page 6](#).)

Rules for the extended SBML class

- dyn-10201** ☑ In all SBML documents using the Dynamic Structures package, the **SBML** object must include a value for the attribute **dyn:required** attribute. (Reference: SBML Level 3 Version 1 Core, Section 4.1.2.)
- dyn-10202** ☑ The value of attribute **dyn:required** on the **SBML** object must be of the data type **boolean**. (Reference: SBML Level 3 Version 1 Core, Section 4.1.2.)

- dyn-10203** ✓ The value of attribute **dyn:required** on the **SBML** object must be set to “**true**” (Reference: SBML Level 3 Package Specification for Dynamic Structures, Version 1, [Section 3.2 on page 6.](#))

Rules for extended Event objects

- dyn-20101** ✓ An **Event** object must have the attributes **dyn:cboTerm** and **comp:applyToAll** because they are required. No other attributes from the Dynamic Structures namespace are permitted on an **Event** object. (References: SBML Level 3 Package Specification for Dynamic Structures, Version 1, [Section 3.4 on page 7.](#))
- dyn-20102** ✓ The value of an **dyn:cboTerm** attribute on a **Event** object must always conform to the syntax of the **CBOTerm** data type. (References: SBML Level 3 Package Specification for Dynamic Structures, Version 1, [Section 3.4.1 on page 7.](#))
- dyn-20103** ✓ The value of an **dyn:applyToAll** attribute on a **Event** object must be of the data type **boolean**. (References: SBML Level 3 Package Specification for Dynamic Structures, Version 1, [Section 3.4.2 on page 8.](#))
- dyn-20104** ✓ The value of an **dyn:cboTerm** attribute on a **Event** object must be a full CBO term identifier taken from the CBO_Process branch defined in CBO and supported in this package. (References: SBML Level 3 Package Specification for Dynamic Structures, Version 1, [Section 4 on page 10.](#))
- dyn-20105** ✓ There may be at most one instance of a **ListOfElements** subobject within an **Event** object that uses the Dynamic Structures package. (References: SBML Level 3 Package Specification for Dynamic Structures, Version 1, [Section 3.4 on page 7.](#))
- dyn-20106** ✓ The **ListOfElements** subobject within an **Event** object is optional, but if present, this container object must not be empty. (References: SBML Level 3 Package Specification for Dynamic Structures, Version 1, [Section 3.4.3 on page 8.](#))
- dyn-20107** ✓ Apart from the general notes and annotation subobjects permitted on all SBML objects, a **ListOfElements** container object may only contain **Element** objects. (References: SBML Level 3 Package Specification for Dynamic Structures, Version 1, [Section 3.4.3 on page 8.](#))
- dyn-20108** ✓ A **ListOfElements** object may have the optional attributes **metaid** and **sboTerm** defined by SBML Level 3 Core. No other attributes from the SBML Level 3 Core namespace or the Dynamic Structures namespace are permitted on a **ListOfElements** object. (References: SBML Level 3 Version 1 Core, Section 3.2.)

Rules for Element objects

- dyn-20201** ✓ An **Element** object must have the attribute **dyn:element** because it is required. No other attributes from the Dynamic Structures namespace are permitted on an **Element** object. (References: SBML Level 3 Package Specification for Dynamic Structures, Version 1, [Section 3.5 on page 8.](#))
- dyn-20202** ✓ An **Element** object may have the optional SBML Level 3 Core attributes **metaid** and **sboTerm**. (References: SBML Level 3 Version 1 Core, Section 3.2.)
- dyn-20203** ✓ An **Element** object may have the optional SBML Level 3 Core subobjects for notes and annotation. No other elements from the SBML Level 3 Core namespace are permitted on an **Element** object. (References: SBML Level 3 Version 1 Core, Section 3.2.)
- dyn-20204** ✓ The value of an **dyn:element** attribute on an **Element** object must always conform to the syntax of the SBML data type **STidRef**. (References: SBML Level 3 Package Specification for Dynamic Structures, Version 1, [Section 3.5.1 on page 8.](#))

- dyn-20205** ✓ The value of an `dyn:element` attribute on an **Element** object must be the identifier of an SBML component whose tokenId is of type `Std`. (References: SBML Level 3 Package Specification for Dynamic Structures, Version 1, [Section 3.5.1 on page 8.](#))

Rules for extended *Compartment* objects

- dyn-20301** ✓ There may be at most one instance of a **ListOfCoordinateComponents** subobject within a **Compartment** object that uses the Dynamic Structures package. (References: SBML Level 3 Package Specification for Dynamic Structures, Version 1, [Section 3.6 on page 8.](#))
- dyn-20302** ✓ The **ListOfCoordinateComponents** subobject within a **Compartment** object is optional, but if present, this container object must not be empty. (References: SBML Level 3 Package Specification for Dynamic Structures, Version 1, [Section 3.6 on page 8.](#))
- dyn-20303** ✓ Apart from the general notes and annotation subobjects permitted on all SBML objects, a **ListOfCoordinateComponents** container object may only contain **CoordinateComponent** objects. (References: SBML Level 3 Package Specification for Dynamic Structures, Version 1, [Section 3.6.1 on page 8.](#))
- dyn-20304** ✓ A **ListOfCoordinateComponents** object may have the optional attributes `metaid` and `sboTerm` defined by SBML Level 3 Core. No other attributes from the SBML Level 3 Core namespace or the Dynamic Structures namespace are permitted on a **ListOfCoordinateComponents** object. (References: SBML Level 3 Version 1 Core, [Section 3.2.](#))

Rules for *CoordinateComponent* objects

- dyn-20401** ✓ A **CoordinateComponent** object must have the attributes `dyn:coordinateIndex` and `token-dyn:variable` because they are required. No other attributes from the Dynamic Structures namespace are permitted on an **CoordinateComponent** object. (References: SBML Level 3 Package Specification for Dynamic Structures, Version 1, [Section 3.7 on page 9.](#))
- dyn-20402** ✓ An **CoordinateComponent** object may have the optional SBML Level 3 Core attributes `metaid` and `sboTerm`. (References: SBML Level 3 Version 1 Core, [Section 3.2.](#))
- dyn-20403** ✓ An **CoordinateComponent** object may have the optional SBML Level 3 Core subobjects for notes and annotation. No other elements from the SBML Level 3 Core namespace are permitted on an **CoordinateComponent** object. (References: SBML Level 3 Version 1 Core, [Section 3.2.](#))
- dyn-20404** ✓ The value of a `dyn:coordinateIndex` attribute on a **CoordinateComponent** object must always conform to the syntax and allowed values of the newly defined `CoordinateKind`. Permitted values for this attribute include “`cartesianX`”, “`cartesianY`”, and “`cartesianZ`”. (References: SBML Level 3 Package Specification for Dynamic Structures, Version 1, [Section 3.7.1 on page 9.](#))
- dyn-20405** ▲ If a given **Compartment** extends a **ListOfCoordinateComponents**, which contains **CoordinateComponent** subobjects to indicate location, values for their `coordinateIndex` attributes must be used in a specific order. For 1-dimensional cases, “`cartesianX`” is used; for 2-dimensional cases, “`cartesianX`” and “`cartesianY`” are used; and for 3-dimensional cases, “`cartesianX`”, “`cartesianY`”, and “`cartesianZ`” are used. (References: SBML Level 3 Package Specification for Dynamic Structures, Version 1, [Section 3.7.1 on page 9.](#))
- dyn-20406** ✓ The value of a `dyn:variable` attribute on a **CoordinateComponent** object must always conform to the syntax of the data type `StdRef`. (References: SBML Level 3 Package Specification for Dynamic Structures, Version 1, [Section 3.7.2 on page 9.](#))

dyn-20407 ✓	The value of a dyn:coordinateIndex attribute on a CoordinateComponent object is set in a Cartesian coordinate system. (References: SBML Level 3 Package Specification for Dynamic Structures, Version 1, Section 3.7.1 on page 9.)	1 2 3
dyn-20408 ✓	The value of a dyn:variable attribute on an CoordinateComponent object must be the identifier of an SBML parameter. (References: SBML Level 3 Package Specification for Dynamic Structures, Version 1, Section 3.7.2 on page 9.)	4 5 6

 **Harold:** Change references to proper sections within the L3V2 Core specification

Acknowledgments

We would like to thank (insert people's names here) for their direct contributions. In addition, we would like to thank other contributors whose helpful discussions, email contributions or input during HARMONY and COMBINE helped us move the project forward to its completion.

References

- Berners-Lee, T., Fielding, R., and Masinter, L. (2005). Uniform resource identifier (uri): Generic syntax.
- Biron, P. V. and Malhotra, A. (2000). XML Schema part 2: Datatypes (W3C candidate recommendation 24 October 2000). Available via the World Wide Web at <http://www.w3.org/TR/xmlschema-2/>.
- Eriksson, H.-E. and Penker, M. (1998). *UML Toolkit*. John Wiley & Sons, New York.
- Fallside, D. C. (2000). XML Schema part 0: Primer (W3C candidate recommendation 24 October 2000). Available via the World Wide Web at <http://www.w3.org/TR/xmlschema-0/>.
- Harold, E. R. and Means, E. S. (2001). *XML in a Nutshell*. O'Reilly.
- Oestereich, B. (1999). *Developing Software with UML: Object-Oriented Analysis and Design in Practice*. AW.
- Sluka, J. P., Shirinifard, A., Swat, M., Cosmanescu, A., Heiland, R. W., and Glazier, J. A. (2014). The Cell Behavior Ontology: Describing the Intrinsic Biological Behaviors of Real and Model Cells Seen as Active Agents. *Bioinformatics*.
- Thompson, H. S., Beech, D., Maloney, M., and Mendelsohn, N. (2000). XML Schema part 1: Structures (W3C candidate recommendation 24 October 2000). Available online via the World Wide Web at the address <http://www.w3.org/TR/xmlschema-1/>.
- W3C (2000). W3C's math home page. Available via the World Wide Web at <http://www.w3.org/Math/>.