CellML Specification
Final Draft — 18 May 2001

2 Fundamentals

2.1 Introduction

This section of the CellML specification introduces some concepts that are used throughout the entire language, and defines rules that apply to all or many of the other parts of the specification. These include the definition of names and 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), this functionality is not used in CellML 1.0 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 contain at least one alphanumeric character.

Convenient code generation is also the reason why colons, periods, and hyphens may not appear in CellML identifiers. 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.

2.2.2 Namespaces in CellML

Namespaces in XML[3] is a companion specification to the XML 1.0 specification[4]. XML namespaces add a

[1] http://www.cellml.org/public/specification/20010518/cellml_specification.html#sec_metadata
[2] http://www.cds.caltech.edu/erato/
[3] http://www.w3.org/TR/1999/REC-xml-names-19990114/
[4] http://www.w3.org/TR/2000/REC-xml-20001006

second level of naming to elements and attributes allowing processing software to distinguish between elements and attributes from different languages. A namespace is identified by a Uniform Resource Identifier[5] (URI), which has the feature of being unique. The value of a namespace URI need have nothing to do with the XML document that uses it, but typically points to a document that defines the rules for the language. 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 namespace for the current element and all of its children.

The CellML 1.0 specification defines a small number of elements and attributes and a namespace with which they must be associated. Putting CellML elements and attributes in the CellML namespace allows them to be distinguished 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. The use of namespaces also allows processing software to distinguish elements and attributes from different versions of the CellML specification. 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.

This specification is primarily concerned with the rules and semantics that relate to the elements and attributes in the CellML namespace, which are used in the definition of model structure. It is an error if documents contain elements and attributes in the CellML namespace that are not defined in this specification. This specification also defines how elements and attributes in the MathML, RDF and CellML Metadata namespaces can be combined with elements and attributes in the CellML namespace, and how processing software should deal with content in those namespaces. MathML is particularly important to CellML, because content in this namespace is considered as fundamental as content in the CellML namespace. Metadata is defined using elements in the RDF namespace, and linked to CellML elements using an attribute in the CellML Metadata namespace, as described in Section 8[6]. Any CellML element may contain elements and attributes in other namespaces, which CellML processing software is free to ignore.

Table 1 lists the names, URIs and recommended prefixes of the namespaces referenced in this specification. For interoperability, the root element of any CellML document should set the default namespace and the map the `cellml` prefix to the CellML 1.0 namespace URI. The latter 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.

Namespace Name	Namespace URI	Recommended Prefix
CellML	`"http://www.cellml.org/cellml/1.0#"`	`cellml`
CellML Metadata	`"http://www.cellml.org/metadata/1.0#"`	`cmeta`
MathML	`"http://www.w3.org/1998/Math/MathML"`	`mathml`
RDF	`"http://www.w3.org/1999/02/22-rdf-syntax-ns#"`	`rdf`

TABLE 1: The names, URIs and recommended prefixes of the namespaces referenced in this specification. See text for more details.

[5] http://www.ietf.org/rfc/rfc2396.txt
[6] http://www.cellml.org/public/specification/20010518/cellml_specification.html#sec_metadata

2.2.3 Extending CellML documents

Any namespace with a URI not defined in Table 1 is an *extension namespace*. Any element in an extension namespace is an *extension element*. Any attribute in an *extension attribute* is an extension attribute. Model authors and CellML processing software may store information not covered by the CellML specification in a CellML document by defining their own extension elements and extension attributes. When authors and implementors define extension namespaces, it is recommended that they use URIs under their jurisdiction. Extension elements and extension attributes may appear anywhere in a CellML document, as long as the result is well-formed XML.

For interoperability, CellML processing software should 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, application B should attempt to include 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 passed through applications unaware of the extensions.

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.

```
<!--
  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 consist purely of underscores or contain colons.
-->

<component name="___" />

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

FIGURE 1: XML elements defining **name** attributes. Valid and invalid CellML identifiers are shown, as noted in the comments.

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 URI 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.

```
<model
    name="simple_electrophysiological_model"
    xmlns="http://www.cellml.org/cellml/1.0#"
    xmlns:cellml="http://www.cellml.org/cellml/1.0#">

  ...

  <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.

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 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 a Extended Backus-Naur Form (EBNF) notation:

```
letter ::= 'a'...'z','A'...'Z'
```

```
<model
    name="cellml_model_with_extensions"
    xmlns="http://www.cellml.org/cellml/1.0#"
    xmlns:app="http://www.software.com/cellml_processor"
    xmlns:cellml="http://www.cellml.org/cellml/1.0#">

  <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 specific data inside a CellML document. The extension namespace was invented for demonstration purposes only.

```
digit  ::= '0'...'9'
name   ::= ('_')* ( letter | digit ) ( letter | '_' | digit )*
```

[A valid CellML identifier must consist of only letters, digits or underscores, and must contain at least one letter or digit. The variant of EBNF used above is defined in Section 6 of the XML 1.0 Recommendation[7].]

2.4.2 Proper use of the CellML namespace

- A document must not contain elements or attributes in the CellML namespace that are not defined in this specification.

  [Documents containing unknown elements or attributes in the CellML namespace are not valid CellML documents. Rules regarding the use of elements in the other namespaces defined in Table 1 are given in the appropriate sections. Note that attributes without an explicit prefix declaration are assumed to be in the same namespace as their parent element.]

2.4.3 Extension namespaces

- Although not explicitly stated throughout this specification, a document author may add extension elements and extension attributes to any CellML element in a CellML document without affecting the validity of the document.

  [Note that attributes without an explicit prefix declaration are assumed to be in the same namespace as their parent element.]

[7] http://www.w3.org/TR/2000/REC-xml-20001006#sec-notation

- For interoperability, elements in the CellML namespace set should not be defined inside extension elements.

  [Specifically, applications should not define important model structure, mathematics or metadata information within extension elements, that other applications are free to ignore.]

2.5 Rules for Processor Behaviour

2.5.1 Treatment of CellML identifiers

- CellML processing software must handle identifiers 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 attribute namespaces

- CellML processing software must treat attributes without an explicit namespace declaration as if they were in the same namespace as their parent element.

2.5.3 Treatment of extension namespaces

- CellML processing software may ignore extension elements and extension attributes.

  [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.]

- CellML processing software may ignore the contents of extension elements.