ANNOTATION

ELEMENTS

angle*[el.angle]*

A "bond" angle between three atoms.

It can be used for:

- Recording experimentally determined bond angles (e.g. in a crystallographic paper).
- Providing the angle component for internal coordinates (e.g. z-matrix).

Example

```
<molecule id="m1" title="angle example">
  <atomArray>
    <atom id="a1"></atom>
    <atom id="a2"></atom>
    <atom id="a3"></atom>
  </atomArray>
  <angle units="degrees" atomRefs3="a1 a2 a3">123.4</angle>
</molecule>
```

Content Model of element

nonNegativeAngleType[st.nonNegativeAngleType]

A non-signed angle, such as a bond angle. Note that we also provide positiveAngleType (e.g. for cell angles) and torsionAngleType for - guess what - **torsion**.

Re-used by **angle**

Example

```
<angle units="degrees" atomRefs3="a1 a2 a3">123.4</angle>
```

[xsd:float]
minInclusive: 0.0
maxInclusive: 180.0

---

title*[att.title]*

A title on an element.

No controlled value.

Example

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

id*[att.id]*

An attribute providing a unique ID for an element.

[link]

convention*[att.convention]*

A reference to a convention.

[link]

dictRef*[att.dictRef]*

A reference to a dictionary entry.

[link]

atomRefs3*[att.atomRefs3]*

A list of three references to atoms.

[link]

units*[att.angleUnits]*

Restricts units to radians or degrees.

[link]

errorValue*[att.errorValue]*

[link]

errorBasis*[att.errorBasis]*

[link]

min*[att.min]*

[link]

max*[att.max]*

[link]

ref*[att.ref]*

A reference to an element of given type.

[link]

arg *[el.arg]*

An argument for a funtion or similar beast.

Experimental. Arguments can be typed and have explicit or free values.

Example

```
<--   Li c O  c Buckingham 0.115E+04 0.280     .000E+00
.000E+00   0.000 10.000 --><potential form="gulp:buckingham">
   <arg name="atom1">
     <atom atomTypeRef="gulp:at.li"></atom>
   </arg>
   <arg name="atom2">
     <atom atomTypeRef="gulp:at.o"></atom>
   </arg>
   <arg ref="gulp:buckingham.a">
     <scalar units="units:ev">0.115E+04</scalar>
   </arg>
   <arg ref="gulp:buckingham.b">
     <scalar units="units:angstrom">0.280</scalar>
   </arg>
   <arg ref="gulp:buckingham.c">
     <scalar units="units:ev">.000E+00</scalar>
   </arg>
 </potential>
```

Content Model of element

(
(atom|scalar|array|matrix|expression)
*)

Attributes of element

title *[att.title]*

A title on an element.

No controlled value.

Example

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

id *[att.id]*

An attribute providing a unique ID for an element.

[link]

convention*[att.convention]*
A reference to a convention.
[link]

dictRef*[att.dictRef]*

A reference to a dictionary entry.
[link]

ref*[att.ref]*
A reference to an element of given type.
[link]

name*[]*

The name of the argument

dataType*[]*

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.
Example
```
<stmml title="namespace example">
<list>
  <scalar dictRef="chem:mpt">123</scalar>
  <scalar dictRef="mpt23">123</scalar>
</list>
</stmml>
```
[xsd:string]

The namespace prefix must start with an alpha character and can only contain alphanumeric and '_'. The suffix can have characters from the XML ID specification (alphanumeric, '_', '.' and '-'
Pattern: ([A-Za-z][A-Za-z0-9_]*:)?[A-Za-z][A-Za-z0-9_\.\-]*
[link]

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.

Example

```
<stmml title="array example 1">
<array size="5" title="value" dataType="xsd:decimal">  1.23
2.34 3.45 4.56 5.67</array>
  </stmml>

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

Example

```
<stmml title="array example 2">
<array size="5" title="initials" dataType="xsd:string"
delimiter="/">/A B//C/D-E/F/</array>
</stmml>

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

Example

```
<stmml title="array example 3">
<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>
</stmml>
```

Content Model of element

[xsd:string]

`title`*[att.title]*

A title on an element.

No controlled value.

Example

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

`id`*[att.id]*

An attribute providing a unique ID for an element.

[link]

`convention`*[att.convention]*

A reference to a convention.

[link]

`dictRef`*[att.dictRef]*

A reference to a dictionary entry.

[link]

`dataType`*[]*REQUIRED

The mandatory data type.

All elements of the array must have the same dataType.

[link]

`errorValues`*[]*

an optional array of error values for numeric arrays.

[link]

`errorBasis`*[att.errorBasis]*

[link]

`minValues`*[]*

an optional array of minimum values for numeric arrays.

[link]

`maxValues`*[]*

an optional array of maximum values for numeric arrays.
[link]

---

units*[att.units]*
Scientific units on an element.
[link]

---

delimiter*[att.delimiter]*
A delimiter character for arrays and matrices.
[link]

---

size*[att.size]*
The size of an array, matrix, list, etc.
[link]

---

ref*[att.ref]*
A reference to an element of given type.
[link]

atom*[el.atom]*

An atom.

Usually within a molecule. It is almost always contained within atomArray.
Example

```
<cml title="single atom example">
<atom id="a1" title="O3'" elementType="O" formalCharge="1"
hydrogenCount="1" isotope="17" occupancy="0.7" x2="1.2"
y2="2.3" x3="3.4" y3="4.5" z3="5.6" convention="ABC"
dictRef="chem:atom">
  <scalar title="dipole" dictRef="d:dip"
units="units:debye">0.2</scalar>
  <atomParity atomRefs4="a3 a7 a2 a4">1</atomParity>
  <electron id="e1" atomRef="a1" count="2"></electron>
</atom>
</cml>
```

Content Model of element

(
(

the electron children: One or more electrons associated with the atom. The **atomRef** on the electron should point to the id on the atom. We may relax this later and allow reference by context.
|name|array|matrix|scalar|atomParity|electron)

*|
(particle*)
)

Attributes of element

id*[att.id]*
An attribute providing a unique ID for an element.
[link]

count*[el.atom.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.
Example

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

[xsd:nonNegativeInteger]
The occurrence count of the atom.
[link]

elementType*[el.atom.elementType]*

Allowed `elementType` values.

The periodic table (up to element number 118. In addition the following strings are allowed:

- `Du`. ("dummy") This does not correspond to a "real" atom and can support a point in space or within a chemical graph.
- `R`. ("R-group") This indicates that an atom or group of atoms could be attached at this point.

Example

```
<cml title="elementType example">
  <atomArray>
    <atom id="a1" elementType="C"></atom>
    <atom id="a2" elementType="N"></atom>
    <atom id="a3" elementType="Pb"></atom>
    <atom id="a4" elementType="Dummy"></atom>
  </atomArray>
</cml>
```

UNION OF
Allowed values

- Ac
- Al
- Ag
- Am
- Ar
- As
- At
- Au
- B
- Ba
- Bh
- Bi
- Be
- Bk
- Br
- C
- Ca
- Cd
- Ce
- Cf
- Cl
- Cm
- Co
- Cr
- Cs
- Cu
- Db
- Dy
- Er
- Es
- Eu
- F
- Fe
- Fm
- Fr
- Ga
- Gd
- Ge
- H

  Any isotope of hydrogen.

  There are no special element symbols for D and T which should use the `isotope` attribute.

- He
- Hf
- Hg

- Ho
- Hs
- I
- In
- Ir
- K
- Kr
- La
- Li
- Lr
- Lu
- Md
- Mg
- Mn
- Mo
- Mt
- N
- Na
- Nb
- Nd
- Ne
- Ni
- No
- Np
- O
- Os
- P
- Pa
- Pb
- Pd
- Pm
- Po
- Pr
- Pt
- Pu
- Ra
- Rb
- Re
- Rf
- Rh
- Rn
- Ru
- S
- Sb
- Sc
- Se
- Sg
- Si
- Sm

- Sn
- Sr
- Ta
- Tb
- Tc
- Te
- Th
- Ti
- Tl
- Tm
- U
- Uun
- Uuu
- Uub
- Uut
- Uuq
- Uup
- Uuh
- Uus
- Uuo
- V
- W
- Xe
- Y
- Yb
- Zn
- Zr
- Dummy
- Du

  A point or object with no chemical semantics.

  Examples can be centroids, bond-midpoints, orienting "atoms" in small z-matrices.

  Note "Dummy" has the same semantics but is now deprecated.
- R

  A point at which an atom or group might be attached.

  Examples are abbreviated organic functional groups, Markush representations, polymers, unknown atoms, etc. Semantics may be determined by the `role` attribute on the atom.

[xsd:string]
Pattern: [A-Za-z]+:[A-Za-z][A-Za-z0-9\-]+
[link]

`formalCharge`*[el.atom.formalCharge]*

The formal charge on an atom.

Used for electron-bookeeping. This has no relation to its calculated (fractional) charge.

Example

```
<cml title="formalCharge example">
  <atomArray>
    <atom id="a1" elementType="N" formalCharge="+1"></atom>
    <atom id="a2" elementType="O" formalCharge="-1"></atom>
  </atomArray>
</cml>
```

[xsd:integer]

The formalCharge on the atom (in electrons).

[link]

`hydrogenCount` *[el.atom.hydrogenCount]*

The total number of hydrogen atoms bonded to an atom.

The total number of hydrogen atoms bonded to an atom, whether explicitly included as atoms or not. It is an error to have hydrogen count less than the explicit hydrogen count. There is no default value and no assumptions about hydrogen Count can be made if it is not given.

If hydrogenCount is given on every atom, then the values can be summed to give the total hydrogenCount for the (sub)molecule. Because of this hydrogenCount should not be used where hydrogen atoms bridge 2 or more atoms.

Example

```
<cml title="single atom example">
<atom id="a1" title="O3'" elementType="O" formalCharge="1"
hydrogenCount="1" isotope="17" occupancy="0.7" x2="1.2"
y2="2.3" x3="3.4" y3="4.5" z3="5.6" convention="ABC"
dictRef="chem:atom">
  <scalar title="dipole" dictRef="d:dip"
units="units:debye">0.2</scalar>
  <atomParity atomRefs4="a3 a7 a2 a4">1</atomParity>
  <electron id="e1" atomRef="a1" count="2"></electron>
</atom>
</cml>
```

[xsd:nonNegativeInteger]

[link]

`nonHydrogenCount` *[el.atom.nonHydrogenCount]*

The number of non-hydrogen atoms attached to an atom.

Obsolete in core CML. Only useful in CML queries

[xsd:nonNegativeInteger]

[link]

`isotope` *[el.atom.isotope]*

The numeric representation of an isotope.

In core CML this represents a single number; either the combined proton/neutron count or a more accurate estimate of the nuclear mass. This is admittedly fuzzy, and requires a more complex object (which can manage conventions, lists of isotopic masses, etc.) See **isotope**.

The default is "natural abundance" - whatever that can be interpreted as.

Delta values (i.e. deviations from the most abundant istopic mass) are never allowed.

[xsd:float]
minInclusive: 0.0
maxInclusive: 99999999999.0
[link]

`occupancy` *[el.atom.occupancy]*

Occupancy of an atomic site.

Primarily for crystallography. Values outside 0-1 are not allowed.

Example
See `atom`.

[xsd:float]
minInclusive: 0
maxInclusive: 1
[link]

`x2` *[el.atom.x2]*

The x coordinate (arbitrary units) of a 2-D representation (unrelated to 3-D structure). Note that x- and y- 2D coordinates are required for graphical stereochemistry such as wedge/hatch. x- and y- coordinates must be both present or both absent.

`x3` *[el.atom.x3]*

The x coordinate (in Angstrom units) of a 3-D cartesian representation. x3 y3 and z3 coordinates must be both present or both absent.

`xFract` *[el.atom.xFract]*

The fractional x coordinate in a crystal structure. xFract, yFract and zFract coordinates must be all present or all absent. A `crystal` element is required

xy2*[el.atom.xy2]*

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
Example

```
<stmml title="coordinate2Type example">
<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>
</stmml>
```

[xsd:string]
Pattern: \s*([\-]|[+])?\d*\.?\d*(\s+|[,])([\-]|[+])?\d*\.?\d*\s*
[link]

---

xyz3*[el.atom.xyz3]*

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
Example

```
<stmml title="coordinate3Type example">
<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>
</stmml>
```

[xsd:string]
Pattern: \s*([\-]|[+])?\d*\.?\d*(\s+|[,])([\-]|[+])?\d*\.?\d*(\s+|[,])([\-]|[+])?\d*\.?\d*\s*
[link]

---

xyzFract*[el.atom.xyzFract]*

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
Example

```
<stmml title="coordinate3Type example">
<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>
</stmml>
```

[xsd:string]
Pattern: \s*([\-]|[+])?\d*\.?\d*(\s+|[,])([\-]|[+])?\d*\.?\d*(\s+|[,])([\-]|[+])?\d*\.?\d*\s*
[link]

y2*[el.atom.y2]*

The y coordinate (arbitrary units) of a 2-D representation (unrelated to 3-D structure). Note that x2 and y2 coordinates are required for graphical stereochemistry such as wedge/hatch. x2 and y2 coordinates must be both present or both absent.

y3*[el.atom.y3]*

The y coordinate (in Angstrom units) of a 3-D cartesian representation. x3 y3 and z3 coordinates must be both present or both absent.

yFract*[el.atom.yFract]*

The fractional x coordinate in a crystal structure. xFract, yFract and zFract coordinates must be all present or all absent. A `crystal` element is required

z3*[el.atom.z3]*

The z coordinate (in Angstrom units) of a 3-D cartesian representation. x3 y3 and z3 coordinates must be both present or both absent.

zFract*[el.atom.zFract]*

The fractional x coordinate in a crystal structure. xFract, yFract and zFract coordinates must be all present or all absent. A `crystal` element is required

title*[att.title]*

A title on an element.

No controlled value.

Example

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

convention*[att.convention]*

A reference to a convention.
[link]

dictRef *[att.dictRef]*

A reference to a dictionary entry.
[link]

ref *[att.ref]*
A reference to an element of given type.
[link]

role *[]*

This can be used to describe the purpose of atoms whose `elementType`s are `dummy` or `locant`.

atomTypeRef *[atom.cmlcomp.att.atomTypeRef]*
a reference to the atomType for the element.

dx3 *[atom.cmlcomp.att.dx3]*

displacement along the Cartesian X axis.

This is a CCML extension to core CML. It represents a change in the Cartesian X coordinate (e.g. for vibrational modes, molecular dynamics, etc.). Whether dx3 can be added to an `x3` value depends on the semantics of the application.

dy3 *[atom.cmlcomp.att.dy3]*

displacement along the Cartesian Y axis.

This is a CCML extension to core CML. It represents a change in the Cartesian Y coordinate (e.g. for vibrational modes, molecular dynamics, etc.). See `dx3`.

dz3 *[atom.cmlcomp.att.dz3]*

displacement along the Cartesian Z axis.

This is a CCML extension to core CML. It represents a change in the Cartesian Z coordinate (e.g. for vibrational modes, molecular dynamics, etc.). See `dx3`.

dxyz3 *[atom.cmlcomp.att.dxyz3]*

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
Example

```
<stmml title="coordinate3Type example">
<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>
</stmml>
```

[xsd:string]
Pattern: \s*([\-]|[+])?\d*\.?\d*(\s+|[,])([\-]|[+])?\d*\.?\d*(\s+|[,])([\-]|[+])?\d*\.?\d*\s*
displacement along the Cartesian X, Y, Z axes.
[link]

vx*[atom.cmlcomp.att.vx]*

velocity along the Cartesian X axis.

This is a CCML extension to core CML.

Units MUST be given in velocityUnits in the grandparent molecule; there are NO defaults.

vy*[atom.cmlcomp.att.vy]*

velocity along the Cartesian Y axis.

This is a CCML extension to core CML.

Units MUST be given in velocityUnits in the grandparent molecule; there are NO defaults.

vz*[atom.cmlcomp.att.vz]*

velocity along the Cartesian Z axis.

This is a CCML extension to core CML.

Units MUST be given in velocityUnits in the grandparent molecule; there are NO defaults.

vxyz*[atom.cmlcomp.att.vxyz]*

A vector in 3-space

No constraints on magnitude (i.e. could be zero)
Example
```
<myVector dataType="stm:vector3Type">2.0 3.0 4.0</myVector>
```
[xsd:string]
Pattern: \s*([\-]|[+])?\d*\.?\d*(\s+|[,])([\-]|[+])?\d*\.?\d*(\s+|[,])([\-]|[+])?\d*\.?\d*\s*
velocity as a vector
[link]

atomArray *[el.atomArray]*

A container for a list of atoms.

A child of **molecule** and contains `atom` information. There are two strategies:

- Create individual **atom** elements under **atomArray** (in any order). This gives the greatest flexibility but is the most verbose.
- Create `*Array` attributes (e.g. of **elementTypeArrayType** under **atomArray**. This requires all arrays to be of identical lengths with explicit values for all atoms in every array. This is NOT suitable for complexType atom children such as **atomParity** or composite types such as `xy2`. It also cannot be checked as easily by schema- and schematron validation. The `atomIDArray` attribute is mandatory. It is allowed (though not yet recommended) to add `*Array` children such as **floatArray**

The attributes are directly related to the scalar attributes under **atom** which should be consulted for more info.

NOTE: The CML-1 specifications are also supported but are deprecated
.
Example
```
<cml title="atomArray CML1">
<list>
  <atomArray>
    <atom id="a1" elementType="O" hydrogenCount="1"></atom>
     <atom id="a2" elementType="N" hydrogenCount="1"></atom>
    <atom id="a3" elementType="C" hydrogenCount="3"></atom>
  </atomArray>
  <atomArray atomID="a1 a2 a3" elementType="O N C"
hydrogenCount="1 1 3"></atomArray>
</list>
</cml>


Example - these are exactly equivalent representations
```

Content Model of element

(atom+|array*)

Attributes of element

`title`*[att.title]*

A title on an element.

No controlled value.

Example

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

`id`*[att.id]*

An attribute providing a unique ID for an element.

[link]

`convention`*[att.convention]*

A reference to a convention.

[link]

`dictRef`*[att.dictRef]*

A reference to a dictionary entry.

[link]

`elementType`*[]*

An array of elementTypes.

Instances of this type will be used in array-style representation of atoms.

Example

```
<cml title="atomArray with elementTypes">
  <atomArray elementType="O N S Pb"></atomArray>
</cml>
```

XSD:LIST of elementTypeType

Allowed `elementType` values.

The periodic table (up to element number 118. In addition the following strings are allowed:

- `Du`. ("dummy") This does not correspond to a "real" atom and can support a point in space or within a chemical graph.
- `R`. ("R-group") This indicates that an atom or group of atoms could be attached at this point.

Example

```
<cml title="elementType example">
  <atomArray>
    <atom id="a1" elementType="C"></atom>
    <atom id="a2" elementType="N"></atom>
    <atom id="a3" elementType="Pb"></atom>
    <atom id="a4" elementType="Dummy"></atom>
  </atomArray>
</cml>
```

UNION OF

Allowed values

- Ac
- Al
- Ag
- Am
- Ar
- As
- At
- Au
- B
- Ba
- Bh
- Bi
- Be
- Bk
- Br
- C
- Ca
- Cd
- Ce
- Cf
- Cl
- Cm
- Co
- Cr
- Cs
- Cu
- Db
- Dy
- Er
- Es
- Eu
- F
- Fe
- Fm
- Fr
- Ga
- Gd
- Ge
- H

Any isotope of hydrogen.

There are no special element symbols for D and T which should use the `isotope` attribute.

- He
- Hf
- Hg
- Ho
- Hs
- I
- In
- Ir
- K
- Kr
- La
- Li
- Lr
- Lu
- Md
- Mg
- Mn
- Mo
- Mt
- N
- Na
- Nb
- Nd
- Ne
- Ni
- No
- Np
- O
- Os
- P
- Pa
- Pb
- Pd
- Pm
- Po
- Pr
- Pt
- Pu
- Ra
- Rb
- Re
- Rf
- Rh
- Rn

- Ru
- S
- Sb
- Sc
- Se
- Sg
- Si
- Sm
- Sn
- Sr
- Ta
- Tb
- Tc
- Te
- Th
- Ti
- Tl
- Tm
- U
- Uun
- Uuu
- Uub
- Uut
- Uuq
- Uup
- Uuh
- Uus
- Uuo
- V
- W
- Xe
- Y
- Yb
- Zn
- Zr
- Dummy
- Du

  A point or object with no chemical semantics.

  Examples can be centroids, bond-midpoints, orienting "atoms" in small z-matrices.

  Note "Dummy" has the same semantics but is now deprecated.
- R

  A point at which an atom or group might be attached.

  Examples are abbreviated organic functional groups, Markush representations,

polymers, unknown atoms, etc. Semantics may be determined by the `role` attribute on the atom.

[xsd:string]
Pattern: [A-Za-z]+:[A-Za-z][A-Za-z0-9\-]+
Almost mandatory. see **elementType**
[link]

---

count[]

An array of integers.

An array of integers; for re-use by other schemas

Not machine-validatable
Example

```
<stmml title="integerArray type">
<atomArray hydrogenCount="3 1 0 2"></atomArray>
</stmml>
```

XSD:LIST of xsd:integer
See **count**
[link]

---

formalCharge[]

An array of integers.

An array of integers; for re-use by other schemas

Not machine-validatable
Example

```
<stmml title="integerArray type">
<atomArray hydrogenCount="3 1 0 2"></atomArray>
</stmml>
```

XSD:LIST of xsd:integer
See **formalCharge**
[link]

---

hydrogenCount[]

An array of integers.

An array of integers; for re-use by other schemas

Not machine-validatable
Example

```
<stmml title="integerArray type">
<atomArray hydrogenCount="3 1 0 2"></atomArray>
```

```
</stmml>
```
XSD:LIST of xsd:integer

See **hydrogenCount**

[link]

---

nonHydrogenCount[]

An array of integers.

An array of integers; for re-use by other schemas

Not machine-validatable

Example

```
<stmml title="integerArray type">
<atomArray hydrogenCount="3 1 0 2"></atomArray>
</stmml>
```
XSD:LIST of xsd:integer

See **nonHydrogenCount**

[link]

---

isotope[]

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.

Example

```
<atomArray x2="1.2 2.3 3.4 5.6"></atomArray>
```
XSD:LIST of xsd:float

See **isotope**

[link]

---

occupancy[]

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.

Example

```
<atomArray x2="1.2 2.3 3.4 5.6"></atomArray>
```
XSD:LIST of xsd:float

See **occupancy**

[link]

---

x2[]

An array of coordinateComponents for a single coordinate.

An array of coordinateComponents for a single coordinate where these all refer to an X-coordinate (NOT x,y,z) Instances of this type will be used in array-style representation of 2-D or 3-D coordinates.

Currently no machine validation

Currently not used in STMML, but re-used by CML (see example)

Example

```
<stmml title="coordinateComponentArrayType">
<cml:atomArray x2="1.2 2.3 4.5 6.7"></cml:atomArray>
</stmml>
```

XSD:LIST of xsd:float

See **x2**

[link]

x3[]

An array of coordinateComponents for a single coordinate.

An array of coordinateComponents for a single coordinate where these all refer to an X-coordinate (NOT x,y,z) Instances of this type will be used in array-style representation of 2-D or 3-D coordinates.

Currently no machine validation

Currently not used in STMML, but re-used by CML (see example)

Example

```
<stmml title="coordinateComponentArrayType">
<cml:atomArray x2="1.2 2.3 4.5 6.7"></cml:atomArray>
</stmml>
```

XSD:LIST of xsd:float

See **x3**

[link]

xFract[]

An array of coordinateComponents for a single coordinate.

An array of coordinateComponents for a single coordinate where these all refer to an X-coordinate (NOT x,y,z) Instances of this type will be used in array-style representation of 2-D or 3-D coordinates.

Currently no machine validation

Currently not used in STMML, but re-used by CML (see example)

Example

```
<stmml title="coordinateComponentArrayType">
<cml:atomArray x2="1.2 2.3 4.5 6.7"></cml:atomArray>
```

```
</stmml>
```

XSD:LIST of xsd:float

See **xFract**

[link]

y2[]

An array of coordinateComponents for a single coordinate.

An array of coordinateComponents for a single coordinate where these all refer to an X-coordinate (NOT x,y,z) Instances of this type will be used in array-style representation of 2-D or 3-D coordinates.

Currently no machine validation

Currently not used in STMML, but re-used by CML (see example)

Example

```
<stmml title="coordinateComponentArrayType">
<cml:atomArray x2="1.2 2.3 4.5 6.7"></cml:atomArray>
</stmml>
```

XSD:LIST of xsd:float

See **y2**

[link]

y3[]

An array of coordinateComponents for a single coordinate.

An array of coordinateComponents for a single coordinate where these all refer to an X-coordinate (NOT x,y,z) Instances of this type will be used in array-style representation of 2-D or 3-D coordinates.

Currently no machine validation

Currently not used in STMML, but re-used by CML (see example)

Example

```
<stmml title="coordinateComponentArrayType">
<cml:atomArray x2="1.2 2.3 4.5 6.7"></cml:atomArray>
</stmml>
```

XSD:LIST of xsd:float

See **y3**

[link]

yFract[]

An array of coordinateComponents for a single coordinate.

An array of coordinateComponents for a single coordinate where these all refer to an

X-coordinate (NOT x,y,z) Instances of this type will be used in array-style representation of 2-D or 3-D coordinates.

Currently no machine validation

Currently not used in STMML, but re-used by CML (see example)
Example

```
<stmml title="coordinateComponentArrayType">
<cml:atomArray x2="1.2 2.3 4.5 6.7"></cml:atomArray>
</stmml>
```

XSD:LIST of xsd:float

See **yFract**

[link]

---

z3[]

An array of coordinateComponents for a single coordinate.

An array of coordinateComponents for a single coordinate where these all refer to an X-coordinate (NOT x,y,z) Instances of this type will be used in array-style representation of 2-D or 3-D coordinates.

Currently no machine validation

Currently not used in STMML, but re-used by CML (see example)
Example

```
<stmml title="coordinateComponentArrayType">
<cml:atomArray x2="1.2 2.3 4.5 6.7"></cml:atomArray>
</stmml>
```

XSD:LIST of xsd:float

See **z3**

[link]

---

zFract[]

An array of coordinateComponents for a single coordinate.

An array of coordinateComponents for a single coordinate where these all refer to an X-coordinate (NOT x,y,z) Instances of this type will be used in array-style representation of 2-D or 3-D coordinates.

Currently no machine validation

Currently not used in STMML, but re-used by CML (see example)
Example

```
<stmml title="coordinateComponentArrayType">
<cml:atomArray x2="1.2 2.3 4.5 6.7"></cml:atomArray>
</stmml>
```

XSD:LIST of xsd:float
See **zFract**
[link]

`atomID`*[]*

An array of atomRefs.

The atomRefs cannot be schema- or schematron-validated. Instances of this type will be used in array-style representation of bonds and atomParitys. It can also be used for arrays of atomIDTypes such as in complex setereochemistry, geometrical definitions, atom groupings, etc.

Example

```
<cml title="atomArray example">
  <molecule id="m1">
    <atomArray atomID="a2 a4 a6" elementType="O N
S"></atomArray>
  </molecule>
</cml>
```

XSD:LIST of atomRefType

A reference to an existing atom.

Example

```
<cml title="atomRef example">
  <molecule id="m1">
    <atomArray>
      <atom id="a1"></atom>
    </atomArray>
    <electron id="e1" atomRef="a1"></electron>
  </molecule>
</cml>
```

[xsd:string]
Pattern: \s*[A-Za-z_][A-Za-z0-9\-:_]*\s*
See **atomID**
[link]

`ref`*[att.ref]*

A reference to an element of given type.
[link]

`atomicBasisFunction`*[el.atomicBasisFunction]*

An atomicBasisFunction.

An atomic atomicBasisFunction which can be linked to atoms, eigenvalues/vectors etc. Normally contained within `basisSet`

Normally these are atom-centered functions, but they can also serve as "ghost" functions which are centered on points. IN CCML these can be dummy atoms so that the atomRef mechanism can still be used.

This information is required to interpret the eignevector components and map them onto the atom list. However this mapping is normally implicit in the program and so it may be necessary to generate `basisSet` information for some programs before XML technology can be automatically used to link the components of the CCML document.

Example

```
This is a 6-311(d) and has 18 components including unsplit d.
```

Content Model of element

(
()
*,
(gradient)
?)

Attributes of element

`atomRef` *[att.atomicBasisFunction.ref]*

A reference to an existing atom.

Example

```
<cml title="atomRef example">
  <molecule id="m1">
    <atomArray>
      <atom id="a1"></atom>
    </atomArray>
    <electron id="e1" atomRef="a1"></electron>
  </molecule>
</cml>
```

[xsd:string]
Pattern: \s*[A-Za-z_][A-Za-z0-9\-:_]*\s*

The atom owning this atomicBasisFunction

[link]

`title` *[att.title]*

A title on an element.

No controlled value.

Example

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

id *[att.id]*

An attribute providing a unique ID for an element.

[link]

convention *[att.convention]*

A reference to a convention.

[link]

dictRef *[att.dictRef]*

A reference to a dictionary entry.

[link]

n *[att.atomicBasisFunction.n]*

Principal quantum number.

1, 2, 3, etc.

l *[att.atomicBasisFunction.l]*

Secondary quantum number.

0, 1, etc.

m *[att.atomicBasisFunction.m]*

Azimuthal quantum number.

-1, 0, 1, etc.

lm *[att.atomicBasisFunction.lm]*

symbolic represention of l amd m.

atomParity *[el.atomParity]*

The stereochemistry round an atom centre.

It follows the convention of the MIF format, and uses 4 distinct atoms to define the chirality. These can be any atoms (though they are normally bonded to the current atom). There is no default order and the order is defined by the atoms in the atomRefs4 attribute. If there are only 3 ligands, the current atom should be included in the 4 atomRefs.

The value of the parity is a signed number. (It can only be zero if two or more atoms are coincident or the configuration is planar). The sign is the sign of the chiral volume created by the four atoms (a1, a2, a3, a4):

```
|  1  1  1  1 |
| x1 x2 x3 x4 |
| y1 y2 y3 y4 |
| z1 z2 z3 z4 |
```

Note that `atomParity` cannot be used with the *Array syntax for atoms.

Example

```
<cml title="atom parity example">
  <atom id="a1">
    <atomParity atomRefs4="a3 a5 a2 a9">1</atomParity>
  </atom>
</cml>
```

Content Model of element

[xsd:float]

`title`*[att.title]*

A title on an element.

No controlled value.

Example

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

`id`*[att.id]*

An attribute providing a unique ID for an element.

[link]

`convention`*[att.convention]*

A reference to a convention.

[link]

dictRef*[att.dictRef]*

A reference to a dictionary entry.
[link]

---

atomRefs4*[att.atomRefs4]*
A list of 4 references to atoms.
[link]

atomSet*[el.atomSet]*

A set of references to atoms.

An atomSet consists of a number of unique references to atoms throught their ids. atomSets need not be related to molecules (which are generally created by aggregation of explicit atoms). Two or more atomSets may reference the same atom, and atomSets may be empty.

atomSets have many potential uses such as:
- identifying functional groups
- results of substructure matching
- identifying atoms with particular roles in a calculation

The atomSet may be referenced from elsewhere in the document and you are encouraged to use locally unique id attributes on atomSets.
Example

```
<cml>
<atomSet id="as1" title="Group 7 elements">a2 a3 a4</atomSet>
<atomSet id="as2" title="first row elements">a2 a3
a4</atomSet>
<molecule>
  <atomArray>
    <atom atomTypeRef="a1" elementType="C"></atom>
    <atom atomTypeRef="a2" elementType="Br"></atom>
    <atom atomTypeRef="a3" elementType="Cl"></atom>
    <atom atomTypeRef="a4" elementType="F"></atom>
    <atom atomTypeRef="a5" elementType="C"
hydrogenCount="3"></atom>
  </atomArray>
</molecule>
</cml>
```

Content Model of element

atomRefArrayType[st.atomRefArrayType]

An array of atomRefs.

The atomRefs cannot be schema- or schematron-validated. Instances of this type will be used in array-style representation of bonds and atomParitys. It can also be used for arrays of atomIDTypes such as in complex setereochemistry, geometrical definitions, atom groupings, etc.

Example

```
<cml title="atomArray example">
  <molecule id="m1">
    <atomArray atomID="a2 a4 a6" elementType="O N
S"></atomArray>
  </molecule>
</cml>
```

XSD:LIST of atomRefType

A reference to an existing atom.

Example

```
<cml title="atomRef example">
  <molecule id="m1">
    <atomArray>
      <atom id="a1"></atom>
    </atomArray>
    <electron id="e1" atomRef="a1"></electron>
  </molecule>
</cml>
```

[xsd:string]

Pattern: \s*[A-Za-z_][A-Za-z0-9\-:_]*\s*

`title` *[att.title]*

A title on an element.

No controlled value.

Example

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

`id` *[att.id]*

An attribute providing a unique ID for an element.

[link]

`convention` *[att.convention]*

A reference to a convention.

[link]

`dictRef` *[att.dictRef]*

A reference to a dictionary entry.

[link]

size*[att.size]*
The size of an array, matrix, list, etc.
[link]

atomType*[el.atomType]*

An atomType.

atomTypes are used in a wide variety of ways in computational chemistry. They are normally labels added to existing atoms (or dummy atoms) in the molecule and have a number of defined properties. These properties are usually in addition to those deducible from the elementType of the atom. AtomTypes usually depend on the chemical or geometrical environment of the atom and are frequently assigned by algorithms with chemical perception. However they are aoften frequently set or "tweaked" by humans initiating a program run.

AtomTypes on an atom have no formal relation to its `elementType`, which only describe the number of protons in the nucleus. It is not unknown (though potentially misleading) to use an "incompatible" atomType to alter the computational properties of an atom (e.g. pretend this K+ is a Ca++ to increase its effective charge). `atomTypes` will also be required to describe pseudoAtoms such as "halogen" (generic) or "methyl group" (unified atom). Atoms in computations can therefore have an `atomTypeRef` attribute.

An atomType contains numeric or other quantities associated with it (charges, masses, use in force-fields, etc.) and also description of any perception algorithms (chemical and/or geometrical) which could be used to compute or constrain it. This is still experimental.

atomTypes are referred to by their mandatory `name` attribute
Example

```
<--  define an atomType for Silicon is a forcefield
calculation  --><cml>
<atomType name="foo:sia" convention="foo" title="Silicon type
A">
  <atom elementType="Si">

    <scalar dictRef="foo:ffcharge">1.7</scalar>
  </atom>
</atomType>
<molecule>
  <atomArray>
    <atom atomTypeRef="foo:sia" x3="0.2" y3="0.2" z3="0.3"
title="silicon defect"></atom>
  </atomArray>
</molecule>
</cml>
```

Content Model of element

(
(molecule|atom)
*,
(scalar|array|matrix|property)
*)

Attributes of element

name*[att.atomType.name]*

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.

Example

```
<stmml title="namespace example">
<list>
  <scalar dictRef="chem:mpt">123</scalar>
  <scalar dictRef="mpt23">123</scalar>
</list>
</stmml>
```

[xsd:string]

The namespace prefix must start with an alpha character and can only contain alphanumeric and '_'. The suffix can have characters from the XML ID specification (alphanumeric, '_', '.' and '-'

Pattern: ([A-Za-z][A-Za-z0-9_]*:)?[A-Za-z][A-Za-z0-9_\.\-]*

A unique name for the atomType

[link]

ref*[att.atomType.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-z][A-Za-z0-9_-]*:)?[A-Za-z][A-Za-z0-9_\-\.]*

foo
[link]

`title` *[att.title]*

A title on an element.

No controlled value.

Example

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

`id` *[att.id]*

An attribute providing a unique ID for an element.
[link]

`convention` *[att.convention]*

A reference to a convention.
[link]

`dictRef` *[att.dictRef]*

A reference to a dictionary entry.
[link]

`atomTypeList` *[el.atomTypeList]*

A container for one or more atomTypes.

atomTypeList can contain several atomTypes.

Example

```
<atomTypeList>
  <atomType name="a:o"><atom
elementType="O"></atom></atomType>
  <atomType name="a:n"><atom
elementType="N"></atom></atomType>
</atomTypeList>
```

Content Model of element

(metadataList*,name*,
(atomType)
*)

Attributes of element

dictRef*[att.dictRef]*

A reference to a dictionary entry.

[link]

convention*[att.convention]*

A reference to a convention.

[link]

title*[att.title]*

A title on an element.

No controlled value.

Example

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

id*[att.id]*

An attribute providing a unique ID for an element.

[link]

ref*[att.ref]*

A reference to an element of given type.

[link]

basisSet*[el.basisSet]*

A container for one or more atomicBasisFunctions.

`basisSet` can contain several orbitals.

Example

```
<basisSet>
  <name>631-G*</name>
  <atomicBasisFunction></atomicBasisFunction>
</basisSet>
```

Content Model of element

(metadataList*,name*,
(atomicBasisFunction)
*)

Attributes of element

`dictRef` *[att.dictRef]*

A reference to a dictionary entry.

[link]

`convention` *[att.convention]*

A reference to a convention.

[link]

`title` *[att.title]*

A title on an element.

No controlled value.

Example

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

`id` *[att.id]*

An attribute providing a unique ID for an element.

[link]

`ref` *[att.ref]*

A reference to an element of given type.

[link]

`role` *[]*

The role of the basisSet. Semantics are not yet controlled

bond*[el.bond]*

A bond between atoms, or between atoms and bonds.

`bond` is a child of `bondArray` and contains bond information. Bond must refer to at least two atoms (using **atomRefs2**) but may also refer to more for multicentre bonds. Bond is often EMPTY but may contain **electron**, **length** or **bondStereo** elements.

Example

```
<cml title="bondArray example">
  <bondArray>
    <bond id="b1" atomRefs2="a3 a8" order="D">
      <electron bondRef="b1"></electron>
      <bondStereo>C</bondStereo>
    </bond>
    <bond id="b2" atomRefs2="a3 a8" order="S">
      <bondStereo convention="MDL"
conventionValue="6"></bondStereo>
    </bond>
  </bondArray>
</cml>
```

Example

```
<cml title="metal-bond example">
  <atomArray>
    <atom id="pt1" elementType="Pt"></atom>
    <atom id="cl1" elementType="Cl"></atom>
    <atom id="cl2" elementType="Cl"></atom>
    <atom id="cl3" elementType="Cl"></atom>
    <atom id="c1" elementType="C" hydrogenCount="2"></atom>
    <atom id="c2" elementType="C" hydrogenCount="2"></atom>
  </atomArray>
  <bondArray>
    <bond id="b1" atomRefs2="c1 c2" order="D"></bond>
    <bond id="b2" atomRefs2="pt1 cl1" order="S"></bond>
    <bond id="b3" atomRefs2="pt1 cl2" order="S"></bond>
    <bond id="b4" atomRefs2="pt1 cl3" order="S"></bond>
    <bond id="b5" atomRefs="pt1" bondRefs="b1"></bond>
  </bondArray>
</cml>
```

Validation

Content Model of element

(
(electron|bondStereo|length)
*)

Attributes of element

`title`*[att.title]*

A title on an element.

No controlled value.

Example

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

`id`*[att.id]*

An attribute providing a unique ID for an element.

[link]

`convention`*[att.convention]*

A reference to a convention.

[link]

`dictRef`*[att.dictRef]*

A reference to a dictionary entry.

[link]

`ref`*[att.ref]*

A reference to an element of given type.

[link]

`atomRefs2`*[]*

A reference to two distinct existing atoms in order.

Example

```
<cml title="atomRefs2 example">
  <molecule id="m1">
    <atomArray>
      <atom id="a1"></atom>
      <atom id="a2"></atom>
    </atomArray>
    <bondArray>
      <bond atomRefs2="a1 a2"></bond>
    </bondArray>
  </molecule>
</cml>
```

[xsd:string]

Pattern: \s*\S+\s+\S+\s*

The two atoms in the bond.
[link]

atomRefs[]

An array of atomRefs.

The atomRefs cannot be schema- or schematron-validated. Instances of this type will be used in array-style representation of bonds and atomParitys. It can also be used for arrays of atomIDTypes such as in complex setereochemistry, geometrical definitions, atom groupings, etc.

Example

```
<cml title="atomArray example">
  <molecule id="m1">
    <atomArray atomID="a2 a4 a6" elementType="O N
S"></atomArray>
  </molecule>
</cml>
```

XSD:LIST of atomRefType

A reference to an existing atom.

Example

```
<cml title="atomRef example">
  <molecule id="m1">
    <atomArray>
      <atom id="a1"></atom>
    </atomArray>
    <electron id="e1" atomRef="a1"></electron>
  </molecule>
</cml>
```

[xsd:string]
Pattern: \s*[A-Za-z_][A-Za-z0-9\-:_]*\s*
The atoms in the bond.
[link]

bondRefs[]

An array of references to bonds.

The references cannot (yet) cannot be schema- or schematron-validated. Instances of this type will be used in array-style representation of electron counts, etc. It can also be used for arrays of bondIDTypes such as in complex stereochemistry, geometrical definitions, bond groupings, etc.

Example

XSD:LIST of bondRefType

A reference to an existing bond.

A reference to a bond may be made by atoms (e.g. for multicentre or pi-bonds), electrons (for annotating reactions or describing electronic properties) or possibly other bonds (no examples yet). The semantics are relatively flexible.

Example

```
<cml title="bondArray example">
  <bondArray>
    <bond id="b1" atomRefs2="a3 a8" order="D">
      <electron bondRef="b1"></electron>
      <bondStereo>C</bondStereo>
    </bond>
    <bond id="b2" atomRefs2="a3 a8" order="S">
      <bondStereo convention="MDL"
conventionValue="6"></bondStereo>
    </bond>
  </bondArray>
</cml>
```

[xsd:string]

Pattern: \s*\S+\s*

Bonds involved in the bond.

[link]

order[]

Bond order (as a string).

This is purely conventional and used for bond/electron counting. There is no default value. The emptyString attribute can be used to indicate a bond of unknown or unspecified type. The interpretation of this is outside the scope of CML-based algorithms. It may be accompanied by a `convention` attribute on the `bond` which links to a dictionary. Example: `<bond convention="ccdc:9" atomRefs2="a1 a2"/>` could represent a delocalised bond in the CCDC convention.

Allowed values

- S

  Single bond.
- 1

  Single bond.
- D

  Double bond.
- 2

  Double bond.
- T

Triple bond.

- 3

Triple bond.

- A

Aromatic bond.

The order of the bond.

[link]

bondArray *[el.bondArray]*

A container for a number of bonds.

`bondArray` is a child of `molecule` and contains `bond` information. There are two strategies:

- Create individual `bond` elements under `bondArray` (in any order). This gives the greatest flexibility but is the most verbose.
- Create `*Array` attributes (e.g. of `orderArrayType` under `bondArray`. This requires all arrays to be of identical lengths with explicit values for all bonds in every array. This is NOT suitable for complexType bond children such as **`bondStereo`** , nor can IDs be added to bonds.. It also cannot be checked as easily by schema- and schematron validation. The `atomRef1Array` and `atomRef2Array` attributes are then mandatory. It is allowed (though not yet recommended) to add `*Array` children such as `floatArray`

The attributes are directly related to the scalar attributes under **`atom`** which should be consulted for more info.

Example

```
<cml title="bondArray example 1">
  <list>
    <bondArray>
      <bond id="b1" atomRefs2="a1 a2" order="1"></bond>
      <bond id="b2" atomRefs2="a1 a3" order="2"></bond>
      <bond id="b3" atomRefs2="a3 a5" order="1"></bond>
    </bondArray>
    <bondArray atomRef1="a1 a1 a3" atomRef2="a2 a3 a5"
order="1 2 1"></bondArray>
  </list>
</cml>


Example - these are exactly equivalent representations
```

Content Model of element

(bond+|array*)

Attributes of element

`title`*[att.title]*

A title on an element.

No controlled value.

Example

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

`id`*[att.id]*

An attribute providing a unique ID for an element.

[link]

`convention`*[att.convention]*

A reference to a convention.

[link]

`dictRef`*[att.dictRef]*

A reference to a dictionary entry.

[link]

`bondID`*[]*

An array of references to bonds.

The references cannot (yet) cannot be schema- or schematron-validated. Instances of this type will be used in array-style representation of electron counts, etc. It can also be used for arrays of bondIDTypes such as in complex stereochemistry, geometrical definitions, bond groupings, etc.

Example

XSD:LIST of bondRefType

A reference to an existing bond.

A reference to a bond may be made by atoms (e.g. for multicentre or pi-bonds), electrons (for annotating reactions or describing electronic properties) or possibly other bonds (no examples yet). The semantics are relatively flexible.

Example

```
<cml title="bondArray example">
  <bondArray>
    <bond id="b1" atomRefs2="a3 a8" order="D">
      <electron bondRef="b1"></electron>
      <bondStereo>C</bondStereo>
    </bond>
    <bond id="b2" atomRefs2="a3 a8" order="S">
      <bondStereo convention="MDL"
conventionValue="6"></bondStereo>
    </bond>
  </bondArray>
</cml>
```

[xsd:string]
Pattern: \s*\S+\s*
[link]

atomRef1[]

An array of atomRefs.

The atomRefs cannot be schema- or schematron-validated. Instances of this type will be used in array-style representation of bonds and atomParitys. It can also be used for arrays of atomIDTypes such as in complex setereochemistry, geometrical definitions, atom groupings, etc.

Example

```
<cml title="atomArray example">
  <molecule id="m1">
    <atomArray atomID="a2 a4 a6" elementType="O N
S"></atomArray>
  </molecule>
</cml>
```

XSD:LIST of atomRefType

A reference to an existing atom.

Example

```
<cml title="atomRef example">
  <molecule id="m1">
    <atomArray>
      <atom id="a1"></atom>
    </atomArray>
    <electron id="e1" atomRef="a1"></electron>
  </molecule>
</cml>
```

[xsd:string]
Pattern: \s*[A-Za-z_][A-Za-z0-9\-:_]*\s*
[link]

atomRef2[]

An array of atomRefs.

The atomRefs cannot be schema- or schematron-validated. Instances of this type will be used in array-style representation of bonds and atomParitys. It can also be used for arrays of atomIDTypes such as in complex setereochemistry, geometrical definitions, atom groupings, etc.

Example

```
<cml title="atomArray example">
  <molecule id="m1">
    <atomArray atomID="a2 a4 a6" elementType="O N
S"></atomArray>
  </molecule>
</cml>
```

XSD:LIST of atomRefType

A reference to an existing atom.

Example

```
<cml title="atomRef example">
  <molecule id="m1">
    <atomArray>
      <atom id="a1"></atom>
    </atomArray>
    <electron id="e1" atomRef="a1"></electron>
  </molecule>
</cml>
```

[xsd:string]

Pattern: \s*[A-Za-z_][A-Za-z0-9\-:_]*\s*

[link]

---

order[]

An array of bond orders.

(seeAlso **orderType**)

XSD:LIST of orderType

Bond order (as a string).

This is purely conventional and used for bond/electron counting. There is no default value. The emptyString attribute can be used to indicate a bond of unknown or unspecified type. The interpretation of this is outside the scope of CML-based algorithms. It may be accompanied by a `convention` attribute on the `bond` which links to a dictionary. Example: `<bond convention="ccdc:9" atomRefs2="a1 a2"/>` could represent a delocalised bond in the CCDC convention.

Allowed values

- S

  Single bond.

- 1

  Single bond.
- D

  Double bond.
- 2

  Double bond.
- T

  Triple bond.
- 3

  Triple bond.
- A

  Aromatic bond.

[link]

bondSet *[el.bondSet]*

A set of references to bonds.

An bondSet consists of a number of unique references to bonds throught their ids. bondSets need not be related to molecules (which are generally created by aggregation of explicit bonds). Two or more bondSets may reference the same bond, and bondSets may be empty.

bondSets have many potential uses such as:

- identifying functional groups
- results of substructure matching
- identifying bonds with particular roles in a calculation

The bondSet may be referenced from elsewhere in the document and you are encouraged to use locally unique id attributes on bondSets.

Example

```
<cml>
<bondSet id="bs1" title="rotatable bonds">b4</bondSet>
<bondSet id="bs2" title="carbon-halogen bonds">b1 b2
b3</bondSet>
<molecule>
  <atomArray>
    <atom atomTypeRef="a1" elementType="C"></atom>
    <atom atomTypeRef="a2" elementType="Br"></atom>
    <atom atomTypeRef="a3" elementType="Cl"></atom>
    <atom atomTypeRef="a4" elementType="F"></atom>
    <atom atomTypeRef="a5" elementType="C"
```

```
/atom>
  </atomArray>
  <bondArray>
    <bond id="b1" atomRefs2="a1 a2"></bond>
    <bond id="b2" atomRefs2="a1 a3"></bond>
    <bond id="b3" atomRefs2="a1 a4"></bond>
    <bond id="b4" atomRefs2="a1 a5"></bond>
  </bondArray>
</molecule>
</cml>
```

Content Model of element

bondRefArrayType[st.bondRefArrayType]

An array of references to bonds.

The references cannot (yet) cannot be schema- or schematron-validated. Instances of this type will be used in array-style representation of electron counts, etc. It can also be used for arrays of bondIDTypes such as in complex stereochemistry, geometrical definitions, bond groupings, etc.

Example

```

```

XSD:LIST of bondRefType

A reference to an existing bond.

A reference to a bond may be made by atoms (e.g. for multicentre or pi-bonds), electrons (for annotating reactions or describing electronic properties) or possibly other bonds (no examples yet). The semantics are relatively flexible.

Example

```
<cml title="bondArray example">
  <bondArray>
    <bond id="b1" atomRefs2="a3 a8" order="D">
      <electron bondRef="b1"></electron>
      <bondStereo>C</bondStereo>
    </bond>
    <bond id="b2" atomRefs2="a3 a8" order="S">
      <bondStereo convention="MDL"
conventionValue="6"></bondStereo>
    </bond>
  </bondArray>
</cml>
```

[xsd:string]

Pattern: \s*\S+\s*

`title`*[att.title]*

A title on an element.

No controlled value.

Example

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

`id`*[att.id]*

An attribute providing a unique ID for an element.

[link]

`convention`*[att.convention]*

A reference to a convention.

[link]

`dictRef`*[att.dictRef]*

A reference to a dictionary entry.

[link]

`size`*[att.size]*

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

[link]

`bondStereo`*[el.bondStereo]*

A container supporting "cis/trans", "wedge hatch" and other stereochemistry.

An explict list of atomRefs must be given, or it must be a child of `bond`. There are no implicit conventions such as E/Z. This will be extended to other types of stereochemistry.

At present the following are supported:

- No atomRefs attribute. **Deprecated, but probably unavoidable**. This must be a child of `bond` where it picks up the two atomRefs in the `atomRefs2` attribute. Possible values are C/T (which only makes sense if there is exactly one ligand at each end of the bond) and W/H. The latter should be raplaced by `atomParity` wherever possible. Note that W/H makes no sense without 2D atom coordinates.
- **atomRefs4 attribute**. The 4 atoms represent a cis or trans configuration. This may or may not be a child of `bond`; if so the second and third atomRefs should be identical with the two atomRefs in the bond. This structure can be used to guide processors in processing stereochemistry and is recommended, since

there is general agreement on the semantics. The semantics of `bondStereo` not related to bonds is less clear (e.g. cumulenes, substituted ring nuclei) etc.It is currently an error to have more than one `bondStereo` referring to the same ordered 4-atom list

- **atomRefs attribute**. There are other stereochemical conventions such as cis/trans for metal complexes which require a variable number of reference atoms. This allows users to create their own - at present we do not see CML creating exhaustive tables. For example cis/trans square-planar complexes might require 4 (or 5) atoms for their definition, octahedral 6 or 7, etc. In principle this is very powerful and could supplement or replace the use of *cis*-, *mer*-, etc.

the `atomRefs` and `atomRefs4` attributes cannot be used simultaneously.

Example

```
<cml title="bondArray example">
  <bondArray>
    <bond id="b1" atomRefs2="a3 a8" order="D">
      <electron bondRef="b1"></electron>
      <bondStereo>C</bondStereo>
    </bond>
    <bond id="b2" atomRefs2="a3 a8" order="S">
      <bondStereo convention="MDL"
conventionValue="6"></bondStereo>
    </bond>
  </bondArray>
</cml>
```

Content Model of element

stereoType[st.stereoType]

(Bond) stereochemistry (as a string).

. This is purely conventional; . There is no default value. The emptyString attribute can be used to indicate a bond of unknown or unspecified type. The interpretation of this is outside the scope of CML-based algorithms. It may be accompanied by a `convention` attribute which links to a dictionary

Example

```
<cml title="bondArray example">
  <bondArray>
    <bond id="b1" atomRefs2="a3 a8" order="D">
      <electron bondRef="b1"></electron>
      <bondStereo>C</bondStereo>
    </bond>
    <bond id="b2" atomRefs2="a3 a8" order="S">
      <bondStereo convention="MDL"
conventionValue="6"></bondStereo>
    </bond>
  </bondArray>
</cml>
```

Allowed values

- C

A cis bond.

- T

A trans bond.

- W

A wedge bond.

- H

A hatch bond.

-

empty or missing.

`atomRefs4` *[att.atomRefs4]*

A list of 4 references to atoms.

[link]

`atomRefArray` *[att.atomRefArray]*

An array of references to atoms.

[link]

`title` *[att.title]*

A title on an element.

No controlled value.

Example

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

`id` *[att.id]*

An attribute providing a unique ID for an element.

[link]

`convention` *[att.convention]*

A reference to a convention.

[link]

`dictRef` *[att.dictRef]*

A reference to a dictionary entry.

[link]

`conventionValue`[]

The stereo value when the `convention` attribute is used.

When convention is used this attribute must be present and element content must be empty.

`cml`*[el.cml]*

A general container for CML elements.

Often the root of the CML (sub)document. Has no explicit function but serves to hold the dictionaries, namespace, and can alert CML processors and search/XMLQuery tools that there is chemistry in the document. Can contain any content, but usually a list of molecules and other CML components. Can be nested

Example

```
<cml id="c1" title="demo of cml subelements">
  <stm:dictionary dictRef="d1"
href="dict1.xml"></stm:dictionary>
  <stm:unitList dictRef="u1" href="units1.xml"></stm:unitList>
  <cml>
    <molecule id="m1"></molecule>
  </cml>
  <molecule id="m2" title="dummy"></molecule>
  <metadata></metadata>
  <reaction>
    <reactantList>
      <molecule id="r1"></molecule>
    </reactantList>
    <productList>
      <molecule id="p1"></molecule>
    </productList>
  </reaction>
  <spectrum>
    <data>
      <stm:array></stm:array>
      <stm:array></stm:array>
    </data>
  </spectrum>
  <substanceList id="subList1"></substanceList>
  <list>
    <scalar title="some scalar"></scalar>
  </list>
</cml>
```

Content Model of element

(

No specific restrictions.

, ANY [lax])

*

Attributes of element

`title`*[att.title]*

A title on an element.

No controlled value.

Example

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

`id`*[att.id]*

An attribute providing a unique ID for an element.

[link]

`convention`*[att.convention]*

A reference to a convention.

[link]

`dictRef`*[att.dictRef]*

A reference to a dictionary entry.

[link]

`crystal`*[el.crystal]*

A container for crystallographic cell parameters and spacegroup.

. Required if fractional coordinates are provided for a molecule.

There are precisely SIX child `scalar`s to represent the cell lengths and angles in that order. There are no default values;

Example

```
<cml title="crystal example">
  <molecule id="m1">
    <crystal z="4">
      <scalar title="a" errorValue="0.001"
units="units:angstrom">4.500</scalar>
      <scalar title="b" errorValue="0.001"
units="units:angstrom">4.500</scalar>
      <scalar title="c" errorValue="0.001"
units="units:angstrom">4.500</scalar>
      <scalar title="alpha" units="units:degree">90</scalar>
```

```
      <scalar title="beta" units="units:degree">90</scalar>
      <scalar title="gamma" units="units:degree">90</scalar>
      <symmetry id="s1" spaceGroup="Fm3m"></symmetry>
    </crystal>
    <atomArray>
      <atom id="a1" elementType="Na" formalCharge="1"
xyzFract="0.0 0.0 0.0" xy2="+23.2 -21.0"></atom>
      <atom id="a2" elementType="Cl" formalCharge="-1"
xyzFract="0.5 0.0 0.0"></atom>
    </atomArray>
  </molecule>
</cml>
```

Content Model of element

(

All 6 cell parameters must be given, even where angles are fixed by symmetry. The order is fixed as a,b,c,alpha,gamma,beta and software can neglect any title or dictRef attributes. Error estimates can be given if required. Any units can be used, but the defaults are Angstrom (10^-10 m) and degrees. .

,scalar{6,6},symmetry?)

Attributes of element

z*[]*

The number of molecules per cell. Molecules are defined as the `molecule` which directly contains the `crystal` element.

`title`*[att.title]*

A title on an element.

No controlled value.

Example

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

`id`*[att.id]*

An attribute providing a unique ID for an element.

[link]

`convention`*[att.convention]*

A reference to a convention.

[link]

dictRef*[att.dictRef]*

A reference to a dictionary entry.
[link]

eigen*[el.eigen]*

An element to hold eigenstuff.

Not yet written
Example

```
<eigen>
  <array dataType="xsd:float">1 2 3</array>
  <matrix dataType="xsd:float" rows="3" columns="3">1 2 3 4 5
6 7 8 9</matrix>
</eigen>
```

Content Model of element

(array?,matrix?)

Attributes of element

foo*[att.eigen.units]*

foo

title*[att.title]*

A title on an element.

No controlled value.
Example

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

id*[att.id]*
An attribute providing a unique ID for an element.
[link]

convention*[att.convention]*
A reference to a convention.
[link]

`dictRef`*[att.dictRef]*

A reference to a dictionary entry.
[link]

`type`*[att.eigen.eigenType]*

Type of eigen.

To be determined.

`electron`*[el.electron]*

One or more electrons.

Since there is very little use of electrons in current chemical information this is a fluid concept. I expect it to be used for electron counting, input and output of theochem operations, descriptions of orbitals, spin states, oxidation states, etc. Electrons can be associated with atoms, bonds and combinations of these. At present there is no hardcoded semantics. However, `atomRef` and similar attributes can be used to associate electrons with atoms or bonds

Example

```
<cml title="electron example">
  <molecule id="m1">
    <atomArray atomID="a1 a2 a3 a4 a5 a6"></atomArray>
    <bondArray order="A A A A A A" bondID="b1 b2 b3 b4 b5 b6"
atomRef1="a1 a2 a3 a4 a5 a6" atomRef2="a6 a1 a2 a3 a4
a5"></bondArray>
    <electron count="6" bondRefs="b1 b2 b3 b4 b5 b6"
atomRefs="a1 a2 a3 a4 a5 a6"></electron>
  </molecule>
</cml>
```

Content Model of element

()

Attributes of element

`title`*[att.title]*

A title on an element.

No controlled value.

Example

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

```
</stmml>
```

id*[att.id]*
An attribute providing a unique ID for an element.
[link]

convention*[att.convention]*
A reference to a convention.
[link]

dictRef*[att.dictRef]*

A reference to a dictionary entry.
[link]

atomRef*[]*

A reference to an existing atom.
Example

```
<cml title="atomRef example">
  <molecule id="m1">
    <atomArray>
      <atom id="a1"></atom>
    </atomArray>
    <electron id="e1" atomRef="a1"></electron>
  </molecule>
</cml>
```

[xsd:string]
Pattern: \s*[A-Za-z_][A-Za-z0-9\-:_]*\s*
[link]

atomRefs*[]*

An array of atomRefs.

The atomRefs cannot be schema- or schematron-validated. Instances of this type will be used in array-style representation of bonds and atomParitys. It can also be used for arrays of atomIDTypes such as in complex setereochemistry, geometrical definitions, atom groupings, etc.
Example

```
<cml title="atomArray example">
  <molecule id="m1">
    <atomArray atomID="a2 a4 a6" elementType="O N
S"></atomArray>
  </molecule>
</cml>
```

XSD:LIST of atomRefType

A reference to an existing atom.

Example

```
<cml title="atomRef example">
  <molecule id="m1">
    <atomArray>
      <atom id="a1"></atom>
    </atomArray>
    <electron id="e1" atomRef="a1"></electron>
  </molecule>
</cml>
```

[xsd:string]
Pattern: \s*[A-Za-z_][A-Za-z0-9\-:_]*\s*
[link]

bondRef[]

A reference to an existing bond.

A reference to a bond may be made by atoms (e.g. for multicentre or pi-bonds), electrons (for annotating reactions or describing electronic properties) or possibly other bonds (no examples yet). The semantics are relatively flexible.

Example

```
<cml title="bondArray example">
  <bondArray>
    <bond id="b1" atomRefs2="a3 a8" order="D">
      <electron bondRef="b1"></electron>
      <bondStereo>C</bondStereo>
    </bond>
    <bond id="b2" atomRefs2="a3 a8" order="S">
      <bondStereo convention="MDL"
conventionValue="6"></bondStereo>
    </bond>
  </bondArray>
</cml>
```

[xsd:string]
Pattern: \s*\S+\s*
[link]

bondRefs[]

An array of references to bonds.

The references cannot (yet) cannot be schema- or schematron-validated. Instances of this type will be used in array-style representation of electron counts, etc. It can also be used for arrays of bondIDTypes such as in complex stereochemistry, geometrical definitions, bond groupings, etc.

Example

XSD:LIST of bondRefType

A reference to an existing bond.

A reference to a bond may be made by atoms (e.g. for multicentre or pi-bonds), electrons (for annotating reactions or describing electronic properties) or possibly other bonds (no examples yet). The semantics are relatively flexible.

Example

```
<cml title="bondArray example">
  <bondArray>
    <bond id="b1" atomRefs2="a3 a8" order="D">
      <electron bondRef="b1"></electron>
      <bondStereo>C</bondStereo>
    </bond>
    <bond id="b2" atomRefs2="a3 a8" order="S">
      <bondStereo convention="MDL"
conventionValue="6"></bondStereo>
    </bond>
  </bondArray>
</cml>
```

[xsd:string]
Pattern: \s*\S+\s*
[link]

count*[]*

The number of electrons.

No formal default, but assumed to be 1. At present restricted to integers.

ref*[att.ref]*

A reference to an element of given type.

[link]

expression*[el.expression]*

An expression, which can be evaluated.

Experimental. This is essentially a mathematical function, expressed currently in reverse Polish notation.

Example

```
<--   Li c O  c Buckingham 0.115E+04 0.280     .000E+00
.000E+00   0.000 10.000 --><potential form="gulp:buckingham">
   <arg name="atom1">
     <atom atomTypeRef="gulp:at.li"></atom>
   </arg>
   <arg name="atom2">
     <atom atomTypeRef="gulp:at.o"></atom>
```

```
   </arg>
   <arg ref="gulp:buckingham.a">
     <scalar units="units:ev">0.115E+04</scalar>
   </arg>
   <arg ref="gulp:buckingham.b">
     <scalar units="units:angstrom">0.280</scalar>
   </arg>
   <arg ref="gulp:buckingham.c">
     <scalar units="units:ev">.000E+00</scalar>
   </arg>
 </potential>
```

Content Model of element

(
(parameter|operator)
*)

Attributes of element

`title`*[att.title]*

A title on an element.

No controlled value.

Example

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

`id`*[att.id]*

An attribute providing a unique ID for an element.

[link]

`convention`*[att.convention]*

A reference to a convention.

[link]

`dictRef`*[att.dictRef]*

A reference to a dictionary entry.

[link]

`dataType`*[]*

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.

Example

```
<stmml title="namespace example">
<list>
  <scalar dictRef="chem:mpt">123</scalar>
  <scalar dictRef="mpt23">123</scalar>
</list>
</stmml>
```

[xsd:string]

The namespace prefix must start with an alpha character and can only contain alphanumeric and '_'. The suffix can have characters from the XML ID specification (alphanumeric, '_', '.' and '-'

Pattern: ([A-Za-z][A-Za-z0-9_]*:)?[A-Za-z][A-Za-z0-9_\.\-]*

[link]

`formula` *[el.formula]*

The stochiometry of the molecule.

It is defined by `atomArray`s each with a list of elementTypes and their counts (or default=1). All other information in the `atomArray` is ignored. `formula` are nestable so that aggregates (e.g. hydrates, salts, etc.) can be described. CML does not require that formula information is consistent with (say) crystallographic information; this allows for experimental variance.

An alternative briefer representation is also available through the `conciseForm`. This must include whitespace round all elements and their counts, which must be explicit.

Example

```
<cml title="formula example">
  <molecule id="sulfuricAcid">
    <formula concise="H 2 S 1 O 4"></formula>
  </molecule>
  <molecule id="CuprammoniumSulfate">
    <formula title="[Cu(NH3)4]2+ SO42-]">
      <formula formalCharge="+2">
        <atomArray elementType="Cu"></atomArray>
        <formula count="4">
          <atomArray elementType="N H" count="1
3"></atomArray>
        </formula>
      </formula>
      <formula formalCharge="-2">
```

```
        <atomArray elementType="S O" count="1 4"></atomArray>
      </formula>
    </formula>
  </molecule>
</cml>
```

Content Model of element

(
(formula|atomArray)
*)

Attributes of element

`title`*[att.title]*

A title on an element.

No controlled value.

Example

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

`id`*[att.id]*

An attribute providing a unique ID for an element.

[link]

`convention`*[att.convention]*

A reference to a convention.

[link]

`dictRef`*[att.dictRef]*

A reference to a dictionary entry.

[link]

`count`*[]*

A positive number. Note that we also provide nonNegativeNumber with inclusive zero. The maximum number is (quite large) since 'unbounded' is more difficult to implement. This is greater than Eddington's estimate of the number of particles in the universe so it should work for most people.

[xsd:double]
minExclusive: 0.0
maxInclusive: 1.0E+99

A multiplier for the formula.
[link]

`formalCharge`[]

The formal charge is normally calculated from the formal charges of the atoms. If the `formalCharge` attribute is given it overrides this information completely. This allows (say) metal complexes to be represented when it is difficult to apportion the charges to atoms.

`concise`[]

A concise representation for a molecular formula.

This MUST adhere to a whitespaced syntax so that it is trivially machine-parsable. Each element is followed by its count, and the string is optionally ended by a formal charge. NO brackets or other nesting is allowed.
Example

```
<cml title="formulaType example (concise)">
  <list>
    <formula id="methane" concise="C 1 H 4"></formula>
    <formula id="chloroacetate" concise="Cl 1 H 2 C 2 O 2
-1"></formula>
    <formula id="sodiumSulfate">
      <formula concise="H 2 O 1" count="10"></formula>
      <formula concise="Na 1 +1" count="2"></formula>
      <formula concise="S 1 O 4 -2"></formula>
    </formula>
  </list>
</cml>
```

[xsd:string]
Pattern: \s*([A-Z][a-z]?\s+[1-9][0-9]*)(\s+[A-Z][a-z]?\s+[1-9][0-9]*)*(\s+[-|+]?[0-9]+)?\s*
A concise string representing an (unstructured) formula.
[link]

`gradient`*[el.gradient]*

A gradient.

A container for a quantity or quantities representing the gradient of other quantities
Example

```
<gradient dictRef="ccml:gradient">
  <scalar units="units:kcal/mole/a">1.3</scalar></gradient>
```

Content Model of element

(
(scalar|array|matrix|property)

*)

Attributes of element

`title`*[att.title]*

A title on an element.

No controlled value.

Example

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

`id`*[att.id]*

An attribute providing a unique ID for an element.

[link]

`convention`*[att.convention]*

A reference to a convention.

[link]

`dictRef`*[att.dictRef]*

A reference to a dictionary entry.

[link]

`identifier` *[el.identifier]*

2003-07-10: Fixed count on identifier children.

2003-03-12: Added isotopic and atoms.

IChI identifier.

Supports compund identifiers such as IChI. At present uses the V0.9 IChI XML representation verbatim but will almost certainly change with future IChIs.

The inclusion of elements from other namespaces causes problems with validation. The content model is deliberately LAX but the actual elements in IChI will fail the validation as they are not declared in CML.

Example

```
<identifier version="0.93Beta" tautomeric="0">
   <basic>C6H3ClN2O5,7-4-1H-3(8(10)11)2H-5(6(4)14H)9(12)13</basic>
   <charge></charge>
  </identifier>
```

Content Model of element

(ANY [lax])
+

Attributes of element

`version`*[]*

`tautomeric`*[]*

`length`*[el.length]*

A length between two atoms.

This is either an experimental measurement or used to build up internal coordinates (as in a z-matrix) (only one allowed)

We expect to move length as a child of **molecule** and remove it from here

Example

```
<cml title="length example">
  <molecule id="m1">
    <atomArray atomID="a1 a2 a3"></atomArray>
    <length atomRefs2="a3 a1">1.534</length>
  </molecule>
</cml>
```

Content Model of element

[xsd:float]

`title`*[att.title]*

A title on an element.

No controlled value.

Example

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

`id`*[att.id]*

An attribute providing a unique ID for an element.

[link]

convention*[att.convention]*
A reference to a convention.
[link]

dictRef*[att.dictRef]*

A reference to a dictionary entry.
[link]

atomRefs2*[att.atomRefs2]*
A list of two references to atoms.
[link]

units*[att.units]*
Scientific units on an element.
[link]

errorValue*[att.errorValue]*
[link]

errorBasis*[att.errorBasis]*
[link]

min*[att.min]*
[link]

max*[att.max]*
[link]

ref*[att.ref]*
A reference to an element of given type.
[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.
Example

```
<stmml title="list example">
<list>
```

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

Content Model of element

(ANY [lax])

Attributes of element

`title`*[att.title]*

A title on an element.

No controlled value.

Example

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

`id`*[att.id]*

An attribute providing a unique ID for an element.

[link]

`convention`*[att.convention]*

A reference to a convention.

[link]

`dictRef`*[att.dictRef]*

A reference to a dictionary entry.

[link]

`type`*[att.list.type]*

Type of the list.

Semeantics undefined.

Example

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

`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.

Example

```
<stmml title="matrix example">
<matrix id="m1" title="matrix-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>
</stmml>
```

Content Model of element

[xsd:string]

`dataType`*[att.matrix.dataType]*REQUIRED[link]

`delimiter`*[att.matrix.delimiter]*[link]

`rows`*[]*REQUIRED

Number of rows
[link]

`columns`*[att.matrix.columns]*REQUIRED

Number of columns
[link]

units*[att.matrix.units]*

units (recommended for numeric quantities!!)
[link]

title*[att.title]*

A title on an element.
No controlled value.
Example

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

id*[att.id]*
An attribute providing a unique ID for an element.
[link]

convention*[att.convention]*
A reference to a convention.
[link]

dictRef*[att.dictRef]*

A reference to a dictionary entry.
[link]

matrixType*[att.matrix.matrixType]*
Type of matrix.
[link]

errorValues*[att.matrix.errorValues]*
an optional array of error values for numeric matrices.
[link]

errorBasis*[att.errorBasis]*
[link]

minValues*[att.matrix.minValues]*
an optional array of minimum values for numeric matrices.
[link]

maxValues*[att.matrix.maxValues]*

an optional array of maximum values for numeric matrices.
[link]

`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).

Example

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

```
    <metadata name="cmlm:identifier"
content="smiles:O=C1C=C(C)C(C)(C)C(C)=C1"></metadata>
  </metadataList>
  <metadataList>
    <metadata name="foo:institution"
content="abc.org"></metadata>
    <metadata name="bar" content="xyzzy"></metadata>
    <metadata name="$deliberateError"
content="error"></metadata>
  </metadataList>
</list>
</stmml>
```

Content Model of element

[xsd:string]

name*[att.metadata.name]*

The metadata type.

[link]

convention*[att.convention]*

A reference to a convention.

[link]

content*[att.metadata.content]*

The metadata.

metadataList*[el.metadataList]*

A general container for metadata elements.

Example

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

```
></metadata>
    <metadata name="dc:contributor" content="Tux
Penguin"></metadata>
    <metadata name="dc:creator" content="author"></metadata>
    <metadata name="dc:publisher" content="Penguinone
publishing"></metadata>
    <metadata name="dc:source"
content="penguinPub"></metadata>
    <metadata name="dc:language" content="en-GB"></metadata>
    <metadata name="dc:date" content="1752-09-10"></metadata>
  </metadataList>
  <metadataList>
    <metadata name="cmlm:safety" content="mostly
harmless"></metadata>
    <metadata name="cmlm:insilico" content="electronically
produced"></metadata>
    <metadata name="cmlm:structure"
content="penguinone"></metadata>
    <metadata name="cmlm:reaction" content="synthesis of
penguinone"></metadata>
    <metadata name="cmlm:identifier"
content="smiles:O=C1C=C(C)C(C)(C)C(C)=C1"></metadata>
  </metadataList>
  <metadataList>
    <metadata name="foo:institution"
content="abc.org"></metadata>
    <metadata name="bar" content="xyzzy"></metadata>
    <metadata name="$deliberateError"
content="error"></metadata>
  </metadataList>
</list>
</stmml>
```

Content Model of element

(metadata+)

`module`*[el.module]*

A module in a calculation.

Many programs are based on discrete modules which produce chunks of output. There are also conceptual chunks such as initialisation, calculation and summary/final which often have finer submodules such as cycle, iteration, snapshot, etc. There is no controlled vocabulary but a typical structure is:

```
<module type="initial">
  <!-- initial values listed here -->
</module>
<module type="geometryOptimisation">
  <module number="1">
  </module>
```

```
  <module number="2">
  </module>
  <!-- ... -->
  <module number="20">
  </module>
  <module type="final">
    <!-- values on last cycle/module listed here -->
  </module>
</module>
<module type="final">
  <!-- final values listed here -->
</module>
```

One of the challenges of CCML is to find communality between different programs and to use agreed abstractions for the modules.

Example

```


Not yet written


```

Content Model of element

(ANY [lax])

Attributes of element

`serial`*[att.module.serial]*

Serial number (or other instance id).

Modules with the same `role` attribute can be distinguished by `serial`. This is often an integer but other schemes may be used.

`title`*[att.title]*

A title on an element.

No controlled value.

Example

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

`id`*[att.id]*

An attribute providing a unique ID for an element.

[link]

convention*[att.convention]*
A reference to a convention.
[link]

dictRef*[att.dictRef]*

A reference to a dictionary entry.
[link]

role*[att.module.role]*

Role of module.

The module can have a program-specific name through its title or dictRef (e.g. "MINIM", "l201") and a generic role ("dynamicsCalculation", "equilibration", etc.). In general role will be controlled by CCML.

molecule*[el.molecule]*

A container for atoms, bonds and submolecules.

`molecule` is a container for atoms, bonds and submolecules along with properties such as crystal and non-builtin properties. It should either contain `molecule` or *Array for atoms and bonds. A molecule can be empty (e.g. we just know its name, id, etc.)

"Molecule" need not represent a chemically meaningful molecule. It can contain atoms with bonds (as in the solid-sate) and it could simply carry a name (e.g. "taxol") without formal representation of the structure. It can contain "sub molecules", which are often discrete subcomponents (e.g. guest-host).

Molecule can contain a <list> element to contain data related to the molecule. Within this can be string/float/integer and other nested lists

Example

```
<cml title="schematic molecule example">
  <molecule id="dummyId">
    <atomArray>
      <atom id="a1" elementType="C" hydrogenCount="0"
x2="6.1964" y2="8.988"></atom>
      <atom id="a2" elementType="C" hydrogenCount="0"
x2="6.1964" y2="7.587"></atom>
      <atom id="a3" elementType="C" hydrogenCount="2"
x2="4.983" y2="6.887"></atom>
      <atom id="a28" elementType="C" hydrogenCount="3"
x2="15.777" y2="6.554"></atom>
      <atom id="a29" elementType="O" hydrogenCount="0"
x2="13.388" y2="6.188"></atom>
    </atomArray>
    <bondArray>
```

```
      <bond atomRefs2="a1 a2" order="1"></bond>
      <bond atomRefs2="a2 a3" order="1"></bond>
      <bond atomRefs2="a3 a4" order="1"></bond>
      <bond atomRefs2="a11 a15" order="1"></bond>
      <bond atomRefs2="a12 a18" order="1">
        <bondStereo>W</bondStereo>
      </bond>
      <bond atomRefs2="a2 a19" order="1">
        <bondStereo>W</bondStereo>
      </bond>
      <bond atomRefs2="a5 a20" order="2"></bond>
      <bond atomRefs2="a17 a21" order="1"></bond>
      <bond atomRefs2="a21 a22" order="1"></bond>
      <bond atomRefs2="a10 a9" order="1"></bond>
      <bond atomRefs2="a16 a29" order="2"></bond>
    </bondArray>
  </molecule>
</cml>
```

Revised content model to allow any order of lengths, angles, torsions 2003-01-01.

Added role attribute 2003-03-19.

Content Model of element

(metadataList*,formula?,identifier?,name*,symmetry?,crystal?,
(molecule*|
(atomArray,bondArray?,electron*,
(length|angle|torsion)
*)
)
,
(scalar*,array*,matrix*,list*)
*)

Attributes of element

`dictRef`*[att.dictRef]*

A reference to a dictionary entry.
[link]

`convention`*[att.convention]*

A reference to a convention.
[link]

`title`*[att.title]*

A title on an element.
No controlled value.

Example

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

id *[att.id]*

An attribute providing a unique ID for an element.

[link]

ref *[att.ref]*

A reference to an element of given type.

[link]

formula *[]*

A concise representation for a molecular formula.

This MUST adhere to a whitespaced syntax so that it is trivially machine-parsable. Each element is followed by its count, and the string is optionally ended by a formal charge. NO brackets or other nesting is allowed.

Example

```
<cml title="formulaType example (concise)">
  <list>
    <formula id="methane" concise="C 1 H 4"></formula>
    <formula id="chloroacetate" concise="Cl 1 H 2 C 2 O 2
-1"></formula>
    <formula id="sodiumSulfate">
      <formula concise="H 2 O 1" count="10"></formula>
      <formula concise="Na 1 +1" count="2"></formula>
      <formula concise="S 1 O 4 -2"></formula>
    </formula>
  </list>
</cml>
```

[xsd:string]

Pattern: \s*([A-Z][a-z]?\s+[1-9][0-9]*)(\s+[A-Z][a-z]?\s+[1-9][0-9]*)*(\s+[-|+]?[0-9]+)?\s*

[link]

count *[]*

A positive number. Note that we also provide nonNegativeNumber with inclusive zero. The maximum number is (quite large) since 'unbounded' is more difficult to implement. This is greater than Eddington's estimate of the number of particles in the universe so it should work for most people.

[xsd:double]

minExclusive: 0.0

maxInclusive: 1.0E+99

The count for the molecule.

[link]

`chirality`*[]*

The chirality of the complete system.

`formalCharge`*[el.molecule.formalCharge]*

The formal charge on an atom.

Used for electron-bookeeping. This has no relation to its calculated (fractional) charge.

Example

```
<cml title="formalCharge example">
  <atomArray>
    <atom id="a1" elementType="N" formalCharge="+1"></atom>
    <atom id="a2" elementType="O" formalCharge="-1"></atom>
  </atomArray>
</cml>
```

[xsd:integer]

The formalCharge on the molecule.

[link]

`spinMultiplicity`*[el.molecule.spinMultiplicity]*

The spin multiplicity for the molecule.

This attribute gives the spin multiplicity of the molecule and is independent of any atomic information. No default, and it may take any positive integer value (though values are normally between 1 and 5)

`symmetryOriented`*[el.molecule.symmetryOriented]*

Is the molecule oriented to the symmetry.

No formal default, but a molecule is assumed to be oriented according to any <symmetry> children. This is required for crystallographic data, but some systems for isolated molecules allow specification of arbitrary Cartesian or internal coordinates, which must be fitted or refined to a prescribed symmetry. In this case the attribute value is `false`.

`role`*[el.molecule.role]*

Role of the molecule

No formal semantics (yet). The role describes the purpose of the molecule element at this stage in the information. Examples can be "conformation", "dynamicsStep", "vibration", "valenceBondIsomer", etc. This attribute may be used by applications to determine how to present a set of molecule elements.

`velocityUnits`*[molecule.cmlcomp.att.velocityUnits]*

units for velocities on molecule or atoms.
This is a CCML extension to core CML.

If any velocities are given for this molecule of its descendants, this attribute MUST be given. There are no defaults and all objects in this molecule must use the same units.

vx*[molecule.cmlcomp.att.vx]*

velocity along the Cartesian X axis.

This is a CCML extension to core CML.

Units MUST be given in velocityUnits; there are NO defaults.

vy*[molecule.cmlcomp.att.vy]*

velocity along the Cartesian Y axis.

This is a CCML extension to core CML.

Units MUST be given in velocityUnits; there are NO defaults.

vz*[molecule.cmlcomp.att.vz]*

velocity along the Cartesian Z axis.

This is a CCML extension to core CML.

Units MUST be given in velocityUnits; there are NO defaults.

vxyz*[molecule.cmlcomp.att.vxyz]*

A vector in 3-space
No constraints on magnitude (i.e. could be zero)
Example
```
<myVector dataType="stm:vector3Type">2.0 3.0 4.0</myVector>
```
[xsd:string]
Pattern: \s*([\-]|[+])?\d*\.?\d*(\s+|[,])([\-]|[+])?\d*\.?\d*(\s+|[,])([\-]|[+])?\d*\.?\d*\s*
velocity as a vector
[link]

name*[el.name]*

A string identifying a molecule, atom or (possibly) other elements.

`name` is used for chemical names (formal and trivial) for molecules and also for identifiers such as CAS registry and RTECS. It can also be used for labelling atoms. It should be used in preference to the `title` attribute because it is repeatable and can be linked to a dictionary.

Constraining patterns can be described in the dictionary and used to validate `name`s.
Example

```
<cml title="name example">
  <molecule id="aspirin">
    <name convention="INN"> name="builtin"
type="xsd:string"in</name>
    <name convention="IUPAC">2-acetoxybenzoic acid</name>
    <name convention="trivial">acetylsalicylic acid</name>
  </molecule>
</cml>
```

Content Model of element

[xsd:string]

`id`*[att.id]*
An attribute providing a unique ID for an element.
[link]

`convention`*[att.convention]*
A reference to a convention.
[link]

`dictRef`*[att.dictRef]*

A reference to a dictionary entry.
[link]

`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
Example

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

```
</observation>
</stmml>
```

Content Model of element

(ANY [lax])
*

Attributes of element

`title`*[att.title]*

A title on an element.

No controlled value.

Example

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

`id`*[att.id]*

An attribute providing a unique ID for an element.

[link]

`convention`*[att.convention]*

A reference to a convention.

[link]

`dictRef`*[att.dictRef]*

A reference to a dictionary entry.

[link]

`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.

Example

```
<stmml title="countType example">
```

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

[xsd:nonNegativeInteger]
[link]

operator *[el.operator]*

An operator within an expression.

Experimental. An operator acts on one or more arguments (at present the number is fixed by the type). The formulation is reverse Polish so the result (with its dataType) is put on a stack for further use

Example

```
<--   Li c O  c Buckingham 0.115E+04 0.280     .000E+00
.000E+00   0.000 10.000 --><potential form="gulp:buckingham">
   <arg name="atom1">
     <atom atomTypeRef="gulp:at.li"></atom>
   </arg>
   <arg name="atom2">
     <atom atomTypeRef="gulp:at.o"></atom>
   </arg>
   <arg ref="gulp:buckingham.a">
     <scalar units="units:ev">0.115E+04</scalar>
   </arg>
   <arg ref="gulp:buckingham.b">
     <scalar units="units:angstrom">0.280</scalar>
   </arg>
   <arg ref="gulp:buckingham.c">
     <scalar units="units:ev">.000E+00</scalar>
   </arg>
 </potential>
```

Content Model of element

(
()
*)

Attributes of element

title *[att.title]*

A title on an element.

No controlled value.

Example

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

`id` *[att.id]*

An attribute providing a unique ID for an element.

[link]

`convention` *[att.convention]*

A reference to a convention.

[link]

`dictRef` *[att.dictRef]*

A reference to a dictionary entry.

[link]

`type` *[]*

The type of the operator (minus, mult, distance, etc.) This will be enumerated when I have time...

`parameter` *[el.parameter]*

A parameter describing the computation.

A parameter is a broad concept and can describe numeric quantities, objects, keywords, etc. The distinction between keywords and parameters is often fuzzy. ("MINIM" might mean "minimize", while "MINIM=3" might require three iterations to be run. It may help to think of control keywords as boolean parameters.

Numeric parameters can describe values in molecules, forcefields or other objects. Often the parameters will be refined or otherwise varied during the calculation. Some parameters may be fixed at particulat values or relaxed at different stages in the calculation. Parameters can have errors, gradients and other indications of uncertainty.

String/character parameters are often abbreviated in program input, and this is supported through the `regex` and `ignoreCase` attributes.

Parameters will usually be defined separately from the objects and use the `ref` attribute to reference them.

Parameters can be used to describe additional constraints. This will probably require the development of a microlanguage and until then may use program-specific mecxhanisms. A common approach will be to use an array of values (or objects) to represent different input values for (parts of) the calculation. Thus a conformational change could be specified by an array of several torsion angles.

A parameter will frequently have a `dictRef` pointing to a dictionary which may have more information about how the parameter is to be used or the values it can take.

The allowable content of `parameter`s may be shown by a "template" in the `appinfo`; this is stil experimental.

Example

```
<parameter role="ccml:keyword"
dictRef="gulp:optimize"></parameter>
```

Content Model of element

(
(scalar|array|matrix|property|expression)
*,
(gradient)
?)

Attributes of element

`ref` *[att.parameter.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-z][A-Za-z0-9_-]*:)?[A-Za-z][A-Za-z0-9_\-\.]*

Reference to an element
[link]

`title`*[att.title]*

A title on an element.

No controlled value.

Example

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

`id`*[att.id]*

An attribute providing a unique ID for an element.
[link]

`convention`*[att.convention]*

A reference to a convention.
[link]

`dictRef`*[att.dictRef]*

A reference to a dictionary entry.
[link]

`value`*[att.parameter.value]*

Value of scalar parameter.

This is a shorthand for a single scalar value of the parameter. It should only be used with the `ref` attribute as it inherits all the dataTyping of the referenced element. It must not be used for defining new parameters as it has no mechanism for units and dataTyping. [This may change?].

`constraint`*[att.parameter.constraint]*

Constraint on a parameter.

Semantics not yet finalised. We anticipate "fixed", "none" and symbolic relationships to other parameters

`name`*[att.parameter.name]*

Name of a parameter.

role*[att.parameter.role]*

Role of a parameter.

Used to define concepts such as independent and dependent variables

parameterList*[el.parameterList]*

A container for one or more parameters.

`parameterList` can contain several parameters.
Example

```
<parameterList>
            <parameter role="ccml:keyword"
dictRef="gulp:optimize"></parameter>
            <parameter role="ccml:keyword"
dictRef="gulp:conp"></parameter>
            <parameter role="ccml:keyword"
dictRef="gulp:property"></parameter>
            <parameter role="ccml:keyword"
dictRef="gulp:phonon"></parameter>
            <parameter role="ccml:keyword"
dictRef="gulp:bond"></parameter>
            <parameter role="ccml:keyword"
dictRef="gulp:angle"></parameter>
            <parameter role="ccml:keyword"
dictRef="gulp:compare"></parameter>
          </parameterList>
```

Content Model of element

(metadataList*,name*,
(parameter)
*)

Attributes of element

dictRef*[att.dictRef]*

A reference to a dictionary entry.
[link]

convention*[att.convention]*
A reference to a convention.

[link]

`title`*[att.title]*

A title on an element.

No controlled value.

Example

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

`id`*[att.id]*

An attribute providing a unique ID for an element.

[link]

`ref`*[att.ref]*

A reference to an element of given type.

[link]

`role`*[]*

The role of the parameterList. Semantics are not yet controlled

`particle`*[el.particle]*

An object in space carrying a set of properties.

`particles` have many of the characteristics of `atom`s but without at atomic nucleus. It does not have an elementType and cannot be involved in bonding, etc. It has coordinates, may carry charge and might have a mass. It represents some aspect of a computational model and should not be used for purely geometrical concepts such as centroid. Examples of particles are "shells" (e.g. in GULP) which are linked to atoms for modelling polarizability or lonepairs and approximations to multipoles.

Properties such as charge, mass should be scalar/array/matrix children.

Example

Content Model of element

(ANY [lax])

Attributes of element

`title`*[att.title]*

A title on an element.

No controlled value.

Example

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

`id`*[att.id]*

An attribute providing a unique ID for an element.

[link]

`convention`*[att.convention]*

A reference to a convention.

[link]

`dictRef`*[att.dictRef]*

A reference to a dictionary entry.

[link]

`particleType`*[att.particleType.ref]*

A reference to an existing particle.

Example

```
<cml title="particleRef example">
  <molecule id="m1">
    <atomArray>
      <atom id="a1">
        <particle id="p1" particleType="pp1"></particle>
      </atom>
    </atomArray>
  </molecule>
</cml>
```

[xsd:string]

Pattern: \s*[A-Za-z_][A-Za-z0-9\-:_]*\s*

The type of the particle

[link]

`x3`*[att.particle.x3]*

An array of coordinateComponents for a single coordinate.

An array of coordinateComponents for a single coordinate where these all refer to an X-coordinate (NOT x,y,z) Instances of this type will be used in array-style representation of 2-D or 3-D coordinates.

Currently no machine validation

Currently not used in STMML, but re-used by CML (see example)

Example

```
<stmml title="coordinateComponentArrayType">
<cml:atomArray x2="1.2 2.3 4.5 6.7"></cml:atomArray>
</stmml>
```

XSD:LIST of xsd:float

The x-coordinate in Cartesian coordinates.

[link]

y3*[att.particle.y3]*

An array of coordinateComponents for a single coordinate.

An array of coordinateComponents for a single coordinate where these all refer to an X-coordinate (NOT x,y,z) Instances of this type will be used in array-style representation of 2-D or 3-D coordinates.

Currently no machine validation

Currently not used in STMML, but re-used by CML (see example)

Example

```
<stmml title="coordinateComponentArrayType">
<cml:atomArray x2="1.2 2.3 4.5 6.7"></cml:atomArray>
</stmml>
```

XSD:LIST of xsd:float

The y-coordinate in Cartesian coordinates.

[link]

z3*[att.particle.z3]*

An array of coordinateComponents for a single coordinate.

An array of coordinateComponents for a single coordinate where these all refer to an X-coordinate (NOT x,y,z) Instances of this type will be used in array-style representation of 2-D or 3-D coordinates.

Currently no machine validation

Currently not used in STMML, but re-used by CML (see example)

Example

```
<stmml title="coordinateComponentArrayType">
<cml:atomArray x2="1.2 2.3 4.5 6.7"></cml:atomArray>
```

```
</stmml>
```

XSD:LIST of xsd:float

The z-coordinate in Cartesian coordinates.

[link]

potential *[el.potential]*

An explicit potential (as in a forcefield).

Experimental. This represents the actual function for the potential (i.e. with explicit values) rather than the functional form, which will normally be referenced from this.

Example

```
<--   Li c O  c Buckingham 0.115E+04 0.280     .000E+00
.000E+00   0.000 10.000 --><potential form="gulp:buckingham">
   <arg name="atom1">
     <atom atomTypeRef="gulp:at.li"></atom>
   </arg>
   <arg name="atom2">
     <atom atomTypeRef="gulp:at.o"></atom>
   </arg>
   <arg ref="gulp:buckingham.a">
     <scalar units="units:ev">0.115E+04</scalar>
   </arg>
   <arg ref="gulp:buckingham.b">
     <scalar units="units:angstrom">0.280</scalar>
   </arg>
   <arg ref="gulp:buckingham.c">
     <scalar units="units:ev">.000E+00</scalar>
   </arg>
 </potential>
```

Content Model of element

(
(arg)
*)

Attributes of element

title *[att.title]*

A title on an element.

No controlled value.

Example

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

```
</stmml>
```

id*[att.id]*
An attribute providing a unique ID for an element.
[link]

convention*[att.convention]*
A reference to a convention.
[link]

dictRef*[att.dictRef]*

A reference to a dictionary entry.
[link]

form*[]*

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.
Example
```
<stmml title="namespace example">
<list>
  <scalar dictRef="chem:mpt">123</scalar>
  <scalar dictRef="mpt23">123</scalar>
</list>
</stmml>
```
[xsd:string]

The namespace prefix must start with an alpha character and can only contain alphanumeric and '_'. The suffix can have characters from the XML ID specification (alphanumeric, '_', '.' and '-'
Pattern: ([A-Za-z][A-Za-z0-9_]*:)?[A-Za-z][A-Za-z0-9_\.\-]*
A reference to the functional form used for this instance of the potential.
[link]

potentialForm*[el.potentialForm]*

The functional form of a potential.

Experimental. This has generic arguments and parameters rather than explicit ones. It is essentially a mathematical function, expressed currently in reverse Polish

notation.

Example

```
<--   Li c O  c Buckingham 0.115E+04 0.280     .000E+00
.000E+00   0.000 10.000 --><potential form="gulp:buckingham">
   <arg name="atom1">
     <atom atomTypeRef="gulp:at.li"></atom>
   </arg>
   <arg name="atom2">
     <atom atomTypeRef="gulp:at.o"></atom>
   </arg>
   <arg ref="gulp:buckingham.a">
     <scalar units="units:ev">0.115E+04</scalar>
   </arg>
   <arg ref="gulp:buckingham.b">
     <scalar units="units:angstrom">0.280</scalar>
   </arg>
   <arg ref="gulp:buckingham.c">
     <scalar units="units:ev">.000E+00</scalar>
   </arg>
 </potential>
```

Content Model of element

(
(arg)
*,
(parameter)
*,
(expression)
?)

Attributes of element

`title`*[att.title]*

A title on an element.

No controlled value.

Example

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

`id`*[att.id]*

An attribute providing a unique ID for an element.

[link]

`convention`*[att.convention]*

A reference to a convention.
[link]

dictRef *[att.dictRef]*

A reference to a dictionary entry.
[link]

name *[]*

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.
Example
```
<stmml title="namespace example">
<list>
  <scalar dictRef="chem:mpt">123</scalar>
  <scalar dictRef="mpt23">123</scalar>
</list>
</stmml>
```
[xsd:string]

The namespace prefix must start with an alpha character and can only contain alphanumeric and '_'. The suffix can have characters from the XML ID specification (alphanumeric, '_', '.' and '-'
Pattern: ([A-Za-z][A-Za-z0-9_]*:)?[A-Za-z][A-Za-z0-9_\.\-]*
[link]

potentialList *[el.potentialList]*

A container for explicit potentials.

Experimental.
Example
```
<--   Li c O  c Buckingham 0.115E+04 0.280     .000E+00
.000E+00   0.000 10.000 --><potential form="gulp:buckingham">
   <arg name="atom1">
     <atom atomTypeRef="gulp:at.li"></atom>
   </arg>
   <arg name="atom2">
     <atom atomTypeRef="gulp:at.o"></atom>
   </arg>
   <arg ref="gulp:buckingham.a">
     <scalar units="units:ev">0.115E+04</scalar>
```

```
    </arg>
    <arg ref="gulp:buckingham.b">
      <scalar units="units:angstrom">0.280</scalar>
    </arg>
    <arg ref="gulp:buckingham.c">
      <scalar units="units:ev">.000E+00</scalar>
    </arg>
  </potential>
```

Content Model of element

(
(potential)
*)

Attributes of element

`title`*[att.title]*

A title on an element.

No controlled value.

Example

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

`id`*[att.id]*

An attribute providing a unique ID for an element.

[link]

`convention`*[att.convention]*

A reference to a convention.

[link]

`dictRef`*[att.dictRef]*

A reference to a dictionary entry.

[link]

`property`*[el.property]*

A container for a property.

`property` can contain one or more children, usually `scalar`, `array` or `matrix`.

The `dictRef` attribute is required, even if there is a single scalar child with the same dictRef. The property may have a different dictRef from the child, thus providing an extension mechanism.

Properties may have a `state` attribute to distinguish the state of matter

Example

```
<molecule>
  <name>buckminsteremptierene</name>
  <list>
  <property title="critical temperature"
dictRef="prop:crittemp">
    <scalar units="unit:celsius">123</scalar>
  </property>
  </list>
</molecule>
```

Content Model of element

(metadataList*,name*,
(scalar|array|matrix)
*,
(parameter)
)

Attributes of element

`dictRef` *[att.dictRef]*

A reference to a dictionary entry.

[link]

`convention` *[att.convention]*

A reference to a convention.

[link]

`title` *[att.title]*

A title on an element.

No controlled value.

Example

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

`id` *[att.id]*

An attribute providing a unique ID for an element.

[link]

`ref` *[att.ref]*

A reference to an element of given type.

[link]

`role` *[property.att.role]*

The role of the property. Semantics are not yet controlled but could include thermochemistry, kinetics or other common properties.

`state` *[property.att.state]*

State of a substance or property.

The state(s) of matter appropriate to a substance or property. It follows a partially controlled vocabulary. It can be extended through namespace codes to dictionaries

Example

Allowed values

- aqueous

  An aqueous solution
- gas

  Gas or vapor. The default state for computation on isolated molecules
- glass

  A glassy state
- liquid

  Normally pure liquid (use solution where appropriate)
- nematic

  The nematic phase
- smectic

  The smectic phase
- solid

  A solid
- solidSolution

  A solid solution
- solution

  A (liquid) solution

[link]

propertyList *[el.propertyList]*

A container for one or more properties.

`propertyList` can contain several properties. These include (but are not limited to observations, or numeric quantities.

Example

```
<molecule>
  <name>penguinone</name>
  <list>
  <propertyList>
    <property title="colour">
      <scalar dictRef="prop:color">black</scalar>
    </property>
    <property title="electric dipole moment">
      <scalar dictRef="ccml:edipole"
units="unit:debye">1.23</scalar>
    </property>
  </propertyList>
  </list>
</molecule>
```

Content Model of element

(metadataList*,name*,
(property|observation)
*)

Attributes of element

dictRef *[att.dictRef]*

A reference to a dictionary entry.

[link]

convention *[att.convention]*

A reference to a convention.

[link]

title *[att.title]*

A title on an element.

No controlled value.

Example

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

```
</stmml>
```

id*[att.id]*
An attribute providing a unique ID for an element.
[link]

ref*[att.ref]*
A reference to an element of given type.
[link]

role*[]*

The role of the propertyList. Semantics are not yet controlled but could include thermochemistry, kinetics or other common properties.

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.

Example

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

Content Model of element

[xsd:string]

title*[att.title]*

A title on an element.

No controlled value.

Example

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

id*[att.id]*

An attribute providing a unique ID for an element.

[link]

convention*[att.convention]*

A reference to a convention.

[link]

dictRef*[att.dictRef]*

A reference to a dictionary entry.

[link]

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

The dataType of an (simple) element or attribute.

[link]

errorValue*[att.errorValue]*

[link]

errorBasis*[att.errorBasis]*

[link]

min*[att.min]*

[link]

max*[att.max]*

[link]

units*[att.units]*

Scientific units on an element.

[link]

symmetry*[el.symmetry]*

Molecular, crystallographic or other symmetry.

`symmetry` provides a label and/or symmetry operations for molecules or crystals. Point and spacegroups can be specified by strings, though these are not enumerated, because of variability in syntax (spaces, case-sensitivity, etc.), potential high symmetries (e.g. TMV disk is D17) and non-standard spacegroup settings. Provision is made for explicit symmetry operations through <matrix> child elements.

By default the axes of symmetry are defined by the symbol - thus C2v requires z to be the unique axis, while P21/c requires b/y. Spacegroups imply the semantics defined in International Tables for Crystallography, (Int Union for Cryst., Munksgaard). Point groups are also defined therein.

The element may also be used to give a label for the symmetry species (irreducible representation) such as "A1u" for a vibration or orbital.

The matrices should be 3x3 for point group operators and 3x4 for spacegroup operators. The use of crystallographic notation ("x,1/2+y,-z") is not supported - this would be <matrix>1 0 0 0.0 0 1 0 0.5 0 0 1 0.0<matrix>.

The default convention for point group symmetry is `Schoenflies` and for spacegroups is "H-M". Other conventions (e.g. "Hall") must be specfied through the `convention` attribute.

This element implies that the Cartesians or fractional coordinates in a molecule are oriented appropriately. In some cases it may be useful to specify the symmetry of an arbitarily oriented molecule and the <molecule> element has the attribute `symmetryOriented` for this purpose.

Example

```
<cml title="symmetry example 1">
<symmetry pointGroup="C2v" id="s1">
  <matrix id="e" rows="3" columns="3" dataType="xsd:float"
matrixType="rotation33">
    1 0 0
    0 1 0
    0 0 1</matrix>
  <matrix id="c2" rows="3" columns="3" dataType="xsd:float"
matrixType="rotation33">
    -1 0 0
    0 -1 0
    0 0 1</matrix>
  <matrix id="sx" rows="3" columns="3" dataType="xsd:float"
matrixType="rotation33">
    -1 0 0
    0 1 0
    0 0 1</matrix>
  <matrix id="sy" rows="3" columns="3" dataType="xsd:float"
matrixType="rotation33">
    1 0 0
    0 -1 0
    0 0 1</matrix>
</symmetry>
</cml>
```

Content Model of element

(matrix*)

Attributes of element

`dictRef`*[att.dictRef]*

A reference to a dictionary entry.
[link]

`convention`*[att.convention]*
A reference to a convention.
[link]

`title`*[att.title]*

A title on an element.

No controlled value.

Example

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

`id`*[att.id]*
An attribute providing a unique ID for an element.
[link]

`pointGroup`*[]*

A point group.

No fixed semantics, though Schoenflies is recommended over Hermann-Mauguin. We may provide a controlled-extensible list in the future.

`spaceGroup`*[]*

A point group.

No fixed semantics, though Hermann-Mauguin or Hall is recommended over Schoenflies. We may provide a controlled-extensible list in the future.

`irreducibleRepresentation`*[]*

A symmetry species.

No fixed semantics, though we may provide a controlled-extensible list in the future.

number*[]*

2003-03-30: added number attribute

The rotational symmetry number

Used for calculation of entropy, etc.

torsion*[el.torsion]*

A torsion angle ("dihedral") between 4 distinct atoms.

The atoms need not be formally bonded. It can be used for:

- Recording experimentally determined torsion angles (e.g. in a crystallographic paper).
- Providing the torsion component for internal coordinates (e.g. z-matrix).

Note that the order of atoms is important.

Example

```
<molecule id="m1">
  <atomArray atomID="a1 a2 a3 a4"></atomArray>
  <torsion atomRefs4="a4 a2 a3 a1"
units="degrees">123</torsion>
</molecule>
```

Content Model of element

torsionAngleType[st.torsionAngleType]

The type of a torsion angle.

[xsd:float]
minInclusive: -360.0
maxInclusive: 360.0

title*[att.title]*

A title on an element.

No controlled value.

Example

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

id*[att.id]*

An attribute providing a unique ID for an element.

[link]

convention*[att.convention]*
A reference to a convention.
[link]

dictRef*[att.dictRef]*

A reference to a dictionary entry.
[link]

atomRefs4*[att.atomRefs4]*
A list of 4 references to atoms.
[link]

units*[att.units]*
Scientific units on an element.
[link]

errorValue*[att.errorValue]*
[link]

errorBasis*[att.errorBasis]*
[link]

min*[att.min]*
[link]

max*[att.max]*
[link]

ref*[att.ref]*
A reference to an element of given type.
[link]

zMatrix*[el.zMatrix]*

A zMatrix.

A container for `length`, `angle` and `torsion`, which must be arranged in the conventional zMatrix format.
Example

```
<molecule id="mol1" instance="initial">
```

```
      <atomArray>
        <atom id="a1" elementType="C"></atom>
        <atom id="a2" elementType="C"></atom>
        <atom id="a3" elementType="C"></atom>
        <atom id="a4" elementType="C"></atom>
      </atomArray>
      <zMatrix>
      <length id="len1" atomRefs2="a2 a1">1.42450</length>
      <length id="len2" atomRefs2="a3 a2">1.42450</length>
      <angle id="ang2" atomRefs3="a3 a2 a1">120.00000</angle>
      <length id="len3" atomRefs2="a4 a3">1.42450</length>
      <angle id="ang3" atomRefs3="a4 a3 a2">120.00000</angle>
      <torsion id="tor3" atomRefs4="a4 a3 a2
a1">0.00000</torsion>
      </zMatrix>
    </molecule>
```

Content Model of element

(
(length|angle|torsion)
*)

Attributes of element

`title`*[att.title]*

A title on an element.

No controlled value.

Example

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

`id`*[att.id]*

An attribute providing a unique ID for an element.

[link]

`convention`*[att.convention]*

A reference to a convention.

[link]

`dictRef`*[att.dictRef]*

A reference to a dictionary entry.

[link]

ATTRIBUTES

atomRef *[att.atomRef]*

A reference to an existing atom.

Example

```
<cml title="atomRef example">
  <molecule id="m1">
    <atomArray>
      <atom id="a1"></atom>
    </atomArray>
    <electron id="e1" atomRef="a1"></electron>
  </molecule>
</cml>
```

[xsd:string]
Pattern: \s*[A-Za-z_][A-Za-z0-9\-:_]*\s*

A reference to an atom.

Typical use would be a bond with only one atom (e.g. the other end is to a bond or electrons).

atomRefArray *[att.atomRefArray]*

An array of atomRefs.

The atomRefs cannot be schema- or schematron-validated. Instances of this type will be used in array-style representation of bonds and atomParitys. It can also be used for arrays of atomIDTypes such as in complex setereochemistry, geometrical definitions, atom groupings, etc.

Example

```
<cml title="atomArray example">
  <molecule id="m1">
    <atomArray atomID="a2 a4 a6" elementType="O N
S"></atomArray>
  </molecule>
</cml>
```

XSD:LIST of atomRefType

A reference to an existing atom.

Example

```
<cml title="atomRef example">
  <molecule id="m1">
    <atomArray>
      <atom id="a1"></atom>
    </atomArray>
    <electron id="e1" atomRef="a1"></electron>
  </molecule>
</cml>
```

[xsd:string]

Pattern: \s*[A-Za-z_][A-Za-z0-9\-:_]*\s*

An array of references to atoms.

Typical use would be to atoms defining a plane.

---

atomRefs2 *[att.atomRefs2]*

A reference to two distinct existing atoms in order.

Example

```
<cml title="atomRefs2 example">
  <molecule id="m1">
    <atomArray>
      <atom id="a1"></atom>
      <atom id="a2"></atom>
    </atomArray>
    <bondArray>
      <bond atomRefs2="a1 a2"></bond>
    </bondArray>
  </molecule>
</cml>
```

[xsd:string]

Pattern: \s*\S+\s+\S+\s*

A list of two references to atoms.

Typically used for defining bonds.

---

atomRefs3 *[att.atomRefs3]*

A reference to three distinct existing atoms in order.

Example

```
<cml title="atomRefs3 example">
  <molecule id="m1">
    <atomArray>
      <atom id="a1"></atom>
      <atom id="a2"></atom>
```

```
      <atom id="a3"></atom>
    </atomArray>
    <angle atomRefs3="a1 a2 a3" units="degrees">123.4</angle>
  </molecule>
</cml>
```

[xsd:string]
Pattern: \s*(\S+\s+){2}\S+\s*

A list of three references to atoms.

Typically used for defining angles, but could also be used to define a three-centre bond.

`atomRefs4` *[att.atomRefs4]*

A reference to four distinct existing atoms in order.

Example

```
<cml title="atomRefs4 example">
  <molecule id="m1">
    <atomArray>
      <atom id="a1"></atom>
      <atom id="a2"></atom>
      <atom id="a3"></atom>
     <atom id="a4"></atom>
    </atomArray>
    <torsion atomRefs4="a1 a2 a3 a4"
units="degrees">123.4</torsion>
  </molecule>
</cml>
```

[xsd:string]
Pattern: \s*(\S+\s+){3}\S+\s*

A list of 4 references to atoms.

Typically used for defining torsions and atomParities, but could also be used to define a four-centre bond.

`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.

Example

```
<stmml title="namespace example">
```

```
<list>
  <scalar dictRef="chem:mpt">123</scalar>
  <scalar dictRef="mpt23">123</scalar>
</list>
</stmml>
```

[xsd:string]

The namespace prefix must start with an alpha character and can only contain alphanumeric and '_'. The suffix can have characters from the XML ID specification (alphanumeric, '_', '.' and '-'

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.

Example

```
<bond convention="fooChem" order="-5"></bond>
```

---

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.

Example

```
<stmml title="dataType example">
<list>
  <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>
</stmml>
```

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.

Example

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

 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.

Example

```
<stmml title="namespace example">
<list>
  <scalar dictRef="chem:mpt">123</scalar>
  <scalar dictRef="mpt23">123</scalar>
</list>
</stmml>
```

[xsd:string]

The namespace prefix must start with an alpha character and can only contain alphanumeric and '_'. The suffix can have characters from the XML ID specification (alphanumeric, '_', '.' and '-'

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

Example

```
<stmml title="dictRef example">
<scalar dataType="xsd:float" title="surfaceArea"
dictRef="cmlPhys:surfArea" units="units:cm2">50</scalar>
  </stmml>
```

Example

```
<stmml title="dictRef example 2">
<stm:list>
  <stm:observation>
    <p>We observed <object count="3"
dictRef="foo:p1"></object>
      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>
</stmml>
```

`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

Example

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

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**.

Example

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

[xsd:float]

`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-z][A-Za-z0-9_-]*:)?[A-Za-z][A-Za-z0-9_\-\.]*

An attribute providing a unique ID for an element.

`id` *[att.mandatoryId]* 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-z][A-Za-z0-9_-]*:)?[A-Za-z][A-Za-z0-9_\-\.]*

An attribute providing a mandatory unique ID for an element.

This is a horrible hack. It should be possible to add 'required' to the attributeGroup where used... (Maybe it is and I am still fighting Schema Wars)

`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

Example

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

[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

Example

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

[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-z][A-Za-z0-9_-]*:)?[A-Za-z][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.

Example

```
<stmml title="ref example">
<cml>
  <molecule id="m1">
    <atomArray>
      <atom elementType="N"></atom>
      <atom elementType="O"></atom>
```

```
    </atomArray>
  </molecule>
  <h:p>The action of <molecule ref="#m1"></molecule> on
cardiac muscle ...</h:p>
</cml>
</stmml>
```

`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:nonNegativeInteger]

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

`title`*[att.title]*

A title on an element.

No controlled value.

Example

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

`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**

Example

```
<stmml title="unitList example">
<stm:unitList>
<stm:unitType id="length" name="length">
  <stm:dimension name="length" power="1"></stm:dimension>
</stm:unitType>
<stm:unitType id="time" name="time">
```

```
  <stm:dimension name="time" power="1"></stm:dimension>
</stm:unitType>
<stm:unitType id="dimensionless" name="dimensionless">
  <stm:dimension name="dimensionless"
power="1"></stm:dimension>
</stm:unitType>
<stm:unitType id="acceleration" name="acceleration">
  <stm:dimension name="length" power="1"></stm:dimension>
  <stm:dimension name="time" power="-2"></stm:dimension>
</stm:unitType>
<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>
<stm:unit id="newton" name="newton" unitType="force">
  <stm:description>The SI unit of force</stm:description>
</stm:unit>
<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>
<stm:unit id="inch" name="inch" parentSI="meter"
abbreviation="in" multiplierToSI="0.0254">
  <stm:description>An imperial measure of
length</stm:description>
</stm:unit>
<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>
</stmml>
```

[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.

`units`*[att.angleUnits]*

an enumeration of allowed angle units.

Example

Allowed values

- degrees
- radians

Restricts units to radians or degrees.

SIMPLETYPES

angleUnitsType[st.angleUnitsType]

an enumeration of allowed angle units.

Example

Allowed values

- degrees
- radians

atomIDType[st.atomIDType]

An identifier for an atom.

Of the form prefix:suffix where prefix and suffix are purely alphanumeric (with _ and -) and prefix is optional. This is similar to XML IDs (and we promote this as good practice for atomIDs. Other punctuation and whitespace is forbidden, so IDs from (say) PDB files are not satisfactory.

The prefix is intended to form a pseudo-namespace so that atom IDs in different molecules may have identical suffixes. It is also useful if the prefix is the ID for the molecule (though this clearly has its limitation). Atom IDs should not be typed as XML IDs since they may not validate.

Example

```
<cml title="example of IDs on atoms">
  <molecule id="m1">
    <atomArray>
      <atom id="a2" elementType="O"></atom>
    </atomArray>
  </molecule>
  <molecule id="m2">
    <atomArray>
```

```
      <atom id="a2" elementType="O"></atom>
    </atomArray>
  </molecule>
</cml>
```

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

---

atomRefArrayType[st.atomRefArrayType]

An array of atomRefs.

The atomRefs cannot be schema- or schematron-validated. Instances of this type will be used in array-style representation of bonds and atomParitys. It can also be used for arrays of atomIDTypes such as in complex setereochemistry, geometrical definitions, atom groupings, etc.

Example

```
<cml title="atomArray example">
  <molecule id="m1">
    <atomArray atomID="a2 a4 a6" elementType="O N
S"></atomArray>
  </molecule>
</cml>
```

XSD:LIST of atomRefType

A reference to an existing atom.

Example

```
<cml title="atomRef example">
  <molecule id="m1">
    <atomArray>
      <atom id="a1"></atom>
    </atomArray>
    <electron id="e1" atomRef="a1"></electron>
  </molecule>
</cml>
```

[xsd:string]
Pattern: \s*[A-Za-z_][A-Za-z0-9\-:_]*\s*

---

atomRefs2Type[st.atomRefs2Type]

A reference to two distinct existing atoms in order.

Example

```
<cml title="atomRefs2 example">
  <molecule id="m1">
    <atomArray>
      <atom id="a1"></atom>
      <atom id="a2"></atom>
    </atomArray>
```

```
    <bondArray>
      <bond atomRefs2="a1 a2"></bond>
    </bondArray>
  </molecule>
</cml>
```

[xsd:string]
Pattern: \s*\S+\s+\S+\s*

---

atomRefs3Type[st.atomRefs3Type]

A reference to three distinct existing atoms in order.
Example

```
<cml title="atomRefs3 example">
  <molecule id="m1">
    <atomArray>
      <atom id="a1"></atom>
      <atom id="a2"></atom>
      <atom id="a3"></atom>
    </atomArray>
    <angle atomRefs3="a1 a2 a3" units="degrees">123.4</angle>
  </molecule>
</cml>
```

[xsd:string]
Pattern: \s*(\S+\s+){2}\S+\s*

---

atomRefs4Type[st.atomRefs4Type]

A reference to four distinct existing atoms in order.
Example

```
<cml title="atomRefs4 example">
  <molecule id="m1">
    <atomArray>
      <atom id="a1"></atom>
      <atom id="a2"></atom>
      <atom id="a3"></atom>
     <atom id="a4"></atom>
    </atomArray>
    <torsion atomRefs4="a1 a2 a3 a4"
units="degrees">123.4</torsion>
  </molecule>
</cml>
```

[xsd:string]
Pattern: \s*(\S+\s+){3}\S+\s*

---

atomRefType[st.atomRefType]

A reference to an existing atom.

Example

```
<cml title="atomRef example">
  <molecule id="m1">
    <atomArray>
      <atom id="a1"></atom>
    </atomArray>
    <electron id="e1" atomRef="a1"></electron>
  </molecule>
</cml>
```

[xsd:string]

Pattern: \s*[A-Za-z_][A-Za-z0-9\-:_]*\s*

---

bondRefArrayType[st.bondRefArrayType]

An array of references to bonds.

The references cannot (yet) cannot be schema- or schematron-validated. Instances of this type will be used in array-style representation of electron counts, etc. It can also be used for arrays of bondIDTypes such as in complex stereochemistry, geometrical definitions, bond groupings, etc.

Example

XSD:LIST of bondRefType

A reference to an existing bond.

A reference to a bond may be made by atoms (e.g. for multicentre or pi-bonds), electrons (for annotating reactions or describing electronic properties) or possibly other bonds (no examples yet). The semantics are relatively flexible.

Example

```
<cml title="bondArray example">
  <bondArray>
    <bond id="b1" atomRefs2="a3 a8" order="D">
      <electron bondRef="b1"></electron>
      <bondStereo>C</bondStereo>
    </bond>
    <bond id="b2" atomRefs2="a3 a8" order="S">
      <bondStereo convention="MDL"
conventionValue="6"></bondStereo>
    </bond>
  </bondArray>
</cml>
```

[xsd:string]

Pattern: \s*\S+\s*

---

bondRefType[st.bondRefType]

A reference to an existing bond.

A reference to a bond may be made by atoms (e.g. for multicentre or pi-bonds), electrons (for annotating reactions or describing electronic properties) or possibly other bonds (no examples yet). The semantics are relatively flexible.

Example

```
<cml title="bondArray example">
  <bondArray>
    <bond id="b1" atomRefs2="a3 a8" order="D">
      <electron bondRef="b1"></electron>
      <bondStereo>C</bondStereo>
    </bond>
    <bond id="b2" atomRefs2="a3 a8" order="S">
      <bondStereo convention="MDL"
conventionValue="6"></bondStereo>
    </bond>
  </bondArray>
</cml>
```

[xsd:string]
Pattern: \s*\S+\s*

---

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

Example

```
<stmml title="coordinate2Type example">
<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>
</stmml>
```

[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
Example

```
<stmml title="coordinate3Type example">
<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>
</stmml>
```

[xsd:string]
Pattern: \s*([\-]|[+])?\d*\.?\d*(\s+|[,])([\-]|[+])?\d*\.?\d*(\s+|[,])([\-]|[+])?\d*\.?\d*\s*

---

coordinateComponentArrayType[st.coordinateComponentArrayType]

An array of coordinateComponents for a single coordinate.

An array of coordinateComponents for a single coordinate where these all refer to an X-coordinate (NOT x,y,z) Instances of this type will be used in array-style representation of 2-D or 3-D coordinates.

Currently no machine validation

Currently not used in STMML, but re-used by CML (see example)
Example

```
<stmml title="coordinateComponentArrayType">
<cml:atomArray x2="1.2 2.3 4.5 6.7"></cml:atomArray>
</stmml>
```

XSD:LIST of xsd:float

---

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.
Example

```
<stmml title="countType example">
<list>
<object title="frog" count="10"></object>
<action title="step3" count="3">
```

```
  <p>Add 10 ml reagent</p>
</action>
</list>
</stmml>
```

[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.

Example

```
<stmml title="dataType example">
<list>
  <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>
</stmml>
```

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.

Example

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

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

[xsd:string]

---

elementTypeArrayType[st.elementTypeArrayType]

An array of elementTypes.

Instances of this type will be used in array-style representation of atoms.

Example

```
<cml title="atomArray with elementTypes">
  <atomArray elementType="O N S Pb"></atomArray>
</cml>
```

XSD:LIST of elementTypeType

Allowed `elementType` values.

The periodic table (up to element number 118. In addition the following strings are allowed:

- `Du`. ("dummy") This does not correspond to a "real" atom and can support a point in space or within a chemical graph.
- `R`. ("R-group") This indicates that an atom or group of atoms could be attached at this point.

Example

```
<cml title="elementType example">
  <atomArray>
    <atom id="a1" elementType="C"></atom>
```

```
    <atom id="a2" elementType="N"></atom>
    <atom id="a3" elementType="Pb"></atom>
    <atom id="a4" elementType="Dummy"></atom>
  </atomArray>
</cml>
```

UNION OF

Allowed values

- Ac
- Al
- Ag
- Am
- Ar
- As
- At
- Au
- B
- Ba
- Bh
- Bi
- Be
- Bk
- Br
- C
- Ca
- Cd
- Ce
- Cf
- Cl
- Cm
- Co
- Cr
- Cs
- Cu
- Db
- Dy
- Er
- Es
- Eu
- F
- Fe
- Fm
- Fr
- Ga
- Gd
- Ge
- H

  Any isotope of hydrogen.

There are no special element symbols for D and T which should use the `isotope` attribute.

- He
- Hf
- Hg
- Ho
- Hs
- I
- In
- Ir
- K
- Kr
- La
- Li
- Lr
- Lu
- Md
- Mg
- Mn
- Mo
- Mt
- N
- Na
- Nb
- Nd
- Ne
- Ni
- No
- Np
- O
- Os
- P
- Pa
- Pb
- Pd
- Pm
- Po
- Pr
- Pt
- Pu
- Ra
- Rb
- Re
- Rf
- Rh
- Rn
- Ru
- S

- Sb
- Sc
- Se
- Sg
- Si
- Sm
- Sn
- Sr
- Ta
- Tb
- Tc
- Te
- Th
- Ti
- Tl
- Tm
- U
- Uun
- Uuu
- Uub
- Uut
- Uuq
- Uup
- Uuh
- Uus
- Uuo
- V
- W
- Xe
- Y
- Yb
- Zn
- Zr
- Dummy
- Du

  A point or object with no chemical semantics.

  Examples can be centroids, bond-midpoints, orienting "atoms" in small z-matrices.

  Note "Dummy" has the same semantics but is now deprecated.
- R

  A point at which an atom or group might be attached.

  Examples are abbreviated organic functional groups, Markush representations, polymers, unknown atoms, etc. Semantics may be determined by the `role` attribute on the atom.

[xsd:string]
Pattern: [A-Za-z]+:[A-Za-z][A-Za-z0-9\-]+

---

elementTypeType[st.elementTypeType]

Allowed `elementType` values.

The periodic table (up to element number 118. In addition the following strings are allowed:

- `Du`. ("dummy") This does not correspond to a "real" atom and can support a point in space or within a chemical graph.
- `R`. ("R-group") This indicates that an atom or group of atoms could be attached at this point.

Example

```
<cml title="elementType example">
  <atomArray>
    <atom id="a1" elementType="C"></atom>
    <atom id="a2" elementType="N"></atom>
    <atom id="a3" elementType="Pb"></atom>
    <atom id="a4" elementType="Dummy"></atom>
  </atomArray>
</cml>
```

UNION OF
Allowed values

- Ac
- Al
- Ag
- Am
- Ar
- As
- At
- Au
- B
- Ba
- Bh
- Bi
- Be
- Bk
- Br
- C
- Ca
- Cd
- Ce
- Cf
- Cl
- Cm
- Co

- Cr
- Cs
- Cu
- Db
- Dy
- Er
- Es
- Eu
- F
- Fe
- Fm
- Fr
- Ga
- Gd
- Ge
- H

  Any isotope of hydrogen.

  There are no special element symbols for D and T which should use the `isotope` attribute.
- He
- Hf
- Hg
- Ho
- Hs
- I
- In
- Ir
- K
- Kr
- La
- Li
- Lr
- Lu
- Md
- Mg
- Mn
- Mo
- Mt
- N
- Na
- Nb
- Nd
- Ne
- Ni
- No
- Np
- O

- Os
- P
- Pa
- Pb
- Pd
- Pm
- Po
- Pr
- Pt
- Pu
- Ra
- Rb
- Re
- Rf
- Rh
- Rn
- Ru
- S
- Sb
- Sc
- Se
- Sg
- Si
- Sm
- Sn
- Sr
- Ta
- Tb
- Tc
- Te
- Th
- Ti
- Tl
- Tm
- U
- Uun
- Uuu
- Uub
- Uut
- Uuq
- Uup
- Uuh
- Uus
- Uuo
- V
- W
- Xe
- Y
- Yb

- Zn
- Zr
- Dummy
- Du

  A point or object with no chemical semantics.

  Examples can be centroids, bond-midpoints, orienting "atoms" in small z-matrices.

  Note "Dummy" has the same semantics but is now deprecated.
- R

  A point at which an atom or group might be attached.

  Examples are abbreviated organic functional groups, Markush representations, polymers, unknown atoms, etc. Semantics may be determined by the `role` attribute on the atom.

[xsd:string]
Pattern: [A-Za-z]+:[A-Za-z][A-Za-z0-9\-]+

---

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

Example

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

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**.

Example

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

[xsd:float]

---

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.

Example

```
<atomArray x2="1.2 2.3 3.4 5.6"></atomArray>
```

XSD:LIST of xsd:float

---

formalChargeType[st.formalChargeType]

The formal charge on an atom.

Used for electron-bookeeping. This has no relation to its calculated (fractional) charge.

Example

```
<cml title="formalCharge example">
  <atomArray>
    <atom id="a1" elementType="N" formalCharge="+1"></atom>
    <atom id="a2" elementType="O" formalCharge="-1"></atom>
  </atomArray>
</cml>
```

[xsd:integer]

---

formulaType[st.formulaType]

A concise representation for a molecular formula.

This MUST adhere to a whitespaced syntax so that it is trivially machine-parsable. Each element is followed by its count, and the string is optionally ended by a formal charge. NO brackets or other nesting is allowed.

Example

```
<cml title="formulaType example (concise)">
  <list>
    <formula id="methane" concise="C 1 H 4"></formula>
    <formula id="chloroacetate" concise="Cl 1 H 2 C 2 O 2
-1"></formula>
    <formula id="sodiumSulfate">
      <formula concise="H 2 O 1" count="10"></formula>
      <formula concise="Na 1 +1" count="2"></formula>
      <formula concise="S 1 O 4 -2"></formula>
    </formula>
  </list>
</cml>
```

[xsd:string]

Pattern: \s*([A-Z][a-z]?\s+[1-9][0-9]*)(\s+[A-Z][a-z]?\s+[1-9][0-9]*)*(\s+[-|+]?[0-9]+)?\s*

---

hydrogenCountType[st.hydrogenCountType]

The total number of hydrogen atoms bonded to an atom.

The total number of hydrogen atoms bonded to an atom, whether explicitly included as atoms or not. It is an error to have hydrogen count less than the explicit hydrogen count. There is no default value and no assumptions about hydrogen Count can be made if it is not given.

If hydrogenCount is given on every atom, then the values can be summed to give the total hydrogenCount for the (sub)molecule. Because of this hydrogenCount should not be used where hydrogen atoms bridge 2 or more atoms.

Example

```
<cml title="single atom example">
<atom id="a1" title="O3'" elementType="O" formalCharge="1"
hydrogenCount="1" isotope="17" occupancy="0.7" x2="1.2"
y2="2.3" x3="3.4" y3="4.5" z3="5.6" convention="ABC"
dictRef="chem:atom">
  <scalar title="dipole" dictRef="d:dip"
units="units:debye">0.2</scalar>
  <atomParity atomRefs4="a3 a7 a2 a4">1</atomParity>
  <electron id="e1" atomRef="a1" count="2"></electron>
</atom>
</cml>
```

[xsd:nonNegativeInteger]

---

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-z][A-Za-z0-9_-]*:)?[A-Za-z][A-Za-z0-9_\-\.]*

---

integerArrayType[st.integerArrayType]

An array of integers.

An array of integers; for re-use by other schemas

Not machine-validatable

Example

```
<stmml title="integerArray type">
<atomArray hydrogenCount="3 1 0 2"></atomArray>
</stmml>
```

XSD:LIST of xsd:integer

---

isotopeType[st.isotopeType]

The numeric representation of an isotope.

In core CML this represents a single number; either the combined proton/neutron count or a more accurate estimate of the nuclear mass. This is admittedly fuzzy, and requires a more complex object (which can manage conventions, lists of isotopic masses, etc.) See **isotope**.

The default is "natural abundance" - whatever that can be interpreted as.

Delta values (i.e. deviations from the most abundant istopic mass) are never allowed.

[xsd:float]
minInclusive: 0.0
maxInclusive: 99999999999.0

---

matrixType[st.matrixType]

Allowed **matrix** types.

Allowed `matrix` types. These are mainly square matrices
Example

```
<stmml title="matrix example">
<matrix id="m1" title="matrix-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>
</stmml>
```

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

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.

Example

```
<stmml title="namespace example">
<list>
  <scalar dictRef="chem:mpt">123</scalar>
  <scalar dictRef="mpt23">123</scalar>
</list>
</stmml>
```

[xsd:string]

The namespace prefix must start with an alpha character and can only contain alphanumeric and '_'. The suffix can have characters from the XML ID specification (alphanumeric, '_', '.' and '-'

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

Example

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

[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. Users can create their own metadata names using the namespaced prefix syntax (e.g.

foo:institution). Ideally these names should be defined in an STMML dictionary.

2003-03-05: Added UNION to manage non-controlled names

UNION OF

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

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.

Example

```
<stmml title="namespace example">
<list>
  <scalar dictRef="chem:mpt">123</scalar>
  <scalar dictRef="mpt23">123</scalar>
</list>
</stmml>
```

[xsd:string]

The namespace prefix must start with an alpha character and can only contain alphanumeric and '_'. The suffix can have characters from the XML ID specification (alphanumeric, '_', '.' and '-'

Pattern: ([A-Za-z][A-Za-z0-9_]*:)?[A-Za-z][A-Za-z0-9_\.\-]*

---

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

Example

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

[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.

Example

```
<stmml title="namespace example">
<list>
  <scalar dictRef="chem:mpt">123</scalar>
  <scalar dictRef="mpt23">123</scalar>
</list>
</stmml>
```

[xsd:string]

The namespace prefix must start with an alpha character and can only contain alphanumeric and '_'. The suffix can have characters from the XML ID specification (alphanumeric, '_', '.' and '-'

Pattern: ([A-Za-z][A-Za-z0-9_]*:)?[A-Za-z][A-Za-z0-9_\.\-]*

nonHydrogenCountType[st.nonHydrogenCountType]

The number of non-hydrogen atoms attached to an atom.

Obsolete in core CML. Only useful in CML queries
[xsd:nonNegativeInteger]

nonNegativeAngleType[st.nonNegativeAngleType]

A non-signed angle, such as a bond angle. Note that we also provide positiveAngleType (e.g. for cell angles) and torsionAngleType for - guess what - **torsion**.

Re-used by **angle**
Example
```
<angle units="degrees" atomRefs3="a1 a2 a3">123.4</angle>
```
[xsd:float]
minInclusive: 0.0
maxInclusive: 180.0

occupancyType[st.occupancyType]

Occupancy of an atomic site.

Primarily for crystallography. Values outside 0-1 are not allowed.
Example
```
See atom.
```
[xsd:float]
minInclusive: 0
maxInclusive: 1

orderArrayType[st.orderArrayType]

An array of bond orders.

(seeAlso **orderType**)
XSD:LIST of orderType

Bond order (as a string).

This is purely conventional and used for bond/electron counting. There is no default value. The emptyString attribute can be used to indicate a bond of unknown or unspecified type. The interpretation of this is outside the scope of CML-based algorithms. It may be accompanied by a `convention` attribute on the `bond` which links to a dictionary. Example: `<bond convention="ccdc:9" atomRefs2="a1 a2"/>` could represent a delocalised bond in the CCDC convention.

Allowed values

- S

  Single bond.
- 1

  Single bond.
- D

  Double bond.
- 2

  Double bond.
- T

  Triple bond.
- 3

  Triple bond.
- A

  Aromatic bond.

---

orderType[st.orderType]

Bond order (as a string).

This is purely conventional and used for bond/electron counting. There is no default value. The emptyString attribute can be used to indicate a bond of unknown or unspecified type. The interpretation of this is outside the scope of CML-based algorithms. It may be accompanied by a `convention` attribute on the `bond` which links to a dictionary. Example: `<bond convention="ccdc:9" atomRefs2="a1 a2"/>` could represent a delocalised bond in the CCDC convention.

Allowed values

- S

  Single bond.

- 1

  Single bond.
- D

  Double bond.
- 2

  Double bond.
- T

  Triple bond.
- 3

  Triple bond.
- A

  Aromatic bond.

---

particleRefType[st.particleRefType]

A reference to an existing particle.
Example

```
<cml title="particleRef example">
  <molecule id="m1">
    <atomArray>
      <atom id="a1">
        <particle id="p1" particleType="pp1"></particle>
      </atom>
    </atomArray>
  </molecule>
</cml>
```

[xsd:string]
Pattern: \s*[A-Za-z_][A-Za-z0-9\-:_]*\s*

---

positiveAngleType[st.positiveAngleType]

A non-signed angle, such as a cell angle. Note that we also provide nonNegativeAngleType (e.g. for bond angles).

Re-used by **crystal**
Example

```
<cml title="positiveAngleType example">
  <list>
    <scalar title="alpha" units="units:degree">70.123</scalar>
```

```
    <scalar title="beta" units="units:degree">80.456</scalar>
    <scalar title="gamma" units="units:degree">90.789</scalar>
  </list>
</cml>
```

[xsd:float]
minExclusive: 0.0
maxInclusive: 180.0

---

positiveNumberType[st.positiveNumberType]

A positive number. Note that we also provide nonNegativeNumber with inclusive zero. The maximum number is (quite large) since 'unbounded' is more difficult to implement. This is greater than Eddington's estimate of the number of particles in the universe so it should work for most people.

[xsd:double]
minExclusive: 0.0
maxInclusive: 1.0E+99

---

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-z][A-Za-z0-9_-]*:)?[A-Za-z][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:nonNegativeInteger]

---

stateType[st.stateType]

State of a substance or property.

The state(s) of matter appropriate to a substance or property. It follows a partially controlled vocabulary. It can be extended through namespace codes to dictionaries

Example

Allowed values

- aqueous

  An aqueous solution
- gas

  Gas or vapor. The default state for computation on isolated molecules
- glass

  A glassy state
- liquid

  Normally pure liquid (use solution where appropriate)
- nematic

  The nematic phase
- smectic

  The smectic phase
- solid

  A solid
- solidSolution

  A solid solution
- solution

  A (liquid) solution

---

stereoType[st.stereoType]

(Bond) stereochemistry (as a string).

. This is purely conventional; . There is no default value. The emptyString attribute can be used to indicate a bond of unknown or unspecified type. The interpretation of this is outside the scope of CML-based algorithms. It may be accompanied by a `convention` attribute which links to a dictionary

Example

```
<cml title="bondArray example">
  <bondArray>
    <bond id="b1" atomRefs2="a3 a8" order="D">
      <electron bondRef="b1"></electron>
      <bondStereo>C</bondStereo>
    </bond>
    <bond id="b2" atomRefs2="a3 a8" order="S">
      <bondStereo convention="MDL"
conventionValue="6"></bondStereo>
    </bond>
  </bondArray>
</cml>
```

Allowed values

- C

  A cis bond.
- T

  A trans bond.
- W

  A wedge bond.
- H

  A hatch bond.
-

  empty or missing.

---

torsionAngleType[st.torsionAngleType]

The type of a torsion angle.

[xsd:float]
minInclusive: -360.0
maxInclusive: 360.0

---

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**

Example

```
<stmml title="unitList example">
<stm:unitList>
<stm:unitType id="length" name="length">
  <stm:dimension name="length" power="1"></stm:dimension>
</stm:unitType>
<stm:unitType id="time" name="time">
  <stm:dimension name="time" power="1"></stm:dimension>
</stm:unitType>
<stm:unitType id="dimensionless" name="dimensionless">
  <stm:dimension name="dimensionless"
power="1"></stm:dimension>
</stm:unitType>
<stm:unitType id="acceleration" name="acceleration">
  <stm:dimension name="length" power="1"></stm:dimension>
  <stm:dimension name="time" power="-2"></stm:dimension>
</stm:unitType>
<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>
<stm:unit id="newton" name="newton" unitType="force">
  <stm:description>The SI unit of force</stm:description>
</stm:unit>
<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>
<stm:unit id="inch" name="inch" parentSI="meter"
abbreviation="in" multiplierToSI="0.0254">
  <stm:description>An imperial measure of
length</stm:description>
</stm:unit>
<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
```

```
/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>
</stmml>
```

[xsd:string]

---

vector3Type[st.vector3Type]

A vector in 3-space

No constraints on magnitude (i.e. could be zero)

Example

```
<myVector dataType="stm:vector3Type">2.0 3.0 4.0</myVector>
```

[xsd:string]

Pattern: \s*([\-]|[+])?\d*\.?\d*(\s+|[,])([\-]|[+])?\d*\.?\d*(\s+|[,])([\-]|[+])?\d*\.?\d*\s*

MISC