

CML Tools and Information Flow in Atomic Scale Simulations

J. Wakelin

Department of Earth Sciences, University of Cambridge, Downing street, Cambridge CB2 3EQ, UK

P. Murray-Rust, S. Tyrrell, and Y. Zhang

*Unilever Centre for Molecular Science Informatics,
Department of Chemistry, Lensfield Road, Cambridge, CB2 1EW, UK.*

H. S. Rzepa

Department of Chemistry, Imperial College, London, SW7 2AY, UK.

A. García

*Departamento de Física de la Materia Condensada, Facultad de Ciencias,
Universidad del País Vasco, E-48080 Bilbao, Spain.*

High-throughput computation of molecules and crystals is supported through an XML infrastructure based on Chemical Markup Language (CML). Tools have been developed for the automatic creation of job input and the extraction of structured XML information from the output. The approach is generic and supports several languages (Java, C++, Python, FORTRAN) so that "black-box" modules can be created.

I. INTRODUCTION

There are several affordable methods for the computation of solid-state properties of crystalline and other condensed matter systems including:

- *ab initio* quantum mechanics
- Semi-empirical quantum mechanics
- Molecular/crystal mechanics

Typical job times for systems of interest range from minutes to weeks on single processor systems. This means that with today's resources, such as Condor-enabled computers, a very large number of jobs can be run in a few weeks. The limiting factor soon becomes not machines, but the human time for preparation of input and the analysis of results (especially unusual output). If in 10000 jobs 0.1 % show pathological behaviour that is still 100 to be critically analysed. Frequently the scientist will wish to vary parameters or protocols in response to early results. The system therefore has to be easily adaptable if it is to be used in the analysis of large amounts of machine output.

Unfortunately most codes still use their own (often very dated) methods for input and output and there is no commonality between them. Even when the science is well understood it is a major effort to move from one protocol to another. Most logfiles are not critically examined because of the lack of common tools. It is also difficult to compare results obtained by different protocols.

In this article we describe a universal information architecture for atomic scale simulations, based on generic XML tools and the domain-specific Chemical Markup Language (CML) XML schemas [1–5]. By converting (or wrapping) scientific codes, it is possible to provide

a common syntax and semantics for all computational processes and to move towards a common ontology [17]

For molecular or condensed matter simulations this would need to cover at least the following information

- Atomic coordinates
- Lattice vectors
- Forcefields, interatomic potentials
- Basis sets
- Methodology (type of calculation, algorithms, etc.)
- Control parameters (precision, number of iterations, etc)
- Job parameters (time, memory, number of processors, etc.)

The scientist may wish to vary one or more of these over the course of a series of calculations. However, most computational codes do not explicitly support the separation of these inputs. Moreover, the majority of data generated by these codes is written to fixed-format logfiles. Typically these output files are used to report calculated properties, errors and warnings, run times, resource usage and other information. Unfortunately, this method of data representation is not well suited to rapid automated analysis for several reasons. *(i)* Such files often omit "obvious" information (e.g. scientific units, conventions). *(ii)* Details of algorithms and methodology are often implicit. *(iii)* Files often use acronyms and symbols which are only knowable by reading the program code, and *(iv)* most importantly such files are designed to be human readable rather than machine readable.

CML provides not only a technical infrastructure but an incentive to pool and coordinate effort in the creation

of components. We now highlight the main components which are desirable to allow the easy construction of applications in simulations.

II. XML AND CML

A. Schemas

XMLSchemas are the emerging *de facto* approach to describing the semantics of a language. They are a formal statement of the allowed vocabulary, datatypes and document structure in an XML instance. Thus the XMLElement `molecule` is defined in the CML Schema and if misspelt or used in the wrong context would be *invalid*. Using XMLSchema validators, now universally available, a scientist can determine whether the CML input into a program is valid. Validity extends to

- element names (e.g. "atomicBasisFunction") which are case-sensitive.
- attributes (e.g. `id="a123"`), also case sensitive.
- format of data items (e.g. a date can be validated against ISO8601 `yyyy-mm-dd`)
- ranges of (numeric) data items (e.g. a unit cell parameter could be required to be a positive float)
- members of an enumeration (thus CML regards elements not in the IUPAC periodic table as invalid)
- element content. Thus a bond would be *invalid* as a child of `atomArray`

The use of XML validation can dramatic reduce the number of incorrect or garbled inputs and outputs to/from computational codes. Note, however, that a valid XML document may be scientific nonsense. We are creating CML-specific validation tools which will recognise some of the most common problems (e.g. bonds between non-existent atoms).

In many cases XML schemas have a fixed set of tags and attributes, but the CML design is more flexible and allows designers to choose from a set of several hundred components (e.g. for molecules, crystals, reactions, spectra). This allows a community to build a language suited to their particular needs, without being forced into accepting extraneous elements and attribute groups. For the work described in this paper we are working with a particular subset of CML known as CMLComp. One potential subset is shown in figure 1, and in more detail in the Appendix.

B. Dictionaries

In principle an XML language requires the elements to have pre-defined semantics and ontology (i.e. the developers have to know how to code these things and what

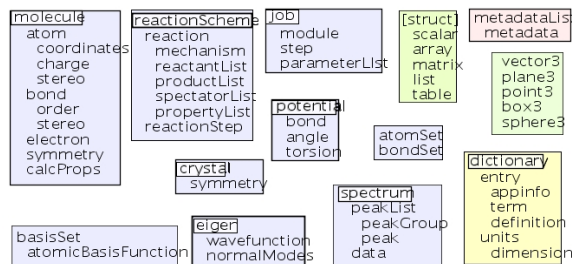


FIG. 1: An example of the kinds of XMLElements available in the CML language. The emphasis is on selecting only those that are relevant to the the user needs

their meanings are). However, in the case of the physical sciences the number of potential concepts is so large that it is neither feasible nor desirable to create a language element for each one. Fortunately, many of the relevant concepts can be abstracted to a small set of generic objects (such as scalar, vector, matrix) and data types (such as float, integer, string). Hence, it is sensible to represent scientific data using generic containers to which additional semantics are added by some other means, such as RDF ontologies [8], or, as in our scheme, by XML based dictionaries. This is desirable because it allows the language to flexibly cover an extremely wide range of concepts without recourse to an unwieldy and ever growing set of elements.

Thus, a concept such as melting point is not hardcoded in the language, instead it is described by a generic element such as `scalar` (see figure 2). The `scalar` element only provides limited information about the data it contains, additional semantics being supplied by the dictionary. An element is linked to a dictionary through its `dictRef` attribute. The dictionary entry has a corresponding `id` attribute that allows it to be referenced.

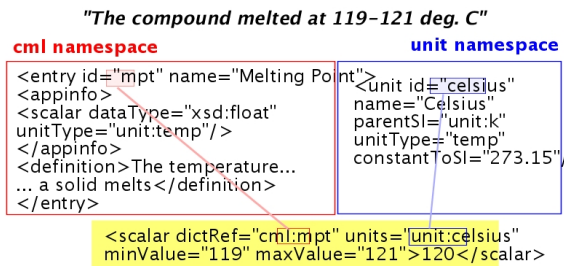


FIG. 2: Describing a "generic" property such as a melting point in CML

The dictionary id/ref scheme is one of the cornerstones of the architecture. Ideally each concept emitted from a program should have a dictionary entry describing its type, use, scientific units, error basis and range, and any

other relevant information. In the simplest case a developer will only create dictionary entries for concepts in their system, and most usually their code (as in figure 3). Later it may be found that there is communal agreement on some of the concepts (definitions and use) in the programs and these can be abstracted into a higher, communal dictionary. This process is well supported in crystallography in the CIF dictionary hierarchy [9]. It is possible for a document to reference multiple dictionaries if necessary (see figure 4)

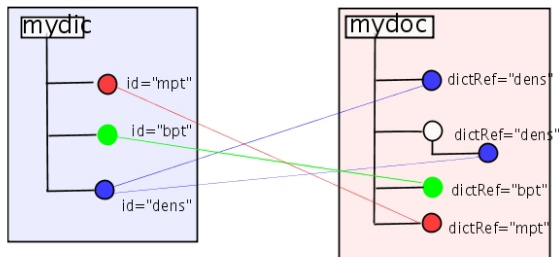


FIG. 3: A dictionary containing multiple entries, and an XML document contain multiple references. Note that many data can be described by the one dictionary entry, here “dens” is referenced twice.

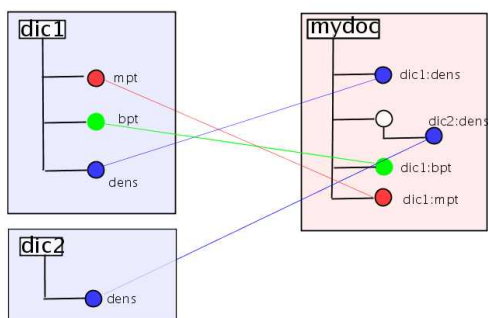


FIG. 4: Here the author has two different meanings for “density” (e.g. volumetric density and charge density). The first would be linked to a dictionary of physical chemistry, the second to one for computational chemistry.

The attraction of this process is that individual groups (e.g. code developers) can create dictionaries for their systems without needing to change the underlying schema.

The borderline between ontologies and dictionaries is flexible as shown by “Gene Ontology” ?? which is currently effectively a thesaurus and dictionary. Like them we can add ontological support to our dictionaries at a later stage and also support their maintenance through tools such as Protege.

III. XML PROCESSING TOOLS

A. Generic XML Processing Tools

There are several generic technologies for processing XML, arguably the three most important being the DOM, the SAX and XSLT.

DOM (the document object model) is an in-memory tree-like representation of an XML document. The DOM allows the user to get, set, add and remove data in an XML document. The entire document is stored in memory meaning that the DOM is not always useful for large processing large documents. There are now DOM implementations available for most programming languages, although at the time we started this work no FORTRAN DOM was available. The majority of theoretical chemistry and physics software is written in FORTRAN, this fact prompted us to develop a FORTRAN 95 DOM implementation (details of which are reported elsewhere [10]).

SAX (the Simple API for XML) allows the user to trap individual XML elements as they are read and process this data with procedural code. The SAX model involves processing an XML document by parts and consequently most SAX implementations have a very small memory footprint. We also provide an implementation of SAX for FORTRAN.

XSLT (eXtensible Stylesheet Language Transformations), allows the user to transform XML documents into other XML or legacy formats. Stylesheet transformations are a particularly efficient way of processing XML when creating displays or rendering. We have a complete set of stylesheets, one for each language element, that allow the user to display data in a natural manner. We also use XSLT for creating the “glueware” that wraps and connects the blackboxes. Examples of XSLT and DOM are discussed in the infrastructure section.

B. Domain Specific XML Processing Tools

CMLDOM, is a domain specific version of the document object model. The CMLDOM has an object for each element in the schema, with explicit methods to get or set the attribute values. Because of the scale of the schemas and the variety of languages we need to use (Java, C++, Python, FORTRAN) the CMLDOM code and its documentation must be generated automatically and packaged into libraries. These are then available for authors who wish to add input/output functionality to their code and also for manipulating and transforming the XML objects. The libraries are enhanced through element-specific tools, which provide additional functionality. Thus `MoleculeTool` adds calculations of molecular mass, chemical structure diagram generation, topological analysis, while `CrystalTool` can calculate reciprocal space operations, process symmetry operations.

JUMBOMarker is a generic parser that allows users to convert legacy logfiles into CML [11]. JUMBOMarker is the appropriate technology when the developer can not modify the source code. JUMBOMarker uses a template based parsing mechanism, in which the user defines a set of XML templates, each containing regular expressions appropriate to extracting a particular piece of information from the logfile.

WXML is a FORTRAN 95 library for generating well-formed XML and CML. The XML formatting library (WXML) can be used to manage multiple XML files and will generate well-formed XML or else inform the user when well-formedness rules are violated. The CML library extends the base XML formatting modules, again checking and enforcing well-formedness, but in addition providing convenience routines for generating large CML elements. In general the CML routines reflect the underlying CML and CMLComp schemas. In theory a DOM could be used for the output of information, however, it is generally not practical to store all the data need to create a DOM. WXML provides a simple piecewise way of outputting well-formed XML.

IV. INFRASTRUCTURE

A. High-Throughput Computing and Workflow

Current technologies, such as the Grid and Condor[18] [12–14], allow the user to perform high-throughput simulations. Moreover, they open up the possibility of creating intricate workflow schemes in which data generated by one program is automatically fed into a second program. A typical workflow scheme will need to take into account the following factors.

- There may be multiple pre- and post-processing steps between simulations.
- There may be dependencies between calculations such that certain calculations can not be performed until others have finished.
- Large numbers of jobs may be able to run in parallel.
- It may be necessary to modify the process as it progresses, dynamically responding to the outcome of calculations.

Workflow issues and tools, and how they relate to this work, are discussed in greater detail in an accompanying paper [15]. We will only mention here that it is possible to automate processes of this kind with off-the-shelf tools, such as Condor’s DAG Manager. Thus, the concern of this paper is not workflow *per se* but *information flow*, i.e. how to deal with data as it “flows” from process to process.

Until now, little concern has been given to data representation in theoretical chemistry and physics applications. Most applications still rely on bespoke binary or text formats that are intimately tied to the software that produces them. The overwhelming majority of text formats are designed be human-readable (rather than machine readable) and often there is no specification for the data format. This is a major barrier to the automatation of complex chemical or physical simulations.

B. Turning Scientific Software Into Workflow Components

Here we describe how the markup languages and tools described in sections II and III can be used to convert existing codes into effective workflow components.

Ideally we will have access to the source code, and can modify the application to read and write CML directly (see figure 5). This approach has been successfully applied to a number of programs, e.g. SIESTA, GULP, DLPOLY, Jmol and OpenBabel. There are a number of ways to convert existing codes, depending on the needs of the application and the language it is written. For instance the Java-based molecular viewer Jmol uses the DOM to read and write a subset of CML. OpenBabel, a file translation program, allows the user to convert between different file formats including CML, also uses a DOM based approach. Both of these programs focus on small subset of the language, primarily structural information, such as atomic coordinates, lattice vectors, connection tables. In contrast simulation packages, such as SIESTA, DLPOLY and GULP, tend to be written in FORTRAN, they generally require large input datasets and rely on their own input formats. For these programs we convert XML to the legacy input formats using either XSLT stylesheets or standalone DOM based applications, converting XML to text is a relatively straightforward process. Unfortunately, converting an arbitrary text file to XML can be far more challenging. For this reason, we have extended these applications to generate CML (in addition to their traditional output). In this case the DOM is not an appropriate technology as jobs may run for several weeks and generate gigabytes of data. Rather than building the document in memory it is more appropriate to write data as it is generated. For this we use the WXML library described above.

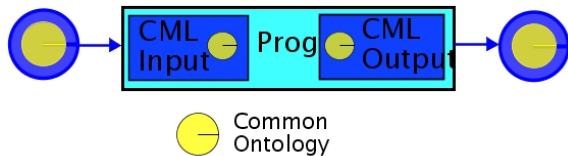


FIG. 5: Software designed to use CML

Where the source code can not be edited (for technical or legal reasons) it is necessary to “wrap” input and output (see figure6). We have used this approach with MOPAC, GUASSIAN and GULP. Legacy input is generated using XSLT. We convert legacy output to XML using the JUMBOMarker utility, defining templates for parsing the text output.

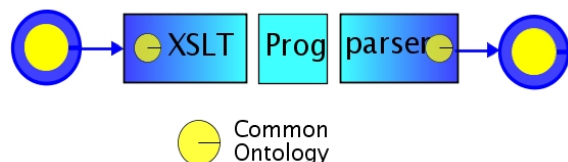


FIG. 6: Generic adaptors for a specific code.

C. Use Case 1: The Simulation of Pollutants in Soils

One of the main aims of this work is to investigate the mechanisms by which pollutant molecules such as DDT, dioxins and biphenyls, become bound to soil minerals. It is not the purpose of this paper to discuss the scientific details of this work rather the aim of the following discussion is to highlight the importance of information flow in this work.

From a scientific point of view, we hope to determine where and how these molecules become bound to soils, systematically searching for trends across families of related compounds. In practical terms this is an extremely large and open-ended task. There are potentially thousands of candidate molecules. Concentrating on the provisional list above there are 420 molecules (including 210 dioxin and 209 biphenyl congeners). There is an equally large choice of minerals, and even for a given mineral there may be many relevant surfaces to investigate. Moreover, even for a single choice of molecule, mineral and surface, we need to investigate the potential energy surface of the system (i.e. how the energetics of the problem change as function of the molecules’ position above the mineral). This in itself requires a large number of calculations. In fact, the nature of the problem requires that calculations be performed, not only on the whole system, but also on molecules and minerals separately, in order to assess their individual contributions to the energetics. A systematic study of this molecule set involves approximately 100,000 calculations per mineral surface. The workflow for this simulation was broken down into the following steps:

1. Select a pollutant molecule (adsorbate)
2. Optimise structure in SIESTA
3. Select a mineral and surface (substrate)

4. Optimise structure in SIESTA
5. Combine molecule and substrate
6. Constrained structural optimisation to find an appropriate starting point
7. Generate a 2D “mesh” in order scan potential energy surface
8. Optimise each system in SIESTA
9. Separate substrate and adsorbate
10. Calculate total energy calculation of adsorbate using SIESTA
11. Calculate total energy calculation of substrate using SIESTA

A series of test calculations have been submitted and their progress through each component carefully monitored. The components perform well and provide an important reduction in error rate which is essential for machine-mediated high-throughput projects.

D. Use Case 2: The World Wide Molecular Matrix

This project aims to compute properties for a large proportion of published molecules, and by extension crystal structures. OpenBabel converts legacy to CML which is then combined with the control parameters and submitted to MOPAC or GAMESS-US. The precise information flow depends on the completeness and consistency of the original information. For molecular crystals a typical computation starts with a refined structure in CIF format but no explicit bonding information. Our information flow is:

1. Read the CIF into an XMLDOM (our cif2cml library)
2. Discard minor disordered components.
3. Convert fractional coordinates to cartesian
4. Join bonds using “reasonable” covalent radii.
5. Apply symmetry operations to generate the minimum number of molecular fragments (here two, as the molecule is a dimer in the crystal).
6. Generate a connection table (CT) for the molecule(s)
7. Check against chemical formula
8. Analyse the CT(s) with to assess chemical validity.
9. Identify potential stereogenic atoms and bonds.
10. Generate CML atomParity and bondStereo if appropriate.

11. Use the CDK library to generate conventional "2D" coordinates.
12. Serialize the result as CML.

This is then used as the coordinate input and the next steps are:

1. Run the computational code
2. Parse the logfile with JUMBOMarker to give XML
3. Tidy this with XSLT stylesheets to create CML-Comp.
4. Compute the IUPAC unique identifier INChI from the connection table
5. Store the result in an XML repository (Xindice) indexed on INChI.

We have already processed ca. 215,000 molecules in this way with MOPAC2003 using the PM5 Hamiltonian and a convergence limit of GRAD=0.1. These are the nucleus of an Open project (the World Wide Molecular Matrix [16]) where anyone can submit molecules for free calculation as long as they are then donated as Open data.

V. CONCLUSIONS

We have demonstrated that a combination of generic and domain specific XML processing tools can be used in conjunction with CML to create a robust architecture for controlling complex scientific calculations. CML-based languages can be created in an extremely flexible way:

- A dynamic approach to constructing schemas - allowing the user to select only those language elements that are relevant

- The use of generic language elements - allowing the language to incorporate an extremely wide range of concepts

- The use of XML based dictionaries - provide a means of adding semantics to an element, allowing a community to develop their language independently of the schema.

We have shown how modifying scientific codes allows them to be used in a component like manner in scientific workflow applications. The way one achieves this depends on several factors, (*i*) availability of the source code, (*ii*) the language the application is written in, (*iii*) the intended use of the application and (*iv*) the size of data sets. We have developed a wide range of libraries and helper applications to facilitate this,

- JUMBOMarker - for wrapping legacy applications, when the user can not modify the code.
- CMLDOM - A domain specific DOM implementation available for a wide range of programming languages
- SAX and DOM implementation for FORTRAN
- WXML - A FORTRAN 90 library allowing developers to create well-formed XML and CML documents in FORTRAN
- XSLT - A complete set of stylesheets for the CML language, providing HTML rendering of chemical information

All of these tools are freely available.

-
- [1] P. Murray-Rust and H. S. Rzepa, *J. Chem. Inf. Comp. Sci.* **39**, 928 (1999).
- [2] P. Murray-Rust and H. S. Rzepa, *J. Chem. Inf. Comp. Sci.* **41**, 1113 (2001).
- [3] D. C. Fallside (2001), URL <http://www.w3.org/TR/xmlschema-0>.
- [4] H. S. Thompson, D. Beech, M. Maloney, and N. Mendelsohn (2001), URL <http://www.w3.org/TR/xmlschema-1/>.
- [5] P. V. Biron and A. Malhotra (2001), URL <http://www.w3.org/TR/xmlschema-2/>.
- [8] D. Brickley and I. R.V. Guha (1999), URL <http://www.w3.org/TR/2004/REC-rdf-schema-20040210>.
- [9] S. R. Hall, F. H. Allen, and I. D. Brown, *Acta Cryst.* **A47**, 655 (1991).
- [10] A. García, P. Murray-Rust, and J. Wakelin, in *Proceeding of the UK e-Science All Hands Meeting* (2004).
- [11] Y. Zhang, P. Murray-Rust, M. Dove, R. C. Glen, H. S. Rzepa, J. A. Townsend, S. Tyrrell, J. Wakelin, and E. L. Willihagen, in *Proceedings of the UK e-Science All Hands Meeting* (2004).
- [12] I. Foster and C. Kesselman, eds., *The Grid: Blueprint for a New Computing Infrastructure* (Morgan Kaufmann, San Francisco, Calif., 1999).
- [13] M. Litzkow, M. Livny, and M. Mutka, in *Proceedings of the 8th International Conference of Distributed Computing Systems* (1988).
- [14] J. Basney and M. Livny, in *High Performance Cluster Computing: Architectures and Systems, Volume 1*, edited by R. Buyya (Prentice Hall PTR, 1999).
- [15] C. Chapman, W. Emmerich, J. Wakelin, E. Artacho, M. Dove, and R. Bruin, to be published in the same

issue.

- [16] P. Murray-Rust, R. C. Glen, H. S. Rzepa, J. J. P. Stewart, J. A. Townsend, E. L. Willighagen, and Y. Zhang, in *Proceedings of the UK e-Science All Hands Meeting* (2003), p. 802.
- [6] M. Hori, J. Euzenat, and P. F. Patel-Schneider (2003), URL <http://www.w3.org/TR/owl-xmlsyntax/>.
- [7] *Proceedings of the UK e-Science All Hands Meeting 2004* (2004), ISBN 1-904425-21-6.
- [17] "Ontology" is increasingly used to represent languages such as OWL and RDF [6] which add machine-processable meaning to the XML components. There is a spectrum of opinion as to whether XMLSchemas (v.i.) should be enhanced with formal ontologies such as provided by OWL/RDF. In a recent BirdsOfFeather (BoF) meeting on ontologies at AHM2004 [7] the consensus was that some communities would concentrate on OWL and use tools such as Protege for authoring and others would base their ontological support on XMLSchema, which is the approach taken here.
- [18] CONDOR is a system to capture free compute cycles on currently unused machines. It includes scheduling, job submission and retrieval of results and can support high-through projects. Condor-G is the extension to run over the GRID

APPENDIX A: XML ELEMENTS COMMONLY USED IN COMPUTATIONAL CHEMISTRY AND PHYSICS

- angle** An angle between three atoms
- annotation** A documentation container similar to annotation in XML Schema
- arg** An argument for a function
- array** A homogenous 1 dimensional array of similar object
- atom** An atom
- atomArray** A container for a list of atoms
- atomicBasisFunction** An atomicBasisFunction
- atomParity** The stereochemistry round an atom centre
- atomSet** A set of references to atoms
- atomType** An atomType
- atomTypeList** A container for one or more atomTypes
- band** A band or Brillouin zone
- bandList** A container for bands
- basisSet** A container for one or more atomicBasisFunctions
- bond** A bond between atoms, or between atoms and bonds
- bondArray** A container for a number of bonds
- bondSet** A set of references to bonds
- bondStereo** A container supporting cis trans wedge hatch and other stereochemistry
- bondType** The type of a bond
- bondTypeList** A container for one or more bondTypes
- cml** A general container for CML elements
- conditionList** A container for one or more experimental condition
- crystal** A crystallographic cell
- eigen** An element to hold eigenstuff
- electron** An electro
- expression** An expression that can be evaluated
- formula** A molecular formula
- gradient** A gradien
- identifier** A structured identifie
- label** A text string qualifying an object
- lattice** A lattice of dimension 3 or less
- latticeVector** A vector3 representing a lattice axis
- length** A length between two atoms
- line3** A line in 3-space
- matrix** A rectangular matrix of any quantities
- metadata** A general container for metadata
- metadataList** A general container for metadata elements
- module** A module in a calculation
- molecule** A container for atoms, bonds and sub-molecules
- name** A string identifying a object
- operator** An operator within an expression
- parameter** A parameter describing the computation
- parameterList** A container for one or more parameters
- particle** An object in space carrying a set of propertie
- plane3** A plane in 3-space
- point3** A point in 3-space
- potential** An explicit potential
- potentialForm** The functional form of a potential

potentialList	A container for explicit potentials	dictRef	A reference to a dictionary entry
property	A container for a property	elementType	The identity of a chemical element
propertyList	A container for one or more properties	formalCharge	The formalCharge on the object
region	A region of the system	hydrogenCount	Number of hydrogen
scalar	An element to hold scalar data	occupancy	Occupancy for an atom
sphere3	A sphere in 3-space	spinMultiplicity	Spin multiplicity
symmetry	Molecular, crystallographic or other symmetry	xy2	x^2/y^2 coordinate for an object
system	The complete system of components in a calculation	xyz3	The $x/y/z$ coordinate of a 3 dimensional object
torsion	torsion angle ("dihedral") between 4 distinct atoms	xyzFract	Fractional $x/y/z$ coordinate
vector3	A vector in 3-space	kpoint	A k vector
zMatrix	A z -Matrix	atomRefs2	References to two different atom
		order	The order of the bond
		spin	The spin of a system
		latticeType	The primitivity of a lattice
		spaceType	The spaceType of the lattice
		periodic	Is the axis periodic?
		state	The physical state of the substance

APPENDIX B: XML ATTRIBUTES COMMONLY USED IN COMPUTATIONAL CHEMISTRY AND PHYSICS

title	A title on an element
id	An attribute providing a unique ID for an element
convention	A reference to a convention

APPENDIX C: CML EXAMPLES

- An atom element with several attributes

```
<cml title="atom1: 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:chem" dictRef="chem:atom">
    <scalar title="dipole" dictRef="cml:dipole" units="units:debye">0.2</scalar>
    <atomParity atomRefs4="a3 a7 a2 a4">1</atomParity>
    <electron id="e1" atomRef="a1" count="2">
    </electron>
  </atom>
</cml>
```

- A crystal cell using crystallographic cell parameters. Note the use of units and error values

```
<cml title="crystal example">
  <molecule id="m1">
    <crystal z="4">
      <scalar id="sc1" title="a" errorValue="0.001" units="units:angstrom">4.500</scalar>
      <scalar id="sc2" title="b" errorValue="0.001" units="units:angstrom">4.500</scalar>
      <scalar id="sc3" title="c" errorValue="0.001" units="units:angstrom">4.500</scalar>
      <scalar id="sc4" title="alpha" units="units:degree">90</scalar>
    </crystal>
  </molecule>
</cml>
```

```
<scalar id="sc5" title="beta" units="units:degree">90</scalar>
<scalar id="sc6" 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>
```