ANNOTATION

- created by hand 2001-11-20
- First draft 2001-11-20

STMML supports domain-independent STM information components

xsd:appinfo
Schematron validation

Data Types and Data Structure

Overview

STMML defines a number of data types suited to STM. It also defines a number of complex data strucures such as arrays, matrices and tables. the constraints are sometimes created through elements and sometimes through attributes. We classify the general components as follows:

Abstract Data Structures

- **scalar**. A scalar quantity, expressible as a string, but with many optional facets such as errors, units, ranges, etc. Most elements may have **countType** attribute to indicate more than one instance.
- **array**. An array of homogeneous `scalar`s whose size is described by **sizeType**. **Delimiter**s in string representations can ve varied.
- **matrix**. A rectangular (often square) matrix of homogeneous `scalar`s. Many matrices have special functions (see **matrixType**) such as geometric transformations
- **table**. An table where the columns are homogeneous `array`s.
- **list**. A list of heterogeneous components from any namespace.
- **sizeType**. Size of arrays
- **delimiterType**. A lexical delimiter

Links and References

- **link**. Support for simple hyperlinks and link structures
- **refType**. A reference to an element
- **namespaceRefType**. A reference to an element, including namespace-like prefixes

Data-based simpleTypes

- **coordinate2Type**. A 2-D coordinate
- **coordinate3Type**. A 3-D coordinate
- **dataTypeType**. An enumeration of data types (similar to those in XML Schema).
- **errorBasisType**. Basis of numeric error estimates
- **minType**. Minimum value
- **maxType**. Maximum value

Common attribute types

- **idType**. Specifies lexical patterns for IDs
- **id**. ID attribute (highly encouraged)
- **title**. Title attribute (highly encouraged)
- **conv**. Convention attribute

Data structure

Data-based simpleTypes

Array-based simpleTypes

General simpleTypes

Numeric types

Links and References

References

General information components

General components

STMML provides a very small number of abstract elements to capture frequently encountered concepts in STM documents. There are no predetermined semantics or ontology; it is expected that descriptive metadata will be added through dictionaries. All elements can contain any element children and can carry the common STM attributes. Currently there are the following:

- **object**. Almost anything - concrete, abstract, representable by a noun. Objects can have properties added through **scalar**, etc.
- **action**. Represents an action performed during a scientific narrative. It has attributes describing a time-line and conditions so that a procedure could be replayed. It has a container **actionList** which shares these attributes and which can describe sets of actions.
- **observation**. Contains narrative or other elements describing an observation, planned or unplanned

Dictionary components

Dictionaries are a major part of STMML and supported as follows:
The dictionary itself:

- **dictionary**. This element defines a dictionary and is often the root element (though a data instance might also be combined with a dictionary). The dictionary play a similar role to a simple schema, by defining data types and other constraints (such as enumerations). By transforming a dictionary to schema format, schema-based tools can be used for validation. A dictionary is normally composed of **entry**s.
- **entry**. An entry contanins information which *describes* or *constrains* elements in a data instance. The link is made through a **dictRef** attribute on the data element. Descriptive information can apply to any type of element (not

necessarily part of or derived from the STM Schema). Constraints are similar to those in XML Schemas and use the same vocabulary (dataTypes, value ranges, enumerations, patterns, etc.). They normally apply to elements from the STM Schema or derived from it.
In addition entrys can constrain elements to have the same higher-level structures and constraints defined by STM Schema. Thus entrys can require a data element to be a matrix, of a given type, with fixed number os rows and columns. These constraints are usually attributes on the entry element, which therefore maps directly onto the instance. Every entry has a mandatory `term` attribute which is the formal text string representing the concept. This string can contain any allowed XML characters (e.g. greek characters) but not markup (e.g. MathML or CML).

- **definition**. An almost mandatory child element of entry, giving a formal definition of the term
- **description**. Additional descriptive informati>on for an entry. This can contain any content, often HTML, but also MathML, CML for description of equations, chemical formulae, etc.
- **alternative**. Alternative strings for describing the concept. These can be any of the stnadard lexical and terminological data categories such as synonyms, abbreviations, homonyms, etc. (see ISO12620 for a full range).
- **enumeration**. A list of allowed values for the data element (or elements in arrays, matrices).
- **relatedEntry**. A related entry. Sometimes this is descriptive (e.g. "seeAlso" provides additional information on related concepts). It can also be used for constraints, and there is a small controlled vocabulary of relationships, but no universal syntax. We support parentage (e.g. through "partitiveParent" = "partOf"). In principle this can be used with **appinfo** to provide algorithmically constructed relationships.
- *attributes*. A wide range of constraints is provided through attributes, several being similar to facets on XML Schema datatypes:
  - **rows** and **columns**, the structure of the data element.
  - **recommendedUnits**, **units** and **unitType**, the units of the data element.
  - **minExclusive**, **minInclusive**, **maxExclusive** and **maxInclusive**, the value of the data element.
  - **totalDigits**, **fractionDigits**, **length**, **maxLength**, **minLength** and **pattern**. The lexical form of the data element.
- **annotation**. Similar to XML Schema, this has children **documentation** for information about the entry (normally curatorial) and **appinfo** to describe entries and constraints in machine-processable fashion. .

Metadata

STMML supports metadata through the element **metadata**. If necessary several of these can be contained in a **metadataList** element.

Scientific Units

Groups (for schema maintenance and re-use)

ELEMENTS

`action`*[el.action]*

An action which might occur in scientific data or narrative.

An action which might occur in scientific data or narrative. The definition is deliberately vague, intending to collect examples of possible usage. Thus an action could be addition of materials, measurement, application of heat or radiation. The content model is unrestricted. `action` iself is normally a child of **actionList**
The start, end and duration attributes should be interpreted as

- XSD dateTimes and XSD durations. This allows precise recording of time of day, etc, or duration after start of actionList. A `convention="xsd"` attribute should be used to enforce XSD.
- a numerical value, with a units attribute linked to a dictionary.
- a human-readable string (unlikely to be machine processable)

`startCondition` and `endCondition` values are not constrained, which allows XSL-like `test` attribute values. The semantics of the conditions are yet to be defined and at present are simply human readable.
The order of the `action` elements in the document may, but will not always, define the order that they actually occur in.
A delay can be shown by an `action` with no content. Repeated actions or actionLists are indicated through the count attribute.

```
<actionList
 xmlns="http://www.xml-cml.org/schema/stmml"
  title="boiling two eggs for breakfast">
  <!-- start cooking at 9am -->
  <action title="turn on heat" start="T09:00:00"
convention="xsd"/>
  <!-- human readable description of time to start action -->
  <action title="put egg into pan" startCondition="water is
boiling" count="2"/>
  <!-- the duration is expressed in ISO8601 format -->
  <action title="boil eggs for 4 minutes" duration="04:00"/>
  <!-- action immediately follows last action -->
  <action title="remove egg from pan" count="1"/>
  <action title="boil second egg for a bit longer"
duration="about half a minute"/>
  <!-- action immediately follows last action -->
  <action title="remove egg from pan" count="1"/>
```

```
</actionList>



<actionList title="preparation of silanols">
  <p>This is a conversion of a chemical synthesis to STMML. We
  have deliberately not marked up the chemistry in this
example!</p>
  <action title="step2">
    <p>Take 1 mmol of the diol and dissolve in dioxan in
      <object title="flask">
        <scalar title="volume" units="units:ml">25</scalar>
      </object>
    </p>
  </action>
  <action title="step2">
    <p>Place flask in water bath with magnetic stirrer</p>
  </action>
  <!-- wait until certain condition -->
  <actionList endCondition="bath temperature stabilised"/>
  <action title="step3">
    <p>Add 0.5 ml 1 M H2SO4</p>
  </action>

  <!-- carry out reaction -->
  <actionList endCondition="reaction complete; no diol spot
remains on TLC">
    <actionList title="check tlc">
      <!-- wait for half an hour -->
      <action duration="half an hour"/>
      <action title="tlc">
        <p>extract solution and check diol spot on TLC</p>
      </action>
    </actionList>
  </actionList>

  <!-- work up reaction -->
  <action title="step5">
    <p>Add 10 ml water to flask</p>
  </action>
  <action title="step6">
    <p>Neutralize acid with 10% NaHCO3</p>
  </action>
  <action title="step7" count="3">
    <p>Extract with 10ml ether</p>
  </action>
  <action title="step8">
    <p>Combine ether layers</p>
  </action>
  <action title="step9" count="2">
    <p>Wash ether with 10 ml water</p>
  </action>
  <action title="step10">
    <p>Wash ether with 10 ml saturated NaCl</p>
  </action>
  <action title="step11">
```

```
    <p>Dry over anhydrous Na2SO4 and remove solvent on rotary
evaporator</p>
  </action>
</actionList>
```

Content Model
(ANY [lax])*

`title`*[att.title]*

A title on an element.

No controlled value.

```
<action title="turn on heat" start="T09:00:00"
convention="xsd"/>
```

`id`*[att.id]*

A unique ID for an element

This is not formally of type ID (an XML NAME which must start with a letter and contain only letters, digits and `.-_:`). It is recommended that IDs start with a letter, and contain no punctuation or whitespace. The function `generate-id()` in XSLT will generate semantically void unique IDs.
It is difficult to ensure uniqueness when documents are merged. We suggest namespacing IDs, perhaps using the containing elements as the base. Thus `mol3:a1` could be a useful unique ID. However this is still experimental.

[xsd:string]
Pattern: [A-Za-z0-9_-]+(:[A-Za-z0-9_-]+)?

An attribute providing a unique ID for an element

`convention`*[att.convention]*

A string referencing a dictionary, units, convention or other metadata.

The namespace is optional but recommended where possible
Note: this convention is only used within STMML and related languages; it is NOT a generic URI.

```
<list>
<!-- dictRef is of namespaceRefType -->
  <scalar dictRef="chem:mpt">123</scalar>
<!-- error -->
  <scalar dictRef="mpt23">123</scalar>
</list>
```

[xsd:string]
Pattern: [A-Za-z][A-Za-z0-9_]*(:[A-Za-z][A-Za-z0-9_]*)?

A reference to a convention

There is no controlled vocabulary for conventions, but the author must ensure that the semantics are openly available and that there are mechanisms for implementation. The convention is inherited by all the subelements, so that a convention for `molecule` would by default extend to its `bond` and `atom` children. This can be overwritten if necessary by an explicit `convention`.
It may be useful to create conventions with namespaces (e.g. `iupac:name`). Use of `convention` will normally require non-STMML semantics, and should be used with caution. We would expect that conventions prefixed with "ISO" would be useful, such as ISO8601 for dateTimes.
There is no default, but the conventions of STMML or the related language (e.g. CML) will be assumed.

```
<bond convention="fooChem" order="-5"
   xmlns:fooChem="http://www.fooChem/conventions"/>
```

`dictRef` *[att.dictRef]*

A string referencing a dictionary, units, convention or other metadata.

The namespace is optional but recommended where possible
Note: this convention is only used within STMML and related languages; it is NOT a generic URI.

```
<list>
<!-- dictRef is of namespaceRefType -->
  <scalar dictRef="chem:mpt">123</scalar>
<!-- error -->
  <scalar dictRef="mpt23">123</scalar>
</list>
```

[xsd:string]
Pattern: [A-Za-z][A-Za-z0-9_]*(:[A-Za-z][A-Za-z0-9_]*)?

A reference to a dictionary entry.

Elements in data instances such as **scalar** may have a `dictRef` attribute to point to an entry in a dictionary. To avoid excessive use of (mutable) filenames and URIs we recommend a namespace prefix, mapped to a namespace URI in the normal manner. In this case, of course, the namespace URI must point to a real XML document containing **entry** elements and validated against STMML Schema.
Where there is concern about the dictionary becoming separated from the document the dictionary entries can be physically included as part of the data instance and the normal XPointer addressing mechanism can be used.
This attribute can also be used on **dictionary** elements to define the namespace prefix

```
<scalar dataType="xsd:float" title="surfaceArea"
  dictRef="cmlPhys:surfArea"
  xmlns:cmlPhys="http://www.xml-cml.org/dict/physical"
  units="units:cm2">50</scalar>
<stm:list xmlns:stm="http://www.xml-cml.org/schema/stmml">
```

```
  <stm:observation>
    <p>We observed <object count="3" dictRef="foo:p1"/>
      constructing dwellings of different material</p>
  </stm:observation>
  <stm:entry id="p1" term="pig">
    <stm:definition>A domesticated animal.</stm:definition>
    <stm:description>Predators include
wolves</stm:description>
    <stm:description class="scientificName">Sus
scrofa</stm:description>
  </stm:entry>
</stm:list>
```

`start` *[att.action.start]*

The start time

The start time in any allowable XSD representation of date, time or dateTime. This will normally be a clock time or date.

`startCondition` *[att.action.startCondition]*

The start condition

This can describe the condition(s) that has to be met before an action can begin, such as in a recipe. Semantics are unexplored but could be used to control robotic operations.

`duration` *[att.action.duration]*

The duration of the action

Semantics undefined.

`end` *[att.action.end]*

The end time

The start time in any allowable XSD representation of date, time or dateTime. This will normally be a clock time or date.

`endCondition` *[att.action.endCondition]*

The end condition

At present a human-readable string describing some condition when the ac tion should end. As XML develops it may be possible to add machine-processable semantics in this field.

`units` *[att.units]*

Scientific units

These will be linked to dictionaries of units with conversion information, using

namespaced references (e.g. si:m)
Distinguish carefully from **unitType** which is an element describing a type of a unit in a **unitList**

```
<stm:unitList xmlns:stm="http://www.xml-cml.org/schema/stmml">

<!--
============================================================================
-->
<!-- ========================= fundamental types
========================== -->
<!--
============================================================================
-->

<stm:unitType id="length" name="length">
  <stm:dimension name="length" power="1"/>
</stm:unitType>

<stm:unitType id="time" name="time">
  <stm:dimension name="time" power="1"/>
</stm:unitType>

<!-- ... -->

<stm:unitType id="dimensionless" name="dimensionless">
  <stm:dimension name="dimensionless" power="1"/>
</stm:unitType>

<!--
============================================================================
-->
<!-- ========================== derived types
============================= -->
<!--
============================================================================
-->

<stm:unitType id="acceleration" name="acceleration">
  <stm:dimension name="length" power="1"/>
  <stm:dimension name="time" power="-2"/>
</stm:unitType>

<!-- ... -->

<!--
============================================================================
-->
<!-- ====================== fundamental SI units
========================== -->
<!--
============================================================================
-->

<stm:unit id="second" name="second" unitType="time">
  <stm:description>The SI unit of time</stm:description>
```

```
</stm:unit>

<stm:unit id="meter" name="meter" unitType="length"
  abbreviation="m">
  <stm:description>The SI unit of length</stm:description>
</stm:unit>

<!-- ... -->

<stm:unit id="kg" name="nameless" unitType="dimensionless"
  abbreviation="nodim">
  <stm:description>A fictitious parent for dimensionless
units</stm:description>
</stm:unit>

<!--
=============================================================================
-->
<!-- ===================== derived SI units
============================== -->
<!--
=============================================================================
-->

<stm:unit id="newton" name="newton" unitType="force">
  <stm:description>The SI unit of force</stm:description>
</stm:unit>

<!-- ... -->

<!-- multiples of fundamental SI units -->

<stm:unit id="g" name="gram" unitType="mass"
  parentSI="kg"
  multiplierToSI="0.001"
  abbreviation="g">
  <stm:description>0.001 kg. </stm:description>
</stm:unit>

<stm:unit id="celsius" name="Celsius" parentSI="k"
  multiplierToSI="1"
  constantToSI="273.18">
  <stm:description><p>A common unit of
temperature</p></stm:description>
</stm:unit>

<!-- fundamental non-SI units -->

<stm:unit id="inch" name="inch" parentSI="meter"
   abbreviation="in"
  multiplierToSI="0.0254" >
  <stm:description>An imperial measure of
length</stm:description>
</stm:unit>

<!-- derived non-SI units -->
```

```
<stm:unit id="l" name="litre" unitType="volume"
   parentSI="meterCubed"
   abbreviation="l"
   multiplierToSI="0.001">
  <stm:description>Nearly 1 dm**3 This is not quite
exact</stm:description>
</stm:unit>

<!-- ... -->

<stm:unit id="fahr" name="fahrenheit" parentSI="k"
   abbreviation="F"
  multiplierToSI="0.55555555555555555"
  constantToSI="-17.777777777777777">
  <stm:description>An obsolescent unit of temperature still
used in popular
  meteorology</stm:description>
</stm:unit>

</stm:unitList>
```

[xsd:string]

Scientific units on an element.

These must be taken from a dictionary of units. There should be some mechanism for validating the type of the units against the possible values of the element.

count*[]*

A count multiplier for an element

Many elements represent objects which can occur an arbitrary number of times in a scientific context. Examples are `action`, `object` or `molecule`s.

```
<list>
<object title="frog" count="10"/>
<action title="step3" count="3">
  <p>Add 10 ml reagent</p>
</action>
</list>
```

[xsd:nonNegativeInteger]

Number of times the action should be repeated

ref*[att.ref]*

A reference to an existing element

A reference to an existing element in the document. The target of the ref attribute must exist. The test for validity will normally occur in the element's `appinfo`
Any DOM Node created from this element will normally be a *reference* to another

Node, so that if the target node is modified a the dereferenced content is modified. At present there are no deep copy semantics hardcoded into the schema.

BASE: idType

A unique ID for an element

This is not formally of type ID (an XML NAME which must start with a letter and contain only letters, digits and `.-_:`). It is recommended that IDs start with a letter, and contain no punctuation or whitespace. The function `generate-id()` in XSLT will generate semantically void unique IDs.
It is difficult to ensure uniqueness when documents are merged. We suggest namespacing IDs, perhaps using the containing elements as the base. Thus `mol3:a1` could be a useful unique ID. However this is still experimental.

[xsd:string]
Pattern: [A-Za-z0-9_-]+(:[A-Za-z0-9_-]+)?

A reference to an element of given type

`ref` modifies an element into a reference to an existing element of that type within the document. This is similar to a pointer and it can be thought of a strongly typed hyperlink. It may also be used for "subclassing" or "overriding" elements.

```
<cml>
  <molecule id="m1">
    <atomArray>
      <atom elementType="N"/>
      <atom elementType="O"/>
    </atomArray>
  </molecule>
  <html:p>The action of <molecule ref="#m1"/> on cardiac
muscle ...</html:p>
</cml>
```

`type` *[att.action.type]*

The type of the action.

Semantics are not controlled.

`actionList` *[el.actionList]*

A container for a group of **actions**

`ActionList` contains a series of `action`s or nested `actionList`s.

See examples in **action**

```
<!-- demonstrating parallel and sequential actions -->
<actionList order="parallel" endCondition="all food cooked">
  <!-- meat and potatoes are cooked in parallel -->
  <actionList title="meat">
```

```
    <action title="cook" endCondition="cooked">
      <p>Roast meat</p>
    </action>
    <action><p>Keep warm in oven</p></action>
  </actionList>
  <actionList title="vegetables">
    <actionList title="cookVeg" endCondition="cooked">
      <action title="boil water" endCondition="water boiling">
        <p>Heat water</p>
      </action>
      <action title="cook" endCondition="potatoes cooked">
        <p>Cook potatoes</p>
      </action>
    </actionList>
    <action><p>Keep warm in oven</p></action>
  </actionList>
</actionList>
```

Content Model

(ANY [lax])*

`title`*[att.title]*

A title on an element.

No controlled value.

```
<action title="turn on heat" start="T09:00:00"
convention="xsd"/>
```

`id`*[att.id]*

A unique ID for an element

This is not formally of type ID (an XML NAME which must start with a letter and contain only letters, digits and `.-_:`). It is recommended that IDs start with a letter, and contain no punctuation or whitespace. The function `generate-id()` in XSLT will generate semantically void unique IDs.
It is difficult to ensure uniqueness when documents are merged. We suggest namespacing IDs, perhaps using the containing elements as the base. Thus `mol3:a1` could be a useful unique ID. However this is still experimental.

[xsd:string]
Pattern: [A-Za-z0-9_-]+(:[A-Za-z0-9_-]+)?

An attribute providing a unique ID for an element

`convention`*[att.convention]*

A string referencing a dictionary, units, convention or other metadata.

The namespace is optional but recommended where possible
Note: this convention is only used within STMML and related languages; it is NOT a generic URI.

```
<list>
<!-- dictRef is of namespaceRefType -->
  <scalar dictRef="chem:mpt">123</scalar>
<!-- error -->
  <scalar dictRef="mpt23">123</scalar>
</list>
```

[xsd:string]
Pattern: [A-Za-z][A-Za-z0-9_]*(:[A-Za-z][A-Za-z0-9_]*)?

A reference to a convention

There is no controlled vocabulary for conventions, but the author must ensure that the semantics are openly available and that there are mechanisms for implementation. The convention is inherited by all the subelements, so that a convention for `molecule` would by default extend to its `bond` and `atom` children. This can be overwritten if necessary by an explicit `convention`.
It may be useful to create conventions with namespaces (e.g. `iupac:name`). Use of `convention` will normally require non-STMML semantics, and should be used with caution. We would expect that conventions prefixed with "ISO" would be useful, such as ISO8601 for dateTimes.
There is no default, but the conventions of STMML or the related language (e.g. CML) will be assumed.

```
<bond convention="fooChem" order="-5"
   xmlns:fooChem="http://www.fooChem/conventions"/>
```

`dictRef` *[att.dictRef]*

A string referencing a dictionary, units, convention or other metadata.

The namespace is optional but recommended where possible
Note: this convention is only used within STMML and related languages; it is NOT a generic URI.

```
<list>
<!-- dictRef is of namespaceRefType -->
  <scalar dictRef="chem:mpt">123</scalar>
<!-- error -->
  <scalar dictRef="mpt23">123</scalar>
</list>
```

[xsd:string]
Pattern: [A-Za-z][A-Za-z0-9_]*(:[A-Za-z][A-Za-z0-9_]*)?

A reference to a dictionary entry.

Elements in data instances such as **scalar** may have a `dictRef` attribute to point to an entry in a dictionary. To avoid excessive use of (mutable) filenames and URIs we recommend a namespace prefix, mapped to a namespace URI in the normal manner. In this case, of course, the namespace URI must point to a real XML

document containing **entry** elements and validated against STMML Schema.
Where there is concern about the dictionary becoming separated from the document the dictionary entries can be physically included as part of the data instance and the normal XPointer addressing mechanism can be used.
This attribute can also be used on **dictionary** elements to define the namespace prefix

```
<scalar dataType="xsd:float" title="surfaceArea"
  dictRef="cmlPhys:surfArea"
  xmlns:cmlPhys="http://www.xml-cml.org/dict/physical"
  units="units:cm2">50</scalar>
<stm:list xmlns:stm="http://www.xml-cml.org/schema/stmml">
  <stm:observation>
    <p>We observed <object count="3" dictRef="foo:p1"/>
      constructing dwellings of different material</p>
  </stm:observation>
  <stm:entry id="p1" term="pig">
    <stm:definition>A domesticated animal.</stm:definition>
    <stm:description>Predators include
wolves</stm:description>
    <stm:description class="scientificName">Sus
scrofa</stm:description>
  </stm:entry>
</stm:list>
```

`start` *[att.action.start]*

The start time

The start time in any allowable XSD representation of date, time or dateTime. This will normally be a clock time or date.

`startCondition` *[att.action.startCondition]*

The start condition

This can describe the condition(s) that has to be met before an action can begin, such as in a recipe. Semantics are unexplored but could be used to control robotic operations.

`duration` *[att.action.duration]*

The duration of the action

Semantics undefined.

`end` *[att.action.end]*

The end time

The start time in any allowable XSD representation of date, time or dateTime. This will normally be a clock time or date.

`endCondition` *[att.action.endCondition]*

The end condition

At present a human-readable string describing some condition when the ac tion should end. As XML develops it may be possible to add machine-processable semantics in this field.

units*[att.units]*

Scientific units

These will be linked to dictionaries of units with conversion information, using namespaced references (e.g. `si:m`)
Distinguish carefully from **unitType** which is an element describing a type of a unit in a **unitList**

```
<stm:unitList xmlns:stm="http://www.xml-cml.org/schema/stmml">

<!--
============================================================================
-->
<!-- ======================== fundamental types
========================== -->
<!--
============================================================================
-->

<stm:unitType id="length" name="length">
  <stm:dimension name="length" power="1"/>
</stm:unitType>

<stm:unitType id="time" name="time">
  <stm:dimension name="time" power="1"/>
</stm:unitType>

<!-- ... -->

<stm:unitType id="dimensionless" name="dimensionless">
  <stm:dimension name="dimensionless" power="1"/>
</stm:unitType>

<!--
============================================================================
-->
<!-- ========================== derived types
============================= -->
<!--
============================================================================
-->

<stm:unitType id="acceleration" name="acceleration">
  <stm:dimension name="length" power="1"/>
  <stm:dimension name="time" power="-2"/>
</stm:unitType>

<!-- ... -->
```

```
<!--
=============================================================================
-->
<!-- ====================== fundamental SI units
========================== -->
<!--
=============================================================================
-->

<stm:unit id="second" name="second" unitType="time">
  <stm:description>The SI unit of time</stm:description>
</stm:unit>

<stm:unit id="meter" name="meter" unitType="length"
  abbreviation="m">
  <stm:description>The SI unit of length</stm:description>
</stm:unit>

<!-- ... -->

<stm:unit id="kg" name="nameless" unitType="dimensionless"
  abbreviation="nodim">
  <stm:description>A fictitious parent for dimensionless
units</stm:description>
</stm:unit>

<!--
=============================================================================
-->
<!-- ===================== derived SI units
============================== -->
<!--
=============================================================================
-->

<stm:unit id="newton" name="newton" unitType="force">
  <stm:description>The SI unit of force</stm:description>
</stm:unit>

<!-- ... -->

<!-- multiples of fundamental SI units -->

<stm:unit id="g" name="gram" unitType="mass"
  parentSI="kg"
  multiplierToSI="0.001"
  abbreviation="g">
  <stm:description>0.001 kg. </stm:description>
</stm:unit>

<stm:unit id="celsius" name="Celsius" parentSI="k"
  multiplierToSI="1"
  constantToSI="273.18">
  <stm:description><p>A common unit of
temperature</p></stm:description>
```

```
</stm:unit>

<!-- fundamental non-SI units -->

<stm:unit id="inch" name="inch" parentSI="meter"
   abbreviation="in"
  multiplierToSI="0.0254" >
  <stm:description>An imperial measure of
length</stm:description>
</stm:unit>

<!-- derived non-SI units -->

<stm:unit id="l" name="litre" unitType="volume"
   parentSI="meterCubed"
   abbreviation="l"
   multiplierToSI="0.001">
  <stm:description>Nearly 1 dm**3 This is not quite
exact</stm:description>
</stm:unit>

<!-- ... -->

<stm:unit id="fahr" name="fahrenheit" parentSI="k"
   abbreviation="F"
  multiplierToSI="0.55555555555555555"
  constantToSI="-17.777777777777777">
  <stm:description>An obsolescent unit of temperature still
used in popular
  meteorology</stm:description>
</stm:unit>

</stm:unitList>
```

[xsd:string]

Scientific units on an element.

These must be taken from a dictionary of units. There should be some mechanism for validating the type of the units against the possible values of the element.

count[]

A count multiplier for an element

Many elements represent objects which can occur an arbitrary number of times in a scientific context. Examples are `action`, `object` or `molecule`s.

```
<list>
<object title="frog" count="10"/>
<action title="step3" count="3">
  <p>Add 10 ml reagent</p>
</action>
</list>
```

[xsd:nonNegativeInteger]

Number of times the action should be repeated

`type`*[]*

The type of the actionList; no defined semantics

`order`*[att.actionList.order]*

Describes whether child elements are sequential or parallel. There is no default.

Allowed values

- sequential
- parallel

`alternative`*[el.alternative]*

An alternative name for an entry

At present a child of **entry** which represents an alternative string that refers to the concept. There is a partial controlled vocabulary in `alternativeType` with values such as :

- synonym
- acronym
- abbreviation

```
 <entry term="ammonia" id="a1">
   <alternative type="synonym">Spirits of
hartshorn</alternative>
   <alternative type="my:formula">NH3</alternative>
 </entry>
```

Content Model

[xsd:string]

`type`*[att.alternative.type]*

UNION OF

Allowed values

- synonym
- quasi-synonym
- acronym
- abbreviation
- homonym
- identifier

BASE: namespaceRefType

A string referencing a dictionary, units, convention or other metadata.

The namespace is optional but recommended where possible

Note: this convention is only used within STMML and related languages; it is NOT a

generic URI.

```
<list>
<!-- dictRef is of namespaceRefType -->
  <scalar dictRef="chem:mpt">123</scalar>
<!-- error -->
  <scalar dictRef="mpt23">123</scalar>
</list>
```

[xsd:string]
Pattern: [A-Za-z][A-Za-z0-9_]*(:[A-Za-z][A-Za-z0-9_]*)?

annotation *[el.annotation]*

A documentation container similar to `annotation` in XML Schema.

A documentation container similar to `annotation` in XML Schema. At present this is experimental and designed to be used for dictionaries, units, etc. One approach is to convert these into XML Schemas when the `documentation` and `appinfo` children will emerge in their correct position in the derived schema.
It is possible that this may develop as a useful tool for annotating components of complex objects such as molecules.

```
<entry term="matrix"
xmlns="http://www.xml-cml.org/schema/stmml">
  <annotation>
    <documentation><!--This refers to mathematical
matrices-->></documentation>
    <appinfo><!-- ...some code to describe and support
matrices ... --></appinfo>
  </annotation>
</entry>
```

Content Model
(documentation|appinfo)*

source *[]*

Source of the annotation

Semantics undefined

```
<entry term="matrix"
xmlns="http://www.xml-cml.org/schema/stmml">
  <annotation>
    <documentation><!--This refers to mathematical
matrices-->></documentation>
    <appinfo><!-- ...some code to describe and support
matrices ... --></appinfo>
  </annotation>
</entry>
```

appinfo *[el.appinfo]*

A container similar to `appinfo` in XML Schema.

A container for machine processable documentation for an entry. This is likely to be platform and/or language specific. It is possible that XSLT, RDF or XBL will emerge as generic languages
See **annotation** and **documentation** for further information

An example in XSLT where an element `foo` calls a bespoke template
.

```
<s:appinfo
  xmlns:s="http://www.xml-cml.org/schema/cml2/core"
  xmlns="http://www.w3.org/1999/XSL/Transform">
  <template match="foo">
    <call-template name="processFoo"/>
  </template>
</s:appinfo>
```

Content Model
(ANY [lax])*

---

source *[]*

Source of the appinfo

Semantics undefined

```
<entry term="matrix"
xmlns="http://www.xml-cml.org/schema/stmml">
  <annotation>
    <documentation><!--This refers to mathematical
matrices-->></documentation>
    <appinfo><!-- ...some code to describe and support
matrices ... --></appinfo>
  </annotation>
</entry>
```

array *[el.array]*

A homogenous 1-dimensional array of similar objects.

`array` manages a homogenous 1-dimensional array of similar objects. These can be encoded as strings (i.e. XSD-like datatypes) and are concatenated as string content. The size of the array should always be >= 1.
The default delimiter is whitespace. The `normalize-space()` function of XSLT could be used to normalize all whitespace to single spaces and this would not affect the value of the array elements. To extract the elements `java.lang.StringTokenizer` could be used. If the elements themselves contain

whitespace then a different delimiter must be used and is identified through the `delimiter` attribute. This method is mandatory if it is required to represent empty strings. If a delimiter is used it MUST start and end the array - leading and trailing whitespace is ignored. Thus `size+1` occurrences of the delimiter character are required. If non-normalized whitespace is to be encoded (e.g. newlines, tabs, etc) you are recommended to translate it character-wise to XML character entities.
Note that normal Schema validation tools cannot validate the elements of **array** (they are defined as `string`) However if the string is split, a temporary schema can be constructed from the type and used for validation. Also the type can be contained in a dictionary and software could decide to retrieve this and use it for validation.
When the elements of the `array` are not simple scalars (e.g. **scalar**s with a value and an error, the `scalar`s should be used as the elements. Although this is verbose, it is simple to understand. If there is a demand for more compact representations, it will be possible to define the syntax in a later version.

```
<array size="5" title="value"
  dataType="xsd:decimal">  1.23 2.34 3.45 4.56 5.67</array>
```

the `size` attribute is not mandatory but provides a useful validity check):

```
<array size="5" title="initials" dataType="xsd:string"
  delimiter="/">/A B//C/D-E/F/</array>
```

Note that the second array-element is the empty string ''.

```
<array title="mass" size="4"
  units="unit:g"
  errorBasis="observedStandardDeviation"
  minValues="10 11 10 9"
  maxValues="12 14 12 11"
  errorValues="1 2 1 1"
  dataType="xsd:float">11 12.5 10.9 10.2
</array>
```

Content Model

[xsd:string]

`title` *[att.title]*

A title on an element.

No controlled value.

```
<action title="turn on heat" start="T09:00:00"
convention="xsd"/>
```

`id` *[att.id]*

A unique ID for an element

This is not formally of type ID (an XML NAME which must start with a letter and contain only letters, digits and `.-_:`). It is recommended that IDs start with a letter, and contain no punctuation or whitespace. The function `generate-id()` in XSLT will generate semantically void unique IDs.
It is difficult to ensure uniqueness when documents are merged. We suggest namespacing IDs, perhaps using the containing elements as the base. Thus `mol3:a1` could be a useful unique ID. However this is still experimental.

[xsd:string]
Pattern: [A-Za-z0-9_-]+(:[A-Za-z0-9_-]+)?

An attribute providing a unique ID for an element

`convention` *[att.convention]*

A string referencing a dictionary, units, convention or other metadata.

The namespace is optional but recommended where possible
Note: this convention is only used within STMML and related languages; it is NOT a generic URI.

```
<list>
<!-- dictRef is of namespaceRefType -->
  <scalar dictRef="chem:mpt">123</scalar>
<!-- error -->
  <scalar dictRef="mpt23">123</scalar>
</list>
```

[xsd:string]
Pattern: [A-Za-z][A-Za-z0-9_]*(:[A-Za-z][A-Za-z0-9_]*)?

A reference to a convention

There is no controlled vocabulary for conventions, but the author must ensure that the semantics are openly available and that there are mechanisms for implementation. The convention is inherited by all the subelements, so that a convention for `molecule` would by default extend to its `bond` and `atom` children. This can be overwritten if necessary by an explicit `convention`.
It may be useful to create conventions with namespaces (e.g. `iupac:name`). Use of `convention` will normally require non-STMML semantics, and should be used with caution. We would expect that conventions prefixed with "ISO" would be useful, such as ISO8601 for dateTimes.
There is no default, but the conventions of STMML or the related language (e.g. CML) will be assumed.

```
<bond convention="fooChem" order="-5"
   xmlns:fooChem="http://www.fooChem/conventions"/>
```

`dictRef` *[att.dictRef]*

A string referencing a dictionary, units, convention or other metadata.

The namespace is optional but recommended where possible
Note: this convention is only used within STMML and related languages; it is NOT a generic URI.

```
<list>
<!-- dictRef is of namespaceRefType -->
  <scalar dictRef="chem:mpt">123</scalar>
<!-- error -->
```

```
  <scalar dictRef="mpt23">123</scalar>
</list>
```

[xsd:string]
Pattern: [A-Za-z][A-Za-z0-9_]*(:[A-Za-z][A-Za-z0-9_]*)?

A reference to a dictionary entry.

Elements in data instances such as **scalar** may have a `dictRef` attribute to point to an entry in a dictionary. To avoid excessive use of (mutable) filenames and URIs we recommend a namespace prefix, mapped to a namespace URI in the normal manner. In this case, of course, the namespace URI must point to a real XML document containing **entry** elements and validated against STMML Schema.
Where there is concern about the dictionary becoming separated from the document the dictionary entries can be physically included as part of the data instance and the normal XPointer addressing mechanism can be used.
This attribute can also be used on **dictionary** elements to define the namespace prefix

```
<scalar dataType="xsd:float" title="surfaceArea"
  dictRef="cmlPhys:surfArea"
  xmlns:cmlPhys="http://www.xml-cml.org/dict/physical"
  units="units:cm2">50</scalar>
<stm:list xmlns:stm="http://www.xml-cml.org/schema/stmml">
  <stm:observation>
    <p>We observed <object count="3" dictRef="foo:p1"/>
      constructing dwellings of different material</p>
  </stm:observation>
  <stm:entry id="p1" term="pig">
    <stm:definition>A domesticated animal.</stm:definition>
    <stm:description>Predators include
wolves</stm:description>
    <stm:description class="scientificName">Sus
scrofa</stm:description>
  </stm:entry>
</stm:list>
```

dataType[]REQUIRED

an enumerated type for all builtin allowed dataTypes in STM

`dataTypeType` represents an enumeration of allowed dataTypes (at present identical with those in XML-Schemas (Part2- datatypes). This means that implementers should be able to use standard XMLSchema-based tools for validation without major implementation problems.
It will often be used an an attribute on **scalar**, **array** or **matrix** elements.

```
<list xmlns="http://www.xml-cml.org/schema/cml2/core">
  <scalar dataType="xsd:boolean" title="she loves
me">true</scalar>
  <scalar dataType="xsd:float" title="x">23.2</scalar>
  <scalar dataType="xsd:duration" title="egg
timer">PM4</scalar>
```

```
  <scalar dataType="xsd:dateTime" title="current data and
time">2001-02-01:00:30</scalar>
  <scalar dataType="xsd:time" title="wake up">06:00</scalar>
  <scalar dataType="xsd:date" title="where is
it">1752-09-10</scalar>
  <scalar dataType="xsd:anyURI" title="CML
site">http://www.xml-cml.org/</scalar>
  <scalar dataType="xsd:QName" title="CML
atom">cml:atom</scalar>
  <scalar dataType="xsd:normalizedString" title="song">the
mouse ran up the clock</scalar>
  <scalar dataType="xsd:language" title="UK
English">en-GB</scalar>
  <scalar dataType="xsd:Name" title="atom">atom</scalar>
  <scalar dataType="xsd:ID" title="XML ID">_123</scalar>
  <scalar dataType="xsd:integer" title="the
answer">42</scalar>
  <scalar dataType="xsd:nonPositiveInteger"
title="zero">0</scalar>
</list>
```

UNION OF
Allowed values

- xsd:string
- xsd:boolean
- xsd:float
- xsd:double
- xsd:decimal
- xsd:duration
- xsd:dateTime
- xsd:time
- xsd:date
- xsd:gYearMonth
- xsd:gYear
- xsd:gMonthDay
- xsd:gDay
- xsd:gMonth
- xsd:hexBinary
- xsd:base64Binary
- xsd:anyURI
- xsd:QName
- xsd:NOTATION
- xsd:normalizedString
- xsd:token
- xsd:language
- xsd:IDREFS
- xsd:ENTITIES
- xsd:NMTOKEN
- xsd:NMTOKENS
- xsd:Name
- xsd:NCName

- xsd:ID
- xsd:IDREF
- xsd:ENTITY
- xsd:integer
- xsd:nonPositiveInteger
- xsd:negativeInteger
- xsd:long
- xsd:int
- xsd:short
- xsd:byte
- xsd:nonNegativeInteger
- xsd:unsignedLong
- xsd:unsignedInt
- xsd:unsignedShort
- xsd:unsignedByte
- xsd:positiveInteger

[xsd:QName]

The mandatory data type.

All elements of the array must have the same dataType

---

errorValues[]

An array of floats

An array of floats or other real numbers. Not used in STM Schema, but re-used by CML and other languages.

```
<atomArray xmlns="http://www.xml-cml.org/schema/cml2/core"
  x2="1.2 2.3 3.4 5.6"/>
```

XSD:LIST of xsd:decimal

an optional array of error values for numeric arrays

---

errorBasis[att.errorBasis]

The basis of an error value

Errors in values can be of several types and this simpleType provides a small controlled vocabulary

```
<scalar
    dataType="xsd:decimal"
    errorValue="1.0"
    errorBasis="observedStandardDeviation"
    title="body weight"
    dictRef="zoo:bodywt"
    units="units:g">34.3</scalar>
```

Allowed values

- observedRange
- observedStandardDeviation
- observedStandardError
- estimatedStandardDeviation

- estimatedStandardError

`minValues`*[]*

An array of floats

An array of floats or other real numbers. Not used in STM Schema, but re-used by CML and other languages.

```
<atomArray xmlns="http://www.xml-cml.org/schema/cml2/core"
  x2="1.2 2.3 3.4 5.6"/>
```

XSD:LIST of xsd:decimal

an optional array of minimum values for numeric arrays

`maxValues`*[]*

An array of floats

An array of floats or other real numbers. Not used in STM Schema, but re-used by CML and other languages.

```
<atomArray xmlns="http://www.xml-cml.org/schema/cml2/core"
  x2="1.2 2.3 3.4 5.6"/>
```

XSD:LIST of xsd:decimal

an optional array of maximum values for numeric arrays

`units`*[att.units]*

Scientific units

These will be linked to dictionaries of units with conversion information, using namespaced references (e.g. `si:m`)
Distinguish carefully from **unitType** which is an element describing a type of a unit in a **unitList**

```
<stm:unitList xmlns:stm="http://www.xml-cml.org/schema/stmml">

<!--
============================================================================
-->
<!-- ========================= fundamental types
========================== -->
<!--
============================================================================
-->

<stm:unitType id="length" name="length">
  <stm:dimension name="length" power="1"/>
</stm:unitType>

<stm:unitType id="time" name="time">
  <stm:dimension name="time" power="1"/>
</stm:unitType>
```

```
<!-- ... -->

<stm:unitType id="dimensionless" name="dimensionless">
  <stm:dimension name="dimensionless" power="1"/>
</stm:unitType>

<!--
============================================================================
-->
<!-- ========================== derived types
============================== -->
<!--
============================================================================
-->

<stm:unitType id="acceleration" name="acceleration">
  <stm:dimension name="length" power="1"/>
  <stm:dimension name="time" power="-2"/>
</stm:unitType>

<!-- ... -->

<!--
============================================================================
-->
<!-- ====================== fundamental SI units
========================== -->
<!--
============================================================================
-->

<stm:unit id="second" name="second" unitType="time">
  <stm:description>The SI unit of time</stm:description>
</stm:unit>

<stm:unit id="meter" name="meter" unitType="length"
  abbreviation="m">
  <stm:description>The SI unit of length</stm:description>
</stm:unit>

<!-- ... -->

<stm:unit id="kg" name="nameless" unitType="dimensionless"
  abbreviation="nodim">
  <stm:description>A fictitious parent for dimensionless
units</stm:description>
</stm:unit>

<!--
============================================================================
-->
<!-- ==================== derived SI units
============================== -->
<!--
============================================================================
```

```
-->

<stm:unit id="newton" name="newton" unitType="force">
  <stm:description>The SI unit of force</stm:description>
</stm:unit>

<!-- ... -->

<!-- multiples of fundamental SI units -->

<stm:unit id="g" name="gram" unitType="mass"
  parentSI="kg"
  multiplierToSI="0.001"
  abbreviation="g">
  <stm:description>0.001 kg. </stm:description>
</stm:unit>

<stm:unit id="celsius" name="Celsius" parentSI="k"
  multiplierToSI="1"
  constantToSI="273.18">
  <stm:description><p>A common unit of
temperature</p></stm:description>
</stm:unit>

<!-- fundamental non-SI units -->

<stm:unit id="inch" name="inch" parentSI="meter"
   abbreviation="in"
  multiplierToSI="0.0254" >
  <stm:description>An imperial measure of
length</stm:description>
</stm:unit>

<!-- derived non-SI units -->

<stm:unit id="l" name="litre" unitType="volume"
   parentSI="meterCubed"
   abbreviation="l"
   multiplierToSI="0.001">
  <stm:description>Nearly 1 dm**3 This is not quite
exact</stm:description>
</stm:unit>

<!-- ... -->

<stm:unit id="fahr" name="fahrenheit" parentSI="k"
   abbreviation="F"
  multiplierToSI="0.55555555555555555"
  constantToSI="-17.777777777777777">
  <stm:description>An obsolescent unit of temperature still
used in popular
  meteorology</stm:description>
</stm:unit>

</stm:unitList>
```

[xsd:string]

Scientific units on an element.

These must be taken from a dictionary of units. There should be some mechanism for validating the type of the units against the possible values of the element.

delimiter *[att.delimiter]*

A non-whitespace character used in arrays to separate components

Some STMML elements (such as **array**) have content representing concatenated values. The default separator is whitespace (which can be normalised) and this should be used whenever possible. However in some cases the values are empty, or contain whitespace or other problematic punctuation, and a delimiter is required. Note that the content string MUST start and end with the delimiter so there is no ambiguity as to what the components are. Only printable characters from the ASCII character set should be used, and character entities should be avoided.
When delimiters are used to separate precise whitespace this should always consist of spaces and not the other allowed whitespace characters (newline, tabs, etc.). If the latter are important it is probably best to redesign the application.

```
<array size="4"  dataType="xsd:string" delimiter="|">|A|B12||D
and   E|</array>


 The values in the array are
  "A", "B12", "" (empty string) and "D and   E"
 note the spaces
```

[xsd:string]

A delimiter character for arrays and matrices

By default array components ('elements' in the non-XML sense) are whitespace-separated. This fails for components with embedded whitespace or missing completely:

```
        Example:
        In the protein database ' CA' and 'CA' are different
atom types, and and array could be:
        <array delimiter="/" dictRef="pdb:atomTypes">/ N/
CA/CA/ N/</array>
```

Note that the array starts and ends with the delimiter, which must be chosen to avoid accidental use. There is currently no syntax for escaping delimiters.

size *[att.size]*

The size of an array

The size of an array. Redundant, but serves as a check for processing software (useful if delimiters are used)

[xsd:positiveInteger]

The size of an array, matrix, list, etc.

`ref` *[att.ref]*

A reference to an existing element

A reference to an existing element in the document. The target of the ref attribute must exist. The test for validity will normally occur in the element's `appinfo`
Any DOM Node created from this element will normally be a *reference* to another Node, so that if the target node is modified a the dereferenced content is modified. At present there are no deep copy semantics hardcoded into the schema.

BASE: idType

A unique ID for an element

This is not formally of type ID (an XML NAME which must start with a letter and contain only letters, digits and `.-_:`). It is recommended that IDs start with a letter, and contain no punctuation or whitespace. The function `generate-id()` in XSLT will generate semantically void unique IDs.
It is difficult to ensure uniqueness when documents are merged. We suggest namespacing IDs, perhaps using the containing elements as the base. Thus `mol3:a1` could be a useful unique ID. However this is still experimental.

[xsd:string]
Pattern: [A-Za-z0-9_-]+(:[A-Za-z0-9_-]+)?

A reference to an element of given type

`ref` modifies an element into a reference to an existing element of that type within the document. This is similar to a pointer and it can be thought of a strongly typed hyperlink. It may also be used for "subclassing" or "overriding" elements.

```
<cml>
  <molecule id="m1">
    <atomArray>
      <atom elementType="N"/>
      <atom elementType="O"/>
    </atomArray>
  </molecule>
  <html:p>The action of <molecule ref="#m1"/> on cardiac
muscle ...</html:p>
</cml>
```

`definition` *[el.definition]*

The definition for a dictionary entry, scientific units, etc.

The definition should be a short nounal phrase defininig the subject of the entry. Definitions should not include commentary, implementations, equations or formulae

(unless the subject is one of these) or examples. The `description` element can be used for these.
The definition can be in any markup language, but normally XHTML will be used, perhaps with links to other XML namespaces such as CML for chemistry.

From the IUPAC Dictionary of Medicinal Chemistry

```
<entry id="a7" term="Allosteric enzyme">
  <definition>An <a href="#e3">enzyme</a>
that contains a region to which small, regulatory molecules
("effectors") may bind in addition to and separate from the
substrate binding site and thereby affect the catalytic
activity.
  </definition>
  <description>On binding the effector, the catalytic activity
of the
<strong>enzyme</strong> towards the substrate may be enhanced,
in
which case the effector is an activator, or reduced, in which
case
it is a de-activator or inhibitor.
  </description>
</entry>
```

Content Model

(ANY [lax])*

`source`*[att.source]*

An attribute linking to the source of the information

A simple way of adding metadata to a piece of information. Likely to be fragile since the URI may disappear.

```
<list>
  <definition source="foo.html#a3">An animal with four
legs</definition>
  <definition source="http://www.foo.com/index.html">
    An animal with six legs</definition>
</list>
```

`description`*[el.description]*

Descriptive information in a dictionary entry, etc.

Entries should have at least one separate **definition**s. `description` is then used for most of the other information, including examples. The `class` attribute has an uncontrolled vocabulary and can be used to clarify the purposes of the `description` elements.

From IUPAC Dictionary of Medicinal Chemistry

```
<entry id="a7" term="Allosteric enzyme">
  <definition>An <a href="#e3">enzyme</a>
```

```
that contains a region to which small, regulatory molecules
("effectors") may bind in addition to and separate from the
substrate binding site and thereby affect the catalytic
activity.
  </definition>
  <description>On binding the effector, the catalytic activity
of the
<strong>enzyme</strong> towards the substrate may be enhanced,
in
which case the effector is an activator, or reduced, in which
case
it is a de-activator or inhibitor.
  </description>
</entry>
```

Content Model

(ANY [lax])*

`source` *[att.source]*

An attribute linking to the source of the information

A simple way of adding metadata to a piece of information. Likely to be fragile since the URI may disappear.

```
<list>
  <definition source="foo.html#a3">An animal with four
legs</definition>
  <definition source="http://www.foo.com/index.html">
    An animal with six legs</definition>
</list>
```

`class` *[att.description.class]*

The type of this information. This is not controlled, but examples might include:

- description
- summary
- note
- usage
- qualifier

`dictionary` *[el.dictionary]*

A dictionary

A dictionary is a container for **entry** elements. Dictionaries can also contain unit-related information.
The dictRef attribute on a `dictionary` element sets a namespace-like prefix allowing the dictionary to be referenced from within the document. In general dictionaries are referenced from an element using the **dictRef** attribute.

```
<stm:dictionary
xmlns:stm="http://www.xml-cml.org/schema/stmml">
  <stm:entry id="a001" term="Amplitude for charge density
mixing"
    dataType="stm:decimal"
    units="arbitrary">
    <stm:annotation>
      <stm:documentation>
        <div class="summary">Amplitude for charge density
mixing</div>
        <div class="description">Not yet filled in...</div>
      </stm:documentation>
    </stm:annotation>
    <stm:alternative
type="abbreviation">CDMixAmp</stm:alternative>
  </stm:entry>
</stm:dictionary>
```

`dictionary` can be used in an instance document to reference the dictionary used.
Example:

```
<list>
  <dictionary
     dictRef="core" href="../dictionary/coreDict.xml"/>
</list>
```

Content Model
(unitList*,annotation*,description*,entry*)

`title`*[att.title]*

A title on an element.

No controlled value.

```
<action title="turn on heat" start="T09:00:00"
convention="xsd"/>
```

`id`*[att.id]*

A unique ID for an element

This is not formally of type ID (an XML NAME which must start with a letter and contain only letters, digits and `.-_:`). It is recommended that IDs start with a letter, and contain no punctuation or whitespace. The function `generate-id()` in XSLT will generate semantically void unique IDs.
It is difficult to ensure uniqueness when documents are merged. We suggest namespacing IDs, perhaps using the containing elements as the base. Thus `mol3:a1` could be a useful unique ID. However this is still experimental.

[xsd:string]
Pattern: [A-Za-z0-9_-]+(:[A-Za-z0-9_-]+)?

An attribute providing a unique ID for an element

`convention`*[att.convention]*

A string referencing a dictionary, units, convention or other metadata.

The namespace is optional but recommended where possible
Note: this convention is only used within STMML and related languages; it is NOT a generic URI.

```
<list>
<!-- dictRef is of namespaceRefType -->
  <scalar dictRef="chem:mpt">123</scalar>
<!-- error -->
  <scalar dictRef="mpt23">123</scalar>
</list>
```

[xsd:string]
Pattern: [A-Za-z][A-Za-z0-9_]*(:[A-Za-z][A-Za-z0-9_]*)?

A reference to a convention

There is no controlled vocabulary for conventions, but the author must ensure that the semantics are openly available and that there are mechanisms for implementation. The convention is inherited by all the subelements, so that a convention for `molecule` would by default extend to its `bond` and `atom` children. This can be overwritten if necessary by an explicit `convention`.
It may be useful to create conventions with namespaces (e.g. `iupac:name`). Use of `convention` will normally require non-STMML semantics, and should be used with caution. We would expect that conventions prefixed with "ISO" would be useful, such as ISO8601 for dateTimes.
There is no default, but the conventions of STMML or the related language (e.g. CML) will be assumed.

```
<bond convention="fooChem" order="-5"
   xmlns:fooChem="http://www.fooChem/conventions"/>
```

dictRef *[att.dictRef]*

A string referencing a dictionary, units, convention or other metadata.

The namespace is optional but recommended where possible
Note: this convention is only used within STMML and related languages; it is NOT a generic URI.

```
<list>
<!-- dictRef is of namespaceRefType -->
  <scalar dictRef="chem:mpt">123</scalar>
<!-- error -->
  <scalar dictRef="mpt23">123</scalar>
</list>
```

[xsd:string]
Pattern: [A-Za-z][A-Za-z0-9_]*(:[A-Za-z][A-Za-z0-9_]*)?

A reference to a dictionary entry.

Elements in data instances such as **scalar** may have a `dictRef` attribute to point to an entry in a dictionary. To avoid excessive use of (mutable) filenames and URIs we recommend a namespace prefix, mapped to a namespace URI in the normal manner. In this case, of course, the namespace URI must point to a real XML document containing **entry** elements and validated against STMML Schema.
Where there is concern about the dictionary becoming separated from the document the dictionary entries can be physically included as part of the data instance and the normal XPointer addressing mechanism can be used.
This attribute can also be used on **dictionary** elements to define the namespace prefix

```
<scalar dataType="xsd:float" title="surfaceArea"
  dictRef="cmlPhys:surfArea"
  xmlns:cmlPhys="http://www.xml-cml.org/dict/physical"
  units="units:cm2">50</scalar>
<stm:list xmlns:stm="http://www.xml-cml.org/schema/stmml">
  <stm:observation>
    <p>We observed <object count="3" dictRef="foo:p1"/>
      constructing dwellings of different material</p>
  </stm:observation>
  <stm:entry id="p1" term="pig">
    <stm:definition>A domesticated animal.</stm:definition>
    <stm:description>Predators include
wolves</stm:description>
    <stm:description class="scientificName">Sus
scrofa</stm:description>
  </stm:entry>
</stm:list>
```

`href`*[att.dictionary.href]*

URI giving the location of the document. Mandatory if `dictRef` present.

`dimension`*[el.dimension]*

A dimension supporting scientific units

This will be primarily used within the defintion of **units**s.

```
<unitType id="energy" name="energy">
  <dimension name="length" power="1"/>
  <dimension name="mass" power="1"/>
  <dimension name="time" power="-1"/>
</unitType>
```

Content Model

()

`name`*[att.dimension.name]*REQUIRED

Allowed values for dimension Types (for quantities).

These are the 7 types prescribed by the SI system, together with the "dimensionless" type. We intend to be somewhat uncoventional and explore enhanced values of "dimensionless", such as "angle". This may be heretical, but we find the present system impossible to implement in many cases.
Used for constructing entries in a dictionary of units

```
<unitType id="energy" name="energy">
  <dimension name="length" power="2"/>
  <dimension name="mass" power="1"/>
  <dimension name="time" power="-2"/>
</unitType>
```

Allowed values

- mass
- length
- time
- charge
- amount
- luminosity
- temperature
- dimensionless
- angle

  An angle (formally dimensionless, but useful to have units).

The type of the dimension

Normally taken from the seven SI types but possibly expandable.

---

power *[att.dimension.power]* REQUIRED

The power to which the dimension should be raised

Normally an integer. Must be included, even if unity.

documentation *[el.documentation]*

Documentation in the **annotation** of an **entry**

A container similar to `documentation` in XML Schema. This is NOT part of the textual content of an entry but is designed to support the transformation of dictionary entrys into schemas for validation. This is experimental and should only be used for dictionaries, units, etc. One approach is to convert these into XML Schemas when the `documentation` and `appinfo` children will emerge in their correct position in the derived schema.
Do NOT confuse documentation with the **definition** or the **definition** which are part of the content of the dictionary

If will probably only be used when there is significant **appinfo** in the entry or where the entry defines an XSD-like datatype of an element in the document.

```
<stm:documentation id="source"
xmlns:stm="http://www.xml-cml.org/schema/stmml">
Transcribed from IUPAC website
</stm:documentation>
```

Content Model
(ANY [lax])*

`source` *[att.source]*

An attribute linking to the source of the information

A simple way of adding metadata to a piece of information. Likely to be fragile since the URI may disappear.

```
<list>
  <definition source="foo.html#a3">An animal with four
legs</definition>
  <definition source="http://www.foo.com/index.html">
    An animal with six legs</definition>
</list>
```

`id` *[att.id]*

A unique ID for an element

This is not formally of type ID (an XML NAME which must start with a letter and contain only letters, digits and `.-_:`). It is recommended that IDs start with a letter, and contain no punctuation or whitespace. The function `generate-id()` in XSLT will generate semantically void unique IDs.
It is difficult to ensure uniqueness when documents are merged. We suggest namespacing IDs, perhaps using the containing elements as the base. Thus `mol3:a1` could be a useful unique ID. However this is still experimental.

[xsd:string]
Pattern: [A-Za-z0-9_-]+(:[A-Za-z0-9_-]+)?

An attribute providing a unique ID for an element

`entry` *[el.entry]*

A dictionary entry

```
<entry id="a003" term="alpha"
  dataType="float"
```

```
  minInclusive="0.0"
  maxInclusive="180.0"
  recommendedUnits="degrees">
  <definition>The alpha cell angle</definition>
</entry>



<entry id="a003"
  term="matrix1"
  dataType="float"
  rows="3"
  columns="4"
  unitType="unit:length"
  minInclusive="0.0"
  maxInclusive="100.0"
  recommendedUnits="unit:m"
  totalDigits="8"
  fractionDigits="3">
    <definition>A matrix of lengths</definition>
    <description>A data instance will have a matrix which
points
  to this entry (e.g. dictRef="foo:matrix1"). The matrix must
  be 3*4, composed of floats in 8.3 format, of type length,
  values between 0 and 100 and with recommended units metres.
    </description>
</entry>
```

Content Model
((alternative|annotation|definition|description|enumeration|relatedEntry)*)

`title` *[att.title]*

A title on an element.

No controlled value.

```
<action title="turn on heat" start="T09:00:00"
convention="xsd"/>
```

`id` *[att.id]*

A unique ID for an element

This is not formally of type ID (an XML NAME which must start with a letter and contain only letters, digits and `.-_:`). It is recommended that IDs start with a letter, and contain no punctuation or whitespace. The function `generate-id()` in XSLT will generate semantically void unique IDs.
It is difficult to ensure uniqueness when documents are merged. We suggest namespacing IDs, perhaps using the containing elements as the base. Thus `mol3:a1` could be a useful unique ID. However this is still experimental.

[xsd:string]
Pattern: [A-Za-z0-9_-]+(:[A-Za-z0-9_-]+)?

An attribute providing a unique ID for an element

convention*[att.convention]*

A string referencing a dictionary, units, convention or other metadata.

The namespace is optional but recommended where possible
Note: this convention is only used within STMML and related languages; it is NOT a generic URI.

```
<list>
<!-- dictRef is of namespaceRefType -->
  <scalar dictRef="chem:mpt">123</scalar>
<!-- error -->
  <scalar dictRef="mpt23">123</scalar>
</list>
```

[xsd:string]
Pattern: [A-Za-z][A-Za-z0-9_]*(:[A-Za-z][A-Za-z0-9_]*)?

A reference to a convention

There is no controlled vocabulary for conventions, but the author must ensure that the semantics are openly available and that there are mechanisms for implementation. The convention is inherited by all the subelements, so that a convention for `molecule` would by default extend to its `bond` and `atom` children. This can be overwritten if necessary by an explicit `convention`.
It may be useful to create conventions with namespaces (e.g. `iupac:name`). Use of `convention` will normally require non-STMML semantics, and should be used with caution. We would expect that conventions prefixed with "ISO" would be useful, such as ISO8601 for dateTimes.
There is no default, but the conventions of STMML or the related language (e.g. CML) will be assumed.

```
<bond convention="fooChem" order="-5"
   xmlns:fooChem="http://www.fooChem/conventions"/>
```

dataType*[]*

rows*[]*

columns*[]*

recommendedUnits*[]*

Scientific units

These will be linked to dictionaries of units with conversion information, using namespaced references (e.g. `si:m`)
Distinguish carefully from **unitType** which is an element describing a type of a unit in a **unitList**

```
<stm:unitList xmlns:stm="http://www.xml-cml.org/schema/stmml">

<!--
============================================================================
-->
<!-- ========================= fundamental types
========================== -->
<!--
============================================================================
-->

<stm:unitType id="length" name="length">
  <stm:dimension name="length" power="1"/>
</stm:unitType>

<stm:unitType id="time" name="time">
  <stm:dimension name="time" power="1"/>
</stm:unitType>

<!-- ... -->

<stm:unitType id="dimensionless" name="dimensionless">
  <stm:dimension name="dimensionless" power="1"/>
</stm:unitType>

<!--
============================================================================
-->
<!-- ========================== derived types
============================= -->
<!--
============================================================================
-->

<stm:unitType id="acceleration" name="acceleration">
  <stm:dimension name="length" power="1"/>
  <stm:dimension name="time" power="-2"/>
</stm:unitType>

<!-- ... -->

<!--
============================================================================
-->
<!-- ====================== fundamental SI units
========================== -->
<!--
============================================================================
-->

<stm:unit id="second" name="second" unitType="time">
  <stm:description>The SI unit of time</stm:description>
</stm:unit>

<stm:unit id="meter" name="meter" unitType="length"
  abbreviation="m">
```

```
  <stm:description>The SI unit of length</stm:description>
</stm:unit>

<!-- ... -->

<stm:unit id="kg" name="nameless" unitType="dimensionless"
  abbreviation="nodim">
  <stm:description>A fictitious parent for dimensionless
units</stm:description>
</stm:unit>

<!--
=========================================================================
-->
<!-- ===================== derived SI units
============================== -->
<!--
=========================================================================
-->

<stm:unit id="newton" name="newton" unitType="force">
  <stm:description>The SI unit of force</stm:description>
</stm:unit>

<!-- ... -->

<!-- multiples of fundamental SI units -->

<stm:unit id="g" name="gram" unitType="mass"
  parentSI="kg"
  multiplierToSI="0.001"
  abbreviation="g">
  <stm:description>0.001 kg. </stm:description>
</stm:unit>

<stm:unit id="celsius" name="Celsius" parentSI="k"
  multiplierToSI="1"
  constantToSI="273.18">
  <stm:description><p>A common unit of
temperature</p></stm:description>
</stm:unit>

<!-- fundamental non-SI units -->

<stm:unit id="inch" name="inch" parentSI="meter"
   abbreviation="in"
  multiplierToSI="0.0254" >
  <stm:description>An imperial measure of
length</stm:description>
</stm:unit>

<!-- derived non-SI units -->

<stm:unit id="l" name="litre" unitType="volume"
   parentSI="meterCubed"
   abbreviation="l"
```

```
   multiplierToSI="0.001">
  <stm:description>Nearly 1 dm**3 This is not quite
exact</stm:description>
</stm:unit>

<!-- ... -->

<stm:unit id="fahr" name="fahrenheit" parentSI="k"
   abbreviation="F"
  multiplierToSI="0.55555555555555555"
  constantToSI="-17.777777777777777">
  <stm:description>An obsolescent unit of temperature still
used in popular
  meteorology</stm:description>
</stm:unit>

</stm:unitList>
```

[xsd:string]

unitType[]

minExclusive[]

minInclusive[]

maxExclusive[]

maxInclusive[]

totalDigits[]

fractionDigits[]

length[]

minLength[]

maxLength[]

units[]

Scientific units

These will be linked to dictionaries of units with conversion information, using namespaced references (e.g. `si:m`)
Distinguish carefully from **unitType** which is an element describing a type of a unit in a **unitList**

```
<stm:unitList xmlns:stm="http://www.xml-cml.org/schema/stmml">

<!--
```

```
<!-- ========================================================================
-->
<!-- ========================= fundamental types
========================== -->
<!--
========================================================================
-->

<stm:unitType id="length" name="length">
  <stm:dimension name="length" power="1"/>
</stm:unitType>

<stm:unitType id="time" name="time">
  <stm:dimension name="time" power="1"/>
</stm:unitType>

<!-- ... -->

<stm:unitType id="dimensionless" name="dimensionless">
  <stm:dimension name="dimensionless" power="1"/>
</stm:unitType>

<!--
========================================================================
-->
<!-- ============================ derived types
============================== -->
<!--
========================================================================
-->

<stm:unitType id="acceleration" name="acceleration">
  <stm:dimension name="length" power="1"/>
  <stm:dimension name="time" power="-2"/>
</stm:unitType>

<!-- ... -->

<!--
========================================================================
-->
<!-- ======================== fundamental SI units
========================== -->
<!--
========================================================================
-->

<stm:unit id="second" name="second" unitType="time">
  <stm:description>The SI unit of time</stm:description>
</stm:unit>

<stm:unit id="meter" name="meter" unitType="length"
  abbreviation="m">
  <stm:description>The SI unit of length</stm:description>
</stm:unit>
```

```
<!-- ... -->

<stm:unit id="kg" name="nameless" unitType="dimensionless"
  abbreviation="nodim">
  <stm:description>A fictitious parent for dimensionless
units</stm:description>
</stm:unit>

<!--
=============================================================================
-->
<!-- ====================== derived SI units
============================== -->
<!--
=============================================================================
-->

<stm:unit id="newton" name="newton" unitType="force">
  <stm:description>The SI unit of force</stm:description>
</stm:unit>

<!-- ... -->

<!-- multiples of fundamental SI units -->

<stm:unit id="g" name="gram" unitType="mass"
  parentSI="kg"
  multiplierToSI="0.001"
  abbreviation="g">
  <stm:description>0.001 kg. </stm:description>
</stm:unit>

<stm:unit id="celsius" name="Celsius" parentSI="k"
  multiplierToSI="1"
  constantToSI="273.18">
  <stm:description><p>A common unit of
temperature</p></stm:description>
</stm:unit>

<!-- fundamental non-SI units -->

<stm:unit id="inch" name="inch" parentSI="meter"
   abbreviation="in"
  multiplierToSI="0.0254" >
  <stm:description>An imperial measure of
length</stm:description>
</stm:unit>

<!-- derived non-SI units -->

<stm:unit id="l" name="litre" unitType="volume"
   parentSI="meterCubed"
   abbreviation="l"
   multiplierToSI="0.001">
  <stm:description>Nearly 1 dm**3 This is not quite
exact</stm:description>
```

```
</stm:unit>

<!-- ... -->

<stm:unit id="fahr" name="fahrenheit" parentSI="k"
   abbreviation="F"
  multiplierToSI="0.55555555555555555"
  constantToSI="-17.777777777777777">
  <stm:description>An obsolescent unit of temperature still
used in popular
  meteorology</stm:description>
</stm:unit>

</stm:unitList>
```

[xsd:string]

whiteSpace[]

pattern[]

term[]REQUIRED

enumeration*[el.enumeration]*

An enumeration of string values associated with an **entry**

An enumeration of string values. Used where a dictionary entry constrains the possible values in a document instance. The dataTypes (if any) must all be identical and are defined by the dataType of the containing element.

```
<entry term="crystal system" id="cryst1" dataType="string">
  <definition>A crystal system</definition>
  <enumeration value="triclinic">
    <annotation>
      <documentation>
        <div class="summary">No constraints on lengths and
angles</div>
      </documentation>
    </annotation>
  </enumeration>
  <enumeration value="monoclinic">
    <annotation>
      <documentation>
        <div class="summary">Two cell angles are right angles;
no other constraints</div>
      </documentation>
    </annotation>
  </enumeration>
  <enumeration value="orthorhombic">
    <annotation>
      <documentation>
        <div class="summary">All three angles are right
angles; no other constraints</div>
```

```
      </documentation>
    </annotation>
  </enumeration>
  <enumeration value="tetragonal">
    <annotation>
      <documentation>
        <div class="summary">Fourfold axis of symmetry; All
three angles are right angles; two equal cell lengths; no
other constraints</div>
      </documentation>
    </annotation>
  </enumeration>
  <enumeration value="trigonal">
    <annotation>
      <documentation>
        <div class="summary">Threefold axis of symmetry; Two
angles are right angles; one is 120 degrees; two equal
lengths; no other constraints</div>
      </documentation>
    </annotation>
  </enumeration>
  <enumeration value="hexagonal">
    <annotation>
      <documentation>
        <div class="summary">Sixfold axis of symmetry; Two
angles are right angles; one is 120 degrees; two equal
lengths; no other constraints</div>
      </documentation>
    </annotation>
  </enumeration>
  <enumeration value="cubic">
    <annotation>
      <documentation>
        <div class="summary">All three angles are right
angles; all cell lengths are equal</div>
      </documentation>
    </annotation>
  </enumeration>
</entry>
```

An enumeration of string values. The dataTypes (if any) must all be identical and are defined by the dataType of the containing element.
Documentation can be added through an **enumeration** child

Content Model
(annotation?)

---

value*[att.enumeration.value]*

The value of the enumerated element.

Must be compatible with the dataType of the containing element (not schema-checkable directly but possible if dictionary is transformed to schema).

link *[el.link]*

An internal or external link to STMML or other object(s)

Semantics are similar to XLink, but simpler and only a subset is implemented. This is intended to make the instances easy to create and read, and software relatively easy to implement. The architecture is:

- **A single element (`link`) used for all linking purposes**.
- **The link types are determined by the `type` attribute and can be:**.
  - **locator**. This points to a single target and must carry either a `ref` or `href` attribute. `locator` links are usually children of an extended link.
    - **arc**. This is a 1:1 link with both ends (`from` and `to`) defined.
    - **extended**. This is usually a parent of several locator links and serves to create a grouping of link ends (i.e. a list of references in documents).

    Many-many links can be built up from arcs linking extended elements

  All links can have optional `role` attributes. The semantics of this are not defined; you are encouraged to use a URI as described in the XLink specification.

  There are two address spaces:
  - The `href` attribute on locators behaves in the same way as `href` in HTML and is of type `xsd:anyURI`. Its primary use is to use XPointer to reference elements outside the document.
  - The `ref` attribute on locators and the `from` and `to` attributes on `arc`s refer to IDs (*without* the '#' syntax).

  Note: several other specific linking mechanisms are defined elsewhere in STM. **relatedEntry** should be used in dictionaries, and **dictRef** should be used to link to dictionaries. There are no required uses of `link` in STMML but we have used it to map atoms, electrons and bonds in reactions in CML

Relation to XLink. At present (2002) we are not aware of generic XLink processors from which we would benefit, so the complete implementation brings little extra value. Among the simplifications from Xlink are:

- `type` supports only `extended`, `locator` and `arc`
- `label` is not supported and `id`s are used as targets of links.
- `show` and `actuate` are not supported.
- `xlink:title` is not supported (all STM elements can have a `title` attribute).
- `xlink:role` supports any string (i.e. does not have to be a namespaced resource). This mechanism can, of course, still be used and we shall promote it where STM benefits from it
- The `to` and `from` attributes point to IDs rather than labels
- The xlink namespace is not used
- It is not intended to create independent linkbases, although some collections of links may have this property and stand outside the documents they link to

Content Model

(ANY)

`title`*[att.title]*

A title on an element.

No controlled value.

```
<action title="turn on heat" start="T09:00:00"
convention="xsd"/>
```

`id`*[att.id]*

A unique ID for an element

This is not formally of type ID (an XML NAME which must start with a letter and contain only letters, digits and `.-_:`). It is recommended that IDs start with a letter, and contain no punctuation or whitespace. The function `generate-id()` in XSLT will generate semantically void unique IDs.
It is difficult to ensure uniqueness when documents are merged. We suggest namespacing IDs, perhaps using the containing elements as the base. Thus `mol3:a1` could be a useful unique ID. However this is still experimental.

[xsd:string]
Pattern: [A-Za-z0-9_-]+(:[A-Za-z0-9_-]+)?

An attribute providing a unique ID for an element

`convention`*[att.convention]*

A string referencing a dictionary, units, convention or other metadata.

The namespace is optional but recommended where possible
Note: this convention is only used within STMML and related languages; it is NOT a generic URI.

```
<list>
<!-- dictRef is of namespaceRefType -->
  <scalar dictRef="chem:mpt">123</scalar>
<!-- error -->
  <scalar dictRef="mpt23">123</scalar>
</list>
```

[xsd:string]
Pattern: [A-Za-z][A-Za-z0-9_]*(:[A-Za-z][A-Za-z0-9_]*)?

A reference to a convention

There is no controlled vocabulary for conventions, but the author must ensure that the semantics are openly available and that there are mechanisms for implementation. The convention is inherited by all the subelements, so that a convention for `molecule` would by default extend to its `bond` and `atom` children. This can be overwritten if necessary by an explicit `convention`.
It may be useful to create conventions with namespaces (e.g. `iupac:name`). Use of `convention` will normally require non-STMML semantics, and should be used with caution. We would expect that conventions prefixed with "ISO" would be useful, such

as ISO8601 for dateTimes.
There is no default, but the conventions of STMML or the related language (e.g. CML) will be assumed.

```
<bond convention="fooChem" order="-5"
   xmlns:fooChem="http://www.fooChem/conventions"/>
```

dictRef *[att.dictRef]*

A string referencing a dictionary, units, convention or other metadata.

The namespace is optional but recommended where possible
Note: this convention is only used within STMML and related languages; it is NOT a generic URI.

```
<list>
<!-- dictRef is of namespaceRefType -->
  <scalar dictRef="chem:mpt">123</scalar>
<!-- error -->
  <scalar dictRef="mpt23">123</scalar>
</list>
```

[xsd:string]
Pattern: [A-Za-z][A-Za-z0-9_]*(:[A-Za-z][A-Za-z0-9_]*)?

A reference to a dictionary entry.

Elements in data instances such as **scalar** may have a `dictRef` attribute to point to an entry in a dictionary. To avoid excessive use of (mutable) filenames and URIs we recommend a namespace prefix, mapped to a namespace URI in the normal manner. In this case, of course, the namespace URI must point to a real XML document containing **entry** elements and validated against STMML Schema.
Where there is concern about the dictionary becoming separated from the document the dictionary entries can be physically included as part of the data instance and the normal XPointer addressing mechanism can be used.
This attribute can also be used on **dictionary** elements to define the namespace prefix

```
<scalar dataType="xsd:float" title="surfaceArea"
  dictRef="cmlPhys:surfArea"
  xmlns:cmlPhys="http://www.xml-cml.org/dict/physical"
  units="units:cm2">50</scalar>
<stm:list xmlns:stm="http://www.xml-cml.org/schema/stmml">
  <stm:observation>
    <p>We observed <object count="3" dictRef="foo:p1"/>
      constructing dwellings of different material</p>
  </stm:observation>
  <stm:entry id="p1" term="pig">
    <stm:definition>A domesticated animal.</stm:definition>
    <stm:description>Predators include
wolves</stm:description>
    <stm:description class="scientificName">Sus
scrofa</stm:description>
  </stm:entry>
```

```
</stm:list>
```

`from`*[att.link.from]*

A string referencing a dictionary, units, convention or other metadata.

The namespace is optional but recommended where possible
Note: this convention is only used within STMML and related languages; it is NOT a generic URI.

```
<list>
<!-- dictRef is of namespaceRefType -->
  <scalar dictRef="chem:mpt">123</scalar>
<!-- error -->
  <scalar dictRef="mpt23">123</scalar>
</list>
```

[xsd:string]
Pattern: [A-Za-z][A-Za-z0-9_]*(:[A-Za-z][A-Za-z0-9_]*)?

The starting point of an arc

`to`*[att.link.to]*

A string referencing a dictionary, units, convention or other metadata.

The namespace is optional but recommended where possible
Note: this convention is only used within STMML and related languages; it is NOT a generic URI.

```
<list>
<!-- dictRef is of namespaceRefType -->
  <scalar dictRef="chem:mpt">123</scalar>
<!-- error -->
  <scalar dictRef="mpt23">123</scalar>
</list>
```

[xsd:string]
Pattern: [A-Za-z][A-Za-z0-9_]*(:[A-Za-z][A-Za-z0-9_]*)?

The endpoint of an arc

`ref`*[att.link.ref]*

A string referencing a dictionary, units, convention or other metadata.

The namespace is optional but recommended where possible
Note: this convention is only used within STMML and related languages; it is NOT a generic URI.

```
<list>
```

```
<!-- dictRef is of namespaceRefType -->
  <scalar dictRef="chem:mpt">123</scalar>
<!-- error -->
  <scalar dictRef="mpt23">123</scalar>
</list>
```

[xsd:string]
Pattern: [A-Za-z][A-Za-z0-9_]*(:[A-Za-z][A-Za-z0-9_]*)?

An ID referenced within a locator

`role`*[att.link.role]*

The role of the link. Xlink adds semantics through a URI; we shall not be this strict. We shall not normally use this mechanism and use dictionaries instead

`href`*[att.link.href]*

The target of the (locator) link, outside the document

`type`*[att.link.type]*

The type of the link

Allowed values

- extended

  A container for locators
- locator

  A link to an element
- arc

  A labelled link

`list`*[el.list]*

A generic container with no implied semantics

A generic container with no implied semantics. It just contains things and can have attributes which bind conventions to it. It could often act as the root element in an STM document.

```
<list>
  <array title="animals" dataType="xsd:string">frog bear
toad</array>
  <scalar title="weight" dataType="xsd:float">3.456</scalar>
</list>
```

Content Model

(ANY [lax])

`title`*[att.title]*

A title on an element.

No controlled value.

```
<action title="turn on heat" start="T09:00:00"
convention="xsd"/>
```

`id`*[att.id]*

A unique ID for an element

This is not formally of type ID (an XML NAME which must start with a letter and contain only letters, digits and `.-_:`). It is recommended that IDs start with a letter, and contain no punctuation or whitespace. The function `generate-id()` in XSLT will generate semantically void unique IDs.
It is difficult to ensure uniqueness when documents are merged. We suggest namespacing IDs, perhaps using the containing elements as the base. Thus `mol3:a1` could be a useful unique ID. However this is still experimental.

[xsd:string]
Pattern: [A-Za-z0-9_-]+(:[A-Za-z0-9_-]+)?

An attribute providing a unique ID for an element

`convention`*[att.convention]*

A string referencing a dictionary, units, convention or other metadata.

The namespace is optional but recommended where possible
Note: this convention is only used within STMML and related languages; it is NOT a generic URI.

```
<list>
<!-- dictRef is of namespaceRefType -->
  <scalar dictRef="chem:mpt">123</scalar>
<!-- error -->
  <scalar dictRef="mpt23">123</scalar>
</list>
```

[xsd:string]
Pattern: [A-Za-z][A-Za-z0-9_]*(:[A-Za-z][A-Za-z0-9_]*)?

A reference to a convention

There is no controlled vocabulary for conventions, but the author must ensure that the semantics are openly available and that there are mechanisms for implementation. The convention is inherited by all the subelements, so that a convention for `molecule` would by default extend to its `bond` and `atom` children.

This can be overwritten if necessary by an explicit `convention`.
It may be useful to create conventions with namespaces (e.g. `iupac:name`). Use of `convention` will normally require non-STMML semantics, and should be used with caution. We would expect that conventions prefixed with "ISO" would be useful, such as ISO8601 for dateTimes.
There is no default, but the conventions of STMML or the related language (e.g. CML) will be assumed.

```
<bond convention="fooChem" order="-5"
   xmlns:fooChem="http://www.fooChem/conventions"/>
```

`dictRef` *[att.dictRef]*

A string referencing a dictionary, units, convention or other metadata.

The namespace is optional but recommended where possible
Note: this convention is only used within STMML and related languages; it is NOT a generic URI.

```
<list>
<!-- dictRef is of namespaceRefType -->
  <scalar dictRef="chem:mpt">123</scalar>
<!-- error -->
  <scalar dictRef="mpt23">123</scalar>
</list>
```

[xsd:string]
Pattern: [A-Za-z][A-Za-z0-9_]*(:[A-Za-z][A-Za-z0-9_]*)?

A reference to a dictionary entry.

Elements in data instances such as **scalar** may have a `dictRef` attribute to point to an entry in a dictionary. To avoid excessive use of (mutable) filenames and URIs we recommend a namespace prefix, mapped to a namespace URI in the normal manner. In this case, of course, the namespace URI must point to a real XML document containing **entry** elements and validated against STMML Schema.
Where there is concern about the dictionary becoming separated from the document the dictionary entries can be physically included as part of the data instance and the normal XPointer addressing mechanism can be used.
This attribute can also be used on **dictionary** elements to define the namespace prefix

```
<scalar dataType="xsd:float" title="surfaceArea"
  dictRef="cmlPhys:surfArea"
  xmlns:cmlPhys="http://www.xml-cml.org/dict/physical"
  units="units:cm2">50</scalar>
<stm:list xmlns:stm="http://www.xml-cml.org/schema/stmml">
  <stm:observation>
    <p>We observed <object count="3" dictRef="foo:p1"/>
      constructing dwellings of different material</p>
  </stm:observation>
  <stm:entry id="p1" term="pig">
    <stm:definition>A domesticated animal.</stm:definition>
```

```
    <stm:description>Predators include
wolves</stm:description>
    <stm:description class="scientificName">Sus
scrofa</stm:description>
  </stm:entry>
</stm:list>
```

type*[att.list.type]*

Type of the list

Semeantics undefined.

```
<list>
  <array title="animals" dataType="xsd:string">frog bear
toad</array>
  <scalar title="weight" dataType="xsd:float">3.456</scalar>
</list>
```

matrix*[el.matrix]*

A rectangular matrix of any quantities

By default `matrix` represents a rectangular matrix of any quantities representable as XSD or STMML dataTypes. It consists of `rows*columns` elements, where `columns` is the fasting moving index. Assuming the elements are counted from 1 they are ordered
`V[1,1],V[1,2],...V[1,columns],V[2,1],V[2,2],...V[2,columns],...V[rows,1],V[rows,2],...V[rows,columns]`
By default whitespace is used to separate matrix elements; see **array** for details. There are NO characters or markup delimiting the end of rows; authors must be careful!. The `columns` and `rows` attributes have no default values; a row vector requires a `rows` attribute of 1.
`matrix` also supports many types of square matrix, but at present we require all elements to be given, even if the matrix is symmetric, antisymmetric or banded diagonal. The `matrixType` attribute allows software to validate and process the type of matrix.

```
<matrix id="m1" title="mattrix-1" dictRef="foo:bar"
  rows="3" columns="3" dataType="xsd:decimal"
  delimiter="|" matrixType="squareSymmetric" units="unit:m"
  >|1.1|1.2|1.3|1.2|2.2|2.3|1.3|2.3|3.3!</matrix>
```

Content Model
[xsd:string]

dataType*[att.matrix.dataType]*REQUIRED

an enumerated type for all builtin allowed dataTypes in STM

`dataTypeType` represents an enumeration of allowed dataTypes (at present

identical with those in XML-Schemas (Part2- datatypes). This means that implementers should be able to use standard XMLSchema-based tools for validation without major implementation problems.
It will often be used an an attribute on **scalar**, **array** or **matrix** elements.

```
<list xmlns="http://www.xml-cml.org/schema/cml2/core">
  <scalar dataType="xsd:boolean" title="she loves
me">true</scalar>
  <scalar dataType="xsd:float" title="x">23.2</scalar>
  <scalar dataType="xsd:duration" title="egg
timer">PM4</scalar>
  <scalar dataType="xsd:dateTime" title="current data and
time">2001-02-01:00:30</scalar>
  <scalar dataType="xsd:time" title="wake up">06:00</scalar>
  <scalar dataType="xsd:date" title="where is
it">1752-09-10</scalar>
  <scalar dataType="xsd:anyURI" title="CML
site">http://www.xml-cml.org/</scalar>
  <scalar dataType="xsd:QName" title="CML
atom">cml:atom</scalar>
  <scalar dataType="xsd:normalizedString" title="song">the
mouse ran up the clock</scalar>
  <scalar dataType="xsd:language" title="UK
English">en-GB</scalar>
  <scalar dataType="xsd:Name" title="atom">atom</scalar>
  <scalar dataType="xsd:ID" title="XML ID">_123</scalar>
  <scalar dataType="xsd:integer" title="the
answer">42</scalar>
  <scalar dataType="xsd:nonPositiveInteger"
title="zero">0</scalar>
</list>
```

UNION OF
Allowed values

- xsd:string
- xsd:boolean
- xsd:float
- xsd:double
- xsd:decimal
- xsd:duration
- xsd:dateTime
- xsd:time
- xsd:date
- xsd:gYearMonth
- xsd:gYear
- xsd:gMonthDay
- xsd:gDay
- xsd:gMonth
- xsd:hexBinary
- xsd:base64Binary
- xsd:anyURI

- xsd:QName
- xsd:NOTATION
- xsd:normalizedString
- xsd:token
- xsd:language
- xsd:IDREFS
- xsd:ENTITIES
- xsd:NMTOKEN
- xsd:NMTOKENS
- xsd:Name
- xsd:NCName
- xsd:ID
- xsd:IDREF
- xsd:ENTITY
- xsd:integer
- xsd:nonPositiveInteger
- xsd:negativeInteger
- xsd:long
- xsd:int
- xsd:short
- xsd:byte
- xsd:nonNegativeInteger
- xsd:unsignedLong
- xsd:unsignedInt
- xsd:unsignedShort
- xsd:unsignedByte
- xsd:positiveInteger

[xsd:QName]

---

delimiter *[att.matrix.delimiter]*

A non-whitespace character used in arrays to separate components

Some STMML elements (such as **array**) have content representing concatenated values. The default separator is whitespace (which can be normalised) and this should be used whenever possible. However in some cases the values are empty, or contain whitespace or other problematic punctuation, and a delimiter is required. Note that the content string MUST start and end with the delimiter so there is no ambiguity as to what the components are. Only printable characters from the ASCII character set should be used, and character entities should be avoided.
When delimiters are used to separate precise whitespace this should always consist of spaces and not the other allowed whitespace characters (newline, tabs, etc.). If the latter are important it is probably best to redesign the application.

```
<array size="4"  dataType="xsd:string" delimiter="|">|A|B12||D
and   E|</array>


 The values in the array are
  "A", "B12", "" (empty string) and "D and   E"
 note the spaces
```

[xsd:string]

rows*[]*REQUIRED

The size of an array

The size of an array. Redundant, but serves as a check for processing software (useful if delimiters are used)

[xsd:positiveInteger]

Number of rows

columns*[att.matrix.columns]*REQUIRED

The size of an array

The size of an array. Redundant, but serves as a check for processing software (useful if delimiters are used)

[xsd:positiveInteger]

Number of columns

units*[att.matrix.units]*

Scientific units

These will be linked to dictionaries of units with conversion information, using namespaced references (e.g. `si:m`)
Distinguish carefully from **unitType** which is an element describing a type of a unit in a **unitList**

```
<stm:unitList xmlns:stm="http://www.xml-cml.org/schema/stmml">

<!--
============================================================================
-->
<!-- ======================== fundamental types
========================== -->
<!--
============================================================================
-->

<stm:unitType id="length" name="length">
  <stm:dimension name="length" power="1"/>
</stm:unitType>

<stm:unitType id="time" name="time">
  <stm:dimension name="time" power="1"/>
</stm:unitType>

<!-- ... -->

<stm:unitType id="dimensionless" name="dimensionless">
```

```
  <stm:dimension name="dimensionless" power="1"/>
</stm:unitType>

<!--
================================================================================
-->
<!-- ========================== derived types
============================== -->
<!--
================================================================================
-->

<stm:unitType id="acceleration" name="acceleration">
  <stm:dimension name="length" power="1"/>
  <stm:dimension name="time" power="-2"/>
</stm:unitType>

<!-- ... -->

<!--
================================================================================
-->
<!-- ====================== fundamental SI units
========================== -->
<!--
================================================================================
-->

<stm:unit id="second" name="second" unitType="time">
  <stm:description>The SI unit of time</stm:description>
</stm:unit>

<stm:unit id="meter" name="meter" unitType="length"
  abbreviation="m">
  <stm:description>The SI unit of length</stm:description>
</stm:unit>

<!-- ... -->

<stm:unit id="kg" name="nameless" unitType="dimensionless"
  abbreviation="nodim">
  <stm:description>A fictitious parent for dimensionless
units</stm:description>
</stm:unit>

<!--
================================================================================
-->
<!-- ==================== derived SI units
============================== -->
<!--
================================================================================
-->

<stm:unit id="newton" name="newton" unitType="force">
  <stm:description>The SI unit of force</stm:description>
```

```
</stm:unit>

<!-- ... -->

<!-- multiples of fundamental SI units -->

<stm:unit id="g" name="gram" unitType="mass"
  parentSI="kg"
  multiplierToSI="0.001"
  abbreviation="g">
  <stm:description>0.001 kg. </stm:description>
</stm:unit>

<stm:unit id="celsius" name="Celsius" parentSI="k"
  multiplierToSI="1"
  constantToSI="273.18">
  <stm:description><p>A common unit of
temperature</p></stm:description>
</stm:unit>

<!-- fundamental non-SI units -->

<stm:unit id="inch" name="inch" parentSI="meter"
   abbreviation="in"
  multiplierToSI="0.0254" >
  <stm:description>An imperial measure of
length</stm:description>
</stm:unit>

<!-- derived non-SI units -->

<stm:unit id="l" name="litre" unitType="volume"
   parentSI="meterCubed"
   abbreviation="l"
   multiplierToSI="0.001">
  <stm:description>Nearly 1 dm**3 This is not quite
exact</stm:description>
</stm:unit>

<!-- ... -->

<stm:unit id="fahr" name="fahrenheit" parentSI="k"
   abbreviation="F"
  multiplierToSI="0.55555555555555555"
  constantToSI="-17.777777777777777">
  <stm:description>An obsolescent unit of temperature still
used in popular
  meteorology</stm:description>
</stm:unit>

</stm:unitList>
```

[xsd:string]

units (recommended for numeric quantities!!)

`title`*[att.title]*

A title on an element.

No controlled value.

```
<action title="turn on heat" start="T09:00:00"
convention="xsd"/>
```

`id`*[att.id]*

A unique ID for an element

This is not formally of type ID (an XML NAME which must start with a letter and contain only letters, digits and `.-_:`). It is recommended that IDs start with a letter, and contain no punctuation or whitespace. The function `generate-id()` in XSLT will generate semantically void unique IDs.
It is difficult to ensure uniqueness when documents are merged. We suggest namespacing IDs, perhaps using the containing elements as the base. Thus `mol3:a1` could be a useful unique ID. However this is still experimental.

[xsd:string]
Pattern: [A-Za-z0-9_-]+(:[A-Za-z0-9_-]+)?

An attribute providing a unique ID for an element

`convention`*[att.convention]*

A string referencing a dictionary, units, convention or other metadata.

The namespace is optional but recommended where possible
Note: this convention is only used within STMML and related languages; it is NOT a generic URI.

```
<list>
<!-- dictRef is of namespaceRefType -->
  <scalar dictRef="chem:mpt">123</scalar>
<!-- error -->
  <scalar dictRef="mpt23">123</scalar>
</list>
```

[xsd:string]
Pattern: [A-Za-z][A-Za-z0-9_]*(:[A-Za-z][A-Za-z0-9_]*)?

A reference to a convention

There is no controlled vocabulary for conventions, but the author must ensure that the semantics are openly available and that there are mechanisms for implementation. The convention is inherited by all the subelements, so that a convention for `molecule` would by default extend to its `bond` and `atom` children. This can be overwritten if necessary by an explicit `convention`.
It may be useful to create conventions with namespaces (e.g. `iupac:name`). Use of

`convention` will normally require non-STMML semantics, and should be used with caution. We would expect that conventions prefixed with "ISO" would be useful, such as ISO8601 for dateTimes.
There is no default, but the conventions of STMML or the related language (e.g. CML) will be assumed.

```
<bond convention="fooChem" order="-5"
   xmlns:fooChem="http://www.fooChem/conventions"/>
```

`dictRef` *[att.dictRef]*

A string referencing a dictionary, units, convention or other metadata.

The namespace is optional but recommended where possible
Note: this convention is only used within STMML and related languages; it is NOT a generic URI.

```
<list>
<!-- dictRef is of namespaceRefType -->
  <scalar dictRef="chem:mpt">123</scalar>
<!-- error -->
  <scalar dictRef="mpt23">123</scalar>
</list>
```

[xsd:string]
Pattern: [A-Za-z][A-Za-z0-9_]*(:[A-Za-z][A-Za-z0-9_]*)?

A reference to a dictionary entry.

Elements in data instances such as **scalar** may have a `dictRef` attribute to point to an entry in a dictionary. To avoid excessive use of (mutable) filenames and URIs we recommend a namespace prefix, mapped to a namespace URI in the normal manner. In this case, of course, the namespace URI must point to a real XML document containing **entry** elements and validated against STMML Schema.
Where there is concern about the dictionary becoming separated from the document the dictionary entries can be physically included as part of the data instance and the normal XPointer addressing mechanism can be used.
This attribute can also be used on **dictionary** elements to define the namespace prefix

```
<scalar dataType="xsd:float" title="surfaceArea"
  dictRef="cmlPhys:surfArea"
  xmlns:cmlPhys="http://www.xml-cml.org/dict/physical"
  units="units:cm2">50</scalar>
<stm:list xmlns:stm="http://www.xml-cml.org/schema/stmml">
  <stm:observation>
    <p>We observed <object count="3" dictRef="foo:p1"/>
      constructing dwellings of different material</p>
  </stm:observation>
  <stm:entry id="p1" term="pig">
    <stm:definition>A domesticated animal.</stm:definition>
    <stm:description>Predators include
wolves</stm:description>
    <stm:description class="scientificName">Sus
```

```
/stm:description>
  </stm:entry>
</stm:list>
```

matrixType *[att.matrix.matrixType]*

Allowed **matrix** types

Allowed `matrix` types. These are mainly square matrices

```
<matrix id="m1" title="mattrix-1" dictRef="foo:bar"
  rows="3" columns="3" dataType="xsd:decimal"
  delimiter="|" matrixType="squareSymmetric" units="unit:m"
  >|1.1|1.2|1.3|1.2|2.2|2.3|1.3|2.3|3.3!</matrix>
```

UNION OF

Allowed values

- rectangular
- square
- squareSymmetric
- squareAntisymmetric
- diagonal

  Symmetric. Elements are zero except on the diagonal
- upperTriangular

  Square. Elements are zero below the diagonal

  ```
  1 2 3 4
  0 3 5 6
  0 0 4 8
  0 0 0 2
  ```
- lowerTriangular

  Symmetric. Elements are zero except on the diagonal
- unitary
- rowEigenvectors
- rotation22
- rotationTranslation32
- homogeneous33
- rotation33
- rotationTranslation43
- homogeneous44
- square
- square

BASE: namespaceRefType

A string referencing a dictionary, units, convention or other metadata.

The namespace is optional but recommended where possible

Note: this convention is only used within STMML and related languages; it is NOT a generic URI.

```
<list>
<!-- dictRef is of namespaceRefType -->
  <scalar dictRef="chem:mpt">123</scalar>
<!-- error -->
  <scalar dictRef="mpt23">123</scalar>
</list>
```

[xsd:string]
Pattern: [A-Za-z][A-Za-z0-9_]*(:[A-Za-z][A-Za-z0-9_]*)?

User-defined matrix-type

This definition must be by reference to a namespaced dictionary entry.

Type of matrix

Mainly square, but extensible through the `xsd:union` mechanism.

`errorValues` *[att.matrix.errorValues]*

An array of floats

An array of floats or other real numbers. Not used in STM Schema, but re-used by CML and other languages.

```
<atomArray xmlns="http://www.xml-cml.org/schema/cml2/core"
  x2="1.2 2.3 3.4 5.6"/>
```

XSD:LIST of xsd:decimal

an optional array of error values for numeric matrices

`errorBasis` *[att.errorBasis]*

The basis of an error value

Errors in values can be of several types and this simpleType provides a small controlled vocabulary

```
<scalar
    dataType="xsd:decimal"
    errorValue="1.0"
    errorBasis="observedStandardDeviation"
    title="body weight"
    dictRef="zoo:bodywt"
    units="units:g">34.3</scalar>
```

Allowed values

- observedRange
- observedStandardDeviation
- observedStandardError
- estimatedStandardDeviation
- estimatedStandardError

`minValues`*[att.matrix.minValues]*

An array of floats

An array of floats or other real numbers. Not used in STM Schema, but re-used by CML and other languages.

```
<atomArray xmlns="http://www.xml-cml.org/schema/cml2/core"
  x2="1.2 2.3 3.4 5.6"/>
```

XSD:LIST of xsd:decimal

an optional array of minimum values for numeric matrices

`maxValues`*[att.matrix.maxValues]*

An array of floats

An array of floats or other real numbers. Not used in STM Schema, but re-used by CML and other languages.

```
<atomArray xmlns="http://www.xml-cml.org/schema/cml2/core"
  x2="1.2 2.3 3.4 5.6"/>
```

XSD:LIST of xsd:decimal

an optional array of maximum values for numeric matrices

`metadata`*[el.metadata]*

A general container for metadata

A general container for metadata, including at least Dublin Core (DC) and CML-specific metadata
In its simple form each element provides a name and content in a similar fashion to the `meta` element in HTML. `metadata` may have simpleContent (i.e. a string for adding further information - this is not controlled).

```
<list>
  <metadataList>
    <metadata name="dc:coverage" content="Europe"/>
    <metadata name="dc:description" content="Ornithological
chemistry"/>
    <metadata name="dc:identifier"  content="ISBN:1234-5678"/>
    <metadata name="dc:format" content="printed"/>
    <metadata name="dc:relation" content="abc:def123"/>
    <metadata name="dc:rights" content="licence:GPL"/>
    <metadata name="dc:subject" content="Informatics"/>
    <metadata name="dc:title" content="birds"/>
    <metadata name="dc:type" content="bird books on
chemistry"/>
    <metadata name="dc:contributor" content="Tux Penguin"/>
    <metadata name="dc:creator" content="author"/>
    <metadata name="dc:publisher" content="Penguinone
```

```
publishing"/>
    <metadata name="dc:source" content="penguinPub"/>
    <metadata name="dc:language" content="en-GB"/>
    <metadata name="dc:date" content="1752-09-10"/>
  </metadataList>
  <metadataList>
    <metadata name="cmlm:safety" content="mostly harmless"/>
    <metadata name="cmlm:insilico" content="electronically
produced"/>
    <metadata name="cmlm:structure" content="penguinone"/>
    <metadata name="cmlm:reaction" content="synthesis of
penguinone"/>
    <metadata name="cmlm:identifier"
content="smiles:O=C1C=C(C)C(C)(C)C(C)=C1"/>
  </metadataList>
</list>
```

Content Model

[xsd:string]

---

name*[att.metadata.name]*

The name of the metadata

Metadata consists of name-value pairs (value is in the "content" attribute). The names are from a semi-restricted vocabulary, mainly Dublin Core. The content is unrestricted. The order of metadata has no implied semantics at present.

Allowed values

- dc:coverage

  The extent or scope of the content of the resource.

  Coverage will typically include spatial location (a place name or geographic coordinates), temporal period (a period label, date, or date range) or jurisdiction (such as a named administrative entity). Recommended best practice is to select a value from a controlled vocabulary (for example, the Thesaurus of Geographic Names [TGN]) and that, where appropriate, named places or time periods be used in preference to numeric identifiers such as sets of coordinates or date ranges.

- dc:description

  An account of the content of the resource.

  Description may include but is not limited to: an abstract, table of contents, reference to a graphical representation of content or a free-text account of the content.

- dc:identifier

  An unambiguous reference to the resource within a given context.

  Recommended best practice is to identify the resource by means of a string or number conforming to a formal identification system. Example formal identification systems include the Uniform Resource Identifier (URI) (including the Uniform Resource Locator (URL)), the Digital Object Identifier (DOI) and the

International Standard Book Number (ISBN).

- dc:format

  The physical or digital manifestation of the resource.

  Typically, Format may include the media-type or dimensions of the resource. Format may be used to determine the software, hardware or other equipment needed to display or operate the resource. Examples of dimensions include size and duration. Recommended best practice is to select a value from a controlled vocabulary (for example, the list of Internet Media Types [MIME] defining computer media formats).

- dc:relation

  A reference to a related resource.

  Recommended best practice is to reference the resource by means of a string or number conforming to a formal identification system.

- dc:rights

  Information about rights held in and over the resource.

  Typically, a Rights element will contain a rights management statement for the resource, or reference a service providing such information. Rights information often encompasses Intellectual Property Rights (IPR), Copyright, and various Property Rights. If the Rights element is absent, no assumptions can be made about the status of these and other rights with respect to the resource.

- dc:subject

  The topic of the content of the resource.

  Typically, a Subject will be expressed as keywords, key phrases or classification codes that describe a topic of the resource. Recommended best practice is to select a value from a controlled vocabulary or formal classification scheme.

- dc:title

  A name given to the resource.

  Typically, a Title will be a name by which the resource is formally known.

- dc:type

  The nature or genre of the content of the resource.

  Type includes terms describing general categories, functions, genres, or aggregation levels for content. Recommended best practice is to select a value from a controlled vocabulary (for example, the working draft list of Dublin Core Types [DCT1]). To describe the physical or digital manifestation of the resource, use the FORMAT element.

- dc:contributor

  An entity responsible for making contributions to the content of the resource.

  Examples of a Contributor include a person, an organisation, or a service. Typically, the name of a Contributor should be used to indicate the entity.

- dc:creator

  An entity primarily responsible for making the content of the resource.

  Examples of a Creator include a person, an organisation, or a service. Typically, the name of a Creator should be used to indicate the entity.
- dc:publisher

  An entity responsible for making the resource available

  Examples of a Publisher include a person, an organisation, or a service. Typically, the name of a Publisher should be used to indicate the entity.
- dc:source

  A Reference to a resource from which the present resource is derived.

  The present resource may be derived from the Source resource in whole or in part. Recommended best practice is to reference the resource by means of a string or number conforming to a formal identification system.
- dc:language

  A language of the intellectual content of the resource.

  Recommended best practice for the values of the Language element is defined by RFC 1766 [RFC1766] which includes a two-letter Language Code (taken from the ISO 639 standard [ISO639]), followed optionally, by a two-letter Country Code (taken from the ISO 3166 standard [ISO3166]). For example, 'en' for English, 'fr' for French, or 'en-uk' for English used in the United Kingdom.
- dc:date

  A date associated with an event in the life cycle of the resource.

  Typically, Date will be associated with the creation or availability of the resource. Recommended best practice for encoding the date value is defined in a profile of ISO 8601 [W3CDTF] and follows the YYYY-MM-DD format.
- cmlm:safety

  Entry contains information relating to chemical safety

  Typically the content will be a reference to a handbook, MSDS, threshhold or other human-readable string
- cmlm:insilico

  Part or whole of the information was computer-generated

  Typically the content will be the name of a method or a program
- cmlm:structure

  3D structure included

  details included
- cmlm:reaction
- cmlm:identifier
- other

The metadata type

content*[att.metadata.content]*

The metadata

metadataList*[el.metadataList]*

A general container for metadata elements

```
<list>
  <metadataList>
    <metadata name="dc:coverage" content="Europe"/>
    <metadata name="dc:description" content="Ornithological
chemistry"/>
    <metadata name="dc:identifier"  content="ISBN:1234-5678"/>
    <metadata name="dc:format" content="printed"/>
    <metadata name="dc:relation" content="abc:def123"/>
    <metadata name="dc:rights" content="licence:GPL"/>
    <metadata name="dc:subject" content="Informatics"/>
    <metadata name="dc:title" content="birds"/>
    <metadata name="dc:type" content="bird books on
chemistry"/>
    <metadata name="dc:contributor" content="Tux Penguin"/>
    <metadata name="dc:creator" content="author"/>
    <metadata name="dc:publisher" content="Penguinone
publishing"/>
    <metadata name="dc:source" content="penguinPub"/>
    <metadata name="dc:language" content="en-GB"/>
    <metadata name="dc:date" content="1752-09-10"/>
  </metadataList>
  <metadataList>
    <metadata name="cmlm:safety" content="mostly harmless"/>
    <metadata name="cmlm:insilico" content="electronically
produced"/>
    <metadata name="cmlm:structure" content="penguinone"/>
    <metadata name="cmlm:reaction" content="synthesis of
penguinone"/>
    <metadata name="cmlm:identifier"
content="smiles:O=C1C=C(C)C(C)(C)C(C)=C1"/>
  </metadataList>
</list>
```

Content Model
(metadata+)

object*[el.object]*

An object which might occur in scientific data or narrative

Deliberately vague. Thus an instrument might be built from sub component objects,

or a program could be composed of smaller modules (objects). Unrestricted content model

```
<object title="frog" type="amphibian" count="5">
  <scalar dataType="xsd:float" title="length"
units="unit:cm">5</scalar>
  <obj1/>
</object>
```

Content Model

(ANY [lax])*

`title`*[att.title]*

A title on an element.

No controlled value.

```
<action title="turn on heat" start="T09:00:00"
convention="xsd"/>
```

`id`*[att.id]*

A unique ID for an element

This is not formally of type ID (an XML NAME which must start with a letter and contain only letters, digits and `.-_:`). It is recommended that IDs start with a letter, and contain no punctuation or whitespace. The function `generate-id()` in XSLT will generate semantically void unique IDs.
It is difficult to ensure uniqueness when documents are merged. We suggest namespacing IDs, perhaps using the containing elements as the base. Thus `mol3:a1` could be a useful unique ID. However this is still experimental.

[xsd:string]
Pattern: [A-Za-z0-9_-]+(:[A-Za-z0-9_-]+)?

An attribute providing a unique ID for an element

`convention`*[att.convention]*

A string referencing a dictionary, units, convention or other metadata.

The namespace is optional but recommended where possible
Note: this convention is only used within STMML and related languages; it is NOT a generic URI.

```
<list>
<!-- dictRef is of namespaceRefType -->
  <scalar dictRef="chem:mpt">123</scalar>
<!-- error -->
  <scalar dictRef="mpt23">123</scalar>
</list>
```

[xsd:string]
Pattern: [A-Za-z][A-Za-z0-9_]*(:[A-Za-z][A-Za-z0-9_]*)?

A reference to a convention

There is no controlled vocabulary for conventions, but the author must ensure that the semantics are openly available and that there are mechanisms for implementation. The convention is inherited by all the subelements, so that a convention for `molecule` would by default extend to its `bond` and `atom` children. This can be overwritten if necessary by an explicit `convention`.
It may be useful to create conventions with namespaces (e.g. `iupac:name`). Use of `convention` will normally require non-STMML semantics, and should be used with caution. We would expect that conventions prefixed with "ISO" would be useful, such as ISO8601 for dateTimes.
There is no default, but the conventions of STMML or the related language (e.g. CML) will be assumed.

```
<bond convention="fooChem" order="-5"
   xmlns:fooChem="http://www.fooChem/conventions"/>
```

`dictRef` *[att.dictRef]*

A string referencing a dictionary, units, convention or other metadata.

The namespace is optional but recommended where possible
Note: this convention is only used within STMML and related languages; it is NOT a generic URI.

```
<list>
<!-- dictRef is of namespaceRefType -->
  <scalar dictRef="chem:mpt">123</scalar>
<!-- error -->
  <scalar dictRef="mpt23">123</scalar>
</list>
```

[xsd:string]
Pattern: [A-Za-z][A-Za-z0-9_]*(:[A-Za-z][A-Za-z0-9_]*)?

A reference to a dictionary entry.

Elements in data instances such as **scalar** may have a `dictRef` attribute to point to an entry in a dictionary. To avoid excessive use of (mutable) filenames and URIs we recommend a namespace prefix, mapped to a namespace URI in the normal manner. In this case, of course, the namespace URI must point to a real XML document containing **entry** elements and validated against STMML Schema.
Where there is concern about the dictionary becoming separated from the document the dictionary entries can be physically included as part of the data instance and the normal XPointer addressing mechanism can be used.
This attribute can also be used on **dictionary** elements to define the namespace prefix

```
<scalar dataType="xsd:float" title="surfaceArea"
  dictRef="cmlPhys:surfArea"
```

```
  xmlns:cmlPhys="http://www.xml-cml.org/dict/physical"
  units="units:cm2">50</scalar>
<stm:list xmlns:stm="http://www.xml-cml.org/schema/stmml">
  <stm:observation>
    <p>We observed <object count="3" dictRef="foo:p1"/>
      constructing dwellings of different material</p>
  </stm:observation>
  <stm:entry id="p1" term="pig">
    <stm:definition>A domesticated animal.</stm:definition>
    <stm:description>Predators include
wolves</stm:description>
    <stm:description class="scientificName">Sus
scrofa</stm:description>
  </stm:entry>
</stm:list>
```

type*[att.object.type]*

Type of the object. Uncontrolled semantics

count*[att.object.count]*

A count multiplier for an element

Many elements represent objects which can occur an arbitrary number of times in a scientific context. Examples are `action`, `object` or `molecule`s.

```
<list>
<object title="frog" count="10"/>
<action title="step3" count="3">
  <p>Add 10 ml reagent</p>
</action>
</list>
```

[xsd:nonNegativeInteger]

observation*[el.observation]*

An observation or occurrence

A container for any events that need to be recorded, whether planned or not. They can include notes, measurements, conditions that may be referenced elsewhere, etc. There are no controlled semantics

```
<observation type="ornithology">
  <object title="sparrow" count="3"/>
  <observ/>
</observation>
```

Content Model

(ANY [lax])*

`title`*[att.title]*

A title on an element.

No controlled value.

```
<action title="turn on heat" start="T09:00:00"
convention="xsd"/>
```

`id`*[att.id]*

A unique ID for an element

This is not formally of type ID (an XML NAME which must start with a letter and contain only letters, digits and `.-_:`). It is recommended that IDs start with a letter, and contain no punctuation or whitespace. The function `generate-id()` in XSLT will generate semantically void unique IDs.
It is difficult to ensure uniqueness when documents are merged. We suggest namespacing IDs, perhaps using the containing elements as the base. Thus `mol3:a1` could be a useful unique ID. However this is still experimental.

[xsd:string]
Pattern: [A-Za-z0-9_-]+(:[A-Za-z0-9_-]+)?

An attribute providing a unique ID for an element

`convention`*[att.convention]*

A string referencing a dictionary, units, convention or other metadata.

The namespace is optional but recommended where possible
Note: this convention is only used within STMML and related languages; it is NOT a generic URI.

```
<list>
<!-- dictRef is of namespaceRefType -->
  <scalar dictRef="chem:mpt">123</scalar>
<!-- error -->
  <scalar dictRef="mpt23">123</scalar>
</list>
```

[xsd:string]
Pattern: [A-Za-z][A-Za-z0-9_]*(:[A-Za-z][A-Za-z0-9_]*)?

A reference to a convention

There is no controlled vocabulary for conventions, but the author must ensure that the semantics are openly available and that there are mechanisms for implementation. The convention is inherited by all the subelements, so that a convention for `molecule` would by default extend to its `bond` and `atom` children. This can be overwritten if necessary by an explicit `convention`.
It may be useful to create conventions with namespaces (e.g. `iupac:name`). Use of `convention` will normally require non-STMML semantics, and should be used with

caution. We would expect that conventions prefixed with "ISO" would be useful, such as ISO8601 for dateTimes.
There is no default, but the conventions of STMML or the related language (e.g. CML) will be assumed.

```
<bond convention="fooChem" order="-5"
   xmlns:fooChem="http://www.fooChem/conventions"/>
```

dictRef *[att.dictRef]*

A string referencing a dictionary, units, convention or other metadata.

The namespace is optional but recommended where possible
Note: this convention is only used within STMML and related languages; it is NOT a generic URI.

```
<list>
<!-- dictRef is of namespaceRefType -->
  <scalar dictRef="chem:mpt">123</scalar>
<!-- error -->
  <scalar dictRef="mpt23">123</scalar>
</list>
```

[xsd:string]
Pattern: [A-Za-z][A-Za-z0-9_]*(:[A-Za-z][A-Za-z0-9_]*)?

A reference to a dictionary entry.

Elements in data instances such as **scalar** may have a `dictRef` attribute to point to an entry in a dictionary. To avoid excessive use of (mutable) filenames and URIs we recommend a namespace prefix, mapped to a namespace URI in the normal manner. In this case, of course, the namespace URI must point to a real XML document containing **entry** elements and validated against STMML Schema.
Where there is concern about the dictionary becoming separated from the document the dictionary entries can be physically included as part of the data instance and the normal XPointer addressing mechanism can be used.
This attribute can also be used on **dictionary** elements to define the namespace prefix

```
<scalar dataType="xsd:float" title="surfaceArea"
  dictRef="cmlPhys:surfArea"
  xmlns:cmlPhys="http://www.xml-cml.org/dict/physical"
  units="units:cm2">50</scalar>
<stm:list xmlns:stm="http://www.xml-cml.org/schema/stmml">
  <stm:observation>
    <p>We observed <object count="3" dictRef="foo:p1"/>
      constructing dwellings of different material</p>
  </stm:observation>
  <stm:entry id="p1" term="pig">
    <stm:definition>A domesticated animal.</stm:definition>
    <stm:description>Predators include
wolves</stm:description>
    <stm:description class="scientificName">Sus
scrofa</stm:description>
```

```
  </stm:entry>
</stm:list>
```

type*[att.observation.type]*

Type of observation (uncontrolled vocabulary)

count*[att.observation.count]*

A count multiplier for an element

Many elements represent objects which can occur an arbitrary number of times in a scientific context. Examples are `action`, `object` or `molecule`s.

```
<list>
<object title="frog" count="10"/>
<action title="step3" count="3">
  <p>Add 10 ml reagent</p>
</action>
</list>
```

[xsd:nonNegativeInteger]

relatedEntry*[el.relatedEntry]*

An entry related in some way to a dictionary entry, scientific units, etc.

The range of relationships is not restricted but should include parents, aggregation, seeAlso etc. dataCategories from ISO12620 can be referenced through the `namespaced` mechanism.

```
<stm:entry id="a14" term="Autoreceptor"
  xmlns:stm="http://www.xml-cml.org/schema/cml2/core">
  <stm:definition>An <strong>autoreceptor</strong>, present at
a nerve ending, is
    a <a href="#r1">receptor</a>
    that regulates, via positive or negative feedback
processes, the
    synthesis and/or release of its own physiological ligand.
  </stm:definition>
  <stm:relatedEntry type="seeAlso"
href="#h4">Heteroreceptor).</stm:relatedEntry>
  <stm:relatedEntry type="my:antonym"
href="#h4">antiheteroreceptor).</stm:relatedEntry>
</stm:entry>
```

Content Model

type*[]*

`relatedEntryType` represents a the type of relationship in a **relatedEntry** element.

UNION OF
Allowed values

- parent
- partitiveParent
- child
- partitiveChild
- related
- synonym
- quasi-synonym
- antonym
- homonym
- see
- seeAlso
- abbreviation
- acronym

BASE: namespaceRefType

A string referencing a dictionary, units, convention or other metadata.

The namespace is optional but recommended where possible
Note: this convention is only used within STMML and related languages; it is NOT a generic URI.

```
<list>
<!-- dictRef is of namespaceRefType -->
  <scalar dictRef="chem:mpt">123</scalar>
<!-- error -->
  <scalar dictRef="mpt23">123</scalar>
</list>
```

[xsd:string]
Pattern: [A-Za-z][A-Za-z0-9_]*(:[A-Za-z][A-Za-z0-9_]*)?

`href`*[]*

The related entry.

`scalar` *[el.scalar]*

An element to hold scalar data.

`scalar` holds scalar data under a single generic container. The semantics are usually resolved by linking to a dictionary. **scalar** defaults to a scalar string but has attributes which affect the type.
`scalar` does not necessarily reflect a physical object (for which **object** should be

used). It may reflect a property of an object such as temperature, size, etc.
Note that normal Schema validation tools cannot validate the data type of **scalar** (it is defined as `string`), but that a temporary schema can be constructed from the type and used for validation. Also the type can be contained in a dictionary and software could decide to retrieve this and use it for validation.

```
<scalar
    dataType="xsd:decimal"
    errorValue="1.0"
    errorBasis="observedStandardDeviation"
    title="body weight"
    dictRef="zoo:bodywt"
    units="units:g">34.3</scalar>
```

Content Model
[xsd:string]

`title` *[att.title]*

A title on an element.

No controlled value.

```
<action title="turn on heat" start="T09:00:00"
convention="xsd"/>
```

`id` *[att.id]*

A unique ID for an element

This is not formally of type ID (an XML NAME which must start with a letter and contain only letters, digits and `.-_:`). It is recommended that IDs start with a letter, and contain no punctuation or whitespace. The function `generate-id()` in XSLT will generate semantically void unique IDs.
It is difficult to ensure uniqueness when documents are merged. We suggest namespacing IDs, perhaps using the containing elements as the base. Thus `mol3:a1` could be a useful unique ID. However this is still experimental.
[xsd:string]
Pattern: [A-Za-z0-9_-]+(:[A-Za-z0-9_-]+)?

An attribute providing a unique ID for an element

`convention` *[att.convention]*

A string referencing a dictionary, units, convention or other metadata.

The namespace is optional but recommended where possible
Note: this convention is only used within STMML and related languages; it is NOT a generic URI.

```
<list>
<!-- dictRef is of namespaceRefType -->
  <scalar dictRef="chem:mpt">123</scalar>
<!-- error -->
  <scalar dictRef="mpt23">123</scalar>
</list>
```

[xsd:string]
Pattern: [A-Za-z][A-Za-z0-9_]*(:[A-Za-z][A-Za-z0-9_]*)?

A reference to a convention

There is no controlled vocabulary for conventions, but the author must ensure that the semantics are openly available and that there are mechanisms for implementation. The convention is inherited by all the subelements, so that a convention for `molecule` would by default extend to its `bond` and `atom` children. This can be overwritten if necessary by an explicit `convention`.
It may be useful to create conventions with namespaces (e.g. `iupac:name`). Use of `convention` will normally require non-STMML semantics, and should be used with caution. We would expect that conventions prefixed with "ISO" would be useful, such as ISO8601 for dateTimes.
There is no default, but the conventions of STMML or the related language (e.g. CML) will be assumed.

```
<bond convention="fooChem" order="-5"
   xmlns:fooChem="http://www.fooChem/conventions"/>
```

`dictRef` *[att.dictRef]*

A string referencing a dictionary, units, convention or other metadata.

The namespace is optional but recommended where possible
Note: this convention is only used within STMML and related languages; it is NOT a generic URI.

```
<list>
<!-- dictRef is of namespaceRefType -->
  <scalar dictRef="chem:mpt">123</scalar>
<!-- error -->
  <scalar dictRef="mpt23">123</scalar>
</list>
```

[xsd:string]
Pattern: [A-Za-z][A-Za-z0-9_]*(:[A-Za-z][A-Za-z0-9_]*)?

A reference to a dictionary entry.

Elements in data instances such as **scalar** may have a `dictRef` attribute to point to an entry in a dictionary. To avoid excessive use of (mutable) filenames and URIs we recommend a namespace prefix, mapped to a namespace URI in the normal manner. In this case, of course, the namespace URI must point to a real XML document containing **entry** elements and validated against STMML Schema.

Where there is concern about the dictionary becoming separated from the document the dictionary entries can be physically included as part of the data instance and the normal XPointer addressing mechanism can be used.
This attribute can also be used on **dictionary** elements to define the namespace prefix

```
<scalar dataType="xsd:float" title="surfaceArea"
  dictRef="cmlPhys:surfArea"
  xmlns:cmlPhys="http://www.xml-cml.org/dict/physical"
  units="units:cm2">50</scalar>
<stm:list xmlns:stm="http://www.xml-cml.org/schema/stmml">
  <stm:observation>
    <p>We observed <object count="3" dictRef="foo:p1"/>
      constructing dwellings of different material</p>
  </stm:observation>
  <stm:entry id="p1" term="pig">
    <stm:definition>A domesticated animal.</stm:definition>
    <stm:description>Predators include
wolves</stm:description>
    <stm:description class="scientificName">Sus
scrofa</stm:description>
  </stm:entry>
</stm:list>
```

`dataType`*[att.dataType]*Default:xsd:string

an enumerated type for all builtin allowed dataTypes in STM

`dataTypeType` represents an enumeration of allowed dataTypes (at present identical with those in XML-Schemas (Part2- datatypes). This means that implementers should be able to use standard XMLSchema-based tools for validation without major implementation problems.
It will often be used an an attribute on **scalar**, **array** or **matrix** elements.

```
<list xmlns="http://www.xml-cml.org/schema/cml2/core">
  <scalar dataType="xsd:boolean" title="she loves
me">true</scalar>
  <scalar dataType="xsd:float" title="x">23.2</scalar>
  <scalar dataType="xsd:duration" title="egg
timer">PM4</scalar>
  <scalar dataType="xsd:dateTime" title="current data and
time">2001-02-01:00:30</scalar>
  <scalar dataType="xsd:time" title="wake up">06:00</scalar>
  <scalar dataType="xsd:date" title="where is
it">1752-09-10</scalar>
  <scalar dataType="xsd:anyURI" title="CML
site">http://www.xml-cml.org/</scalar>
  <scalar dataType="xsd:QName" title="CML
atom">cml:atom</scalar>
  <scalar dataType="xsd:normalizedString" title="song">the
mouse ran up the clock</scalar>
  <scalar dataType="xsd:language" title="UK
English">en-GB</scalar>
  <scalar dataType="xsd:Name" title="atom">atom</scalar>
  <scalar dataType="xsd:ID" title="XML ID">_123</scalar>
```

```
  <scalar dataType="xsd:integer" title="the
answer">42</scalar>
  <scalar dataType="xsd:nonPositiveInteger"
title="zero">0</scalar>
</list>
```

UNION OF
Allowed values

- xsd:string
- xsd:boolean
- xsd:float
- xsd:double
- xsd:decimal
- xsd:duration
- xsd:dateTime
- xsd:time
- xsd:date
- xsd:gYearMonth
- xsd:gYear
- xsd:gMonthDay
- xsd:gDay
- xsd:gMonth
- xsd:hexBinary
- xsd:base64Binary
- xsd:anyURI
- xsd:QName
- xsd:NOTATION
- xsd:normalizedString
- xsd:token
- xsd:language
- xsd:IDREFS
- xsd:ENTITIES
- xsd:NMTOKEN
- xsd:NMTOKENS
- xsd:Name
- xsd:NCName
- xsd:ID
- xsd:IDREF
- xsd:ENTITY
- xsd:integer
- xsd:nonPositiveInteger
- xsd:negativeInteger
- xsd:long
- xsd:int
- xsd:short
- xsd:byte
- xsd:nonNegativeInteger
- xsd:unsignedLong
- xsd:unsignedInt

- xsd:unsignedShort
- xsd:unsignedByte
- xsd:positiveInteger

[xsd:QName]

The dataType of an (simple) element or attribute

`errorValue` *[att.errorValue]*

An observed or calculated estimate of the error in the value of a numeric quantity

An observed or calculated estimate of the error in the value of a numeric quantity. . It should be ignored for dataTypes such as URL, date or string. The statistical basis of the `errorValueType` is not defined - it could be a range, an estimated standard deviation, an observed standard error, etc. This information can be added through **errorBasisType**.

```
<scalar
    dataType="xsd:decimal"
    errorValue="1.0"
    errorBasis="observedStandardDeviation"
    title="body weight"
    dictRef="zoo:bodywt"
    units="units:g">34.3</scalar>
```

[xsd:decimal]

`errorBasis` *[att.errorBasis]*

The basis of an error value

Errors in values can be of several types and this simpleType provides a small controlled vocabulary

```
<scalar
    dataType="xsd:decimal"
    errorValue="1.0"
    errorBasis="observedStandardDeviation"
    title="body weight"
    dictRef="zoo:bodywt"
    units="units:g">34.3</scalar>
```

Allowed values

- observedRange
- observedStandardDeviation
- observedStandardError
- estimatedStandardDeviation
- estimatedStandardError

`min` *[att.min]*

The minimum INCLUSIVE value of a quantity

The minimum INCLUSIVE value of a sortable quantity such as numeric, date or

string. It should be ignored for dataTypes such as URL. The use of `min` and `min` attributes can be used to give a range for the quantity. The statistical basis of this range is not defined. The value of `min` is usually an observed quantity (or calculated from observations). To restrict a value, the `minExclusive` type in a dictionary should be used.
The type of the minimum is the same as the quantity to which it refers - numeric, date and string are currently allowed

```
<scalar dataType="xsd:float" max="20" min="12">15</scalar>
```

[xsd:string]

max *[att.max]*

The maximum INCLUSIVE value of a quantity

The maximum INCLUSIVE value of a sortable quantity such as numeric, date or string. It should be ignored for dataTypes such as URL. The use of `min` and `max` attributes can be used to give a range for the quantity. The statistical basis of this range is not defined. The value of `max` is usually an observed quantity (or calculated from observations). To restrict a value, the `maxExclusive` type in a dictionary should be used.
The type of the maximum is the same as the quantity to which it refers - numeric, date and string are currently allowed

```
<scalar dataType="xsd:float" max="20" min="12">15</scalar>
```

[xsd:string]

units *[att.units]*

Scientific units

These will be linked to dictionaries of units with conversion information, using namespaced references (e.g. `si:m`)
Distinguish carefully from **unitType** which is an element describing a type of a unit in a **unitList**

```
<stm:unitList xmlns:stm="http://www.xml-cml.org/schema/stmml">

<!--
============================================================================
-->
<!-- ======================== fundamental types
========================== -->
<!--
============================================================================
-->

<stm:unitType id="length" name="length">
  <stm:dimension name="length" power="1"/>
</stm:unitType>

<stm:unitType id="time" name="time">
```

```
  <stm:dimension name="time" power="1"/>
</stm:unitType>

<!-- ... -->

<stm:unitType id="dimensionless" name="dimensionless">
  <stm:dimension name="dimensionless" power="1"/>
</stm:unitType>

<!--
============================================================================
-->
<!-- ========================== derived types
============================== -->
<!--
============================================================================
-->

<stm:unitType id="acceleration" name="acceleration">
  <stm:dimension name="length" power="1"/>
  <stm:dimension name="time" power="-2"/>
</stm:unitType>

<!-- ... -->

<!--
============================================================================
-->
<!-- ======================== fundamental SI units
============================ -->
<!--
============================================================================
-->

<stm:unit id="second" name="second" unitType="time">
  <stm:description>The SI unit of time</stm:description>
</stm:unit>

<stm:unit id="meter" name="meter" unitType="length"
  abbreviation="m">
  <stm:description>The SI unit of length</stm:description>
</stm:unit>

<!-- ... -->

<stm:unit id="kg" name="nameless" unitType="dimensionless"
  abbreviation="nodim">
  <stm:description>A fictitious parent for dimensionless
units</stm:description>
</stm:unit>

<!--
============================================================================
-->
<!-- ====================== derived SI units
============================== -->
```

```
<!--
================================================================================
-->

<stm:unit id="newton" name="newton" unitType="force">
  <stm:description>The SI unit of force</stm:description>
</stm:unit>

<!-- ... -->

<!-- multiples of fundamental SI units -->

<stm:unit id="g" name="gram" unitType="mass"
  parentSI="kg"
  multiplierToSI="0.001"
  abbreviation="g">
  <stm:description>0.001 kg. </stm:description>
</stm:unit>

<stm:unit id="celsius" name="Celsius" parentSI="k"
  multiplierToSI="1"
  constantToSI="273.18">
  <stm:description><p>A common unit of
temperature</p></stm:description>
</stm:unit>

<!-- fundamental non-SI units -->

<stm:unit id="inch" name="inch" parentSI="meter"
   abbreviation="in"
  multiplierToSI="0.0254" >
  <stm:description>An imperial measure of
length</stm:description>
</stm:unit>

<!-- derived non-SI units -->

<stm:unit id="l" name="litre" unitType="volume"
   parentSI="meterCubed"
   abbreviation="l"
   multiplierToSI="0.001">
  <stm:description>Nearly 1 dm**3 This is not quite
exact</stm:description>
</stm:unit>

<!-- ... -->

<stm:unit id="fahr" name="fahrenheit" parentSI="k"
   abbreviation="F"
  multiplierToSI="0.55555555555555555"
  constantToSI="-17.777777777777777">
  <stm:description>An obsolescent unit of temperature still
used in popular
  meteorology</stm:description>
</stm:unit>
```

```
</stm:unitList>
```
[xsd:string]

Scientific units on an element.

These must be taken from a dictionary of units. There should be some mechanism for validating the type of the units against the possible values of the element.

stmml *[el.stmml]*

An element to hold stmml data.

`stmml` holds stmml data under a single generic container. Other namespaces may be present as children. No semantics implied.

```
<stmml>
  <actionList>
    <action></action>
  </actionList>
  <object></object>
  <observation></observation>

<!-- ==================== DICTIONARY ========== -->

  <dictionary>
    <annotation>
      <documentation></documentation>
      <appinfo></appinfo>
    </annotation>
    <entry term="foo">
      <definition></definition>
      <alternative></alternative>
      <description></description>
      <enumeration></enumeration>
      <relatedEntry></relatedEntry>
    </entry>
  </dictionary>

<!-- ================  METADATA ================= -->

  <metadataList>
    <metadata></metadata>
  </metadataList>

<!-- ================  SCIENTIFIC UNITS ================
-->

  <unitList>
    <unitType id="ut1" name="u">
      <dimension name="mass" power="1"></dimension>
    </unitType>
    <unit id="u1"></unit>
  </unitList>
```

```
</stmml>
```

Content Model
(ANY [lax])*

`title`*[att.title]*

A title on an element.

No controlled value.

```
<action title="turn on heat" start="T09:00:00"
convention="xsd"/>
```

`id`*[att.id]*

A unique ID for an element

This is not formally of type ID (an XML NAME which must start with a letter and contain only letters, digits and `.-_:`). It is recommended that IDs start with a letter, and contain no punctuation or whitespace. The function `generate-id()` in XSLT will generate semantically void unique IDs.
It is difficult to ensure uniqueness when documents are merged. We suggest namespacing IDs, perhaps using the containing elements as the base. Thus `mol3:a1` could be a useful unique ID. However this is still experimental.

[xsd:string]
Pattern: [A-Za-z0-9_-]+(:[A-Za-z0-9_-]+)?

An attribute providing a unique ID for an element

`convention`*[att.convention]*

A string referencing a dictionary, units, convention or other metadata.

The namespace is optional but recommended where possible
Note: this convention is only used within STMML and related languages; it is NOT a generic URI.

```
<list>
<!-- dictRef is of namespaceRefType -->
  <scalar dictRef="chem:mpt">123</scalar>
<!-- error -->
  <scalar dictRef="mpt23">123</scalar>
</list>
```

[xsd:string]
Pattern: [A-Za-z][A-Za-z0-9_]*(:[A-Za-z][A-Za-z0-9_]*)?

A reference to a convention

There is no controlled vocabulary for conventions, but the author must ensure that the semantics are openly available and that there are mechanisms for implementation. The convention is inherited by all the subelements, so that a convention for `molecule` would by default extend to its `bond` and `atom` children. This can be overwritten if necessary by an explicit `convention`.
It may be useful to create conventions with namespaces (e.g. `iupac:name`). Use of `convention` will normally require non-STMML semantics, and should be used with caution. We would expect that conventions prefixed with "ISO" would be useful, such as ISO8601 for dateTimes.
There is no default, but the conventions of STMML or the related language (e.g. CML) will be assumed.

```
<bond convention="fooChem" order="-5"
   xmlns:fooChem="http://www.fooChem/conventions"/>
```

`dictRef` *[att.dictRef]*

A string referencing a dictionary, units, convention or other metadata.

The namespace is optional but recommended where possible
Note: this convention is only used within STMML and related languages; it is NOT a generic URI.

```
<list>
<!-- dictRef is of namespaceRefType -->
  <scalar dictRef="chem:mpt">123</scalar>
<!-- error -->
  <scalar dictRef="mpt23">123</scalar>
</list>
```

[xsd:string]
Pattern: [A-Za-z][A-Za-z0-9_]*(:[A-Za-z][A-Za-z0-9_]*)?

A reference to a dictionary entry.

Elements in data instances such as **scalar** may have a `dictRef` attribute to point to an entry in a dictionary. To avoid excessive use of (mutable) filenames and URIs we recommend a namespace prefix, mapped to a namespace URI in the normal manner. In this case, of course, the namespace URI must point to a real XML document containing **entry** elements and validated against STMML Schema.
Where there is concern about the dictionary becoming separated from the document the dictionary entries can be physically included as part of the data instance and the normal XPointer addressing mechanism can be used.
This attribute can also be used on **dictionary** elements to define the namespace prefix

```
<scalar dataType="xsd:float" title="surfaceArea"
  dictRef="cmlPhys:surfArea"
  xmlns:cmlPhys="http://www.xml-cml.org/dict/physical"
  units="units:cm2">50</scalar>
<stm:list xmlns:stm="http://www.xml-cml.org/schema/stmml">
  <stm:observation>
    <p>We observed <object count="3" dictRef="foo:p1"/>
```

```
      constructing dwellings of different material</p>
  </stm:observation>
  <stm:entry id="p1" term="pig">
    <stm:definition>A domesticated animal.</stm:definition>
    <stm:description>Predators include
wolves</stm:description>
    <stm:description class="scientificName">Sus
scrofa</stm:description>
  </stm:entry>
</stm:list>
```

`table`*[el.table]*

A rectangular table of any quantities

By default `table` represents a rectangular table of any quantities representable as XSD or STMML dataTypes. The default layout is columnwise, with `columns` columns, where each column is a (homogeneous) **array** of size `rows` data. This is the "normal" orientation of data tables but the table display could be transposed by XSLT transformation if required. Access is to columns, and thence to the data within them. DataTyping, delimiters, etc are delegated to the arrays, which must all be of the same size. For verification it is recommended that every array carries a `size` attribute.

```
<table rows="3" columns="2" title="people">
  <array title="age" dataType="xsd:integer">3 5 7</array>
  <array title="name" dataType="xsd:string">Sue Fred
Sandy</array>
</table>
```

Content Model

(array+)

`rows`[]REQUIRED

The size of an array

The size of an array. Redundant, but serves as a check for processing software (useful if delimiters are used)

[xsd:positiveInteger]

Number of rows

`columns`[]REQUIRED

The size of an array

The size of an array. Redundant, but serves as a check for processing software (useful if delimiters are used)

[xsd:positiveInteger]

Number of columns

`title`*[att.title]*

A title on an element.

No controlled value.

```
<action title="turn on heat" start="T09:00:00"
convention="xsd"/>
```

`id`*[att.id]*

A unique ID for an element

This is not formally of type ID (an XML NAME which must start with a letter and contain only letters, digits and `.-_:`). It is recommended that IDs start with a letter, and contain no punctuation or whitespace. The function `generate-id()` in XSLT will generate semantically void unique IDs.
It is difficult to ensure uniqueness when documents are merged. We suggest namespacing IDs, perhaps using the containing elements as the base. Thus `mol3:a1` could be a useful unique ID. However this is still experimental.

[xsd:string]
Pattern: [A-Za-z0-9_-]+(:[A-Za-z0-9_-]+)?

An attribute providing a unique ID for an element

`convention`*[att.convention]*

A string referencing a dictionary, units, convention or other metadata.

The namespace is optional but recommended where possible
Note: this convention is only used within STMML and related languages; it is NOT a generic URI.

```
<list>
<!-- dictRef is of namespaceRefType -->
  <scalar dictRef="chem:mpt">123</scalar>
<!-- error -->
  <scalar dictRef="mpt23">123</scalar>
</list>
```

[xsd:string]
Pattern: [A-Za-z][A-Za-z0-9_]*(:[A-Za-z][A-Za-z0-9_]*)?

A reference to a convention

There is no controlled vocabulary for conventions, but the author must ensure that the semantics are openly available and that there are mechanisms for implementation. The convention is inherited by all the subelements, so that a convention for `molecule` would by default extend to its `bond` and `atom` children.

This can be overwritten if necessary by an explicit `convention`.
It may be useful to create conventions with namespaces (e.g. `iupac:name`). Use of `convention` will normally require non-STMML semantics, and should be used with caution. We would expect that conventions prefixed with "ISO" would be useful, such as ISO8601 for dateTimes.
There is no default, but the conventions of STMML or the related language (e.g. CML) will be assumed.

```
<bond convention="fooChem" order="-5"
   xmlns:fooChem="http://www.fooChem/conventions"/>
```

dictRef *[att.dictRef]*

A string referencing a dictionary, units, convention or other metadata.

The namespace is optional but recommended where possible
Note: this convention is only used within STMML and related languages; it is NOT a generic URI.

```
<list>
<!-- dictRef is of namespaceRefType -->
  <scalar dictRef="chem:mpt">123</scalar>
<!-- error -->
  <scalar dictRef="mpt23">123</scalar>
</list>
```

[xsd:string]
Pattern: [A-Za-z][A-Za-z0-9_]*(:[A-Za-z][A-Za-z0-9_]*)?

A reference to a dictionary entry.

Elements in data instances such as **scalar** may have a `dictRef` attribute to point to an entry in a dictionary. To avoid excessive use of (mutable) filenames and URIs we recommend a namespace prefix, mapped to a namespace URI in the normal manner. In this case, of course, the namespace URI must point to a real XML document containing **entry** elements and validated against STMML Schema.
Where there is concern about the dictionary becoming separated from the document the dictionary entries can be physically included as part of the data instance and the normal XPointer addressing mechanism can be used.
This attribute can also be used on **dictionary** elements to define the namespace prefix

```
<scalar dataType="xsd:float" title="surfaceArea"
  dictRef="cmlPhys:surfArea"
  xmlns:cmlPhys="http://www.xml-cml.org/dict/physical"
  units="units:cm2">50</scalar>
<stm:list xmlns:stm="http://www.xml-cml.org/schema/stmml">
  <stm:observation>
    <p>We observed <object count="3" dictRef="foo:p1"/>
      constructing dwellings of different material</p>
  </stm:observation>
  <stm:entry id="p1" term="pig">
    <stm:definition>A domesticated animal.</stm:definition>
```

```
    <stm:description>Predators include
wolves</stm:description>
    <stm:description class="scientificName">Sus
scrofa</stm:description>
  </stm:entry>
</stm:list>
```

unit *[el.unit]*

A scientific unit

A scientific unit. Units are of the following types:

- SI Units. These may be one of the seven fundamental types (e.g. meter) or may be derived (e.g. joule). An SI unit is identifiable because it has no parentSI attribute and will have a unitType attribute.
- nonSI Units. These will normally have a parent SI unit (e.g. calorie has joule as an SI parent).

Example:

```
<unit id="units:fahr" name="fahrenheit" parentSI="units:K"
  multiplierToSI="0.55555555555555555"
  constantToSI="-17.777777777777777">
    <description>An obsolescent unit of temperature still used
in popular
      meteorology</description>
</unit>
```

xsd:appinfo

Content Model

(description|annotation)*

id *[]* REQUIRED

abbreviation *[]*

name *[]*

parentSI *[]*

A reference to the parent SI unit (forbidden for SI Units themselves).

unitType *[]*

A reference to the unitType (required for SI Units).

multiplierToSI *[]*

The factor by which the non-SI unit should be multiplied to convert a quantity to its

representation in SI Units. This is applied **before** `constantToSI`. necessarily unity for SI units

`constantToSI`[]

The amount to add to a quantity in non-SI units to convert its representation to SI Units. This is applied **after** `multiplierToSI`. necessarily zero for SI units.

`unitList`*[el.unitList]*

A container for several unit entries

Usually forms the complete units dictionary (along with metadata)

```
<stm:unitList xmlns:stm="http://www.xml-cml.org/schema/stmml">

<!--
============================================================================
-->
<!-- ========================= fundamental types
========================== -->
<!--
============================================================================
-->

<stm:unitType id="length" name="length">
  <stm:dimension name="length" power="1"/>
</stm:unitType>

<stm:unitType id="time" name="time">
  <stm:dimension name="time" power="1"/>
</stm:unitType>

<!-- ... -->

<stm:unitType id="dimensionless" name="dimensionless">
  <stm:dimension name="dimensionless" power="1"/>
</stm:unitType>

<!--
============================================================================
-->
<!-- ========================== derived types
============================= -->
<!--
============================================================================
-->

<stm:unitType id="acceleration" name="acceleration">
  <stm:dimension name="length" power="1"/>
  <stm:dimension name="time" power="-2"/>
</stm:unitType>
```

```
<!-- ... -->

<!--
============================================================================
-->
<!-- ====================== fundamental SI units
========================== -->
<!--
============================================================================
-->

<stm:unit id="second" name="second" unitType="time">
  <stm:description>The SI unit of time</stm:description>
</stm:unit>

<stm:unit id="meter" name="meter" unitType="length"
  abbreviation="m">
  <stm:description>The SI unit of length</stm:description>
</stm:unit>

<!-- ... -->

<stm:unit id="kg" name="nameless" unitType="dimensionless"
  abbreviation="nodim">
  <stm:description>A fictitious parent for dimensionless
units</stm:description>
</stm:unit>

<!--
============================================================================
-->
<!-- ===================== derived SI units
============================== -->
<!--
============================================================================
-->

<stm:unit id="newton" name="newton" unitType="force">
  <stm:description>The SI unit of force</stm:description>
</stm:unit>

<!-- ... -->

<!-- multiples of fundamental SI units -->

<stm:unit id="g" name="gram" unitType="mass"
  parentSI="kg"
  multiplierToSI="0.001"
  abbreviation="g">
  <stm:description>0.001 kg. </stm:description>
</stm:unit>

<stm:unit id="celsius" name="Celsius" parentSI="k"
  multiplierToSI="1"
  constantToSI="273.18">
```

```
  <stm:description><p>A common unit of
temperature</p></stm:description>
</stm:unit>

<!-- fundamental non-SI units -->

<stm:unit id="inch" name="inch" parentSI="meter"
   abbreviation="in"
  multiplierToSI="0.0254" >
  <stm:description>An imperial measure of
length</stm:description>
</stm:unit>

<!-- derived non-SI units -->

<stm:unit id="l" name="litre" unitType="volume"
   parentSI="meterCubed"
   abbreviation="l"
   multiplierToSI="0.001">
  <stm:description>Nearly 1 dm**3 This is not quite
exact</stm:description>
</stm:unit>

<!-- ... -->

<stm:unit id="fahr" name="fahrenheit" parentSI="k"
   abbreviation="F"
  multiplierToSI="0.55555555555555555"
  constantToSI="-17.777777777777777">
  <stm:description>An obsolescent unit of temperature still
used in popular
  meteorology</stm:description>
</stm:unit>

</stm:unitList>
<stm:unitList
    xmlns:stm="http://www.xml-cml.org/schema/stmml"
    dictRef="unit" href="units.xml" />
```

Content Model

(unitType*,unit*)

`title`*[att.title]*

A title on an element.

No controlled value.

```
<action title="turn on heat" start="T09:00:00"
convention="xsd"/>
```

`id`*[att.id]*

A unique ID for an element

This is not formally of type ID (an XML NAME which must start with a letter and contain only letters, digits and `.-_:`). It is recommended that IDs start with a letter, and contain no punctuation or whitespace. The function `generate-id()` in XSLT will generate semantically void unique IDs.
It is difficult to ensure uniqueness when documents are merged. We suggest namespacing IDs, perhaps using the containing elements as the base. Thus `mol3:a1` could be a useful unique ID. However this is still experimental.

[xsd:string]
Pattern: [A-Za-z0-9_-]+(:[A-Za-z0-9_-]+)?

An attribute providing a unique ID for an element

`convention` *[att.convention]*

A string referencing a dictionary, units, convention or other metadata.

The namespace is optional but recommended where possible
Note: this convention is only used within STMML and related languages; it is NOT a generic URI.

```
<list>
<!-- dictRef is of namespaceRefType -->
  <scalar dictRef="chem:mpt">123</scalar>
<!-- error -->
  <scalar dictRef="mpt23">123</scalar>
</list>
```

[xsd:string]
Pattern: [A-Za-z][A-Za-z0-9_]*(:[A-Za-z][A-Za-z0-9_]*)?

A reference to a convention

There is no controlled vocabulary for conventions, but the author must ensure that the semantics are openly available and that there are mechanisms for implementation. The convention is inherited by all the subelements, so that a convention for `molecule` would by default extend to its `bond` and `atom` children. This can be overwritten if necessary by an explicit `convention`.
It may be useful to create conventions with namespaces (e.g. `iupac:name`). Use of `convention` will normally require non-STMML semantics, and should be used with caution. We would expect that conventions prefixed with "ISO" would be useful, such as ISO8601 for dateTimes.
There is no default, but the conventions of STMML or the related language (e.g. CML) will be assumed.

```
<bond convention="fooChem" order="-5"
   xmlns:fooChem="http://www.fooChem/conventions"/>
```

`dictRef` *[att.dictRef]*

A string referencing a dictionary, units, convention or other metadata.

The namespace is optional but recommended where possible
Note: this convention is only used within STMML and related languages; it is NOT a generic URI.

```
<list>
<!-- dictRef is of namespaceRefType -->
  <scalar dictRef="chem:mpt">123</scalar>
<!-- error -->
  <scalar dictRef="mpt23">123</scalar>
</list>
```

[xsd:string]
Pattern: [A-Za-z][A-Za-z0-9_]*(:[A-Za-z][A-Za-z0-9_]*)?

A reference to a dictionary entry.

Elements in data instances such as **scalar** may have a `dictRef` attribute to point to an entry in a dictionary. To avoid excessive use of (mutable) filenames and URIs we recommend a namespace prefix, mapped to a namespace URI in the normal manner. In this case, of course, the namespace URI must point to a real XML document containing **entry** elements and validated against STMML Schema.
Where there is concern about the dictionary becoming separated from the document the dictionary entries can be physically included as part of the data instance and the normal XPointer addressing mechanism can be used.
This attribute can also be used on **dictionary** elements to define the namespace prefix

```
<scalar dataType="xsd:float" title="surfaceArea"
  dictRef="cmlPhys:surfArea"
  xmlns:cmlPhys="http://www.xml-cml.org/dict/physical"
  units="units:cm2">50</scalar>
<stm:list xmlns:stm="http://www.xml-cml.org/schema/stmml">
  <stm:observation>
    <p>We observed <object count="3" dictRef="foo:p1"/>
      constructing dwellings of different material</p>
  </stm:observation>
  <stm:entry id="p1" term="pig">
    <stm:definition>A domesticated animal.</stm:definition>
    <stm:description>Predators include
wolves</stm:description>
    <stm:description class="scientificName">Sus
scrofa</stm:description>
  </stm:entry>
</stm:list>
```

`href`[]

Maps a **dictRef** prefix to the location of a dictionary.

This requires the prefix and the physical URI address to be contained within the same file. We can anticipate that better mechanisms will arise - perhaps through XMLCatalogs. At least it works at present.

unitType *[el.unitType]*

An element containing the description of a scientific unit

Mandatory for SI Units, optional for nonSI units since they should be able to obtain this from their parent. For complex derived units without parents it may be useful. Used within a unitList
Distinguish carefully from **unitsType** which is primarily used for attributes describing the units that elements carry

```
<stm:unitList xmlns:stm="http://www.xml-cml.org/schema/stmml">

<!--
============================================================================
-->
<!-- ========================= fundamental types
========================== -->
<!--
============================================================================
-->

<stm:unitType id="length" name="length">
  <stm:dimension name="length" power="1"/>
</stm:unitType>

<stm:unitType id="time" name="time">
  <stm:dimension name="time" power="1"/>
</stm:unitType>

<!-- ... -->

<stm:unitType id="dimensionless" name="dimensionless">
  <stm:dimension name="dimensionless" power="1"/>
</stm:unitType>

<!--
============================================================================
-->
<!-- ========================== derived types
============================== -->
<!--
============================================================================
-->

<stm:unitType id="acceleration" name="acceleration">
  <stm:dimension name="length" power="1"/>
  <stm:dimension name="time" power="-2"/>
</stm:unitType>

<!-- ... -->

<!--
============================================================================
```

```
-->
<!-- ====================== fundamental SI units
========================== -->
<!--
=============================================================================
-->

<stm:unit id="second" name="second" unitType="time">
  <stm:description>The SI unit of time</stm:description>
</stm:unit>

<stm:unit id="meter" name="meter" unitType="length"
  abbreviation="m">
  <stm:description>The SI unit of length</stm:description>
</stm:unit>

<!-- ... -->

<stm:unit id="kg" name="nameless" unitType="dimensionless"
  abbreviation="nodim">
  <stm:description>A fictitious parent for dimensionless
units</stm:description>
</stm:unit>

<!--
=============================================================================
-->
<!-- ===================== derived SI units
============================== -->
<!--
=============================================================================
-->

<stm:unit id="newton" name="newton" unitType="force">
  <stm:description>The SI unit of force</stm:description>
</stm:unit>

<!-- ... -->

<!-- multiples of fundamental SI units -->

<stm:unit id="g" name="gram" unitType="mass"
  parentSI="kg"
  multiplierToSI="0.001"
  abbreviation="g">
  <stm:description>0.001 kg. </stm:description>
</stm:unit>

<stm:unit id="celsius" name="Celsius" parentSI="k"
  multiplierToSI="1"
  constantToSI="273.18">
  <stm:description><p>A common unit of
temperature</p></stm:description>
</stm:unit>

<!-- fundamental non-SI units -->
```

```
<stm:unit id="inch" name="inch" parentSI="meter"
   abbreviation="in"
  multiplierToSI="0.0254" >
  <stm:description>An imperial measure of
length</stm:description>
</stm:unit>

<!-- derived non-SI units -->

<stm:unit id="l" name="litre" unitType="volume"
   parentSI="meterCubed"
   abbreviation="l"
   multiplierToSI="0.001">
  <stm:description>Nearly 1 dm**3 This is not quite
exact</stm:description>
</stm:unit>

<!-- ... -->

<stm:unit id="fahr" name="fahrenheit" parentSI="k"
   abbreviation="F"
  multiplierToSI="0.55555555555555555"
  constantToSI="-17.777777777777777">
  <stm:description>An obsolescent unit of temperature still
used in popular
  meteorology</stm:description>
</stm:unit>

</stm:unitList>
```

Content Model
(dimension*)

---

id[]REQUIRED

A unique ID for an element

This is not formally of type ID (an XML NAME which must start with a letter and contain only letters, digits and `.-_:`). It is recommended that IDs start with a letter, and contain no punctuation or whitespace. The function `generate-id()` in XSLT will generate semantically void unique IDs.
It is difficult to ensure uniqueness when documents are merged. We suggest namespacing IDs, perhaps using the containing elements as the base. Thus `mol3:a1` could be a useful unique ID. However this is still experimental.

[xsd:string]
Pattern: [A-Za-z0-9_-]+(:[A-Za-z0-9_-]+)?

---

name[]REQUIRED

ATTRIBUTES

---

convention*[att.convention]*

A string referencing a dictionary, units, convention or other metadata.

The namespace is optional but recommended where possible
Note: this convention is only used within STMML and related languages; it is NOT a generic URI.

```
<list>
<!-- dictRef is of namespaceRefType -->
  <scalar dictRef="chem:mpt">123</scalar>
<!-- error -->
  <scalar dictRef="mpt23">123</scalar>
</list>
```

[xsd:string]
Pattern: [A-Za-z][A-Za-z0-9_]*(:[A-Za-z][A-Za-z0-9_]*)?

A reference to a convention

There is no controlled vocabulary for conventions, but the author must ensure that the semantics are openly available and that there are mechanisms for implementation. The convention is inherited by all the subelements, so that a convention for `molecule` would by default extend to its `bond` and `atom` children. This can be overwritten if necessary by an explicit `convention`.
It may be useful to create conventions with namespaces (e.g. `iupac:name`). Use of `convention` will normally require non-STMML semantics, and should be used with caution. We would expect that conventions prefixed with "ISO" would be useful, such as ISO8601 for dateTimes.
There is no default, but the conventions of STMML or the related language (e.g. CML) will be assumed.

```
<bond convention="fooChem" order="-5"
   xmlns:fooChem="http://www.fooChem/conventions"/>
```

---

dataType*[att.dataType]*Default:xsd:string

an enumerated type for all builtin allowed dataTypes in STM

`dataTypeType` represents an enumeration of allowed dataTypes (at present identical with those in XML-Schemas (Part2- datatypes). This means that implementers should be able to use standard XMLSchema-based tools for validation without major implementation problems.
It will often be used an an attribute on **scalar**, **array** or **matrix** elements.

```
<list xmlns="http://www.xml-cml.org/schema/cml2/core">
  <scalar dataType="xsd:boolean" title="she loves
me">true</scalar>
  <scalar dataType="xsd:float" title="x">23.2</scalar>
  <scalar dataType="xsd:duration" title="egg
timer">PM4</scalar>
  <scalar dataType="xsd:dateTime" title="current data and
time">2001-02-01:00:30</scalar>
  <scalar dataType="xsd:time" title="wake up">06:00</scalar>
  <scalar dataType="xsd:date" title="where is
it">1752-09-10</scalar>
  <scalar dataType="xsd:anyURI" title="CML
site">http://www.xml-cml.org/</scalar>
  <scalar dataType="xsd:QName" title="CML
atom">cml:atom</scalar>
  <scalar dataType="xsd:normalizedString" title="song">the
mouse ran up the clock</scalar>
  <scalar dataType="xsd:language" title="UK
English">en-GB</scalar>
  <scalar dataType="xsd:Name" title="atom">atom</scalar>
  <scalar dataType="xsd:ID" title="XML ID">_123</scalar>
  <scalar dataType="xsd:integer" title="the
answer">42</scalar>
  <scalar dataType="xsd:nonPositiveInteger"
title="zero">0</scalar>
</list>
```

UNION OF
Allowed values

- xsd:string
- xsd:boolean
- xsd:float
- xsd:double
- xsd:decimal
- xsd:duration
- xsd:dateTime
- xsd:time
- xsd:date
- xsd:gYearMonth
- xsd:gYear
- xsd:gMonthDay
- xsd:gDay
- xsd:gMonth

- xsd:hexBinary
- xsd:base64Binary
- xsd:anyURI
- xsd:QName
- xsd:NOTATION
- xsd:normalizedString
- xsd:token
- xsd:language
- xsd:IDREFS
- xsd:ENTITIES
- xsd:NMTOKEN
- xsd:NMTOKENS
- xsd:Name
- xsd:NCName
- xsd:ID
- xsd:IDREF
- xsd:ENTITY
- xsd:integer
- xsd:nonPositiveInteger
- xsd:negativeInteger
- xsd:long
- xsd:int
- xsd:short
- xsd:byte
- xsd:nonNegativeInteger
- xsd:unsignedLong
- xsd:unsignedInt
- xsd:unsignedShort
- xsd:unsignedByte
- xsd:positiveInteger

[xsd:QName]

The dataType of an (simple) element or attribute

`delimiter` *[att.delimiter]*

A non-whitespace character used in arrays to separate components

Some STMML elements (such as **array**) have content representing concatenated values. The default separator is whitespace (which can be normalised) and this should be used whenever possible. However in some cases the values are empty, or contain whitespace or other problematic punctuation, and a delimiter is required. Note that the content string MUST start and end with the delimiter so there is no ambiguity as to what the components are. Only printable characters from the ASCII character set should be used, and character entities should be avoided.
When delimiters are used to separate precise whitespace this should always consist of spaces and not the other allowed whitespace characters (newline, tabs, etc.). If the

latter are important it is probably best to redesign the application.

```
<array size="4"  dataType="xsd:string" delimiter="|">|A|B12||D
and   E|</array>


 The values in the array are
  "A", "B12", "" (empty string) and "D and   E"
 note the spaces
```

[xsd:string]

A delimiter character for arrays and matrices

By default array components ('elements' in the non-XML sense) are whitespace-separated. This fails for components with embedded whitespace or missing completely:

```
        Example:
        In the protein database ' CA' and 'CA' are different
atom types, and and array could be:
        <array delimiter="/" dictRef="pdb:atomTypes">/ N/
CA/CA/ N/</array>
```

Note that the array starts and ends with the delimiter, which must be chosen to avoid accidental use. There is currently no syntax for escaping delimiters.

---

dictRef *[att.dictRef]*

A string referencing a dictionary, units, convention or other metadata.

The namespace is optional but recommended where possible
Note: this convention is only used within STMML and related languages; it is NOT a generic URI.

```
<list>
<!-- dictRef is of namespaceRefType -->
  <scalar dictRef="chem:mpt">123</scalar>
<!-- error -->
  <scalar dictRef="mpt23">123</scalar>
</list>
```

[xsd:string]
Pattern: [A-Za-z][A-Za-z0-9_]*(:[A-Za-z][A-Za-z0-9_]*)?

A reference to a dictionary entry.

Elements in data instances such as **scalar** may have a `dictRef` attribute to point to an entry in a dictionary. To avoid excessive use of (mutable) filenames and URIs we recommend a namespace prefix, mapped to a namespace URI in the normal manner. In this case, of course, the namespace URI must point to a real XML

document containing **entry** elements and validated against STMML Schema.
Where there is concern about the dictionary becoming separated from the document the dictionary entries can be physically included as part of the data instance and the normal XPointer addressing mechanism can be used.
This attribute can also be used on **dictionary** elements to define the namespace prefix

```
<scalar dataType="xsd:float" title="surfaceArea"
  dictRef="cmlPhys:surfArea"
  xmlns:cmlPhys="http://www.xml-cml.org/dict/physical"
  units="units:cm2">50</scalar>
<stm:list xmlns:stm="http://www.xml-cml.org/schema/stmml">
  <stm:observation>
    <p>We observed <object count="3" dictRef="foo:p1"/>
      constructing dwellings of different material</p>
  </stm:observation>
  <stm:entry id="p1" term="pig">
    <stm:definition>A domesticated animal.</stm:definition>
    <stm:description>Predators include
wolves</stm:description>
    <stm:description class="scientificName">Sus
scrofa</stm:description>
  </stm:entry>
</stm:list>
```

---

errorBasis *[att.errorBasis]*

The basis of an error value

Errors in values can be of several types and this simpleType provides a small controlled vocabulary

```
<scalar
    dataType="xsd:decimal"
    errorValue="1.0"
    errorBasis="observedStandardDeviation"
    title="body weight"
    dictRef="zoo:bodywt"
    units="units:g">34.3</scalar>
```

Allowed values

- observedRange
- observedStandardDeviation
- observedStandardError
- estimatedStandardDeviation
- estimatedStandardError

---

errorValue *[att.errorValue]*

An observed or calculated estimate of the error in the value of a numeric quantity

An observed or calculated estimate of the error in the value of a numeric quantity. . It should be ignored for dataTypes such as URL, date or string. The statistical basis of the `errorValueType` is not defined - it could be a range, an estimated standard deviation, an observed standard error, etc. This information can be added through **errorBasisType**.

```
<scalar
    dataType="xsd:decimal"
    errorValue="1.0"
    errorBasis="observedStandardDeviation"
    title="body weight"
    dictRef="zoo:bodywt"
    units="units:g">34.3</scalar>
```

[xsd:decimal]

`id` *[att.id]*

A unique ID for an element

This is not formally of type ID (an XML NAME which must start with a letter and contain only letters, digits and `.-_:`). It is recommended that IDs start with a letter, and contain no punctuation or whitespace. The function `generate-id()` in XSLT will generate semantically void unique IDs.
It is difficult to ensure uniqueness when documents are merged. We suggest namespacing IDs, perhaps using the containing elements as the base. Thus `mol3:a1` could be a useful unique ID. However this is still experimental.

[xsd:string]
Pattern: [A-Za-z0-9_-]+(:[A-Za-z0-9_-]+)?

An attribute providing a unique ID for an element

`max` *[att.max]*

The maximum INCLUSIVE value of a quantity

The maximum INCLUSIVE value of a sortable quantity such as numeric, date or string. It should be ignored for dataTypes such as URL. The use of `min` and `max` attributes can be used to give a range for the quantity. The statistical basis of this range is not defined. The value of `max` is usually an observed quantity (or calculated from observations). To restrict a value, the `maxExclusive` type in a dictionary should be used.
The type of the maximum is the same as the quantity to which it refers - numeric, date and string are currently allowed

```
<scalar dataType="xsd:float" max="20" min="12">15</scalar>
```

[xsd:string]

`min` *[att.min]*

The minimum INCLUSIVE value of a quantity

The minimum INCLUSIVE value of a sortable quantity such as numeric, date or string. It should be ignored for dataTypes such as URL. The use of `min` and `min` attributes can be used to give a range for the quantity. The statistical basis of this range is not defined. The value of `min` is usually an observed quantity (or calculated from observations). To restrict a value, the `minExclusive` type in a dictionary should be used.
The type of the minimum is the same as the quantity to which it refers - numeric, date and string are currently allowed

```
<scalar dataType="xsd:float" max="20" min="12">15</scalar>
```

[xsd:string]

`ref` *[att.ref]*

A reference to an existing element

A reference to an existing element in the document. The target of the ref attribute must exist. The test for validity will normally occur in the element's `appinfo`
Any DOM Node created from this element will normally be a *reference* to another Node, so that if the target node is modified a the dereferenced content is modified. At present there are no deep copy semantics hardcoded into the schema.

BASE: idType

A unique ID for an element

This is not formally of type ID (an XML NAME which must start with a letter and contain only letters, digits and `.-_:`). It is recommended that IDs start with a letter, and contain no punctuation or whitespace. The function `generate-id()` in XSLT will generate semantically void unique IDs.
It is difficult to ensure uniqueness when documents are merged. We suggest namespacing IDs, perhaps using the containing elements as the base. Thus `mol3:a1` could be a useful unique ID. However this is still experimental.

[xsd:string]
Pattern: [A-Za-z0-9_-]+(:[A-Za-z0-9_-]+)?

A reference to an element of given type

`ref` modifies an element into a reference to an existing element of that type within the document. This is similar to a pointer and it can be thought of a strongly typed

hyperlink. It may also be used for "subclassing" or "overriding" elements.

```
<cml>
  <molecule id="m1">
    <atomArray>
      <atom elementType="N"/>
      <atom elementType="O"/>
    </atomArray>
  </molecule>
  <html:p>The action of <molecule ref="#m1"/> on cardiac
muscle ...</html:p>
</cml>
```

`size` *[att.size]*

The size of an array

The size of an array. Redundant, but serves as a check for processing software (useful if delimiters are used)

[xsd:positiveInteger]

The size of an array, matrix, list, etc.

`source` *[att.source]*

An attribute linking to the source of the information

A simple way of adding metadata to a piece of information. Likely to be fragile since the URI may disappear.

```
<list>
  <definition source="foo.html#a3">An animal with four
legs</definition>
  <definition source="http://www.foo.com/index.html">
    An animal with six legs</definition>
</list>
```

`title` *[att.title]*

A title on an element.

No controlled value.

```
<action title="turn on heat" start="T09:00:00"
convention="xsd"/>
```

---

units*[att.units]*

Scientific units

These will be linked to dictionaries of units with conversion information, using namespaced references (e.g. `si:m`)
Distinguish carefully from **unitType** which is an element describing a type of a unit in a **unitList**

```
<stm:unitList xmlns:stm="http://www.xml-cml.org/schema/stmml">

<!--
============================================================================
-->
<!-- ========================= fundamental types
========================== -->
<!--
============================================================================
-->

<stm:unitType id="length" name="length">
  <stm:dimension name="length" power="1"/>
</stm:unitType>

<stm:unitType id="time" name="time">
  <stm:dimension name="time" power="1"/>
</stm:unitType>

<!-- ... -->

<stm:unitType id="dimensionless" name="dimensionless">
  <stm:dimension name="dimensionless" power="1"/>
</stm:unitType>

<!--
============================================================================
-->
<!-- ========================== derived types
============================= -->
<!--
============================================================================
-->

<stm:unitType id="acceleration" name="acceleration">
  <stm:dimension name="length" power="1"/>
  <stm:dimension name="time" power="-2"/>
</stm:unitType>

<!-- ... -->
```

```
<!--
=============================================================================
-->
<!-- ====================== fundamental SI units
========================== -->
<!--
=============================================================================
-->

<stm:unit id="second" name="second" unitType="time">
  <stm:description>The SI unit of time</stm:description>
</stm:unit>

<stm:unit id="meter" name="meter" unitType="length"
  abbreviation="m">
  <stm:description>The SI unit of length</stm:description>
</stm:unit>

<!-- ... -->

<stm:unit id="kg" name="nameless" unitType="dimensionless"
  abbreviation="nodim">
  <stm:description>A fictitious parent for dimensionless
units</stm:description>
</stm:unit>

<!--
=============================================================================
-->
<!-- ===================== derived SI units
============================== -->
<!--
=============================================================================
-->

<stm:unit id="newton" name="newton" unitType="force">
  <stm:description>The SI unit of force</stm:description>
</stm:unit>

<!-- ... -->

<!-- multiples of fundamental SI units -->

<stm:unit id="g" name="gram" unitType="mass"
  parentSI="kg"
  multiplierToSI="0.001"
  abbreviation="g">
  <stm:description>0.001 kg. </stm:description>
</stm:unit>

<stm:unit id="celsius" name="Celsius" parentSI="k"
  multiplierToSI="1"
  constantToSI="273.18">
  <stm:description><p>A common unit of
temperature</p></stm:description>
</stm:unit>
```

```
<!-- fundamental non-SI units -->

<stm:unit id="inch" name="inch" parentSI="meter"
   abbreviation="in"
  multiplierToSI="0.0254" >
  <stm:description>An imperial measure of
length</stm:description>
</stm:unit>

<!-- derived non-SI units -->

<stm:unit id="l" name="litre" unitType="volume"
   parentSI="meterCubed"
   abbreviation="l"
   multiplierToSI="0.001">
  <stm:description>Nearly 1 dm**3 This is not quite
exact</stm:description>
</stm:unit>

<!-- ... -->

<stm:unit id="fahr" name="fahrenheit" parentSI="k"
   abbreviation="F"
  multiplierToSI="0.55555555555555555"
  constantToSI="-17.777777777777777">
  <stm:description>An obsolescent unit of temperature still
used in popular
  meteorology</stm:description>
</stm:unit>

</stm:unitList>
```

[xsd:string]

Scientific units on an element.

These must be taken from a dictionary of units. There should be some mechanism for validating the type of the units against the possible values of the element.

SIMPLETYPES

coordinate2Type[st.coordinate2Type]

An x/y coordinate pair

An x/y coordinate pair consisting of two real numbers, separated by whitespace or a comma. In arrays and matrices, it may be useful to set a separate delimiter

```
<list>
  <array dataType="xsd:decimal"
      >1.2,3.4   3.2,4.5   6.7,23.1 </array>
  <array delimiter="/" dataType="xsd:decimal"
      >/1.2 3.4/3.2 4.5/6.7 23.1/</array>
</list>
```

[xsd:string]
Pattern: \s*([-]|[+])?\d*\.?\d*(\s+|[,])([-]|[+])?\d*\.?\d*\s*

coordinate3Type[st.coordinate3Type]

An x/y/z coordinate triple

An x/y/z coordinate triple consisting of three real numbers, separated by whitespace or commas. In arrays and matrices, it may be useful to set a separate delimiter

```
<list>
  <array dataType="xsd:decimal">1.2,3.4,1.2
    3.2,4.5,7.3   6.7,23.1,5.6 </array>
  <array delimiter="/" dataType="xsd:decimal"
  >/1.2 3.4 3.3/3.2 4.5 4.5/6.7 23.1 5.6/</array>
</list>
```

[xsd:string]
Pattern: \s*([-]|[+])?\d*\.?\d*(\s+|[,])([-]|[+])?\d*\.?\d*(\s+|[,])([-]|[+])?\d*\.?\d*\s*

countType[st.countType]

A count multiplier for an element

Many elements represent objects which can occur an arbitrary number of times in a scientific context. Examples are `action`, `object` or `molecule`s.

```
<list>
<object title="frog" count="10"/>
<action title="step3" count="3">
  <p>Add 10 ml reagent</p>
</action>
</list>
```

[xsd:nonNegativeInteger]

---

dataTypeType[st.dataTypeType]

an enumerated type for all builtin allowed dataTypes in STM

`dataTypeType` represents an enumeration of allowed dataTypes (at present identical with those in XML-Schemas (Part2- datatypes). This means that implementers should be able to use standard XMLSchema-based tools for validation without major implementation problems.
It will often be used an an attribute on **scalar**, **array** or **matrix** elements.

```
<list xmlns="http://www.xml-cml.org/schema/cml2/core">
  <scalar dataType="xsd:boolean" title="she loves
me">true</scalar>
  <scalar dataType="xsd:float" title="x">23.2</scalar>
  <scalar dataType="xsd:duration" title="egg
timer">PM4</scalar>
  <scalar dataType="xsd:dateTime" title="current data and
time">2001-02-01:00:30</scalar>
  <scalar dataType="xsd:time" title="wake up">06:00</scalar>
  <scalar dataType="xsd:date" title="where is
it">1752-09-10</scalar>
  <scalar dataType="xsd:anyURI" title="CML
site">http://www.xml-cml.org/</scalar>
  <scalar dataType="xsd:QName" title="CML
atom">cml:atom</scalar>
  <scalar dataType="xsd:normalizedString" title="song">the
mouse ran up the clock</scalar>
  <scalar dataType="xsd:language" title="UK
English">en-GB</scalar>
  <scalar dataType="xsd:Name" title="atom">atom</scalar>
  <scalar dataType="xsd:ID" title="XML ID">_123</scalar>
  <scalar dataType="xsd:integer" title="the
answer">42</scalar>
  <scalar dataType="xsd:nonPositiveInteger"
title="zero">0</scalar>
```

```
</list>
```

UNION OF
Allowed values

- xsd:string
- xsd:boolean
- xsd:float
- xsd:double
- xsd:decimal
- xsd:duration
- xsd:dateTime
- xsd:time
- xsd:date
- xsd:gYearMonth
- xsd:gYear
- xsd:gMonthDay
- xsd:gDay
- xsd:gMonth
- xsd:hexBinary
- xsd:base64Binary
- xsd:anyURI
- xsd:QName
- xsd:NOTATION
- xsd:normalizedString
- xsd:token
- xsd:language
- xsd:IDREFS
- xsd:ENTITIES
- xsd:NMTOKEN
- xsd:NMTOKENS
- xsd:Name
- xsd:NCName
- xsd:ID
- xsd:IDREF
- xsd:ENTITY
- xsd:integer
- xsd:nonPositiveInteger
- xsd:negativeInteger
- xsd:long
- xsd:int
- xsd:short
- xsd:byte
- xsd:nonNegativeInteger
- xsd:unsignedLong
- xsd:unsignedInt
- xsd:unsignedShort
- xsd:unsignedByte
- xsd:positiveInteger

[xsd:QName]

---

delimiterType[st.delimiterType]

A non-whitespace character used in arrays to separate components

Some STMML elements (such as **array**) have content representing concatenated values. The default separator is whitespace (which can be normalised) and this should be used whenever possible. However in some cases the values are empty, or contain whitespace or other problematic punctuation, and a delimiter is required. Note that the content string MUST start and end with the delimiter so there is no ambiguity as to what the components are. Only printable characters from the ASCII character set should be used, and character entities should be avoided.
When delimiters are used to separate precise whitespace this should always consist of spaces and not the other allowed whitespace characters (newline, tabs, etc.). If the latter are important it is probably best to redesign the application.

```
<array size="4"  dataType="xsd:string" delimiter="|">|A|B12||D
and   E|</array>


 The values in the array are
  "A", "B12", "" (empty string) and "D and   E"
 note the spaces
```

[xsd:string]

---

dimensionType[st.dimensionType]

Allowed values for dimension Types (for quantities).

These are the 7 types prescribed by the SI system, together with the "dimensionless" type. We intend to be somewhat uncoventional and explore enhanced values of "dimensionless", such as "angle". This may be heretical, but we find the present system impossible to implement in many cases.
Used for constructing entries in a dictionary of units

```
<unitType id="energy" name="energy">
  <dimension name="length" power="2"/>
  <dimension name="mass" power="1"/>
  <dimension name="time" power="-2"/>
</unitType>
```

Allowed values

- mass
- length
- time

- charge
- amount
- luminosity
- temperature
- dimensionless
- angle

  An angle (formally dimensionless, but useful to have units).

---

errorBasisType[st.errorBasisType]

The basis of an error value

Errors in values can be of several types and this simpleType provides a small controlled vocabulary

```
<scalar
    dataType="xsd:decimal"
    errorValue="1.0"
    errorBasis="observedStandardDeviation"
    title="body weight"
    dictRef="zoo:bodywt"
    units="units:g">34.3</scalar>
```

Allowed values

- observedRange
- observedStandardDeviation
- observedStandardError
- estimatedStandardDeviation
- estimatedStandardError

---

errorValueType[st.errorValueType]

An observed or calculated estimate of the error in the value of a numeric quantity

An observed or calculated estimate of the error in the value of a numeric quantity. . It should be ignored for dataTypes such as URL, date or string. The statistical basis of the `errorValueType` is not defined - it could be a range, an estimated standard deviation, an observed standard error, etc. This information can be added through **errorBasisType**.

```
<scalar
    dataType="xsd:decimal"
    errorValue="1.0"
    errorBasis="observedStandardDeviation"
    title="body weight"
    dictRef="zoo:bodywt"
    units="units:g">34.3</scalar>
```

[xsd:decimal]

floatArrayType[st.floatArrayType]

An array of floats

An array of floats or other real numbers. Not used in STM Schema, but re-used by CML and other languages.

```
<atomArray xmlns="http://www.xml-cml.org/schema/cml2/core"
  x2="1.2 2.3 3.4 5.6"/>
```

XSD:LIST of xsd:decimal

idType[st.idType]

A unique ID for an element

This is not formally of type ID (an XML NAME which must start with a letter and contain only letters, digits and `.-_:`). It is recommended that IDs start with a letter, and contain no punctuation or whitespace. The function `generate-id()` in XSLT will generate semantically void unique IDs.
It is difficult to ensure uniqueness when documents are merged. We suggest namespacing IDs, perhaps using the containing elements as the base. Thus `mol3:a1` could be a useful unique ID. However this is still experimental.

[xsd:string]
Pattern: [A-Za-z0-9_-]+(:[A-Za-z0-9_-]+)?

matrixType[st.matrixType]

Allowed **matrix** types

Allowed `matrix` types. These are mainly square matrices

```
<matrix id="m1" title="mattrix-1" dictRef="foo:bar"
  rows="3" columns="3" dataType="xsd:decimal"
  delimiter="|" matrixType="squareSymmetric" units="unit:m"
  >|1.1|1.2|1.3|1.2|2.2|2.3|1.3|2.3|3.3!</matrix>
```

UNION OF
Allowed values

- rectangular
- square
- squareSymmetric
- squareAntisymmetric
- diagonal

Symmetric. Elements are zero except on the diagonal

- upperTriangular

Square. Elements are zero below the diagonal

```
1 2 3 4
0 3 5 6
0 0 4 8
0 0 0 2
```

- lowerTriangular

Symmetric. Elements are zero except on the diagonal

- unitary
- rowEigenvectors
- rotation22
- rotationTranslation32
- homogeneous33
- rotation33
- rotationTranslation43
- homogeneous44
- square
- square

BASE: namespaceRefType

A string referencing a dictionary, units, convention or other metadata.

The namespace is optional but recommended where possible
Note: this convention is only used within STMML and related languages; it is NOT a generic URI.

```
<list>
<!-- dictRef is of namespaceRefType -->
  <scalar dictRef="chem:mpt">123</scalar>
<!-- error -->
  <scalar dictRef="mpt23">123</scalar>
</list>
```

[xsd:string]
Pattern: [A-Za-z][A-Za-z0-9_]*(:[A-Za-z][A-Za-z0-9_]*)?

User-defined matrix-type

This definition must be by reference to a namespaced dictionary entry.

---

maxType[st.maxType]

The maximum INCLUSIVE value of a quantity

The maximum INCLUSIVE value of a sortable quantity such as numeric, date or string. It should be ignored for dataTypes such as URL. The use of `min` and `max` attributes can be used to give a range for the quantity. The statistical basis of this range is not defined. The value of `max` is usually an observed quantity (or calculated from observations). To restrict a value, the `maxExclusive` type in a dictionary should be used.
The type of the maximum is the same as the quantity to which it refers - numeric, date and string are currently allowed

```
<scalar dataType="xsd:float" max="20" min="12">15</scalar>
```

[xsd:string]

---

metadataType[md.metadataType]

The name of the metadata

Metadata consists of name-value pairs (value is in the "content" attribute). The names are from a semi-restricted vocabulary, mainly Dublin Core. The content is unrestricted. The order of metadata has no implied semantics at present.

Allowed values

- dc:coverage

  The extent or scope of the content of the resource.

  Coverage will typically include spatial location (a place name or geographic coordinates), temporal period (a period label, date, or date range) or jurisdiction (such as a named administrative entity). Recommended best practice is to select a value from a controlled vocabulary (for example, the Thesaurus of Geographic Names [TGN]) and that, where appropriate, named places or time periods be used in preference to numeric identifiers such as sets of coordinates or date ranges.

- dc:description

  An account of the content of the resource.

  Description may include but is not limited to: an abstract, table of contents, reference to a graphical representation of content or a free-text account of the content.

- dc:identifier

  An unambiguous reference to the resource within a given context.

  Recommended best practice is to identify the resource by means of a string or number conforming to a formal identification system. Example formal identification systems include the Uniform Resource Identifier (URI) (including the Uniform Resource Locator (URL)), the Digital Object Identifier (DOI) and the International Standard Book Number (ISBN).

- dc:format

  The physical or digital manifestation of the resource.

Typically, Format may include the media-type or dimensions of the resource. Format may be used to determine the software, hardware or other equipment needed to display or operate the resource. Examples of dimensions include size and duration. Recommended best practice is to select a value from a controlled vocabulary (for example, the list of Internet Media Types [MIME] defining computer media formats).

- dc:relation

A reference to a related resource.

Recommended best practice is to reference the resource by means of a string or number conforming to a formal identification system.

- dc:rights

Information about rights held in and over the resource.

Typically, a Rights element will contain a rights management statement for the resource, or reference a service providing such information. Rights information often encompasses Intellectual Property Rights (IPR), Copyright, and various Property Rights. If the Rights element is absent, no assumptions can be made about the status of these and other rights with respect to the resource.

- dc:subject

The topic of the content of the resource.

Typically, a Subject will be expressed as keywords, key phrases or classification codes that describe a topic of the resource. Recommended best practice is to select a value from a controlled vocabulary or formal classification scheme.

- dc:title

A name given to the resource.

Typically, a Title will be a name by which the resource is formally known.

- dc:type

The nature or genre of the content of the resource.

Type includes terms describing general categories, functions, genres, or aggregation levels for content. Recommended best practice is to select a value from a controlled vocabulary (for example, the working draft list of Dublin Core Types [DCT1]). To describe the physical or digital manifestation of the resource, use the FORMAT element.

- dc:contributor

An entity responsible for making contributions to the content of the resource.

Examples of a Contributor include a person, an organisation, or a service. Typically, the name of a Contributor should be used to indicate the entity.

- dc:creator

An entity primarily responsible for making the content of the resource.

Examples of a Creator include a person, an organisation, or a service. Typically,

the name of a Creator should be used to indicate the entity.

- dc:publisher

  An entity responsible for making the resource available

  Examples of a Publisher include a person, an organisation, or a service. Typically, the name of a Publisher should be used to indicate the entity.

- dc:source

  A Reference to a resource from which the present resource is derived.

  The present resource may be derived from the Source resource in whole or in part. Recommended best practice is to reference the resource by means of a string or number conforming to a formal identification system.

- dc:language

  A language of the intellectual content of the resource.

  Recommended best practice for the values of the Language element is defined by RFC 1766 [RFC1766] which includes a two-letter Language Code (taken from the ISO 639 standard [ISO639]), followed optionally, by a two-letter Country Code (taken from the ISO 3166 standard [ISO3166]). For example, 'en' for English, 'fr' for French, or 'en-uk' for English used in the United Kingdom.

- dc:date

  A date associated with an event in the life cycle of the resource.

  Typically, Date will be associated with the creation or availability of the resource. Recommended best practice for encoding the date value is defined in a profile of ISO 8601 [W3CDTF] and follows the YYYY-MM-DD format.

- cmlm:safety

  Entry contains information relating to chemical safety

  Typically the content will be a reference to a handbook, MSDS, threshhold or other human-readable string

- cmlm:insilico

  Part or whole of the information was computer-generated

  Typically the content will be the name of a method or a program

- cmlm:structure

  3D structure included

  details included

- cmlm:reaction
- cmlm:identifier
- other

---

minType[st.minType]

The minimum INCLUSIVE value of a quantity

The minimum INCLUSIVE value of a sortable quantity such as numeric, date or string. It should be ignored for dataTypes such as URL. The use of `min` and `min` attributes can be used to give a range for the quantity. The statistical basis of this range is not defined. The value of `min` is usually an observed quantity (or calculated from observations). To restrict a value, the `minExclusive` type in a dictionary should be used.
The type of the minimum is the same as the quantity to which it refers - numeric, date and string are currently allowed

```
<scalar dataType="xsd:float" max="20" min="12">15</scalar>
```

[xsd:string]

---

namespaceRefType[st.namespaceRefType]

A string referencing a dictionary, units, convention or other metadata.

The namespace is optional but recommended where possible
Note: this convention is only used within STMML and related languages; it is NOT a generic URI.

```
<list>
<!-- dictRef is of namespaceRefType -->
  <scalar dictRef="chem:mpt">123</scalar>
<!-- error -->
  <scalar dictRef="mpt23">123</scalar>
</list>
```

[xsd:string]
Pattern: [A-Za-z][A-Za-z0-9_]*(:[A-Za-z][A-Za-z0-9_]*)?

---

refType[st.refType]

A reference to an existing element

A reference to an existing element in the document. The target of the ref attribute must exist. The test for validity will normally occur in the element's `appinfo`
Any DOM Node created from this element will normally be a *reference* to another Node, so that if the target node is modified a the dereferenced content is modified. At present there are no deep copy semantics hardcoded into the schema.

BASE: idType

A unique ID for an element

This is not formally of type ID (an XML NAME which must start with a letter and contain only letters, digits and `.-_:`). It is recommended that IDs start with a letter,

and contain no punctuation or whitespace. The function `generate-id()` in XSLT will generate semantically void unique IDs.
It is difficult to ensure uniqueness when documents are merged. We suggest namespacing IDs, perhaps using the containing elements as the base. Thus `mol3:a1` could be a useful unique ID. However this is still experimental.
[xsd:string]
Pattern: [A-Za-z0-9_-]+(:[A-Za-z0-9_-]+)?

---

sizeType[st.sizeType]

The size of an array

The size of an array. Redundant, but serves as a check for processing software (useful if delimiters are used)
[xsd:positiveInteger]

---

unitsType[st.unitsType]

Scientific units

These will be linked to dictionaries of units with conversion information, using namespaced references (e.g. `si:m`)
Distinguish carefully from **unitType** which is an element describing a type of a unit in a **unitList**

```
<stm:unitList xmlns:stm="http://www.xml-cml.org/schema/stmml">

<!--
============================================================================
-->
<!-- ========================= fundamental types
========================== -->
<!--
============================================================================
-->

<stm:unitType id="length" name="length">
  <stm:dimension name="length" power="1"/>
</stm:unitType>

<stm:unitType id="time" name="time">
  <stm:dimension name="time" power="1"/>
</stm:unitType>

<!-- ... -->

<stm:unitType id="dimensionless" name="dimensionless">
  <stm:dimension name="dimensionless" power="1"/>
</stm:unitType>
```

```
<!--
============================================================================
-->
<!-- ========================== derived types
============================== -->
<!--
============================================================================
-->

<stm:unitType id="acceleration" name="acceleration">
  <stm:dimension name="length" power="1"/>
  <stm:dimension name="time" power="-2"/>
</stm:unitType>

<!-- ... -->

<!--
============================================================================
-->
<!-- ======================== fundamental SI units
============================ -->
<!--
============================================================================
-->

<stm:unit id="second" name="second" unitType="time">
  <stm:description>The SI unit of time</stm:description>
</stm:unit>

<stm:unit id="meter" name="meter" unitType="length"
  abbreviation="m">
  <stm:description>The SI unit of length</stm:description>
</stm:unit>

<!-- ... -->

<stm:unit id="kg" name="nameless" unitType="dimensionless"
  abbreviation="nodim">
  <stm:description>A fictitious parent for dimensionless
units</stm:description>
</stm:unit>

<!--
============================================================================
-->
<!-- ====================== derived SI units
============================== -->
<!--
============================================================================
-->

<stm:unit id="newton" name="newton" unitType="force">
  <stm:description>The SI unit of force</stm:description>
</stm:unit>
```

```
<!-- ... -->

<!-- multiples of fundamental SI units -->

<stm:unit id="g" name="gram" unitType="mass"
  parentSI="kg"
  multiplierToSI="0.001"
  abbreviation="g">
  <stm:description>0.001 kg. </stm:description>
</stm:unit>

<stm:unit id="celsius" name="Celsius" parentSI="k"
  multiplierToSI="1"
  constantToSI="273.18">
  <stm:description><p>A common unit of
temperature</p></stm:description>
</stm:unit>

<!-- fundamental non-SI units -->

<stm:unit id="inch" name="inch" parentSI="meter"
   abbreviation="in"
  multiplierToSI="0.0254" >
  <stm:description>An imperial measure of
length</stm:description>
</stm:unit>

<!-- derived non-SI units -->

<stm:unit id="l" name="litre" unitType="volume"
   parentSI="meterCubed"
   abbreviation="l"
   multiplierToSI="0.001">
  <stm:description>Nearly 1 dm**3 This is not quite
exact</stm:description>
</stm:unit>

<!-- ... -->

<stm:unit id="fahr" name="fahrenheit" parentSI="k"
   abbreviation="F"
  multiplierToSI="0.55555555555555555"
  constantToSI="-17.777777777777777">
  <stm:description>An obsolescent unit of temperature still
used in popular
  meteorology</stm:description>
</stm:unit>

</stm:unitList>
```

[xsd:string]

MISC

`maxValue`[]

The maximum INCLUSIVE value of a quantity

The maximum INCLUSIVE value of a sortable quantity such as numeric, date or string. It should be ignored for dataTypes such as URL. The use of `min` and `max` attributes can be used to give a range for the quantity. The statistical basis of this range is not defined. The value of `max` is usually an observed quantity (or calculated from observations). To restrict a value, the `maxExclusive` type in a dictionary should be used.
The type of the maximum is the same as the quantity to which it refers - numeric, date and string are currently allowed

```
<scalar dataType="xsd:float" max="20" min="12">15</scalar>
```

[xsd:string]

`minValue`[]

The minimum INCLUSIVE value of a quantity

The minimum INCLUSIVE value of a sortable quantity such as numeric, date or string. It should be ignored for dataTypes such as URL. The use of `min` and `min` attributes can be used to give a range for the quantity. The statistical basis of this range is not defined. The value of `min` is usually an observed quantity (or calculated from observations). To restrict a value, the `minExclusive` type in a dictionary should be used.
The type of the minimum is the same as the quantity to which it refers - numeric, date and string are currently allowed

```
<scalar dataType="xsd:float" max="20" min="12">15</scalar>
```

[xsd:string]
xsd:group
dataGroupgp.dataGroup

A grouping of `list` and `scalar` elements for use by other schemas. Experimental.

This is to allow non-STM schemas to include a generic "data" component in their elements. It is messy and probably should not survive. Better extensibility mechanisms should be found.
Use only within Schemas

(list|scalar)

`foo`*[att.foo345]*