CellML Specification

Draft — 2 March 2001

2 Fundamentals

2.1 Introduction

The fundamentals section of the CellML specification introduces some concepts that are used throughout the entire language, and defines rules that are referenced in all or many of the other parts of the specification. These include the definition of names in CellML and recommended practice for the use of namespaces in CellML.

2.2 Basic Structure

2.2.1 Definition of a valid CellML identifier

The most common use of a CellML identifier is the **`name`** attribute required on many basic elements in CellML. The value of this attribute can be used to reference that element from elsewhere in the model definition or from another model definition altogether. An object's name can generally be thought of as a unique identifier for that object. Although the XML specification defines a mechanism for specifying that the value of an attribute is unique across an entire document (with the ID attribute type), we choose not to make use of that functionality because an object's name need only be unique across its own class of objects.

The generation of computer code for running simulations is one of the target applications for CellML. The value of an object's **`name`** attribute is intended to be a suitable name for the same object when it is represented in computer code. For this reason, CellML identifiers must consist of only alphanumeric characters and the underscore character ("_") and are subject to some additional constraints outlined below. These names will generally not be the most effective way of identifying the object to humans working with CellML models, as it is not possible to include whitespace or formatting. More human readable names can be defined and associated with CellML objects using the metadata functionality introduced in Section 8[1].

The XML specification is based on the Unicode standard, which defines a scheme for 16 bit character encoding. Thus it is possible to include, for instance, Japanese characters in a valid XML document. In the interests of making the code generation process as convenient as possible for those using mainstream programming languages, CellML identifiers are subject to the following constraints:

- An identifier must consist only of alphanumeric characters from the US-ASCII character set and underscore characters,
- An identifier must start with a letter or underscore,
- If an identifier starts with an underscore, then the second character must be a letter.

Convenient code generation is also the reason why colons, periods, and hyphens may not appear in CellML identifier. CellML identifiers are case sensitive: a variable with an identifier of **`ABC`** is different from a variable with an identifier of **`abc`**.

The specification of a valid CellML identifier is identical to the definition of a valid object name in SBML[2]. This should simplify the process of translating model definitions between the two languages.

[1]http://www.cellml.org/public/specification/20010302/cellml_specification.html#sec_metadata
[2]http://www.cds.caltech.edu/erato/

2.2.2 Namespaces in CellML

Namespaces in XML[3] is a companion specification to the main XML specification. It provides a facility for associating the elements and/or attributes in all or part of a document with a particular schema, as indicated by a Uniform Resource Identifier[4] (URI). The key aspect of the URI is that it is unique. The value of the URI need not have anything to do with the XML document that uses it, although typically it would be a good location for the XML Schema or DTD that defines the rules for the document type. The URI may be mapped to a prefix, which may then be used in front of element and attribute names, separated by a colon. If not mapped to a prefix, the URI sets the default schema for the current element and all of its children.

The CellML specification defines a small number of elements and attributes and a namespace with which they must be associated. Associating CellML elements and attributes with the CellML namespace allows them to be differentiated from elements and attributes from other vocabularies with which CellML syntax might be combined in a CellML document. For instance, CellML makes use of the MathML vocabulary for the definition of equations, and all MathML elements must be placed in the MathML namespace in order for CellML processing software to recognise those elements. Applications that store their own proprietary data within a CellML document must define their own namespaces, and associate their own elements and attributes with those namespaces, as discussed in Section 2.2.3.

The scope of CellML is specifically limited to the definition of model structure. The CellML namespace includes all elements and attributes that define the structure of a model. All other information that may be included in a CellML document, such as mathematics and metadata, is included using other namespaces. Metadata is placed in a variety of namespaces, as described in Section 8[5]. The MathML namespace is given special importance, because content in this namespace is considered as fundamental as content in the CellML namespace. An empty CellML element may not contain content in either the CellML or MathML namespace, although it may contain content in other namespaces, including the metadata namespaces defined in this specification.

Table 1 defines all of the namespaces used in the rules defined in this specification. The first three namespaces are in the cellml.org domain, and are associated with the core model structure elements, some custom metadata elements, and an XML serialization of the Object Management Group's bibliographic query service[6] (BQS) data model created for storing citations in CellML. The MathML and RDF namespaces are defined in standards administered by the World Wide Web Consortium. Finally, the Dublin Core and Dublin Core Qualifiers namespaces reference standards for metadata specification administered by the Dublin Core[7] organisation.

Table 1 also gives the recommended prefix to be mapped to each namespace declaration for use in CellML documents. It is recommended that when a CellML element such as **`<model>`** is the root element of a document, CellML be declared as the default namespace for the document and also be explicitly mapped to the `cellml` prefix. This simplifies the association of elements and attributes with the CellML namespace in regions of the document where the default namespace is not the CellML namespace. For instance, the MathML elements used to define equations are typically placed inside a **`<math>`** element that changes the default namespace to the MathML namespace. A **`cellml:units`** attribute in the CellML namespace can then be added to each of MathML's **`<cn>`** elements without having to redeclare the CellML namespace every time it is used.

[3]http://www.w3.org/TR/1999/REC-xml-names-19990114/
[4]http://www.ietf.org/rfc/rfc2396.txt
[5]http://www.cellml.org/public/specification/20010302/cellml_specification.html#sec_metadata
[6]http://www.omg.org/lsr/
[7]http://dublincore.org/

Namespace Name	Namespace URI	Preferred Prefix
CellML	`http://www.cellml.org/2001/03/cellml`	`cellml`
CellML Metadata	`http://www.cellml.org/2001/03/metadata`	`cmeta`
CellML BQS	`http://www.cellml.org/2001/03/bqs`	`bqs`
MathML	`http://www.w3.org/1998/Math/MathML`	`mathml`
RDF	`http://www.w3c.org/1999/02/22-rdf-syntax-ns#`	`rdf`
Dublin Core	`http://purl.org/dc/elements/1.0`	`dc`
Dublin Core Qualifiers	`http://purl.org/dc/qualifiers/1.0`	`dcq`

TABLE 1: The CellML specifi cation defi nes the expected behaviour of CellML processing software for XML elements and attributes that are in these namespaces. Applications may not place their own elements and attributes in these namespaces. See text for more details.

2.2.3 Extending CellML documents

CellML processing software may store information not covered by the CellML specification in a CellML document by defining its own elements and attributes and placing them in a namespace other than one of those defined in Table 1. (This specification only defines the content models of elements in the namespaces in Table 1 with respect to other elements in those namespaces.) Elements and attributes in extension namespaces may appear anywhere in a CellML document, as long as the result is well-formed XML. Because the CellML specification is only concerned with content in the CellML or MathML namespaces, elements in extension namespaces may even appear inside elements defined by the CellML specification to be empty.

It is hoped that CellML processing applications will respect the extension elements and attributes of other applications. If a model is created in application A, which adds its own extension elements, and is subsequently edited in application B, it would be polite if application B included application A's extension elements in its output, even if these extension elements are now invalid. Applications will need to validate their own extension data if a CellML document is read in from a non-trusted location.

The namespace extension mechanism provides a convenient way to associate a small amount of application-specific information with a model defined in CellML. However, it is recommended that applications needing to store large amounts of information, such as rendering or simulation information, do so in a separate document. This will make CellML documents easier to exchange, and will prevent the loss of application-specific information if the model is read into another application.

2.3 Examples

Figure 1 contains some example CellML elements, each of which defines a **`name`** attribute. The values of the **`name`** attribute on the first three elements are valid CellML identifiers. The values of the **`name`** attribute on the last two elements are invalid identifiers.

Figure 2 contains portions of a typical CellML document that demonstrate the recommended use of namespaces. The root element sets the default namespace to the CellML namespace and also explicitly maps the CellML namespace to the `cellml` prefix. The **`<math>`** element that encloses a set of equations inside a component element resets the default namespace to the MathML namespace. The **`units`** attribute on the **`<cn>`** element (which is in the MathML namespace) is placed in the CellML namespace by using the previously-defined `cellml` prefix.

Figure 3 demonstrates how software can embed its own information inside a valid CellML document using XML namespaces. The **`<model>`** element sets the default namespace to the CellML namespace, and maps the `app` prefix to an extension namespace (i.e., one not defined in Table 1). The `app` prefix is

```
<!--
  The following elements have name attributes with valid values.
-->

<component name="my_favorite_component" />

<variable name="_ca2_conc" units="millimolar" />

<model name="model1345" />

<!--
  The following elements have name attributes with invalid values.
  Names may not start with numbers or contain colons.
-->

<component name="123component" />

<component name="my_model:my_component" />
```

FIGURE 1: XML elements defi ning **`name`** attributes. Valid and invalid CellML identifi ers are shown, as noted in the comments.

then used to define an **`<app:component_rendering_information>`** element and two attributes on a **`<component>`** element.

2.4 Rules for CellML Documents

2.4.1 Valid CellML identifiers

- The following is definition of a valid CellML identifier using Backus-Naur notation:

```
letter ::= ’a’...’z’,’A’...’Z’
digit  ::= ’0’...’9’
name   ::= {’_’} letter {letter | ’_’ | digit }
```

  [A valid CellML name must start with a letter or underscore (“_”). If a name starts with an underscore, the next character must be a letter. A name may continue with any alphanumeric character or an underscore. Backus-Naur notation is described in Naur, P. (1960) “Revised Report on the Algorithmic Language ALGOL 60”, *Communications of the ACM*, 3(5):299-314.]

2.4.2 Extension namespaces

- Any element not in one of the namespaces defined in Table 1 is an extension element. Any attribute not in one of the namespaces defined in Table 1 is an extension attribute. Attributes without an explicit namespace declaration are assumed to be in the same namespace as their parent element. Although not explicitly stated, a document author may add elements and attributes in other namespaces anywhere in a CellML document without affecting its validity.

```
<model
    name="simple_electrophysiological_model"
    xmlns="http://www.cellml.org/2001/03/cellml"
    xmlns:cellml="http://www.cellml.org/2001/03/cellml">

  ...

  <component name="extra_cellular_space">
    ...
    <math xmlns="http://www.w3.org/1998/Math/MathML">
      <apply><eq />
        <apply><diff />
          <bvar><ci> time </ci></bvar>
          <ci> Na </ci>
        </apply>
        <apply><times />
          <cn cellml:units="dimensionless"> -1.0 </cn>
          <ci> I_Na </ci>
        </apply>
       </apply>
       ...
    </math>
  </component>

  ...

</model>
```

FIGURE 2: A CellML fragment demonstrating the recommended use of namespaces in a CellML document. This fragment is taken from the simple electro-physiological model example on the CellML website.

```
<model
    name="cellml_model_with_extensions"
    xmlns="http://www.cellml.org/2001/03/cellml"
    xmlns:app="http://www.software.com/cellml_processor">

  <app:component_rendering_information>
    cell : blue
    membrane : yellow
    channel : red
  </app:component_rendering_information>

  <component
      name="cell"
      app:component_type="cell"
      app:render_corners="100, 100, 400, 400" />

</model>
```

FIGURE 3: A CellML document demonstrating the use of XML namespaces to embed application specifi c data inside a CellML document. The extension namespace was invented for demonstration purposes only.

2.4.3 Proper use of the CellML namespace

- Only elements and attributes defined in the CellML specification may be placed in the CellML namespace.

  [Documents containing unknown elements or attributes in the CellML namespace are invalid CellML documents. Rules regarding the use of elements in the other namespaces defined in Table 1 are given in the appropriate sections.]

2.5 Rules for Processor Behaviour

2.5.1 Treatment of CellML identifiers

- CellML identifiers must be handled in a case-sensitive manner.

  [Two CellML elements of the same type may be defined with identifiers of **A** and **a**. Processing software is expected to match the identifiers in a case-sensitive manner when those elements are referenced at other places in the document.]

2.5.2 Treatment of extension namespaces

- CellML processing software is free to do whatever it wishes when it encounters elements and attributes that are not in one of the namespaces defined in Table 1.

  [If the namespace is unrecognised, then software should probably alert the user to its presence. Polite software should attempt to store non-CellML data, so that it can write it out again when it exports the document. Software should validate its own non-CellML data carefully when reading documents from a non-trusted location.]