Date: January 2007

Chemical Structure Access & Representation

OMG Available Specification

formal/07-01-02

OBJECT MANAGEMENT GROUP

USE OF SPECIFICATION - TERMS, CONDITIONS & NOTICES

The material in this document details an Object Management Group specification in accordance with the terms, conditions and notices set forth below. This document does not represent a commitment to implement any portion of this specification in any company's products. The information contained in this document is subject to change without notice.

LICENSES

The companies listed above have granted to the Object Management Group, Inc. (OMG) a nonexclusive, royalty-free, paid up, worldwide license to copy and distribute this document and to modify this document and distribute copies of the modified version. Each of the copyright holders listed above has agreed that no person shall be deemed to have infringed the copyright in the included material of any such copyright holder by reason of having used the specification set forth herein or having conformed any computer software to the specification.

Subject to all of the terms and conditions below, the owners of the copyright in this specification hereby grant you a fully-paid up, non-exclusive, nontransferable, perpetual, worldwide license (without the right to sublicense), to use this specification to create and distribute software and special purpose specifications that are based upon this specification, and to use, copy, and distribute this specification as provided under the Copyright Act; provided that: (1) both the copyright notice identified above and this permission notice appear on any copies of this specification; (2) the use of the specifications is for informational purposes and will not be copied or posted on any network computer or broadcast in any media and will not be otherwise resold or transferred for commercial purposes; and (3) no modifications are made to this specification. This limited permission automatically terminates without notice if you breach any of these terms or conditions. Upon termination, you will destroy immediately any copies of the specifications in your possession or control.

PATENTS

The attention of adopters is directed to the possibility that compliance with or adoption of OMG specifications may require use of an invention covered by patent rights. OMG shall not be responsible for identifying patents for which a license may be required by any OMG specification, or for conducting legal inquiries into the legal validity or scope of those patents that are brought to its attention. OMG specifications are prospective and advisory only. Prospective users are responsible for protecting themselves against liability for infringement of patents.

GENERAL USE RESTRICTIONS

Any unauthorized use of this specification may violate copyright laws, trademark laws, and communications regulations and statutes. This document contains information which is protected by copyright. All Rights Reserved. No part of this work covered by copyright herein may be reproduced or used in any form or by any means--graphic, electronic, or mechanical, including photocopying, recording, taping, or information storage and retrieval systems--without permission of the copyright owner.

DISCLAIMER OF WARRANTY

RESTRICTED RIGHTS LEGEND

TRADEMARKS

COMPLIANCE

OMG's Issue Reporting Procedure

All OMG specifications are subject to continuous review and improvement. As part of this process we encourage readers to report any ambiguities, inconsistencies, or inaccuracies they may find by completing the Issue Reporting Form listed on the main web page *http://www.omg.org*, under Documents, Report a Bug/Issue (http://www.omg.org/technology/agreement.htm).

Table of Contents

Preface

About the Object Management Group

OMG

Founded in 1989, the Object Management Group, Inc. (OMG) is an open membership, not-for-profit computer industry standards consortium that produces and maintains computer industry specifications for interoperable, portable and reusable enterprise applications in distributed, heterogeneous environments. Membership includes Information Technology vendors, end users, government agencies, and academia.

OMG member companies write, adopt, and maintain its specifications following a mature, open process. OMG's specifications implement the Model Driven Architecture® (MDA®), maximizing ROI through a full-lifecycle approach to enterprise integration that covers multiple operating systems, programming languages, middleware and networking infrastructures, and software development environments. OMG's specifications include: UML® (Unified Modeling Language™); CORBA® (Common Object Request Broker Architecture); CWM™ (Common Warehouse Metamodel); and industry-specific standards for dozens of vertical markets.

More information on the OMG is available at *http://www.omg.org/*.

OMG Specifications

As noted, OMG specifications address middleware, modeling and vertical domain frameworks. A catalog of all OMG Specifications Catalog is available from the OMG website at:

http://www.omg.org/technology/documents/spec_catalog.htm

Specifications within the Catalog are organized by the following categories:

OMG Modeling Specifications

- UML
- MOF
- XMI
- CWM
- Profile specifications.

OMG Middleware Specifications

- CORBA/IIOP
- IDL/Language Mappings
- Specialized CORBA specifications
- CORBA Component Model (CCM).

Platform Specific Model and Interface Specifications

- CORBAservices

- CORBAfacilities
- OMG Domain specifications
- OMG Embedded Intelligence specifications
- OMG Security specifications.

All of OMG's formal specifications may be downloaded without charge from our website. (Products implementing OMG specifications are available from individual suppliers.) Copies of specifications, available in PostScript and PDF format, may be obtained from the Specifications Catalog cited above or by contacting the Object Management Group, Inc. at:

OMG Headquarters
140 Kendrick Street
Building A - Suite 300
Needham, MA 02494
USA
Tel: +1-781-444-0404
Fax: +1-781-444-0320
Email: *pubs@omg.org*

Certain OMG specifications are also available as ISO standards. Please consult *http://www.iso.org*

Typographical Conventions

The type styles shown below are used in this document to distinguish programming statements from ordinary English. However, these conventions are not used in tables or section headings where no distinction is necessary.

Times/Times New Roman - 10 pt.: Standard body text

Helvetica/Arial - 10 pt. Bold: OMG Interface Definition Language (OMG IDL) and syntax elements.

`Courier - 10 pt. Bold:` Programming language elements.

Helvetica/Arial - 10 pt: Exceptions

Note – Terms that appear in *italics* are defined in the glossary. Italic text also represents the name of a document, specification, or other publication.

Issues

The reader is encouraged to report any technical or editing issues/problems with this specification to *http://www.omg.org/technology/agreement.htm*.

1 Scope

A set of specifications describing the management of molecular information, chemical structure access, and retrieval processes is provided herein. The specification is based in the Chemical Markup Language (CML) as described at http://www.xml-cml.org/ and http://wwmm.ch.cam.ac.uk/moin/CmlCore. The Chemical Structure Access and Representation (CSAR) makes use of the representational and translational capabilities offered by CML and complement them with classes that facilitate searches and allow the creation of collections. Situations that are not described explicitly can be addressed by extending the types and using the conventions described in this document. The standard was developed starting from the practical need to represent complex bodies of data in a natural way. Existing practice and terminology is used wherever this was available and practical.

2 Conformance

Since the CSAR specification makes use or extends the conversion interfaces provided by CML, there are two points of compliance for this specification. One explicit which is the interface described in the CSAR specification and the second one implicit which is the conversion interface described in the CML specification. The compliance points are described below.

2.1 CSAR

The CSAR specification describes interfaces to:

- Transform legacy file formats into CML representations that make use of CML's Jumbo interfaces.
- Validate the CML representation that makes use of the CMLDOM interface.
- Search, create (register), replace, update, and delete chemical components.
- Estimate the loss of information when searching different proprietary (legacy) databases.

CSAR is a mandatory conformance point.

2.2 CML Core

CoreCML is a subset of CML which has a tighter specification and is designed for representing "small molecules." It is completely consistent with full CML V1.0 but its strictness encourages interoperability. By writing CoreCML you will be writing CML, but taking advantage of many conventions described below which will help the development of conforming software.

Implicitly CoreCML is a mandatory conformance point.

3 Normative References

None

4 Terms and Definitions

There are no terms and definitions associated with this specification.

5 Symbols

There are no symbols associated with this specification.

6 Additional Information

6.1 Relationship to Other OMG Specifications

- Model Driven Architecture - While the model provides only the data structures needed for the interchange and manipulation of Chemical Representation data and specifies no services, it does share the MDA approach of using the UML model as the basis of generating the DTD from the XMI. Given this model it is now possible to generate data structures that meet different language standards.
- XML Metadata Interchange - The model is submitted in XMI format, v1.1, generated from Rational Rose Enterprise Edition 2001 using the Unisys Rose XML Tools Version 1.3.2 add-in.
- Query Service - The Query service was not used; instead the authors suggest that it is possible to implement a Query Service for Chemical Representation data using Xdb. Xdb is an XML document repository providing structured storage of XML data, at present using a Relational Database Management System (RDBMS) mapping over PostgreSQL. XDB provides a fast and scalable XML database framework with support for both XML Path Language (XPath) and XML Query Language (XQL), and the ability to store XML documents and SAX APIs.
- Bibliographic Query Service - Although the BQS specification is not directly incorporated, the attributes and annotation from associations of the Bibliographic Reference can be used in queries to data sources that support the interfaces in the BQS specification.
- Collection Service - The Collection Service is not used. Instead, the W3C DOM and XML-DEV SAX parsers provide similar capabilities for XML data as the Collection Service for IDL data. The DOM parser is ideal for smaller XML data and provides full navigation backwards and forwards. The SAX parser provides a forward only traversal of the data, which is ideal for parsing large XML documents.

6.2 Document Structure

The document is structured as follows: 7 Scope provides a synopsis of the data model and data types; 8 CSAR presents the CSAR standard; 9 Glossary provides a list of terms used throughout this document; Annex A contains the UML use cases; Annex B lists the use cases for chemistry; Annex C contains the UML related interface documentation for the model; Annex D contains the Java code segments; and Annex E provides the complete XMI of the specification.

6.3 Acknowledgements

The following company submitted this specification:

- Intelligent Solutions

7 Introduction

Several code segments have been developed to test the use case scenarios presented below.

The general functionality is summarized in Figure 7.1. The types of requests are outlined below and a sample work flow is provided.

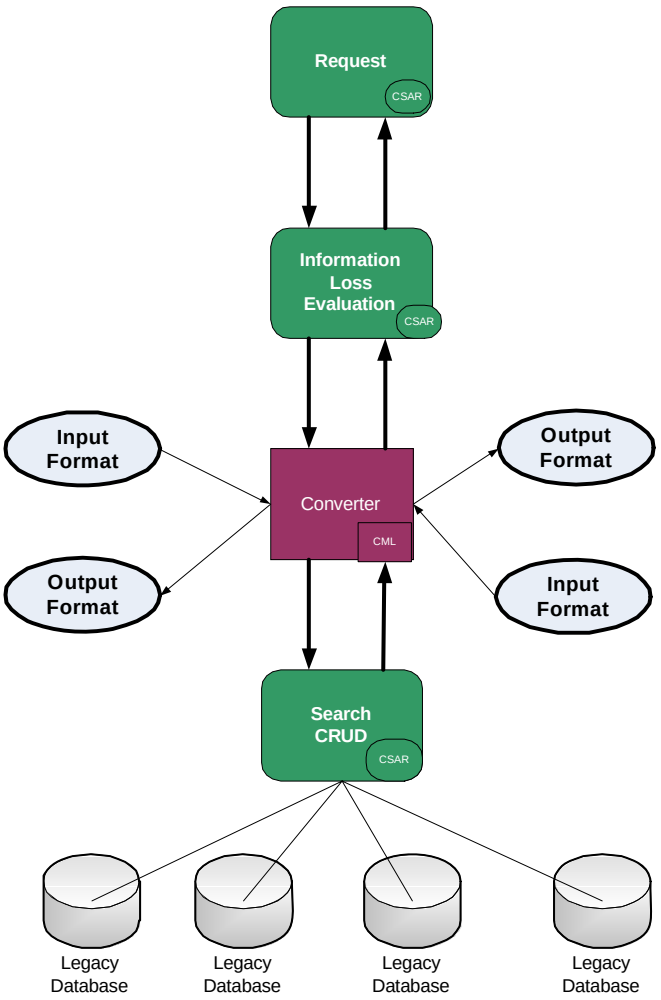


Figure 7.1 - General Functionality

7.1 Request

1. Search Chemical Structure (against legacy database)
   Create CML molecule from a legacy format.
   Select the values for use as parameters in a query.
   Construct a query using interfaces in Search and DB modules.
   Transport resulting query to server.

Vendor Implementation translates the query into the legacy format.
Vendor Implementation performs the search from the database.
Vendor Implementation translates the result into the CML object model.
Transport the result back to client.
Transform the result into a legacy format.

2. Search Chemical Structure (within collection of CML molecules)
   Create CML molecule from a legacy format.
   Select the values for use as parameters in a query.
   Construct a query using interfaces in Search and Collection modules.
   Search the collection using the query built above.
   Transform the result into a legacy format.

3. Add Chemical Structure (Register)
   Create CML molecule.
   Transport the new molecule to server.
   Vendor Implementation translates the insert query into the legacy format.
   Vendor Implementation performs the insert into the database.
   Vendor Implementation translates the status of the insert into standard format.
   Transport the result back to client.

4. Update Chemical Structure
   Search the database for the chemical structure to be updated.
   Replace the corresponding CML molecule with a newly constructed CML molecule.
   Transport the update query along with the new molecule to server.
   Vendor Implementation translates the query into the legacy format.
   Vendor Implementation performs the update to the database.
   Vendor Implementation translates the status of the update into standard format.
   Transport the result back to client.

5. Delete Chemical Structure
   Search the database for the chemical structure to be deleted.
   Delete the corresponding CML molecule. [Actual deletion is performed by Vendor Implementation]
   Transport the delete query to server.
   Vendor Implementation translates the query into the legacy format.
   Vendor Implementation performs the delete to the database.
   Vendor Implementation translates the status of the delete into standard format.
   Transport the result back to client.

6. Translate Chemical Structures between Representations
   Forward transformation will create a CML representation from a legacy representation. Reverse transformation will create a legacy representation from a CML representation.

7. Manipulate Associated Intrinsic Properties
   Given a legacy representation of a molecule or result set, create CML molecule. Then gain access to its corresponding intrinsic properties - including atoms, bond types, connectivity, molecular weight, and molecular formula.

7.2 Sample Workflow

Figure 7.2 illustrates the general workflow that is followed when making use of CSAR.

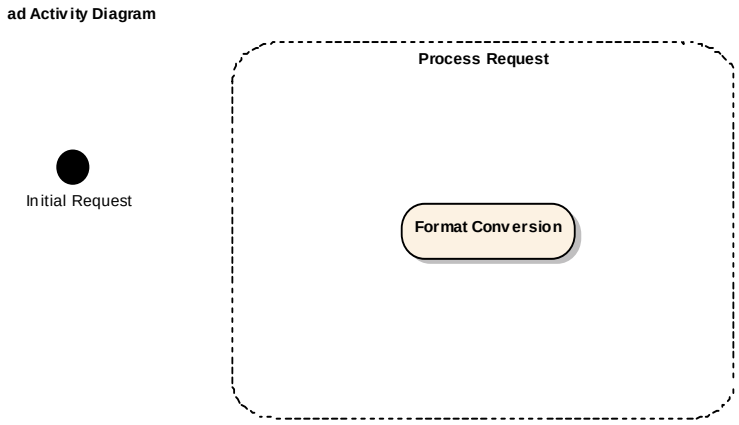


Figure 7.2 - General Workflow

The workflow is outlined below.

1. A chemist could issue an initial request to the system, for instance a search for a given compound. The request will be accompanied with a source chemical table file (source format) and a destination format. For example, the source format could be an MDL MOL file and the destination a Daylight database containing files in its proprietary format named SMILES.
2. The source file will be converted to CML. For instance, Figure 7.3 shows an example of a Mol file and its corresponding conversion into CML.

MOL File: caffeine.mol

```
24 25  0  0  0  0  0  0  0  0  0  0
    -2.8709   -1.0499    0.1718 C   0  0  0  0  0
    -2.9099    0.2747    0.1062 N   0  0  0  0  0
    -1.8026    0.9662   -0.1184 C   0  0  0  0  0
    -0.6411    0.2954   -0.2316 C   0  0  0  0  0
    -0.6549   -1.0889   -0.1279 C   0  0  0  0  0
    -1.7352   -1.7187    0.0624 N   0  0  0  0  0
     0.6052    0.7432   -0.4434 N   0  0  0  0  0
     1.2863   -0.4175   -0.4514 C   0  0  0  0  0
     0.5994   -1.5633   -0.2698 N   0  0  0  0  0
     1.0875    2.0867   -0.6139 C   0  0  0  0  0
    -1.8349    2.1699   -0.2205 O   0  0  0  0  0
```

The corresponding CML for the MOL file for caffeine is partially show below

```
<?xml version="1.0"?>
<!--<?xml-stylesheet type="text/xsl" href="generic.xsl" ?>-->
<document>

<!-- CML document - example_mol - karne - 18/4/00 -->
<!-- file converted from: MDL .mol -->
<cml title="example_mol" id="cml_example_mol_karne">
<molecule title="example_mol" id="mol_example_mol_karne">
<list title="atoms">
<atom id="example_mol_karne_a_1">
<integer builtin="atomId">1</integer>
<float builtin="x3" units="A">-2.8709</float>
<float builtin="y3" units="A">-1.0499</float>
```

```
 -4.2178    0.9810    0.2003 C   0  0  0  0  0
 -3.8944   -1.6746    0.3323 O   0  0  0  0  0
 -1.6764   -3.1997    0.1458 C   0  0  0  0  0
  2.3776   -0.4481   -0.6036 H   0  0  0  0  0
  2.1902    2.0944   -0.7699 H   0  0  0  0  0
  0.6074    2.5547   -1.5032 H   0  0  0  0  0
  0.8606    2.6915    0.2934 H   0  0  0  0  0
 -4.0942    2.0097    0.6091 H   0  0  0  0  0
 -4.6699    1.0338   -0.8167 H   0  0  0  0  0
 -4.9101    0.4518    0.8943 H   0  0  0  0  0
 -2.3049   -3.6334   -0.6659 H   0  0  0  0  0
 -0.6444   -3.6030    0.0359 H   0  0  0  0  0
 -2.0682   -3.5218    1.1381 H   0  0  0  0  0
  1  2  1  0  0  0
  1  6  1  0  0  0
  1 13  2  0  0  0
  2  3  1  0  0  0
  2 12  1  0  0  0
  3  4  1  0  0  0
  3 11  2  0  0  0
  4  5  2  0  0  0
  4  7  1  0  0  0
  5  6  1  0  0  0
  5  9  1  0  0  0
  6 14  1  0  0  0
  7  8  1  0  0  0
  7 10  1  0  0  0
  8  9  2  0  0  0
  8 15  1  0  0  0
 10 16  1  0  0  0
 10 17  1  0  0  0
 10 18  1  0  0  0
 12 19  1  0  0  0
 12 20  1  0  0  0
 12 21  1  0  0  0
 14 22  1  0  0  0
 14 23  1  0  0  0
 14 24  1  0  0  0
M  END
```

```
<float builtin="z3" units="A">0.1718</float>
<string builtin="elementType">C</string>
</atom>
<atom id="example_mol_karne_a_2">
<integer builtin="atomId">2</integer>
<float builtin="x3" units="A">-2.9099</float>
<float builtin="y3" units="A">0.2747</float>
<float builtin="z3" units="A">0.1062</float>
<string builtin="elementType">N</string>
</atom>
atom id="example_mol_karne_a_3">
<integer builtin="atomId">3</integer>
<float builtin="x3" units="A">-1.8026</float>
<float builtin="y3" units="A">0.9662</float>
<float builtin="z3" units="A">-0.1184</float>
<string builtin="elementType">C</string>
</atom>
<atom id="example_mol_karne_a_4">
<integer builtin="atomId">4</integer>
<float builtin="x3" units="A">-0.6411</float>
<float builtin="y3" units="A">0.2954</float>
<float builtin="z3" units="A">-0.2316</float>
<string builtin="elementType">C</string>
</atom>
                                                    :
                                                    :
convention="MDL">1</integer>
</bond>
<bond id="example_mol_karne_b_24">
<integer title="bondId">24</integer>
<integer builtin="atomRef">14</integer>
<integer builtin="atomRef">23</integer>
<integer builtin="order" convention="MDL">1</integer>
</bond>
<bond id="example_mol_karne_b_25">
<integer title="bondId">25</integer>
<integer builtin="atomRef">14</integer>
<integer builtin="atomRef">24</integer>
<integer builtin="order" convention="MDL">1</integer>
</bond>
</list>
</molecule>
</cml>
</document>
```

Figure 7.3 - MOL file for Caffeine and corresponding CML representation

3. A validity check of the data in the source file will be conducted.

- Current molecular ‘file formats ‘ and database entries normally choose a subset of available information that can be captured. Many older formats are based on fixed length records (often 80 characters) and a restricted order for those records; this limits or completely denies extensibility. In general, two different formats have different ontologies and cover a different subset of chemical information space.

- Every piece of chemical software uses its own ontology, usually implicitly. The relevant information has to be supplied in the input files but, because the implementation of ontologies is very expensive, the program is usually built to accept a small number of file types. When chemical information is passed from one program to another, ontological conversion is necessary. However, only the concepts present in both ontologies can be passed, and this normally leads to information loss. For example, a PDB file does not explicitly hold bond orders, while an MDL-molfile does not hold occupancies. Both these concepts are therefore lost in an interconversion. There is also often an ontological loss since the meaning of a common concept (e.g., bond order) may be different in both.

See example below:

MOL file representation for ethane	SMILES file representation for ethane:
SMI2MOL 2 1 0 0 0 0 0 0 0 0999 V2000 -0.5100 1.5300 0.0000 C 0 0 0 0 0 0 0 0 0 0 0 0 0.5100 1.5300 0.0000 C 0 0 0 0 0 0 0 0 0 0 0 0 1 2 1 0 0 0 0 M END	CC

- CML has been designed to allow conversion from legacy files to CML *without information loss*. In some cases, this is because the information can be represented in an abstract, convention-free form. In other cases, however, this is essentially a syntactic conversion with annotation of the original **convention** (i.e., ontology) used. While we do not believe a single ontology is possible for chemistry, the use of CML may highlight agreement on some ontological subsets. We continue to emphasize that conversion from CML to other formats will probably involve information loss. The use of CML as input to programs should make it easier to identify chemical information in files, and to convert when possible.

4. The CML file is then used to generate a query against the chemical repository.

The following section provides the problem definition and the problem solution via CSAR.

7.3 The Chemical Exchange Problem

Storing chemical information in a computer is not a trivial task. Many different approaches and formats have been used. Most that worked at all are still with us. For instance, there are between 30-40 important chemical formats—managing them is a formidable and expensive task.

A small part of the problem is that arbitrary methods are used for encoding data, e.g., double bonds can be represented as the integer 2, the real bond order 2.0, the symbol "=", the enumeration DOUBLE, and as repeated connections.

A bigger part of the problem is that most chemical file formats contain information that is meaningless except in terms of a specific program, e.g., “tautomeric,” “ring-double,” “exo-double,” and “fragment double” bonds.

The biggest part of the problem is that different programs that process chemical information use different underlying models, e.g., in ab initio or M.O. programs, the idea of “bond” isn’t a particularly useful concept. To be useful, we must provide an accurate method for representing the underlying data model.

7.3.1 Approaches that don't work well[1]

Various attempts have been made to solve the problem created by the plethora of chemical file formats. Experience has shown that the two most common approaches don't work well. They haven't solved the practical problem and they keep being rewritten.

7.3.1.1 Comprehensive file format converters

Software is provided which converts files from one format to another. This is usually implemented by creating "readers" and "writers" which share a common data structure or format. However, most of the big players in this industry make use of proprietary formats that make conversion of files a very hard problem that might require reverse engineering of the files.

This approach works well for encoding problems only, i.e., where representational issues don't exist. Such systems are inevitably reactive (must be modified as formats evolve) and usually either inaccurate (there is no central authority) or LCD (lowest common denominator).

7.3.1.2 "Kitchen sink" formats

An all-encompassing format is suggested which purports to represent every possible kind of chemical information entity.

This approach has all the failings of the previous approach, plus it complicates the problem by introducing yet another format. Furthermore, the new format is so complex that a comprehensive reader/writer can't provide a universal interface because it is prohibitively expensive in size, complexity, and support.

Therefore, a common representation that provides a base functionality for atomic, molecular, and crystallographic information and allows extensibility for other chemical applications was sought. As a result, the Chemical Markup Language (CML)—an application of XML, the eXtensible Markup Language--was developed for containing chemical information components within documents. Its design supports interoperability with the XML family of tools and protocols. Legacy files can be imported into CML with limited information loss and can carry any desired chemical ontology[2].

7.4 Chemical Markup Language within the Context of CSAR

7.4.1 Introduction[3]

Peter Murray-Rust *and* Henry Rzepa state that there are a number of problems when trying to capture the contextual meaning of chemistry that impede the use of this information. For instance, the following extract[4] from a typical molecule science journal illustrates not only how precisely data and information must be represented, but also how much human perception is required to translate this information as presented in this (linear) form into e.g., a reproducible experiment or a mechanistic interpretation;

1. Ideas taken from "Computer Representation of Chemical Information" http://www.ccl.net/cca/documents/molecular-modeling/node2.shtml
2. "The CML ontology is the theory underlying the CML language. It defines the basic concepts, such as model-fragment and time-dependent-relation, that are assumed in the language. It gives axiomatic semantics for the notion of time and change inherent in CML. The CML ontology is built upon the Engineering Math ontologies, extending the unary-scalar-functions and standard-units theories.", Theory CML, http://www.ksl.stanford.edu/htw/dme/thermal-kb-tour/cml.html
3. Many ideas, concepts and descriptions in this section are taken (some times verbatim some times modified by the authors) from P. Murray-Rust and H. Repka, "Chemical Markup, XML and the World-Wide Web. Part II: Information Objects and the CMLDOM"
4. D. H. Peapus, H. J. Chiu and N. Campobasso, *Biochemistry*, 2001, **40**, 10103-10114.

"Thiamin phosphate synthase catalyzes the formation of thiamin phosphate from 4-amino-5- (hydroxymethyl)-2-methylpyrimidine pyrophosphate and 5-(hydroxyethyl)-4-methylthiazole phosphate. The reaction involves... dissociative mechanism...carbenium ion intermediate...and pyrimidine iminemethide observed in the crystal..."

Note - The profusion of chemical structure information, concepts, and terms, which only a trained human chemist could easily process. Quantitative concepts and units are also ubiquitous;

> "A 500 ul aliquot of 0.8 uM TP synthase in 50 mM Tris-HCl (pH 7.5) and 6 mM $MgCl_2$ incubated at room temperature with 50uM CF_3HMP-PP."

An even greater degree of human perception is required when handling graphical chemical representations which may contain many, often fuzzy and dangerous, human-only semantics (e.g., 2D representations of 3D properties, relative stereochemistry, aromaticity, hydrogen and other "weak" bonding, use of generic and "R" groups, reaction arrows and mechanisms, etc.). The challenge therefore is to develop an infrastructure that can be routinely used to capture, store, and appropriately filter and display such information. Moreover, each discovery informatics company makes use of proprietary formats to describe chemical objects. For instance, MDL Information Systems Inc., the largest chemical informatics company, supports a number of file formats for representation and communication of chemical information group under the generic name of MDL's Chemical Table files (CTfiles), see Figure 7.4. These file formats are:

1. Molfiles, RGfiles, SDfiles, Rxnfiles, RDfiles
2. Code names: **mol**, **mol:V3**, **mol:V3ec**, **mol:V3ea**, **rgf**, **sdf**, **rxn**, **rxn:V3**, **rdf**
3. file extensions: **.mol**, **.sdf**, **.rxn**, **.rdf**

Table 7.1 provides a description of each file format shown in Figure 7.4.

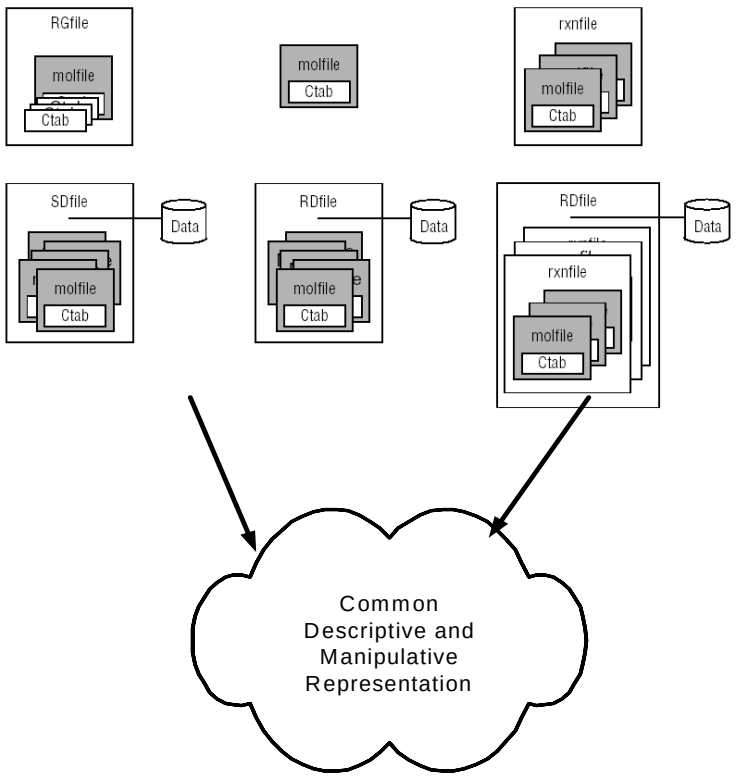


Figure 7.4 - The representation problem [5]

Table 7.1 - Chemical Table File Types[a]

Chemical Table File Type	Description
Molecule files (molfiles)	Each molfile describes a single molecular structure which can contain disjoint fragments.
Rgroup files (RGfiles)	An RGfile describes a single molecular query with Rgroups. Each RGfile is a combination of Ctabs defining the root molecule and each member of each Rgroup in the query.
Reaction files (rxnfiles)	Each rxnfile contains the structural information for the reactants and products of a single reaction. MDL currently supports only the REACCS type of rxnfile. The CPSS type of rxnfile written by CPSS programs is no longer supported and is not described in this document.
Structure-data files (SDfiles)	An SDfile contains structures and data for any number of molecules. Together with RDfiles, SDfiles are the primary format for large-scale data transfer between MDL databases.
Reaction-data files (RDfiles)	Similar to SDfiles in concept, the RDfile is a more general format that can include reactions as well as molecules, together with their associated data. Although RDfiles are used primarily by ISIS and REACCS, MACCS-II can also read and write RDfiles except for the reaction structure information (indicated by the square brackets in the MDL Program table).
XML-data files (XDfiles)	XML-based data format for transferring recordsets of structure or reaction information with associated data. An XDfile can contain structures or reactions that use any of the CTfile formats, Chime strings, or SMILES strings. (Chime is an encrypted format that is used to render structures and reactions on a Web page. SMILES is a line notation format that uses character strings and SMILES, Simplified Molecule Input Line Entry System, syntax to represent a structure.)

a. MDL Systems CTL formats.

In addition to MDL systems there are a number of companies that support additional file formats, such as, Day Light's SMILES and others.

7.5 CML and CSAR

The goal of the model described in this specification is to make use and/or extend the representational and translational capabilities offered by CML and in addition complement them with classes that facilitate transactional operations such as searches and creation of collections (see Figure 7.5).

5. The picture has been taken from MDL Information Systems manual., CTL File Formats and modified by the authors to explain the common descriptive and manipulation representation.

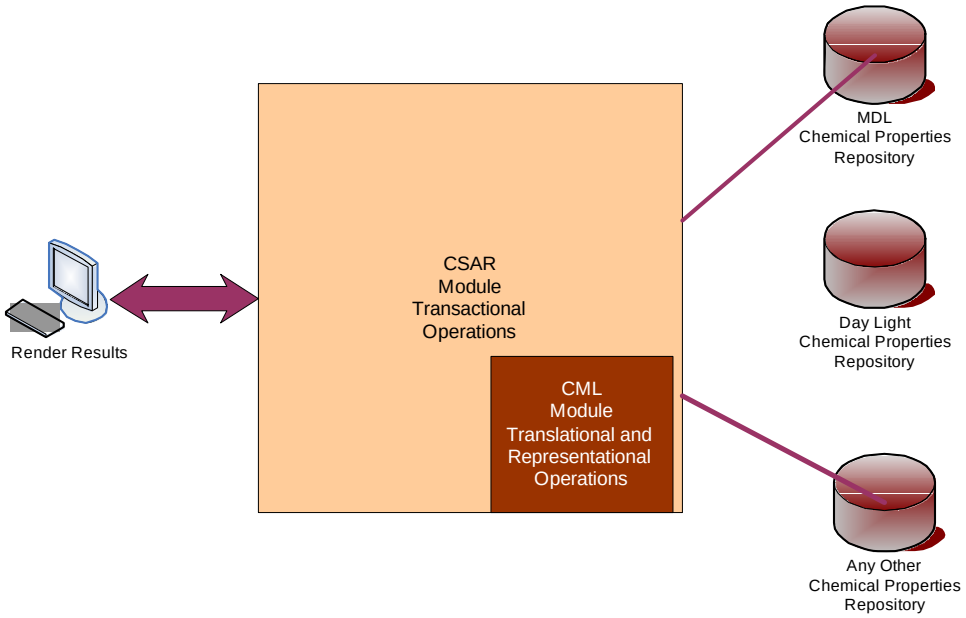


Figure 7.5 - Integrated CML and CSAR

7.6 Processing Classes Involved

Figure 7.6 provides an overview of the CSAR's elements.

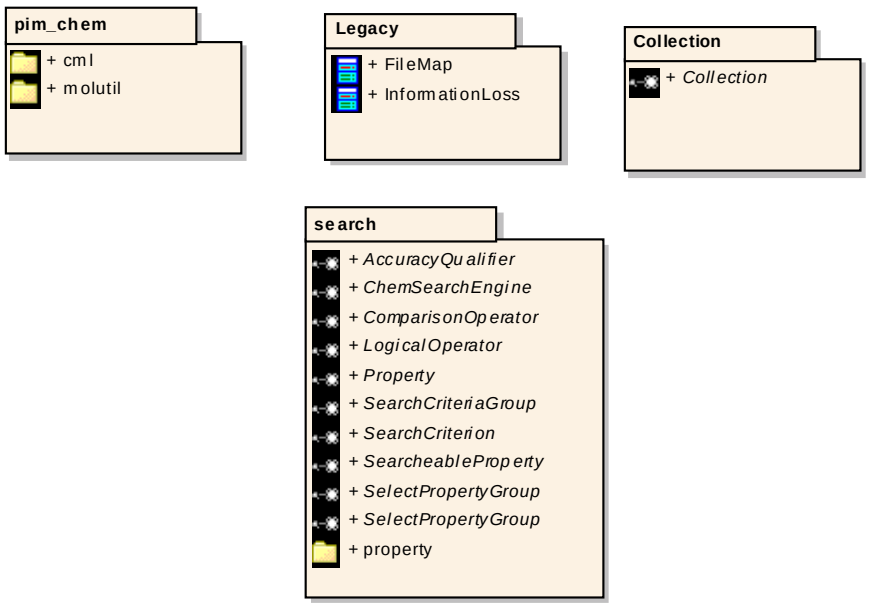


Figure 7.6 - CSAR Elements Overview

- Legacy Module – Provides interfaces to convert legacy database formats to CML and in between them. It also provides the basic file mappings.
- PIM_CHEM Module – Provides the classes that provide the representational capabilities of the standard and a number of utility classes.
- Search Module – Holds interfaces that are used to search against legacy databases as well as collections.
- Collection Module – Provides a common repository of chemical information.

8 CSAR

Basically the CSAR standard provides the following capabilities:

- Interfaces and/or classes to represent chemical information elements.
- Interfaces and/or classes to facilitate searches.
- Interfaces and/or classes to facilitate searches for properties.
- Interfaces and/or classes for Cartesian coordinates as well as classes for Polar coordinates.
- Interfaces and/or classes to calculate the information loss when working with different legacy formats.

8.1 [Chemical] Elements within CSAR

The following [chemical] elements of the CSAR specification recast (or extend) elements (with similar names) included in the CML specification to satisfy the functional requirements of CSAR. Figure 7.1 in the previous chapter provides an overview of the elements.

Figure 8.1 provides an overview of the components of the pim_chem module.

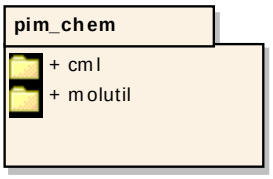


Figure 8.1 - pim_chem Module

The pim_chem module provides representational interfaces or classes for chemical information. Figure 8.2 provides a more detailed perspective of these components.

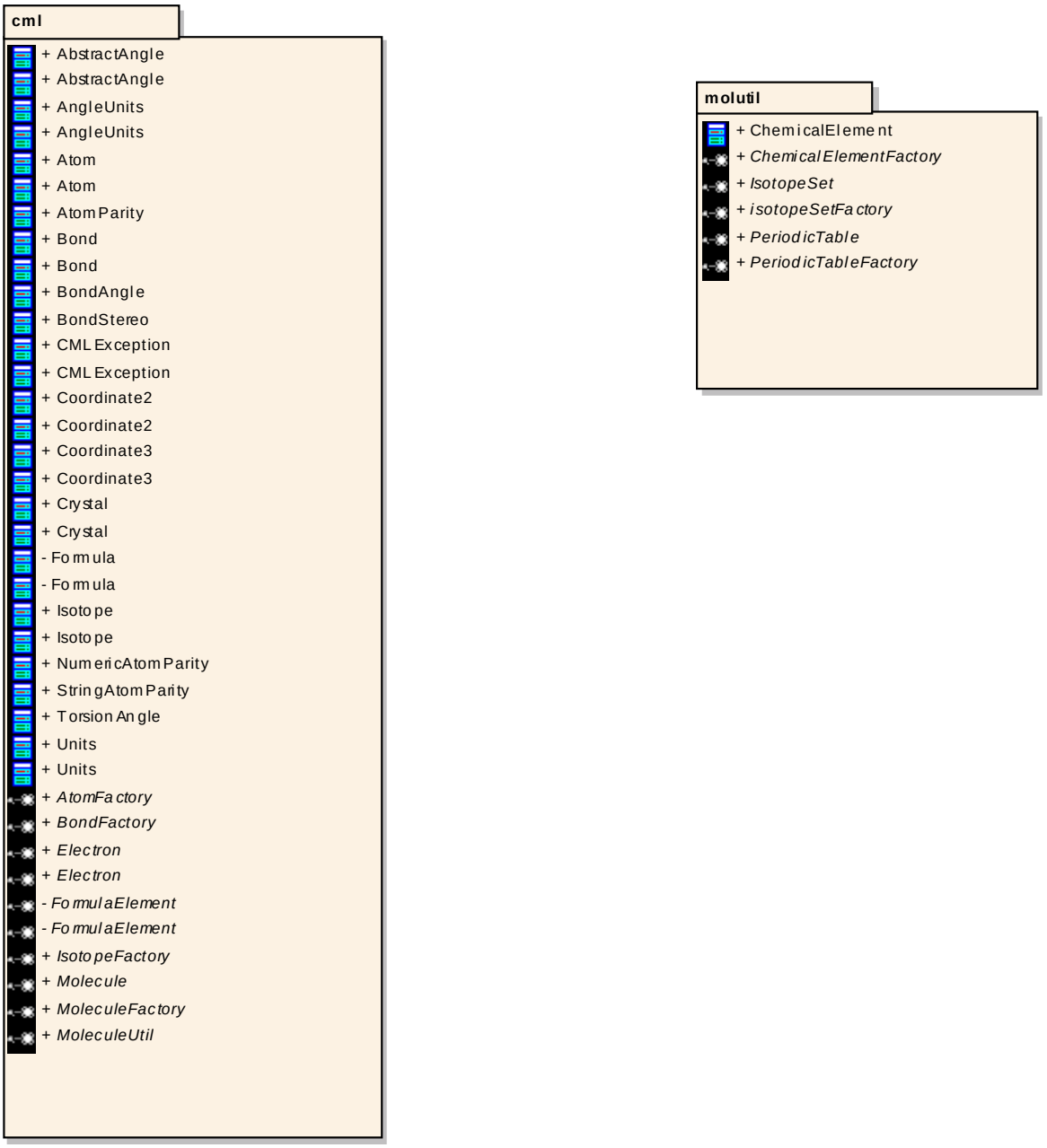


Figure 8.2 - pim_chem components detailed view

The purpose of this module is to provide means to represent the chemical elements of the model. For instance, the components provide representational interfaces or classes for Atom, Molecule, Bond, Electrons, Formulas, and others.

8.2 CML Module

8.2.1 Molecule

At the heart of our model is the Molecule entity that represents a chemical substance, see Figure 8.3. One Molecule contains zero or more sub-Molecules (no limit on the depth with which Molecules can be nested), zero or more Atoms, and zero or more Bonds. Molecules nested within a Molecule give our model the ability to accommodate sets of tautomers, conformers, residues, mixtures (not required for this specification, but definitely useful in chemistry), and other complex chemical entities.

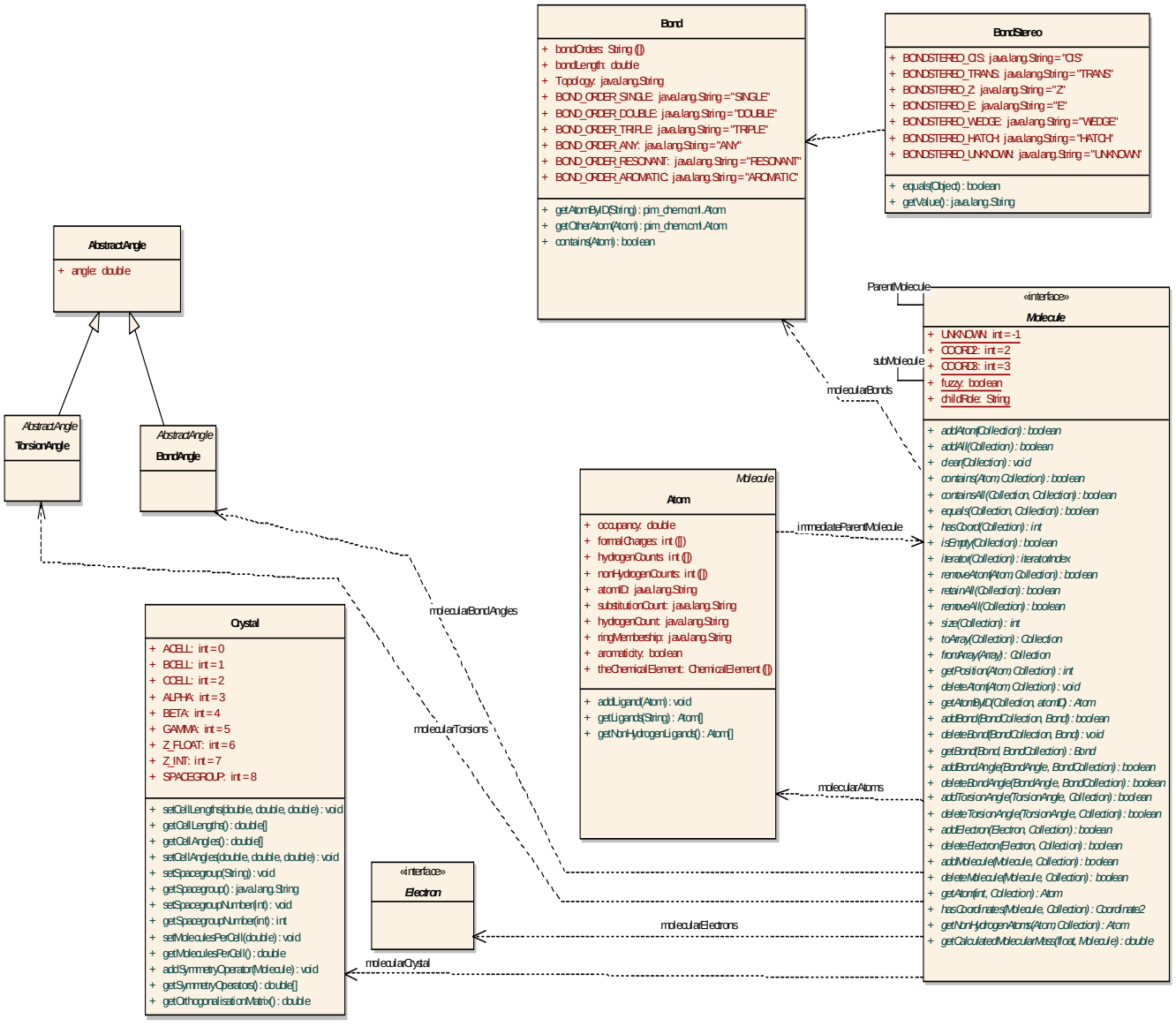


Figure 8.3 - Molecule and interrelated classes/interfaces

Molecules have the following properties.

8.2.1.1 Properties

Molecule Attributes

Attribute	Details
public static int **UNKNOWN**	*Initial: -1* *Notes:* Basically is a flag to identify those molecules that are not registered yet in the database and consequently they are unknown to the system.
public static int **COORD2**	*Initial: 2* *Notes:* It contains a 2-dimensional coordinates
public static int **COORD3**	*Initial: 3* *Notes:* Contains a tri-dimensional set of coordinates
public static boolean **fuzzy**	*Notes:* when true, this molecule is only intended to query databases and probably does not reflect a real chemical substance that a researcher would make or isolate and then store in a database.
public static String **childRole**	*Notes:* string; when this Molecule is associated with another Molecule, childRole contains a description of the relationship.

8.2.1.2 Associations

We use *Associations* to record the relationship between **Molecule** and other entities:

- Dependency link from class [cml].Atom – A pointer to the set of Atoms that belong to this Molecule.
- Dependency link from class [cml].Bond – A pointer to the set of Bonds that belong to this Molecule.
- Association link from interface [cml].Molecule – Identifies the next Molecule up in the hierarchy.
- Association link from interface [cml].Molecule – Identifies the set of sub-Molecules under this molecule.
- Dependency link from class [cml].Atom – Identifies the Molecule immediately above the set of Atoms.
- Dependency link to class [cml].BondAngle – A pointer to the set of BondAngles that belong to Molecule.
- Dependency link to class [cml].TorsionAngle – A pointer to the set of TorsionAngles that belong to this Molecule .
- Dependency link to class [cml].Crystal – A pointer to a Crystal unit, defining the crystal structure of this molecule.
- Dependency link to interface [cml].Electron – A pointer to the set of Electrons that belong to this Molecule.

8.2.1.3 Operations

The following operations define the behavior of Molecule.

Molecule Methods

Operation	Details
public **addAtom**(Collection atoms):boolean	*Sequential* *Notes:* Appends an Atom to the set indicated by the Collection of atoms.
public **addAll**(Collection atoms):boolean	*Sequential* *Notes:* Appends an entire Collection of atoms.
public **clear**(Collection atoms):void	*Sequential* *Notes:* Clears the Collection container
public **contains**(Atom atom, Collection atoms):boolean	*Sequential* *Notes:* Determines if a given Atom is present within the Collection
public **containsAll**(Collection searchatoms, Collection atoms):boolean	*Sequential* *Notes:* Determines if a given Collection is part of another Collection
public **equals**(Collection atomsTwo, Collection atomsOne):boolean	*Sequential* *Notes:* Determines if both Collections are equal
public **hasCoord**(Collection atoms):int	*Sequential* *Notes:* Indicates where this Molecule has coordinates of a specified type (either 2D or 3D)
public **isEmpty**(Collection atoms):boolean	*Sequential* *Notes:* Determines if the Collection is empty
public **iterator**(Collection atoms):iteratorIndex	*Sequential* *Notes:* Iterates over the given Collection
public **removeAtom**(Atom givenAtom, Collection atoms):boolean	*Sequential* *Notes:* Removes a given Atom from a Collection.
public **retainAll**(Collection atoms):boolean	*Sequential* *Notes:* Asserts the Collection is correct
public **removeAll**(Collection atoms):boolean	*Sequential* *Notes:* Remove all elements of a given Collection or sub collection

public **size**(Collection atoms):int	*Sequential* *Notes:* Determines the size of the Collection
public **toArray**(Collection atoms):Collection	*Sequential* *Notes:* Transfer the contents of a given Collection into an array of the same name.
public **fromArray**(Array arrayofAtoms):Collection	*Sequential* *Notes:* Populates a Collection from a given array given th eCollection the same name as the original array.
public **getPosition**(Atom satom, Collection atoms):int	*Sequential* *Notes:* Provides the position of a given Atom within the Collection
public **deleteAtom**(Atom datom, Collection atom):void	*Sequential* *Notes:* Deletes a given Atom from a Collection
public **getAtomByID**(Collection atoms, atomID atom):Atom	*Sequential* *Notes:* A convenience method for locating an Atom given an (alphanumeric) ID.
public **addBond**(BondCollection molecularBonds, Bond bond):boolean	*Sequential* *Notes:* Appends a Bond to the set indicated by molecularBonds
public **deleteBond**(BondCollection molecularBonds, Bond bond):void	*Sequential* *Notes:* Removes a Bond from the set indicated by molecularBonds
public **getBond**(Bond sbond, BondCollection bond):Bond	*Sequential* *Notes:* Get a given Bond from a given Collection of Bonds
public **addBondAngle**(BondAngle bondAngle, BondCollection molecularBondAngle):boolean	*Sequential* *Notes:* Appends a BondAngle to the set indicated by molecularBondAngles.
public **deleteBondAngle**(BondAngle bondangle, BondCollection dbondAngle):boolean	*Sequential* *Notes:* Deletes a BondAngle from a Collection

public **addTorsionAngle**(TorsionAngle torsionangle, Collection molecularTorsionAngles):boolean	*Sequential* *Notes:* Appends a TorsionAngle to the set indicated by molecularTorsionAngles
public **deleteTorsionAngle**(TorsionAngle torsionangle, Collection molecularTorsionAngleAngles):boolean	*Sequential* *Notes:* Removes a TorsionAngleAngle from the set indicated by molecularTorsionAngleAngles.
public **addElectron**(Electron electron, Collection molecularElectrons):boolean	*Sequential* *Notes:* Appends an Electron to the set indicated by molecularElectrons.
public **deleteElectron**(Electron electron, Collection molecularElectrons):boolean	*Sequential* *Notes:* Removes an Electron from the set indicated by molecularElectrons
public **addMolecule**(Molecule molecule, Collection subMolecules):boolean	*Sequential* *Notes:* Appends a sub-Molecule to the set indicated by subMolecules.
public **deleteMolecule**(Molecule molecule, Collection subMolecules):boolean	*Sequential* *Notes:* Removes a Molecule from the set indicated by subMolecules
public **getAtom**(int num, Collection atoms):Atom	*Sequential* *Notes:* Given an integer, n, return the nth Atom in this Molecule's molecularAtoms.
public **hasCoordinates**(Molecule molecule, Collection atoms):Coordinate2	*Sequential* *Notes:* Indicates where this Molecule has coordinates of a specified type (either 2D or 3D)
public **getNonHydrogenAtoms**(Atom nonhydro, Collection atoms):Atom	*Sequential* *Notes:* Returns an array of Atoms associated with this Molecule (molecularAtoms), omitting hydrogens.
public **getCalculatedMolecularMass**(float weight, Molecule molecule):double	*Sequential* *Notes:* Returns the molecular weight of this Molecule by summing the weights of constituent Atoms.

8.2.2 MoleculeFactory

MoleculeFactory is an interface that defines the behavior of a factory that creates **Molecules**. Figure 8.4 and Table 8.2 provide more detailed information.

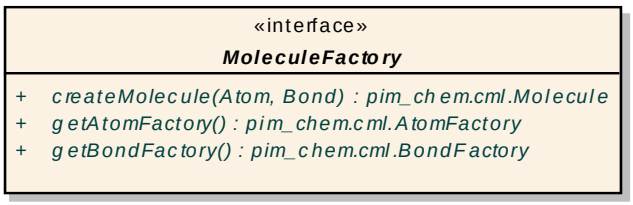


Figure 8.4 - Molecule Factory

Table 8.2 - Molecule Factory Methods

Operation	Details
public **createMolecule**(Atom atoms, Bond bonds):pim_chem.cml.Molecule	*Sequential* *Notes:* return pim_chem.cml.Molecule
public **getAtomFactory**():pim_chem.cml.AtomFactory	*Sequential* *Notes:* return pim_chem.cml.AtomFactory
public **getBondFactory**():pim_chem.cml.BondFactory	*Sequential* *Notes:* return pim_chem.cml.BondFactory

8.2.3 MoleculeUtil

The interface **MoleculeUtil** defines the behavior of something that calculates properties for a given **Molecule**. Figure 8.5 and Table 8.3 provide detailed information.

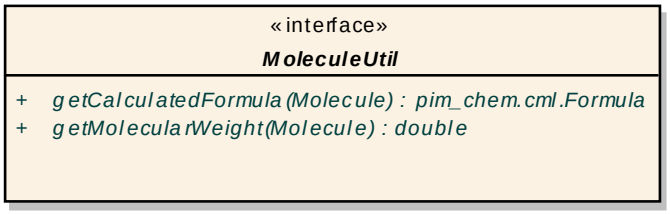


Figure 8.5 - MoleculeUtil Interface

Table 8.3 - MoleculeUtil Interface

Operation	Details
public **getCalculatedFormula**(Molecule structure):pim_chem.cml.Formula	*Sequential* *Notes:* return pim_chem.cml.Formula
public **getMolecularWeight**(Molecule structure):double	*Sequential* *Notes:* return double

8.2.4 Atom

Atom represents a location within a molecule, generally a chemical atom (see Glossary). Figure 8.6 shows the CSAR Atom and interrelated classes and interfaces.

A set of properties is defined. Properties listed as arrays, are generally single-valued for registerable structures (see Glossary) but may have zero to many values for query structures. Table 8.5 provides detailed description of the properties and methods.

Table 8.4 - Atom attributes and methods

Attribute	Details
public double **occupancy**	*whether the position occupied in coordinate space by this **Atom** actually has a chemical atom within it.*
public int **formalCharges**	a value or set of values for this **Atom** indicating whether it has gained (<0) or lost (>0) electrons relative to the uncombined form of the [chemical] element.
public int **hydrogenCounts**	a value or set of values indicating the number of hydrogen atoms attached to this **Atom**. These hydrogen atoms may be used for substructure querying or display.
public int **nonHydrogenCounts**	the number or allowed numbers of heavy **Atoms** attached to this **Atom**.
public java.lang.String **atomID**	string identifier attached to this **Atom**.
public java.lang.String **substitutionCount**	a query property of this **Atom**, used exclusively in database queries, indicating the number of heavy atoms attached. It can take non-negative integral values, plus '*' to indicate 'as drawn.'
public java.lang.String **hydrogenCount**	this is a string property, distinct from the array of integers *hydrogenCounts* that corresponds to the MDL molfile field indicating the number of hydrogens that must be present. It can take non-negative integral values, plus '*' to indicate 'as drawn.'
public java.lang.String **ringMembership**	a query property indicating the number of rings in which this **Atom** participates. It can take non-negative integral values, plus '*' to indicate 'as drawn.'

Table 8.5 - Atom attributes and methods

Attribute	Details
public double **occupancy**	*whether the position occupied in coordinate space by this **Atom** actually has a chemical atom within it.*
public int **formalCharges**	a value or set of values for this **Atom** indicating whether it has gained (<0) or lost (>0) electrons relative to the uncombined form of the [chemical] element.
public int **hydrogenCounts**	a value or set of values indicating the number of hydrogen atoms attached to this **Atom**. These hydrogen atoms may be used for substructure querying or display.
public int **nonHydrogenCounts**	the number or allowed numbers of heavy **Atoms** attached to this **Atom**.
public java.lang.String **atomID**	string identifier attached to this **Atom**.
public java.lang.String **substitutionCount**	a query property of this **Atom**, used exclusively in database queries, indicating the number of heavy atoms attached. It can take non-negative integral values, plus '*' to indicate 'as drawn.'
public java.lang.String **hydrogenCount**	this is a string property, distinct from the array of integers *hydrogenCounts* that corresponds to the MDL molfile field indicating the number of hydrogens that must be present. It can take non-negative integral values, plus '*' to indicate 'as drawn.'
public java.lang.String **ringMembership**	a query property indicating the number of rings in which this **Atom** participates. It can take non-negative integral values, plus '*' to indicate 'as drawn.'

Atom Methods

Operation	Details
public **addLigand**(Atom ligand):void	*Sequential* *Tags:* throws=CMLException appends a new **Atom** to this **Atom**'s list of attachments
public **getLigands**(String atomID):Atom	*Sequential* *Notes:* @return Atom[] @roseuid 4280B2C800DE returns an array of bonded **Atoms**
public **getNonHydrogenLigands**():Atom	*Sequential* *Notes:* @return Atom[] @roseuid 4280B2C800E8 convenience method to provide a list of bonded heavy **Atoms**

8.2.4.1 Associations

- Dependency link to interface [cml].Molecule – A pointer to the set of Atoms that belong to this Molecule.
- Dependency link to interface [cml].Molecule – Identifies the Molecule immediately above the set of Atoms.
- Dependency link to class [cml].NumericAtomParity – Defines the chirality (if any) of this Atom.
- Dependency link to class [cml].Coordinate3 – Relates the Atom to a set of 3D coordinates specifying location in space.
- Dependency link to class [cml].Coordinate2 – Relates the Atom to a set of 2D screen coordinates for display.
- Dependency link from class [cml].Atom – Defines a set of Atoms that are bonded to this Atom.
- Dependency link to interface [cml].FormulaElement – Defines that Atom type by relating it to a ChemicalElement in a periodic table.
- Dependency link to class [cml].Crystal – Relates the Atom to a set of fractional crystal coordinates.

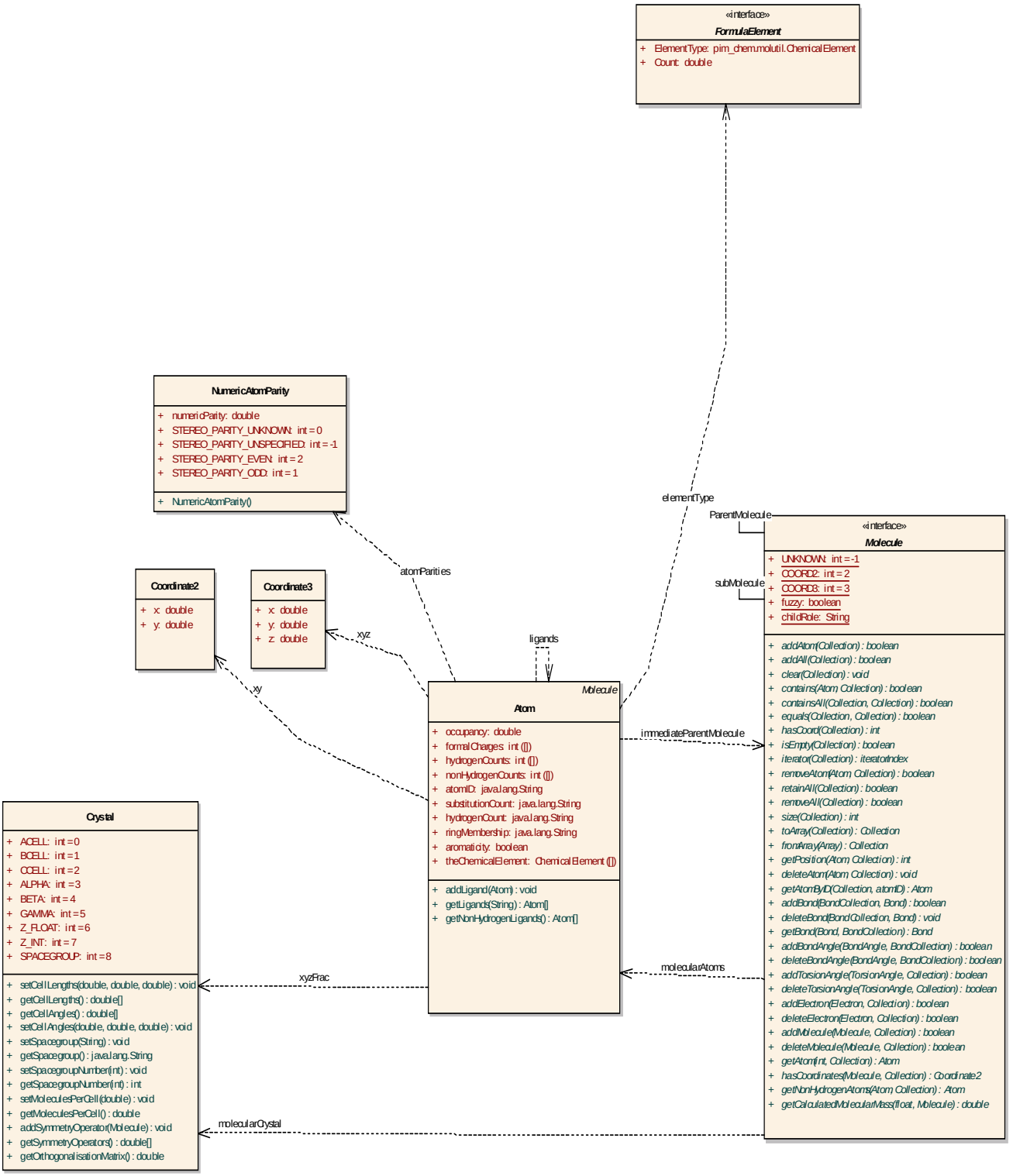


Figure 8.6 - Atom and interrelated classes/interfaces

8.2.5 AtomFactory

The **AtomFactory** interface specifies the behavior of things that create **Atoms**.

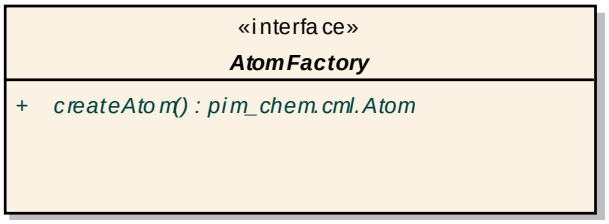


Figure 8.7 - Atom factory

8.2.5.1 Properties

None.

8.2.5.2 Associations

None.

8.2.5.3 Operations

createAtom – instantiates a new **Atom**.

8.2.6 AtomParity

The **AtomParity** interface defines a generalized pattern of behavior for definitions of atom-level chirality. Atom-level chirality means that the atom has some 'handedness' as a tetrahedral atom with 4 unlike groups around it. The setting for this chirality – returned by the getStereoCenter method – can be either numeric (as in MDL software) or a string (as in Daylight software).

Atom parity is optional in some systems; molecular chirality may be fully specified using bond markings.

The **AtomParity** interface is realized by two classes: **NumericAtomParity** and **StringAtomParity**, which store the parity as a double-precision real and a string, respectively (see Figure 8.8).

8.2.6.1 Properties

None.

8.2.6.2 Associations

- Association link from class [cml].Atom
- Realization <<realize>> link from class [cml].NumericAtomParity
- Realization <<realize>> link from class [cml].StringAtomParity

8.2.6.3 Operations

Operation	Details
public **AtomParity**():AtomParity	*Sequential* *Notes:* The AtomParity interface defines a generalized pattern of behavior for definitions of atom-level chirality. Atom-level chirality means that the atom has some 'handedness' as a tetrahedral atom with 4 unlike groups around it. The setting for this chirality – returned by the getStereoCenter method – can be either numeric (as in MDL software) or a string (as in Daylight software).
public **equals**(Object obj):boolean	*Sequential* *Notes:* compares two different AtomParities which may be in different formats
public **isChiral**():boolean	*Sequential* *Notes:* returns true if the Atom has defined asymmetry, for example, as a tetrahedral atom with 4 different substituents .
public **isSpecified**():boolean	*Sequential* *Notes:* returns true when isChiral returns true AND a specific parity is set.
public **getStereoCenter**():String	*Sequential* *Notes:* returns the actual value for this stereocenter

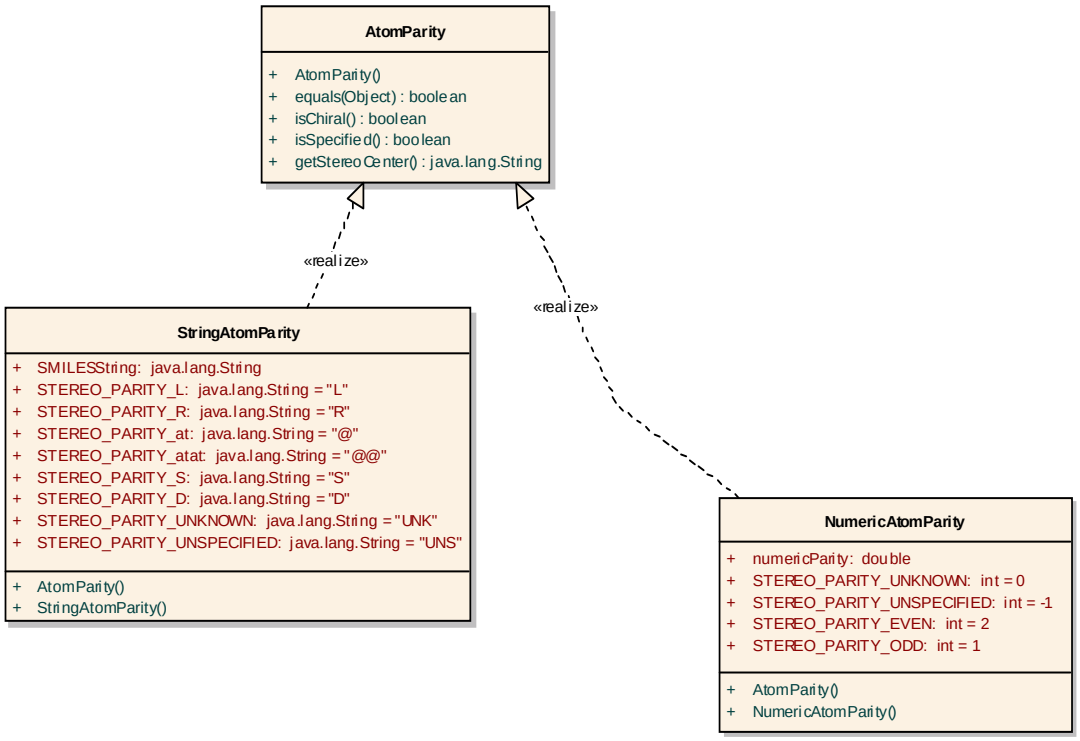


Figure 8.8 - Atom parity

8.2.7 Bond

Bond represents a chemical linkage between two Atoms. Properties listed as arrays are generally single-valued for registerable structures (see glossary) but may have zero to many values for query structures.

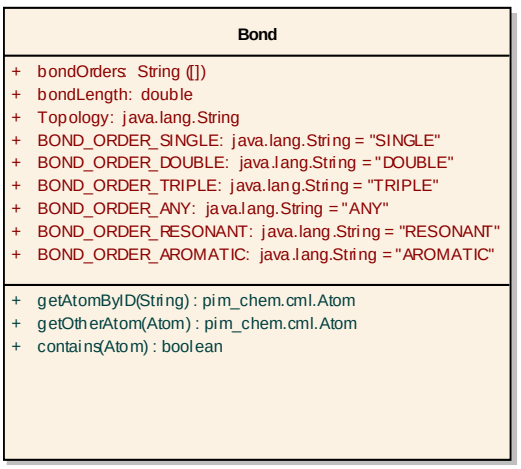


Figure 8.9 - Bond

8.2.7.1 Properties

Attribute	Details
public String **bondOrders**	*Notes:* an indication of the strength of the Bond. Common values include SINGLE, DOUBLE, TRIPLE, AROMATIC, RESONANT, ANY.
public double **bondLength**	*Notes:* the distance between the atoms on either side of the Bond. This only has meaning for 3D structures.
public java.lang.String **Topology**	*Notes:* a query property that indicates whether the Bond is part of a ring, excluded from ring membership or unspecified.
public java.lang.String **BOND_ORDER_SINGLE**	*Initial: "SINGLE"*
public java.lang.String **BOND_ORDER_DOUBLE**	*Initial: "DOUBLE"*
public java.lang.String **BOND_ORDER_TRIPLE**	*Initial: "TRIPLE"*
public java.lang.String **BOND_ORDER_ANY**	*Initial: "ANY"*
public java.lang.String **BOND_ORDER_RESONANT**	*Initial: "RESONANT"*
public java.lang.String **BOND_ORDER_AROMATIC**	*Initial: "AROMATIC"*

8.2.7.2 Associations

- Dependency link to interface [cml].Molecule – A pointer to the set of Bonds that belong to this Molecule.
- Dependency link from class [cml].BondStereo
- Association link to class [cml].Atom – Relates the Bond to its constituent (pair of) Atoms.

8.2.7.3 Operations

Operation	Details
public **getAtomByID**(String atomID):pim_chem.cml.Atom	*Sequential* *Notes:* returns the constituent Atom of this Bond having the specified ID.
public **getOtherAtom**(Atom thisAtom):pim_chem.cml.Atom	*Sequential* *Notes:* given one Atom, return the second Atom constituent of this Bond.
public **contains**(Atom thisAtom):boolean	*Sequential* *Notes:* returns true if the specified Atom is part of this Bond.

8.2.8 BondStereo

Interface **BondStereo** defines the behavior of classes that define a **Bond**'s stereochemistry.

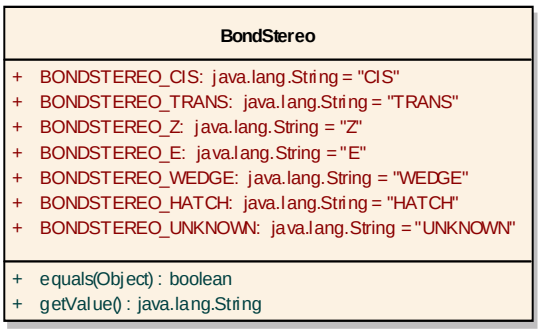


Figure 8.10 - BondStereo

8.2.8.1 Associations

- Association link from class [cml].Bond
- Dependency link to class [cml].Bond

8.2.8.2 Operations

Operation	Details
public **equals**(Object obj):boolean	*Sequential* *Notes:* return boolean
public **getValue**():java.lang.String	*Sequential* *Notes:* return java.lang.String

8.2.9 Electron

Electron is reserved for future use.

8.2.9.1 Properties

None.

8.2.9.2 Associations

None.

8.2.9.3 Operations

None.

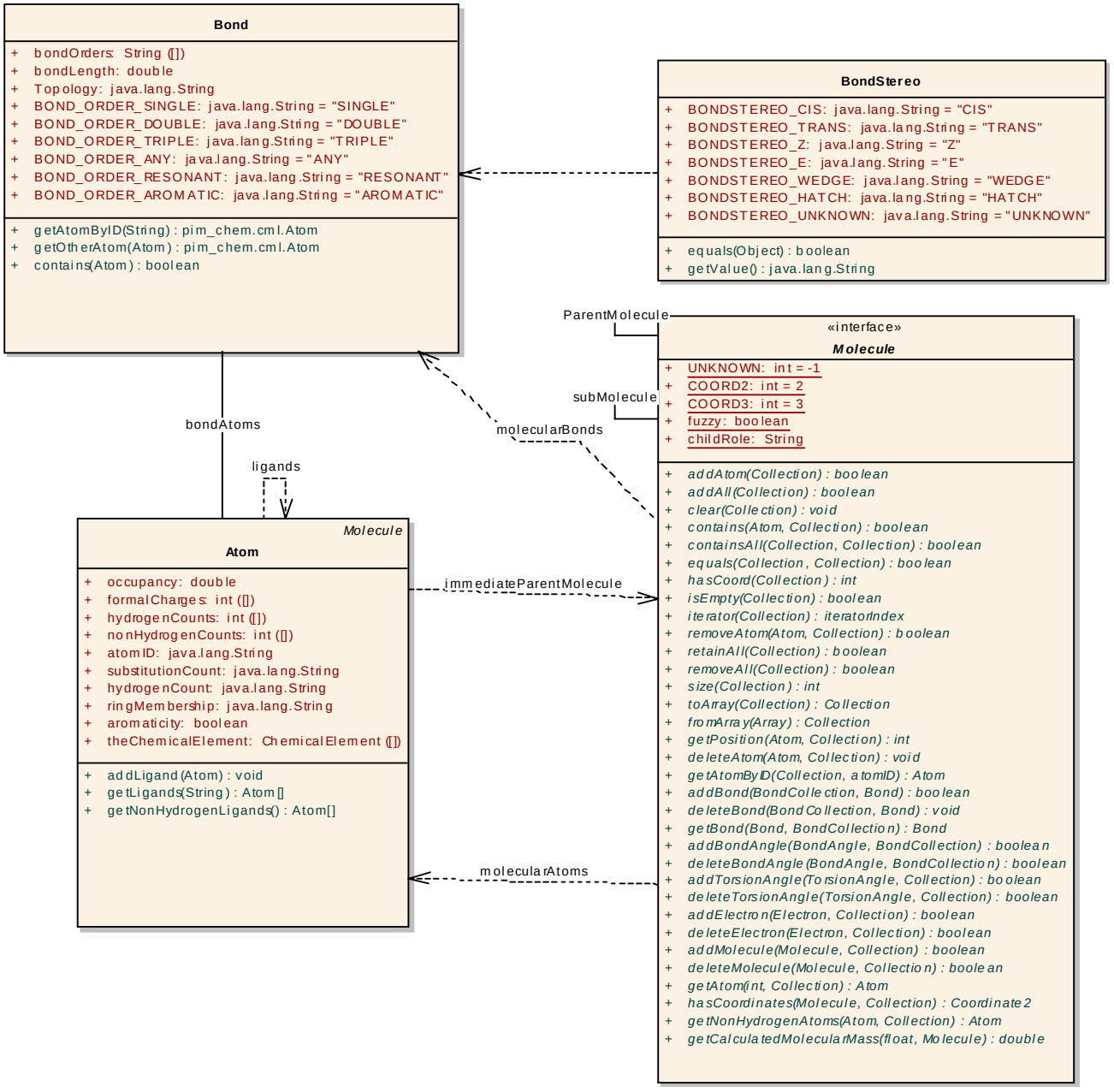


Figure 8.11 - Bond and interrelated classes/interfaces

8.2.10 Isotope

Isotope represents one possible configuration of an atom of a given [chemical] element, defined by its mass. (Isotopes of a given [chemical] element differ from one another because of the number of neutrons.)

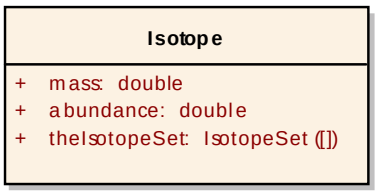


Figure 8.12 - Isotope

8.2.10.1 Properties

Attribute	Details
public double **mass**	*Notes:* inertial property of the atom
public double **abundance**	*Notes:* fraction of the [chemical] containing this
public IsotopeSet **theIsotopeSet**	*Notes:* set of isotopes

8.2.10.2 Associations

None.

8.2.10.3 Operations

None.

8.2.11 IsotopeFactory

IsotopeFactory provides a uniform interface for things that create **Isotopes**.

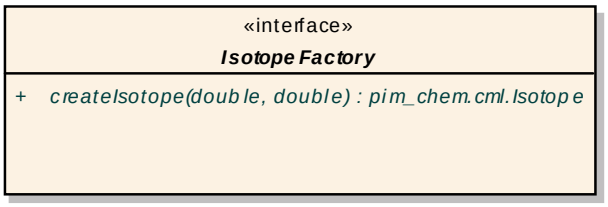


Figure 8.13 - IsotopeFactory

8.2.11.1 Properties

None.

8.2.11.2 Associations

None.

8.2.11.3 Operations

Operation	Details
public **createIsotope**(double mass, double abundance):pim_chem.cml.Isotope	*Sequential* *Notes:* instantiates an **Isotope** object.

8.2.12 AtomFactory

AtomFactory provides a uniform interface for things that create **Atoms.**

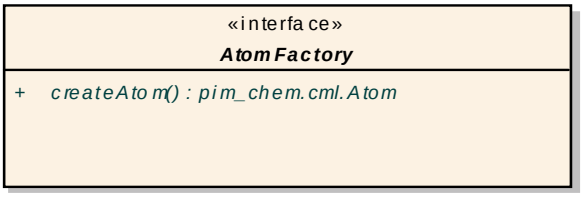


Figure 8.14 - AtomFactory

8.2.12.1 Properties

None.

8.2.12.2 Associations

None.

8.2.12.3 Operations

Operation	Details
public **createAtom**():pim_chem.cml.Atom	*Sequential* *Notes:* instantiates an Atom object.

8.2.13 BondFactory

BondFactory provides a uniform interface for things that create **Bonds**.

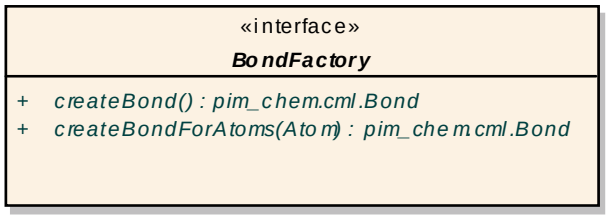


Figure 8.15 - BondFactory

8.2.13.1 Properties

None.

8.2.13.2 Associations

None.

8.2.13.3 Operations

Operation	Details
public **createBond**():pim_chem.cml.Bond	*Sequential* *Notes:* instantiates an empty Bond object.
public **createBondForAtoms**(Atom atm):pim_chem.cml.Bond	*Sequential* *Tags:* throws=CMLException *Notes:* instantiates a Bond given a pair of Atoms.

8.2.14 NumericAtomParity

NumericAtomParity is a realization of the **AtomParity** interface that uses numbers to hold the parity information.

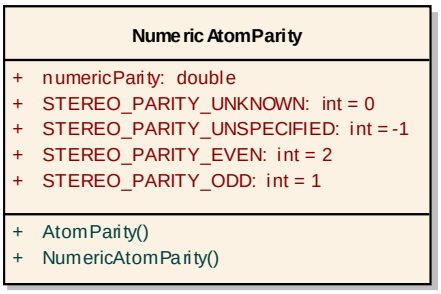


Figure 8.16 - NumericAtomParity

8.2.14.1 Properties

Attribute	Details
public double **numericParity**	*Notes:* the value of the current Atom's stereochemical parity.
public int **STEREO_PARITY_UNKNOWN**	*Initial: 0* *Notes:* an indication that no information is available about the current Atom's stereochemical parity, probably because of a lack of experimental data.
public int **STEREO_PARITY_UNSPECIFIED**	*Initial: -1* *Notes:* an indication that no information has been provided about the current Atom's stereochemical parity, probably because the property has not been given a value.
public int **STEREO_PARITY_EVEN**	*Initial: 2* *Notes:* an indication that the mathematical function describing the current Atom's stereochemical parity gives a value divisible by 2.
public int **STEREO_PARITY_ODD**	*Initial: 1* *Notes:* an indication that the mathematical function describing the current Atom's stereochemical parity gives a value not evenly divisible by 2.

8.2.14.2 Associations

None.

8.2.14.3 Operations

Operation	Details
public **AtomParity**():	*Sequential* *Notes:* The AtomParity interface defines a generalized pattern of behavior for definitions of atom-level chirality. Atom-level chirality means that the atom has some 'handedness' as a tetrahedral atom with 4 unlike groups around it. The setting for this chirality – returned by the getStereoCenter method – can be either numeric (as in MDL software) or a string (as in Daylight software).
public **NumericAtomParity**():	*Sequential* *Notes:* constructor

8.2.15 StringAtomParity

StringAtomParity is a realization of **AtomParity** interface that uses text to hold the parity information.

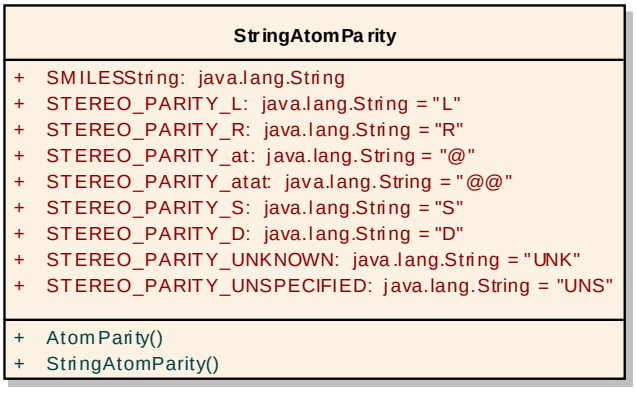


Figure 8.17 - StringAtomParity

8.2.15.1 Properties

Attribute	Details
public java.lang.String **SMILESString**	*Notes:* the value of the current Atom's stereochemical parity.
public java.lang.String **STEREO_PARITY_L**	*Initial: "L"* *Notes:* an indication that the current Atom's stereochemical configuration resembles a standard 'L' Atom.
public java.lang.String **STEREO_PARITY_R**	*Initial: "R"* *Notes:* an indication that the current Atom's stereochemical configuration is classified as 'R' according to the Cahn-Ingold-Prelog rules.
public java.lang.String **STEREO_PARITY_at**	*Initial: "@"* *Notes:* an indication that current Atom has substituents arranged in an 'anticlockwise' fashion.
public java.lang.String **STEREO_PARITY_atat**	*Initial: "@@"* *Notes:* an indication that current Atom has substituents arranged in a 'clockwise' fashion.
public java.lang.String **STEREO_PARITY_S**	*Initial: "S"* *Notes:* an indication that the current Atom's stereochemical parity is classified as 'S' according to the Cahn-Ingold-Prelog rules.
public java.lang.String **STEREO_PARITY_D**	*Initial: "D"* *Notes:* an indication that the current Atom's stereochemical parity resembles a standard 'D' Atom.
public java.lang.String **STEREO_PARITY_UNKNOWN**	*Initial: "UNK"* *Notes:* an indication that no information is available about the current Atom's stereochemical parity, probably because of experimental limitations.
public java.lang.String **STEREO_PARITY_UNSPECIFIED**	*Initial: "UNS"* *Notes:* an indication that no information has been provided about the current Atom's stereochemical parity, possibly because the property has not been given a value.

8.2.15.2 Associations

None.

8.2.15.3 Operations

Operation	Details
public **AtomParity**():	*Sequential* *Notes:* The AtomParity interface defines a generalized pattern of behavior for definitions of atom-level chirality. Atom-level chirality means that the atom has some 'handedness' as a tetrahedral atom with 4 unlike groups around it. The setting for this chirality – returned by the getStereoCenter method – can be either numeric (as in MDL software) or a string (as in Daylight software).
public **StringAtomParity**():	*Sequential* *Notes:* constructor

8.2.16 Coordinate2

Coordinate2 represents a set of x, y coordinates which specify the placement of an atom on a 2D display grid.

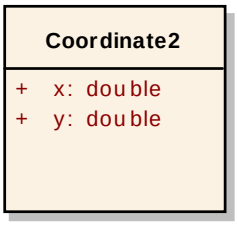


Figure 8.18 - Coordinate2

8.2.16.1 Properties

Attribute	Details
public double **x**	*Notes:* the abscissa of this coordinate set.
public double **y**	*Notes:* the ordinate of this coordinate set.

8.2.16.2 Associations

- Dependency link from class [cml].Atom – Relates the Atom to a set of 2D screen coordinates for display.

8.2.16.3 Operations

None.

8.2.17 Coordinate3

Coordinate3 represents a set of x, y, z coordinates which specify the placement of an atom on a 2D display grid.

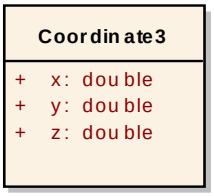


Figure 8.19 - Coordinate3

8.2.17.1 Properties

Attribute	Details
public double **x**	*Notes:* the abscissa of this coordinate set.
public double **y**	*Notes:* the ordinate of this coordinate set
public double **z**	*Notes:* the depth coordinate of this set.

8.2.17.2 Associations

- Dependency link from class [cml].Atom – Relates the Atom to a set of 3D coordinates specifying location in space.

8.2.17.3 Operations

None.

8.2.18 AbstractAngle

A generalization of the behavior of bond (or 3-center) angles and torsional (or 4-center) angles. Realizations: **BondAngle** and **TorsionAngle**.

8.2.18.1 Properties

Attribute	Details
public double **angle**	

8.2.18.2 Associations

- Association link to class [cml].AngleUnits
- Generalization link from class [cml].BondAngle
- Generalization link from class [cml].TorsionAngle

8.2.18.3 Operations

None.

8.2.19 Formula

Formula represents a listing of the atoms and quantities within a **Molecule**. It is built of **formulaElements** (q.v.) blocks. Since there are multiple ways of calculating the molecular formula for a given molecule, (depending on, for example, counting salt fragments that are not explicitly included in the structure), there may be more than one **Formula** for a given **Molecule** and therefore, **Formulas** may contain sub-**Formulas**.

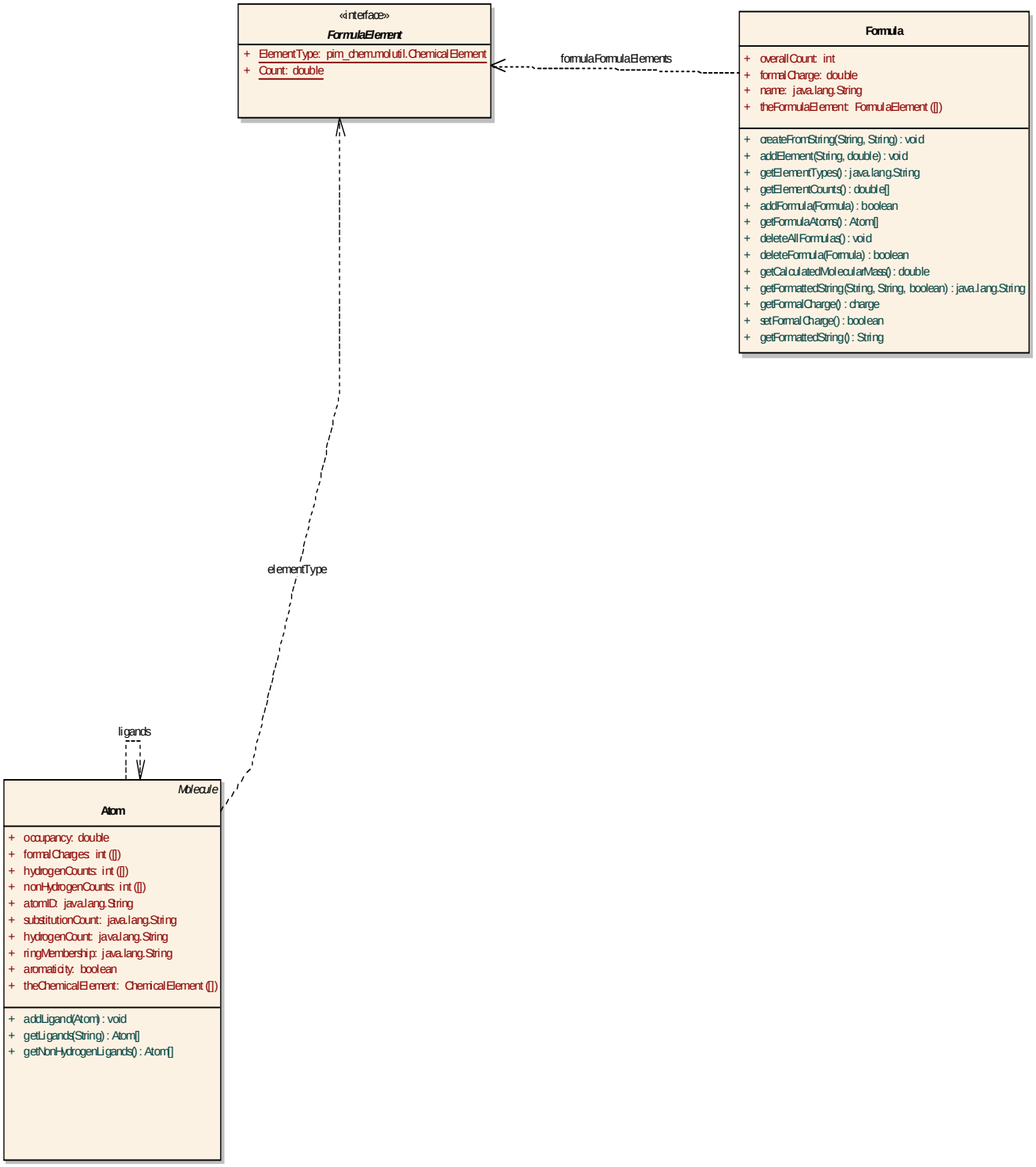


Figure 8.20 - Formula Elements

8.2.19.1 Properties

Attribute	Details
public int **overallCount**	*Notes:* total number of atoms in this formula.
public double **formalCharge**	
public java.lang.String **name**	
public FormulaElement **theFormulaElement**	

8.2.19.2 Associations

- Dependency link to interface [cml].FormulaElement – Relates the Formula to the constituent formulaElements.

8.2.19.3 Operations

Operation	Details
public **createFromString**(String formulaString, String formulaConvention):void	*Sequential* *Tags:* throws=CMLException *Notes:* initializes a Formula from text input
public **addElement**(String elementType, double count):void	*Sequential* *Notes:* append a [chemical] element (including a count) to the Formula
public **getElementTypes**():java.lang.String	*Sequential* *Notes:* returns an array representing the types of atoms present
public **getElementCounts**():double	*Sequential* *Notes:* returns an array of numbers representing the number of times each type of atom (from the array returned by getElementTypes) occurs in the Formula.
public **addFormula**(Formula form):boolean	*Sequential* *Notes:* appends a Formula representation to this Formula.
public **getFormulaAtoms**():Atom	*Sequential* *Notes:* returns an array of Atoms represented herein.
public **deleteAllFormulas**():void	*Sequential* *Notes:* remove all sub-Formulas from this Formula.
public **deleteFormula**(Formula form):boolean	*Sequential* *Notes:* remove one sub-Formula from this Formula.

public **getCalculatedMolecularMass**():double	*Sequential* *Tags:* throws=CMLException *Notes:* @return double
public **getFormattedString**(String convention, String sort, boolean omitCount):java.lang.String	*Sequential* *Notes:* @return java.lang.String @roseuid 4280B2C90016
public **getFormalCharge**():charge	*Sequential* *Notes:* return the surfeit or deficit of electrons in this Formula
public **setFormalCharge**():boolean	*Sequential* *Notes:* change the surfeit or deficit of electrons in this Formula.
public **getCalculatedMolecularMass**():molecularmass	*Sequential* *Notes:* returns the weight generated for this Formula.
public **getFormattedString**():String	*Sequential* *Notes:* generate a printable text representation of Formula, given a display mode.

8.2.20 FormulaElement

It refers to a combination of atom type ([chemical] element) and count. It is used in defining molecular formulas.

8.2.20.1 Properties

Attribute	Details
public pim_chem.molutil.ChemicalElement **ElementType**	pointer to an entry in a periodic table defining a kind of atom
public double **Count**	the number of times an *ElementType* occurs in a formula unit.

8.2.20.2 Associations

- Dependency link from class [cml].Atom – Defines that Atom type by relating it to a ChemicalElement in a periodic table.
- Dependency link from class [cml].Formula – Relates the Formula to the constituent formulaElements.

8.2.20.3 Operations

None.

8.2.21 Crystal

Crystal is a homogenous solid formed by a repeating, three-dimensional pattern of atoms, ions, or molecules and having fixed distances between constituent parts.

Figure 8.21 - Crystal

8.2.21.1 Properties

Attribute	Details
public const int **ACELL**	*Initial: 0*
public const int **BCELL**	*Initial: 1*
public const int **CCELL**	*Initial: 2*
public const int **ALPHA**	*Initial: 3*
public const int **BETA**	*Initial: 4*
public const int **GAMMA**	*Initial: 5*
public const int **Z_FLOAT**	*Initial: 6*
public const int **Z_INT**	*Initial: 7*
public const int **SPACEGROUP**	*Initial: 8*

8.2.21.2 Associations

- Dependency link from interface [cml].Molecule – A pointer to a Crystal unit, defining the crystal structure of this molecule.
- Dependency link from class [cml].Atom – Relates the Atom to a set of fractional crystal coordinates.

8.2.21.3 Operations

None.

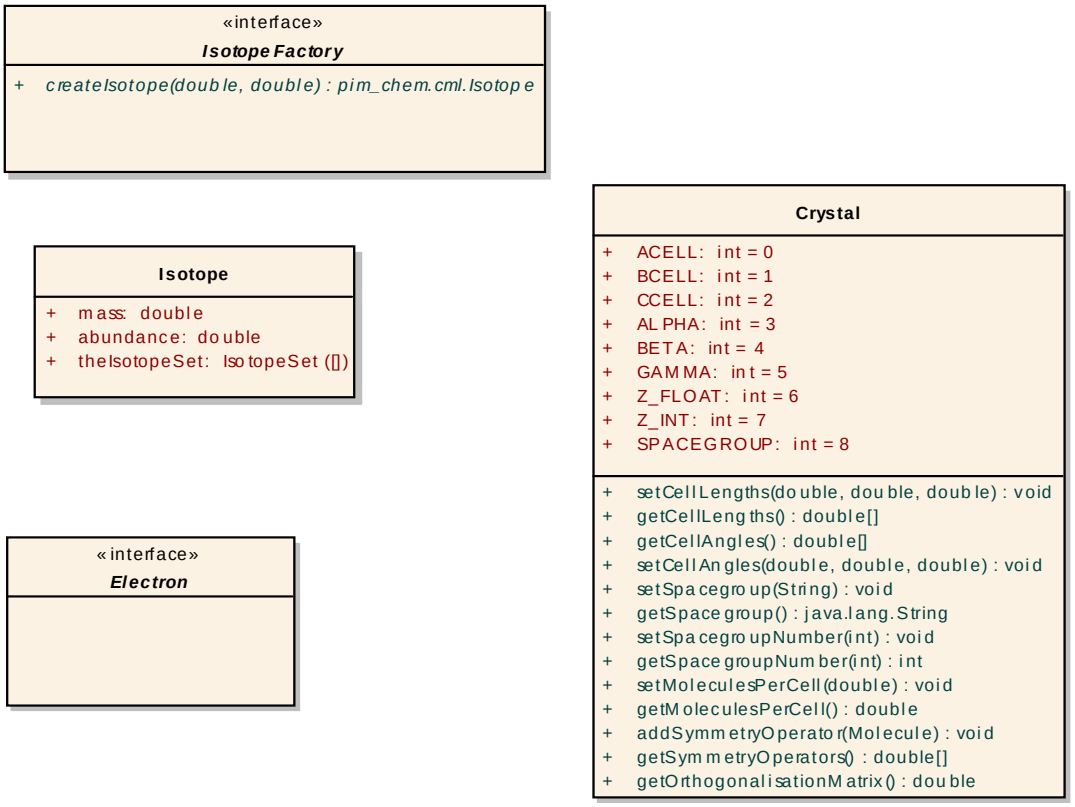


Figure 8.22 - CSAR Crystal and Others

8.3 Molutil Module

Before we describe this module, general information regarding [Chemical] elements is provided here.

Periodic Table of Elements - This table gives information about the chemical elements. Elements are grouped into eight classes according to their properties.

Elements - Each element has a fixed number of positively charged protons in its nucleus and an equal number of electrons orbiting the nucleus. For example, hydrogen (H) has one proton and one electron, but lead (Pb) has 82 protons and 82 electrons. There are about 115 known elements of which 82 are naturally abundant.

Isotopes - The nucleus contains both protons and neutrons. An element has a fixed number of protons but may exist with various numbers of neutrons. The sum of the protons and neutrons is the mass number. For example, helium exists as 3He(2 protons and one neutron) or as 4He (2 protons and 2 neutrons). The two forms of helium are called isotopes of helium. Isotopes of an element have the same chemical properties but different weights. Some elements have many isomers. Tin (Sn) has about 38 known isotopes.

The MOLUtl module, see Figure 8.23, contains a number of interfaces whose main function is to provide utility functionality. For example, there are the PeriodicTableFactory and PeriodicTable interfaces that are used to construct periodic tables. In addition this module provides the ChemicalElementFactory and the ChemicalElement interfaces used to manufacture chemical elements. Moreover since each chemical element may appear as an isotope, an isotopeSetFactory, isotopeSet, and isotope classes/interfaces are provided.

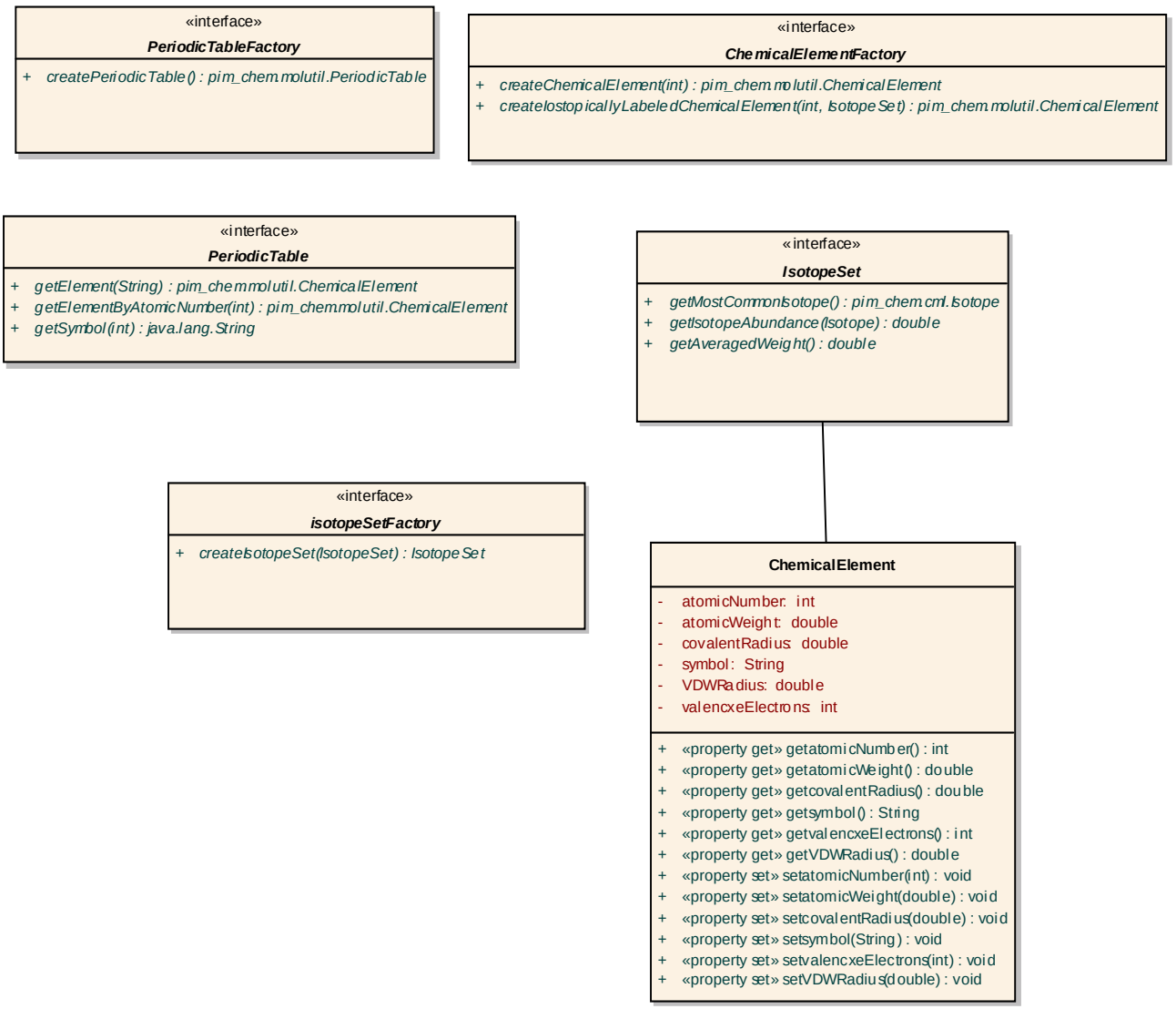


Figure 8.23 - MOLUtil Module

8.3.1 ChemicalElement

ChemicalElement provides a way of describing the properties of a category of **Atoms**, related by having the same number of protons in the nucleus.

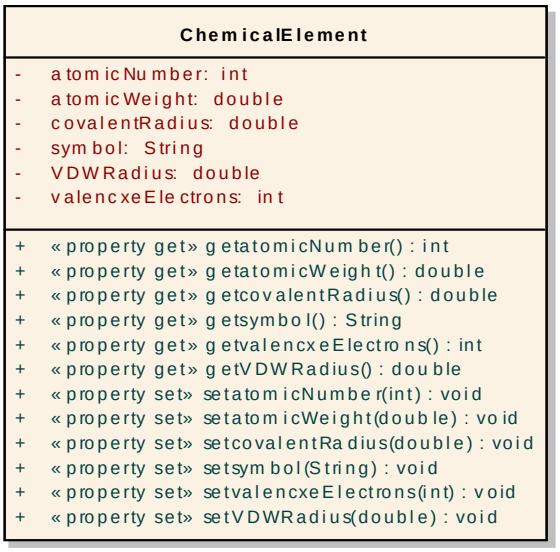


Figure 8.24 - Chemical Element

8.3.1.1 Properties

Attribute	Details
private int **atomicNumber**	*Notes:* an integer that identifies this [chemical] element, equal to the number of protons in the nucleus
private double **atomicWeight**	*Notes:* the gravitational mass of the [chemical] element relative to carbon (a standard). This number is generally a weighted average of the Isotopes that make up the [chemical] element.
private double **covalentRadius**	*Notes:* one half of the distance between two singly-bonded atoms of the [chemical] element.
private String **symbol**	*Notes:* 1-3 letters that are used to represent the [chemical] element
private double **VDWRadius**	*Notes:* the closest a non-bonded atom can approach without incurring very strong repulsive forces.
private int **valencxeElectrons**	*Notes:* number of electrons in the outermost shell of an atom of this [chemical] element.

8.3.1.2 Associations

- Association link to interface [molutil].IsotopeSet

8.3.1.3 Operations

Operation	Details
public **getatomicNumber**():int	<<*property get*>> *Tags:* attribute_name=atomicNumber
public **getatomicWeight**():double	<<*property get*>> *Tags:* attribute_name=atomicWeight
public **getcovalentRadius**():double	<<*property get*>> *Tags:* attribute_name=covalentRadius
public **getsymbol**():String	<<*property get*>> *Tags:* attribute_name=symbol
public **getvalencxeElectrons**():int	<<*property get*>> *Tags:* attribute_name=valencxeElectrons
public **getVDWRadius**():double	<<*property get*>> *Tags:* attribute_name=VDWRadius
public **setatomicNumber**(int newVal):void	<<*property set*>> *Tags:* attribute_name=atomicNumber
public **setatomicWeight**(double newVal):void	<<*property set*>> *Tags:* attribute_name=atomicWeight
public **setcovalentRadius**(double newVal):void	<<*property set*>> *Tags:* attribute_name=covalentRadius
public **setsymbol**(String newVal):void	<<*property set*>> *Tags:* attribute_name=symbol
public **setvalencxeElectrons**(int newVal):void	<<*property set*>> *Tags:* attribute_name=valencxeElectrons
public **setVDWRadius**(double newVal):void	<<*property set*>> *Tags:* attribute_name=VDWRadius

8.3.2 ChemicalElementFactory

ChemicalElementFactory provides a uniform interface for things that create **ChemicalElements**.

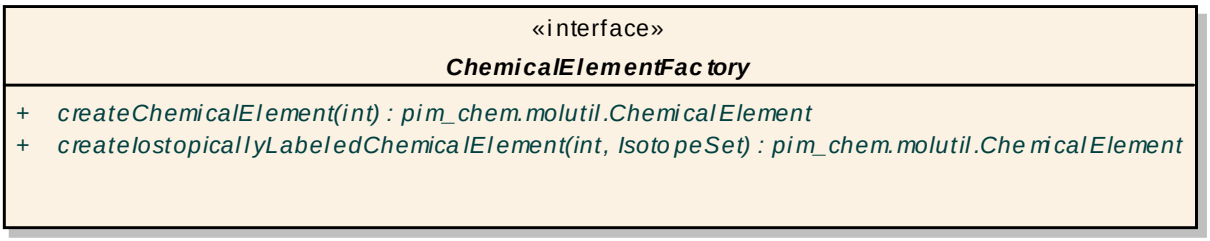


Figure 8.25 - ChemicalElementFactory

8.3.2.1 Properties

None.

8.3.2.2 Associations

None.

8.3.2.3 Operations

Operation	Details
public **createChemicalElement**(int atomicNumber):pim_chem.molutil.ChemicalElement	*Sequential* *Notes:* instantiates a ChemicalElement having a specified atomic number.
public **createIsotopicallyLabeledChemicalElement**(int atomicNumber, IsotopeSet isotopes):pim_chem.molutil.ChemicalElement	*Sequential* *Notes:* instantiates a ChemicalElement having a specified atomic number and a given set of Isotopes.

8.3.3 IsotopeSet

An **IsotopeSet** is a grouping of **Isotopes** that define the composition of a sample of the [chemical] element.

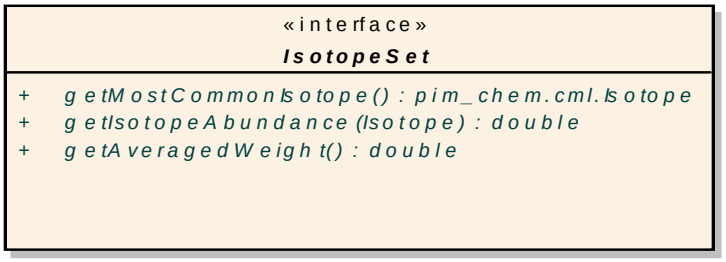


Figure 8.26 - IsotopeSet

8.3.3.1 Properties

None.

8.3.3.2 Associations

- Association link from class [cml].Isotope
- Association link to class [cml].Isotope
- Association link to class [cml].Isotope
- Association link from class [molutil].ChemicalElement

8.3.3.3 Operations

Operation	Details
public **getMostCommonIsotope**():pim_chem.cml.Isotope	*Sequential* *Notes:* returns the most prominent Isotope of the current [chemical] element.
public **getIsotopeAbundance**(Isotope isotope):double	*Sequential* *Notes:* returns the fraction of a given Isotope within the set.
public **getAveragedWeight**():double	*Sequential* *Notes:* returns the mean of the weights of the Isotopes making up the set

8.3.4 IsotopeSetFactory

IsotopeSetFactory provides a uniform interface for things that create **IsotopeSets**.

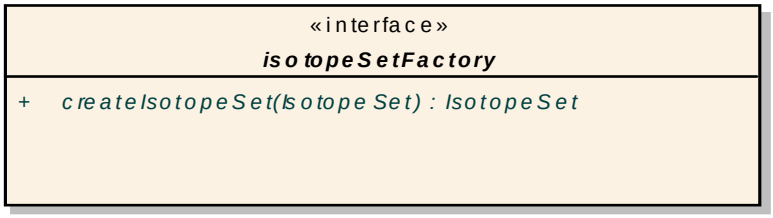


Figure 8.27 - IsotopeSetFactory

8.3.4.1 Properties

None.

8.3.4.2 Associations

None.

8.3.4.3 Operations

Operation	Details
public **createIsotopeSet**(IsotopeSet isotopes):IsotopeSet	*Sequential* *Notes:* instantiates an IsotopeSet

8.3.5 PeriodicTable

A grouping of **Chemical Elements** that provides a complete representation of all the atom types used in some chemical system.

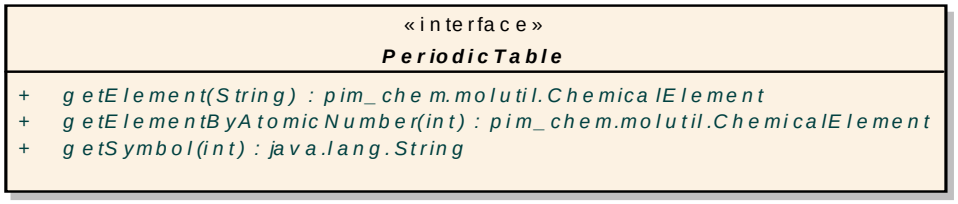


Figure 8.28 - Periodic Table

8.3.5.1 Properties

None.

8.3.5.2 Associations

None.

8.3.5.3 Operations

Operation	Details
public **getElement**(String symbol):pim_chem.molutil.ChemicalElement	*Sequential* *Notes:* given an atomic symbol, return the corresponding ChemicalElement.
public **getElementByAtomicNumber**(int atomicNumber):pim_chem.molutil.ChemicalElement	*Sequential* *Notes:* given a number, return the ChemicalElement with that many protons.
public **getSymbol**(int atomicNumber):java.lang.String	*Sequential* *Notes:* given a number, return the atomic symbol for the ChemicalElement with that many protons.

8.3.6 PeriodicTableFactory

PeriodicTableFactory provides a uniform interface for things that create **PeriodicTables**.

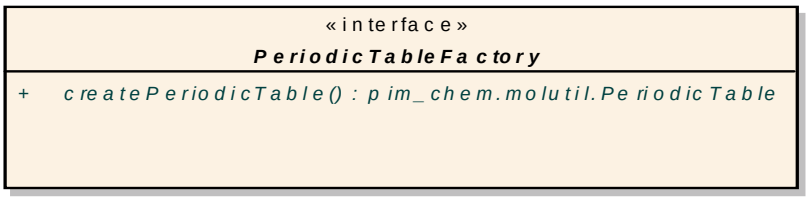


Figure 8.29 PeriodicTable Factory

8.3.6.1 Properties

None.

8.3.6.2 Associations

None.

8.3.6.3 Operations

Operation	Details
public **createPeriodicTable**():pim_chem.molutil.PeriodicTable	*Sequential* *Notes:* instantiates an empty PeriodicTable

8.4 Search Component

Search is one of the more important transactional operations. It is required for every type of processing such as registering components, comparing elements, and others. For example, a typical interaction will begin as follows: using ISIS/Draw to sketch a molecule for a substructure search of a Daylight database.

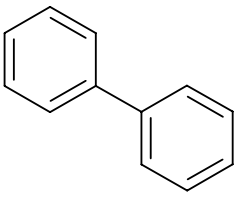


Figure 8.30 - Substructure search

Another typical search will use both intrinsic and extrinsic properties. In this particular case, the intrinsic properties are stored in proprietary databases and the extrinsic are stored in relational databases, in most cases Oracle. This specification only deals with the search of intrinsic properties. Figure 8.31 and Figure 8.32 illustrate the UML description of this component.

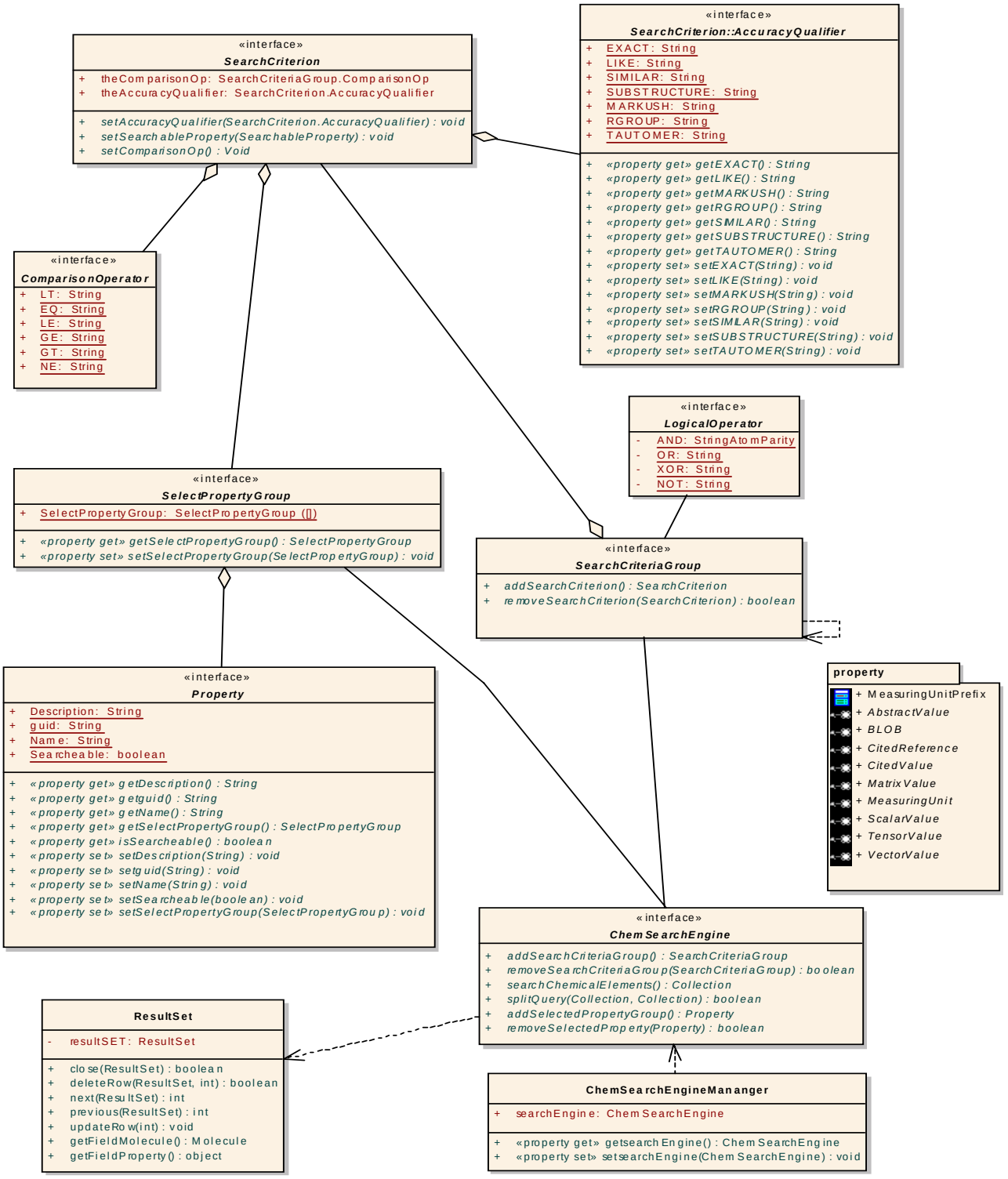


Figure 8.31 - Search Module

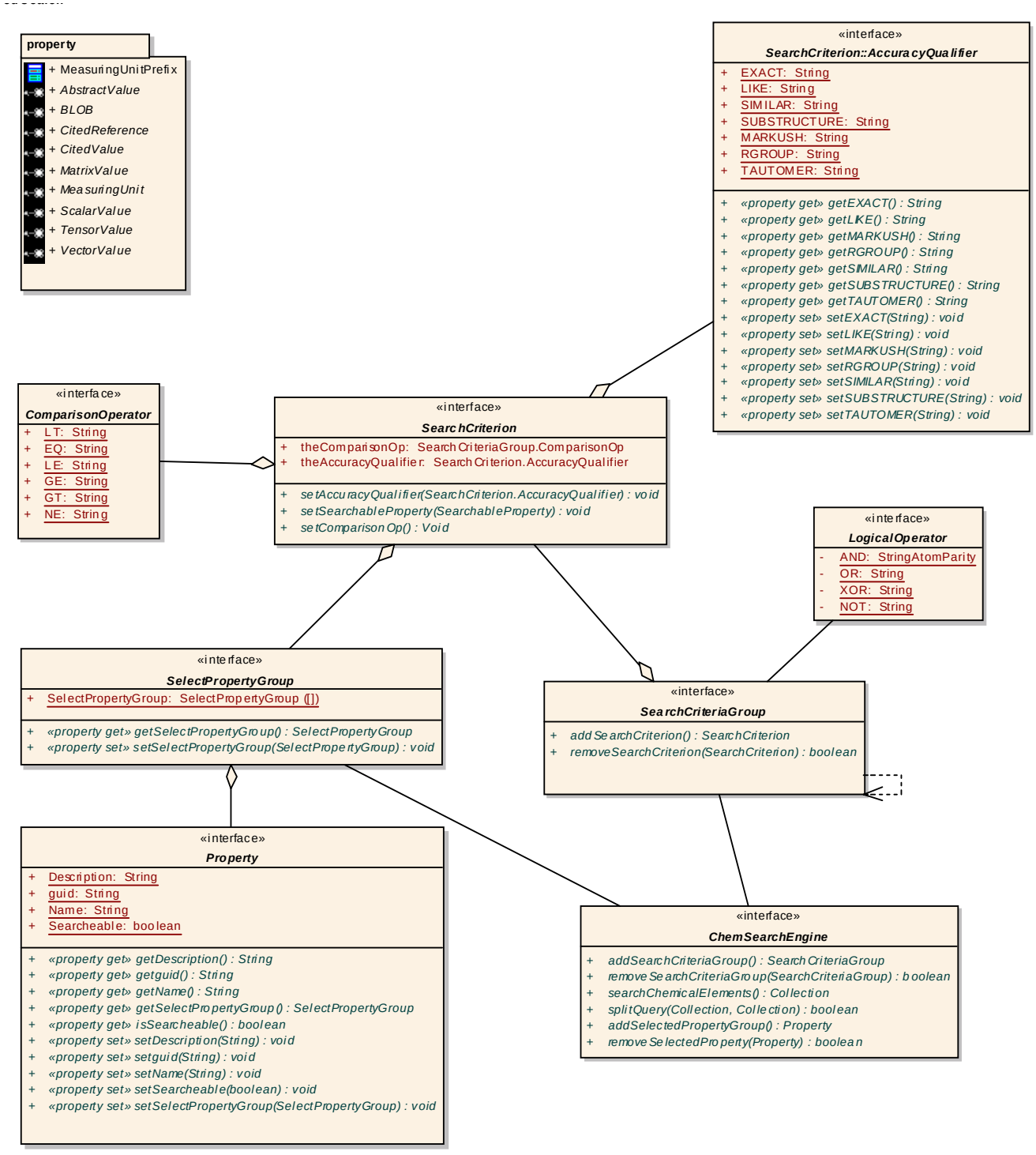


Figure 8.32 - Search Module (Properties Classes)

The following sections describe the modules in detail.

8.4.1 ChemSearchEngineManager

The ChemSearchEngineManager interface acts like a factory creating a given ChemSearchEngine.

8.4.1.1 Connections

- Dependency link to interface [search].ChemSearchEngine

8.4.1.2 ChemSearchEngineManager Attributes

Attribute	Details
public ChemSearchEngine **searchEngine**	

8.4.1.3 ChemSearchEngineManager Methods

Operation	Details
public **getsearchEngine**():ChemSearchEngine	*<<property get>>* *Tags:* attribute_name=searchEngine
public **setsearchEngine**(ChemSearchEngine newVal):void	*<<property set>>* *Tags:* attribute_name=searchEngine

8.4.2 ResultSet

The ResultSet is a class that instantiates the result of executing the query against the proprietary databases. As any typical result set, the set is composed of one or more rows for each molecule and property pair. Methods are provided to close() the result set and to navigate the result set such as move next() and previous() and to manipulate the result set such as updateRow() and deleteRow().

8.4.2.1 Connections

- Dependency link from interface [search].ChemSearchEngine

8.4.2.2 ResultSet Attributes

Attribute	Details
private ResultSet **resultSET**	*Notes:* Containier

8.4.2.3 ResultSet Methods

Operation	Details
public **close**(ResultSet set):boolean	*Sequential* *Notes:* Closes the set
public **deleteRow**(ResultSet set, int row):boolean	*Sequential* *Notes:* Deletes one row of the given set returns true if done
public **next**(ResultSet set):int	*Sequential* *Notes:* iterates over the set
public **previous**(ResultSet set):int	*Sequential* *Notes:* Moves to previous record
public **updateRow**(int row):void	*Sequential* *Notes:* Updates a row in the set
public **getFieldMolecule**():Molecule	*Sequential*
public **getFieldProperty**():object	*Sequential*

8.4.3 AccuracyQualifier

8.4.3.1 Connections

- Aggregation link to interface [search].SearchCriterion

8.4.3.2 AccuracyQualifier Attributes

public static String **EXACT**	
public static String **LIKE**	
public static String **SIMILAR**	
public static String **SUBSTRUCTURE**	
public static String **MARKUSH**	
public static String **RGROUP**	
public static String **TAUTOMER**	

8.4.3.3 AccuracyQualifier Methods

Operation	Details
public **getEXACT**():String	<<*property get*>> *Tags:* attribute_name=EXACT
public **getLIKE**():String	<<*property get*>> *Tags:* attribute_name=LIKE
public **getMARKUSH**():String	<<*property get*>> *Tags:* attribute_name=MARKUSH
public **getRGROUP**():String	<<*property get*>> *Tags:* attribute_name=RGROUP
public **getSIMILAR**():String	<<*property get*>> *Tags:* attribute_name=SIMILAR
public **getSUBSTRUCTURE**():String	<<*property get*>> *Tags:* attribute_name=SUBSTRUCTURE
public **getTAUTOMER**():String	<<*property get*>> *Tags:* attribute_name=TAUTOMER
public **setEXACT**(String newVal):void	<<*property set*>> *Tags:* attribute_name=EXACT
public **setLIKE**(String newVal):void	<<*property set*>> *Tags:* attribute_name=LIKE
public **setMARKUSH**(String newVal):void	<<*property set*>> *Tags:* attribute_name=MARKUSH
public **setRGROUP**(String newVal):void	<<*property set*>> *Tags:* attribute_name=RGROUP
public **setSIMILAR**(String newVal):void	<<*property set*>> *Tags:* attribute_name=SIMILAR
public **setSUBSTRUCTURE**(String newVal):void	<<*property set*>> *Tags:* attribute_name=SUBSTRUCTURE
public **setTAUTOMER**(String newVal):void	<<*property set*>> *Tags:* attribute_name=TAUTOMER

8.4.4 ChemSearchEngine

8.4.4.1 Connections

- Association link to interface [search].SearchCriteriaGroup
- Association link from interface [search].SelectPropertyGroup
- Dependency link to class [search].ResultSet
- Dependency link from class [search].ChemSearchEngineMananger

8.4.4.2 ChemSearchEngine Methods

Operation	Details
public **addSearchCriteriaGroup**():SearchCriteriaGroup	*Sequential*
public **removeSearchCriteriaGroup**(SearchCriteriaGroup criteriaGroup):boolean	*Sequential*
public **searchChemicalElements**():Collection	*Sequential*
public **splitQuery**(Collection chemicalPropertiesOracle, Collection atoms):boolean	*Sequential*
public **addSelectedPropertyGroup**():Property	*Sequential*
public **removeSelectedProperty**(Property property):boolean	*Sequential*

8.4.5 ComparisonOperator

8.4.5.1 Connections

- Aggregation link to interface [search].SearchCriterion

8.4.5.2 ComparisonOperator Attributes

Attribute	Details
public static String **LT**	
public static String **EQ**	
public static String **LE**	
public static String **GE**	
public static String **GT**	
public static String **NE**	

8.4.6 LogicalOperator

8.4.6.1 Connections

- Association link to interface [search].SearchCriteriaGroup

8.4.6.2 LogicalOperator Attributes

Attribute	Details
private static StringAtomParity **AND**	
private static String **OR**	
private static String **XOR**	
private static String **NOT**	

8.4.7 Property

8.4.7.1 Connections

- Aggregation link to interface [search].SelectPropertyGroup

8.4.7.2 Property Attributes

Attribute	Details
public static String **Description**	
public static String **guid**	
public static String **Name**	
public static boolean **Searcheable**	

8.4.7.3 Property Methods

Operation	Details
public **getDescription**():String	*<<property get>>* *Tags:* attribute_name=Description
public **getguid**():String	*<<property get>>* *Tags:* attribute_name=guid
public **getName**():String	*<<property get>>* *Tags:* attribute_name=Name
public **getSelectPropertyGroup**():SelectPropertyGroup	*<<property get>>* *Tags:* attribute_name=SelectPropertyGroup
public **isSearcheable**():boolean	*<<property get>>* *Tags:* attribute_name=Searcheable
public **setDescription**(String newVal):void	*<<property set>>* *Tags:* attribute_name=Description
public **setguid**(String newVal):void	*<<property set>>* *Tags:* attribute_name=guid
public **setName**(String newVal):void	*<<property set>>* *Tags:* attribute_name=Name
public **setSearcheable**(boolean newVal):void	*<<property set>>* *Tags:* attribute_name=Searcheable
public **setSelectPropertyGroup**(SelectPropertyGroup newVal):void	*<<property set>>* *Tags:* attribute_name=SelectPropertyGroup

8.4.8 SearchCriteriaGroup

8.4.8.1 Connections

- Aggregation link from interface [search].SearchCriterion
- Association link from interface [search].LogicalOperator
- Dependency link from interface [search].SearchCriteriaGroup
- Association link from interface [search].ChemSearchEngine

8.4.8.2 SearchCriteriaGroup Methods

Operation	Details
public **addSearchCriterion**():SearchCriterion	*Sequential*
public **removeSearchCriterion**(SearchCriterion criterion):boolean	*Sequential*

8.4.9 SearchCriterion

Search criterion. Consists of various properties the searchable structure should possess to.

8.4.9.1 Connections

- Aggregation link from interface [search].AccuracyQualifier
- Aggregation link from interface [search].SelectPropertyGroup
- Aggregation link from interface [search].ComparisonOperator
- Aggregation link to interface [search].SearchCriteriaGroup

8.4.9.2 SearchCriterion Attributes

Attribute	Details
public *SearchCriteriaGroup.ComparisonOp* **theComparisonOp**	
public *SearchCriterion.AccuracyQualifier* **theAccuracyQualifier**	

8.4.9.3 SearchCriterion Methods

Operation	Details
public **setAccuracyQualifier**(SearchCriterion.AccuracyQualifier arg0):void	*Sequential*
public **setSearchableProperty**(SearchableProperty arg0):void	*Sequential*
public **setComparisonOp**():Void	*Sequential* *Notes:* return Void

8.4.10 SearcheableProperty

8.4.10.1 SearchableProperty Attributes

Attribute	Details
private static String **Description**	
public static String **guid**	
public static String **Name**	

8.4.10.2 SearchableProperty Methods

Operation	Details
public **getDescription**():String	<<*property get*>> *Tags:* attribute_name=Description
public **getguid**():String	<<*property get*>> *Tags:* attribute_name=guid
public **getName**():String	<<*property get*>> *Tags:* attribute_name=Name
public **setDescription**(String newVal):void	<<*property set*>> *Tags:* attribute_name=Description
public **setguid**(String newVal):void	<<*property set*>> *Tags:* attribute_name=guid
public **setName**(String newVal):void	<<*property set*>> *Tags:* attribute_name=Name

8.4.11 SelectPropertyGroup

This is abstract property which could be used in search through chemical collections.

8.4.11.1 Connections

- Aggregation link from interface [search].Property
- Aggregation link to interface [search].SearchCriterion
- Association link to interface [search].ChemSearchEngine

8.4.11.2 SelectPropertyGroup Attributes

Attribute	Details
public static SelectPropertyGroup **SelectPropertyGroup**	

8.4.11.3 SelectPropertyGroup Methods

Operation	Details
public **getSelectPropertyGroup**():SelectPropertyGroup	*<<property get>>* *Tags:* attribute_name=SelectPropertyGroup
public **setSelectPropertyGroup**(SelectPropertyGroup newVal):void	*<<property set>>* *Tags:* attribute_name=SelectPropertyGroup

8.5 Property

8.5.1 MeasuringUnitPrefix

8.5.1.1 Connections

- Aggregation link to interface [property].AbstractValue

8.5.1.2 MeasuringUnitPrefix Attributes

Attribute	Details
public java.lang.String **MEASURING_UNIT_PREFIX_KILO**	*Initial: KILO*
public java.lang.String **MEASURING_UNIT_PREFIX_MEGA**	*Initial: MEGA*
public java.lang.String **MEASURING_UNIT_PREFIX_GIGA**	*Initial: GIGA*
public java.lang.String **MEASURING_UNIT_PREFIX_TERA**	*Initial: TERA*
public java.lang.String **MEASURING_UNIT_PREFIX_MILLI**	*Initial: MILLI*
public java.lang.String **MEASURING_UNIT_PREFIX_MICRO**	*Initial: MICRO*
public java.lang.String **MEASURING_UNIT_PREFIX_NANO**	*Initial: NANO*
public java.lang.String **MEASURING_UNIT_PREFIX_PICO**	*Initial: PICO*
public java.lang.String **MEASURING_UNIT_PREFIX_FEMTO**	*Initial: FEMTO*
public java.lang.String **MEASURING_UNIT_PREFIX_ATTO**	*Initial: ATTO*
public java.lang.String **MEASURING_UNIT_PREFIX_PETA**	*Initial: PETA*
public java.lang.String **MEASURING_UNIT_PREFIX_EXA**	*Initial: EXA*
public java.lang.String **prefix**	
public java.lang.String **description**	

8.5.1.3 MeasuringUnitPrefix Methods

Operation	Details
public **MeasuringUnitPrefix**():	*Sequential* *Notes:*

8.5.2 AbstractValue

8.5.2.1 Connections

- Aggregation link from interface [property].MeasuringUnit
- Aggregation link from class [property].MeasuringUnitPrefix
- Aggregation link from interface [property].VectorValue
- Aggregation link from interface [property].TensorValue
- Aggregation link from interface [property].ScalarValue
- Aggregation link from interface [property].MatrixValue
- Aggregation link from interface [property].BLOB
- Aggregation link from interface [property].CitedValue

8.5.3 BLOB

8.5.3.1 Connections

- Aggregation link to interface [property].AbstractValue

8.5.4 CitedReference

8.5.4.1 Connections

- Aggregation link to interface [property].CitedValue

8.5.4.2 CitedReference Attributes

Attribute	Details
private static wstring **source**	*Notes:* It could be reference to journal, conference, ...
private static String **date**	*Notes:* Value obtained date

8.5.4.3 CitedReference Methods

Operation	Details
public **getdate**():String	<<*property get*>> *Tags:* attribute_name=date *Notes:* Value obtained date
public **getsource**():wstring	<<*property get*>> *Tags:* attribute_name=source *Notes:* It could be reference to journal, conference, ...
public **setdate**(String newVal):void	<<*property set*>> *Tags:* attribute_name=date *Notes:* Value obtained date
public **setsource**(wstring newVal):void	<<*property set*>> *Tags:* attribute_name=source *Notes:* It could be reference to journal, conference, ...

8.5.5 CitedValue

8.5.5.1 Connections

- Aggregation link from interface [property].CitedReference
- Aggregation link to interface [property].AbstractValue

8.5.5.2 CitedValue Attributes

Attribute	Details
public static String **property**	
public static CitedReference **reference**	

8.5.5.3 CitedValue Methods

Operation	Details
public **getproperty**():String	*<<property get>>* *Tags:* attribute_name=property
public **getrefernce**():CitedReference	*<<property get>>* *Tags:* attribute_name=refernce
public **setproperty**(String newVal):void	*<<property set>>* *Tags:* attribute_name=property
public **setrefernce**(CitedReference newVal):void	*<<property set>>* *Tags:* attribute_name=refernce

8.5.6 MatrixValue

8.5.6.1 Connections

- Aggregation link to interface [property].AbstractValue

8.5.7 MeasuringUnit

8.5.7.1 Connections

- Aggregation link to interface [property].AbstractValue

8.5.7.2 MeasuringUnit Attributes

Attribute	Details
public java.lang.String **MEASURING_UNIT__METRE**	*Initial: METRE*
public java.lang.String **MEASURING_UNIT_LITRE**	*Initial: LITRE*
public java.lang.String **MEASUREING_UNIT_GRAM**	*Initial: GRAM*
public java.lang.String **MEASURING_UNIT_SECOND**	*Initial: SECOND*
public java.lang.String **MEASURING_UNIT_AMPERE**	*Initial: AMPERE*
public java.lang.String **MEASURING_UNIT_KELVIN**	*Initial: KELVIN*
public java.lang.String **MEASURING_UNIT_MOLE**	*Initial: MOLE*
public java.lang.String **MEASURING_UNIT_CANDELLA**	*Initial: CANDELLA*
public java.lang.String **MEASURING_UNIT_RADIAN**	*Initial: RADIAN*
public java.lang.String **MEASURING_UNIT_STERADIAN**	*Initial: STERADIAN*
public java.lang.String **MEASURING_UNIT_HERTZ**	*Initial: HERTZ*
public java.lang.String **MEASURING_UNIT_NEWTON**	*Initial: NEWTON*
public java.lang.String **MEASURING_UNIT_PASCAL**	*Initial: PASCAL*
public java.lang.String **MEASURING_UNIT_JOULE**	*Initial: JOULE*
public java.lang.String **MEASURING_UNIT_WATT**	*Initial: WATT*
public java.lang.String **MEASURING_UNIT_COULOMB**	*Initial: COULOMB*
public java.lang.String **MEASURING_UNIT_VOLT**	*Initial: VOLT*
public java.lang.String **MEASURING_UNIT_FARAD**	*Initial: FARAD*
public java.lang.String **MEASURING_UNIT_OHM**	*Initial: OHM*

public java.lang.String **MEASURING_UNIT_SIEMENS**	*Initial: SIEMENS*
public java.lang.String **MEASURING_UNIT_WEBER**	*Initial: WEBER*
public java.lang.String **MEASURING_UNIT_TESLA**	*Initial: TESLA*
public java.lang.String **MEASURING_UNIT_HENRY**	*Initial: HENRY*
public java.lang.String **MEASURING_UNIT_CELCIUS_DEGREE**	*Initial: CELCIUS_DEGREE*
public java.lang.String **MEASURING_UNIT_LUMEN**	*Initial: LUMEN*
public java.lang.String **MEASURING_UNIT_LUX**	*Initial: LUX*
public java.lang.String **MEASURING_UNIT_BECQUEREL**	*Initial: BECQUEREL*
public java.lang.String **description**	
public java.lang.String **name**	
public java.lang.String **MEASURING_UNIT_ZIEVERT**	*Initial: ZIEVERT*
public java.lang.String **MEASURING_UNIT_GRAY**	*Initial: GRAY*
public java.lang.String **reference**	

8.5.8 ScalarValue

8.5.8.1 Connections

- Aggregation link to interface [property].AbstractValue

8.5.9 TensorValue

8.5.9.1 Connections

- Aggregation link to interface [property].AbstractValue

8.5.10 VectorValue

8.5.10.1 Connections

- Aggregation link to interface [property].AbstractValue

8.6 Search General Functionality

The interfaces shown above allow you to create a query using the CML representation of the item being searched on by selecting properties and creating a search criterion.

[Chemical] compounds have a number of [Chemical] properties. Some of these [Chemical] properties are searchable (that is, the [Chemical] property and is corresponding value has been measured and they are kept in proprietary databases) and some are not. Each `SearchableProperty` consists of `SelectedPropertyGroup` and a `SearchCriterion`. Each `SearchCriterion` is formed by one or more `SearchCriteriaGroups` associated via `Logical`operators, an `AccuracyQualifier`, and a `ComparisonOpe`rator. [Chemical] properties have `MeasuringUnits` (such as AMPERE) and `MeasuringUnitPrefix`es (such as PICO) and could be `ScalarValues`, `BLOB` values, `VectorValues`, `MatrixValues`, and `TensorValues`. Moreover, sometimes the [Chemical] properties have also a `CitedValue` and a `CitedReference` (as in 34.56 PICOFARADs Journal Of Chemistry, Vol. 9, pp 37-49, March 7, 1999) that will need to also be searched.

The central class of this module is the `ChemSearchEngine` which drives the entire functionality of this module. This interface provides methods to add or remove criteria groups, and add or remove property groups which are components of a given search string. In addition this interface provides functionality to perform the search, to insert new search criteria and to close the criteria search. A method is also provided for future use; that is, the `split()` method will allow the split of a request into two main search strings, one for the intrinsic properties and one for the extrinsic properties. The code segments are shown below.

```
public interface ChemSearchEngine
{
   public SelectPropertyGroup theSelectPropertyGroup[];
   public SearchCriteriaGroup theSearchCriteriaGroupSCCSE;

   /**
    * @param group
    * @return boolean
    * @throws Chem::InvalidCriterion
    * @throws Chem::InvalidCriteriaCombination
    */
   public boolean addSearchCriteriaGroup(in Chem::SearchCriteriaGroup group) throws
Chem::InvalidCriterion, Chem::InvalidCriteriaCombination;

   /**
    * @param group
    * @return boolean
    * @throws Chem::InvalidCriterion
    */
   public boolean removeSearchCriteriaGroup(in Chem::SearchCriteriaGroup group) throws
Chem::InvalidCriterion;
```

```
    /**
     */
    public void search();

    /**
     * @param arg0
     * @return boolean
     */
    public boolean insert(MoleculeImpl arg0);

    /**
     * @return boolean
     */
    public boolean split();


    public void close();

    /**
     * @param arg0
     * @return boolean
     */
    public boolean addSelectPropertyGroup(SelectPropertyGroup arg0);

    /**
     * @param arg0
     * @return boolean
     */
    public boolean removeSelectPropertyGroup(SelectPropertyGroup arg0);
}
```

8.7 Legacy Module

The Legacy module contains classes to determine the source and destination file formats (inherited from the CML Jumbo classes), a file map that stores the information loss between the different file formats, and a class to calculate that information loss. Figure 8.33 shows the legacy module classes and interfaces. This module holds interfaces to interact with legacy databases.

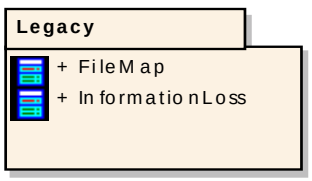


Figure 8.33 - Legacy Module

8.7.1 FileMap

This class contains all possible chemical information files and their corresponding mappings.

8.7.1.1 FileMap Attributes

Attribute	Details
public List **MDL_ICHI_IL**	
public List **MDL_SMILES-IL**	
public List **MDL_SDF_IL**	
public List **MDL_MOPACINC-IL**	
public List **MDL_CIF-IL**	
public List **SMILES_ICHI-IL**	
public List **SMILES_SDF-IL**	
public List **SMILES_MOPACINC-IL**	
public List **SMILES_CIF-IL**	
public List **ICHI_SDF-IL**	
public List **ICHI_MOPACINC-IL**	
public List **ICHI_CIF-IL**	
public List **SDF_MOPACINC-IL**	
public List **SDF_CIF-IL**	
public List **MOPACINC_CIF-IL**	

8.7.1.2 FileMap Methods

Operation	Details
public **getICHI_CIF-IL**():List	<<*property get*>> *Tags:* attribute_name=ICHI_CIF-IL
public **getICHI_MOPACINC-IL**():List	<<*property get*>> *Tags:* attribute_name=ICHI_MOPACINC-IL
public **getICHI_SDF-IL**():List	<<*property get*>> *Tags:* attribute_name=ICHI_SDF-IL
public **getMDL_CIF-IL**():List	<<*property get*>> *Tags:* attribute_name=MDL_CIF-IL
public **getMDL_ICHI_IL**():List	<<*property get*>> *Tags:* attribute_name=MDL_ICHI_IL
public **getMDL_MOPACINC-IL**():List	<<*property get*>> *Tags:* attribute_name=MDL_MOPACINC-IL
public **getMDL_SDF_IL**():List	<<*property get*>> *Tags:* attribute_name=MDL_SDF_IL
public **getMDL_SMILES-IL**():List	<<*property get*>> *Tags:* attribute_name=MDL_SMILES-IL
public **getMOPACINC_CIF-IL**():List	<<*property get*>> *Tags:* attribute_name=MOPACINC_CIF-IL
public **getSDF_CIF-IL**():List	<<*property get*>> *Tags:* attribute_name=SDF_CIF-IL
public **getSDF_MOPACINC-IL**():List	<<*property get*>> *Tags:* attribute_name=SDF_MOPACINC-IL
public **getSMILES_CIF-IL**():List	<<*property get*>> *Tags:* attribute_name=SMILES_CIF-IL
public **getSMILES_ICHI-IL**():List	<<*property get*>> *Tags:* attribute_name=SMILES_ICHI-IL
public **getSMILES_MOPACINC-IL**():List	<<*property get*>> *Tags:* attribute_name=SMILES_MOPACINC-IL
public **getSMILES_SDF-IL**():List	<<*property get*>> *Tags:* attribute_name=SMILES_SDF-IL
public **setICHI_CIF-IL**(List newVal):void	<<*property set*>> *Tags:* attribute_name=ICHI_CIF-IL
public **setICHI_MOPACINC-IL**(List newVal):void	<<*property set*>> *Tags:* attribute_name=ICHI_MOPACINC-IL

public **setICHI_SDF-IL**(List newVal):void	<<*property set*>> *Tags:* attribute_name=ICHI_SDF-IL
public **setMDL_CIF-IL**(List newVal):void	<<*property set*>> *Tags:* attribute_name=MDL_CIF-IL
public **setMDL_ICHI_IL**(List newVal):void	<<*property set*>> *Tags:* attribute_name=MDL_ICHI_IL
public **setMDL_MOPACINC-IL**(List newVal):void	<<*property set*>> *Tags:* attribute_name=MDL_MOPACINC-IL
public **setMDL_SDF_IL**(List newVal):void	<<*property set*>> *Tags:* attribute_name=MDL_SDF_IL
public **setMDL_SMILES-IL**(List newVal):void	<<*property set*>> *Tags:* attribute_name=MDL_SMILES-IL
public **setMOPACINC_CIF-IL**(List newVal):void	<<*property set*>> *Tags:* attribute_name=MOPACINC_CIF-IL
public **setSDF_CIF-IL**(List newVal):void	<<*property set*>> *Tags:* attribute_name=SDF_CIF-IL
public **setSDF_MOPACINC-IL**(List newVal):void	<<*property set*>> *Tags:* attribute_name=SDF_MOPACINC-IL
public **setSMILES_CIF-IL**(List newVal):void	<<*property set*>> *Tags:* attribute_name=SMILES_CIF-IL
public **setSMILES_ICHI-IL**(List newVal):void	<<*property set*>> *Tags:* attribute_name=SMILES_ICHI-IL
public **setSMILES_MOPACINC-IL**(List newVal):void	<<*property set*>> *Tags:* attribute_name=SMILES_MOPACINC-IL
public **setSMILES_SDF-IL**(List newVal):void	<<*property set*>> *Tags:* attribute_name=SMILES_SDF-IL
public **getFileMap**():FileMap	*Sequential*
public **setFileMap**():FileMap	*Sequential*

public **addFileMap**(FileMap Map):boolean	*Sequential*
public **deleteMap**(FileMap Map):boolean	*Sequential*

8.7.2 InformationLoss

This class contains methods to estimate the information loss when converting between different formats. Please refer to previous sections for detailed descriptions of the conversion process and results.

8.7.2.1 InformationLoss Attributes

Attribute	Details
public FileMap **inputFileType**	*Notes:* Enter the file type for the input
public FileMap **outputFileType**	*Notes:* Gets the output file format
public BLOB **LossofInformation**	*Notes:* Computes information loss

8.7.2.2 InformationLoss Methods

Operation	Details
public **getinputFileType**():FileMap	*Sequential* <<*property get*>> *Tags:* attribute_name=inputFileType *Notes:* Returns the input file type
public **getLossofInformation**():BLOB	*Sequential* <<*property get*>> *Tags:* attribute_name=LossofInformation *Notes:* Computes and reports the lkoss of information
public **getoutputFileType**():FileMap	*Sequential* <<*property get*>> *Tags:* attribute_name=outputFileType *Notes:* Gets output file type
public **setinputFileType**(FileMap newVal):void	*Sequential* <<*property set*>> *Tags:* attribute_name=inputFileType *Notes:* Set input file type
public **setLossofInformation**(BLOB newVal):void	*Sequential* <<*property set*>> *Tags:* attribute_name=LossofInformation *Notes:* Set loss of information
public **setoutputFileType**(FileMap newVal):void	*Sequential* <<*property set*>> *Tags:* attribute_name=outputFileType *Notes:* Set output file type
public **getSourceFileType**():FileMap	*Sequential* *Notes:* return CSAR.Legacy.FileMap
public **getDestinationFileType**():FileMap	*Sequential* *Notes:* return CSAR.Legacy.FileMap

public **removeAll**():boolean	*Sequential* *Notes:* Removes from the target Collection all its elements that are also contained in the specified Collection.
public **size**(Collection collection):int	*Sequential* *Notes:* Returns the size of the given Collection
public **toArray**(	*Sequential*

8.7.3 Collection Module

The collection module, see Figure 40, extends the Java API public class Collections which extends Object. This class consists exclusively of static methods that operate on or return collections of Molecules. It contains polymorphic algorithms that operate on collections, "wrappers," which return a new collection backed by a specified collection, and a few other odds and ends. This module could be replaced when and if the LSR Collection standard is accepted.

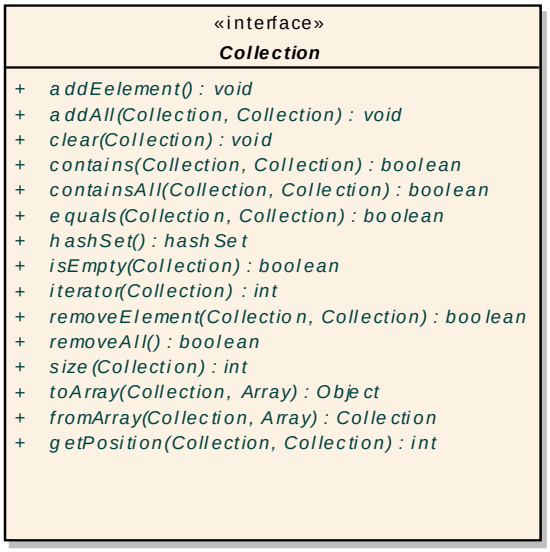


Figure 8.34 - Collection Module

The functionality is described in the table below.

Operation	Details
public **addEelement**():void	*Sequential* *Notes:* Add given element to the appropriate collection
public **addAll**(Collection collection, Collection gSet):void	*Sequential* *Notes:* Adds all the elements in the specified Collection to the target Collection.
public **clear**(Collection collection):void	*Sequential* *Notes:* Removes all elements from the Collection.
public **contains**(Collection collection, Collection element):boolean	*Sequential* *Notes:* Returns true if the Collection contains the element
public **containsAll**(Collection collection2, Collection collection1):boolean	*Sequential* *Notes:* Returns true if the target Collection contains all of the elements in the specified Collection.
public **equals**(Collection collection2, Collection collection1):boolean	*Sequential* *Notes:* Returns True if the two collections are equal.
public **hashSet**():hashSet	*Sequential* *Notes:* Set interface, backed by a hash table (actually a HashMap instance). It makes no guarantees as to the iteration order of the set; in particular, it does not guarantee that the order will remain constant over time. This class permits the null element.
public **isEmpty**(Collection colelction):boolean	*Sequential* *Notes:* Returns True if the collection is empty
public **iterator**(Collection collection):int	*Sequential* *Notes:* Iterates over the collection
public **removeElement**(Collection collection, Collection element):boolean	*Sequential* *Notes:* Removes a given element from the Collection
public **removeAll**():boolean	*Sequential* *Notes:* Removes from the target Collection all its elements that are also contained in the specified Collection.

public **size**(Collection collection):int	*Sequential* *Notes:* Returns the size of the given Collection
public **toArray**(Collection collection, Array array):Object	*Sequential* *Notes:* The toArray methods are provided as a bridge between collections and older APIs that expect arrays on input.
public **fromArray**(Collection collection, Array array):Collection	*Sequential* *Notes:* The fromArray methods are provided as a bridge between collections and older APIs that provide arrays as input.
public **getPosition**(Collection collection, Collection element):int	*Sequential* *Notes:* Get the position of a given element in a Collection

9 Glossary

Aromaticity

This is a quality possessed by many, many common compounds, from benzene to phenylalanine in which multiple double bonds, ‘conjugate,’ sharing electrons. This sharing produces a structure of lower energy than one in which the double bonds are isolated. (The above is a very simple explanation of a common phenomenon. For more information, see any basic text on organic chemistry.)

Different software systems define aromaticity differently. The biggest differences are whether aromaticity is specified in atom types or in bonds or perceived from the arrangement of bonds. Daylight’s SMILES notation generally specifies aromatic atoms using lower-case letters. For example, benzene (an aromatic ring of 6 carbon atoms) is defined as 'c1ccccc1.' Because the atoms are denoted with lower-case letters, they are distinguished from cyclohexane (a non-aromatic or 'aliphatic' ring of carbon atoms (C1CCCCC1)).

MDL’s molfile, on the other hand, defines aromatics from an arrangement of alternating single and double bonds within a ring of appropriate size.

Still other systems explicitly define aromaticity using explicitly designated aromatic bonds.

To make matters even more complicated, MDL supports an aromatic bond type that can be used for bonds within a molfile that can be used to query a database but not registered.

Atom

The smallest particle of an [chemical] element that can exist either alone or in combination, retaining any properties of the [chemical] element. We extend this definition to include points in space (that can be used to define the position of other points); 'superatoms' or atoms that represent a collection of other atoms.

Bit string

A contiguous set of characters consisting entirely of 1s and 0s. A bit string can be used to encode a good deal of information in a compact way.

Bond

A chemical link between two atoms. Bonds are classified as ionic (transfer of electrons from one atom to another); covalent (sharing of electrons, generally an equal number from each atom); dative (sharing of two electrons from a single atom); or hydrogen (attraction of electron-starved hydrogen atom to electron rich heavy atom.)

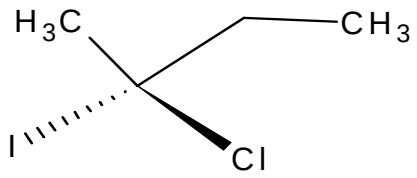
Charge

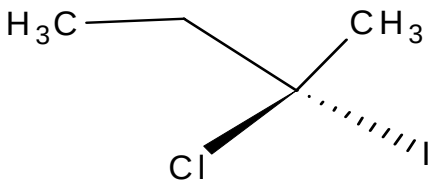
A deficiency or excess of electrons on a particular object, giving rise to a positive or negative charge, respectively. (www.allwords.com) Molecules can carry charges, which are often attributed to specific atoms.

Chiral

Adjective applied to a molecule that cannot be superimposed on its mirror image. (‘Chirality’ is the corresponding noun.)

An example of a chiral structure for 2-chloro-2-iodo-butane is shown below:





The compound has a mirror image that cannot be superimposed and is therefore termed its enantiomer.

Connection table

A means of representing the atoms contained within a molecule and the bonds that hold them together.

Counterion

A set of one or more bonded atoms, with opposite charge and generally smaller size, that accompanies another charged set of bonded atoms.

Cyclic, acyclic bonds

When chemical bonds occur within a ring, they are termed ‘cyclic.’ ‘Acyclic bonds’ by contrast, occur in open chain structures.

Electrons (π or pi)

Molecules containing double or triple bonds typically have electrons that project outside the line between the atoms in the bond. When more than one double or triple bond are in close proximity, the electrons in the pi bonds interact and spread out over all the atoms involved.

Fingerprints

Fingerprints are bit strings that are based on features of a chemical structure. In this regard, they are similar to **Structural keys** (q.v.). Fingerprints are different from keys in that the bits they contain are typically ‘folded over’ or combined (using a logical OR) with one another to reduce the size of the string.

Heteroatoms

Atoms that are neither carbon nor hydrogen are considered ‘heteroatoms’ and are often handled differently by software systems.

Markush structure

It is common to represent chemical structures as a common core containing marked substitution sites, plus a set of possible structures for each substitution point. These Markush structures can be used in several ways:

- To represent a set of compounds analyzed in order to determine the effect of varying substituents on compound activity (SAR, short of 'Structure Activity Relationships).
- To represent a set of compound produced using combinatorial techniques (synthesized by serially attaching different chemical groups to a common core).
- To produce a fine-tuned substructure query.

In this document, we use one Assembly to represent the core, (atoms of type 'R' designate the substitution points), plus one additional Assembly for each R-Group.

Molecule

The smallest particle into which an [chemical] element or a compound can be divided without changing its chemical and physical properties; a group of like or different atoms held together by chemical forces (www.allwords.com); generally, composed of atoms held together by bonds.

A molecule can represent:

1. An entire chemical entity.
2. One portion of a complex chemical entity (such as a mixture, set of tautomers or conformers).
3. A collection of other molecules as defined in 2) above.

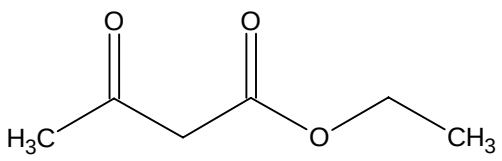
Orbital

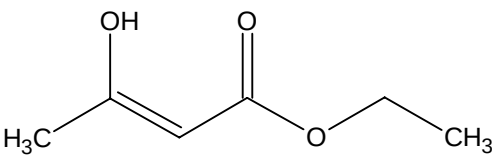
A subdivision of a nuclear shell containing zero, one, or two electrons (m-w.com).

Query structure

Most chemical software systems require structures to meet certain requirements in order to be entered into a database or used for calculation. A structure that meets these criteria is classified as ‘registerable.’ (Generally, atom types must correspond to entities in the periodic table and bond types must be well defined.)

Additionally, the software will allow a chemist to draw structures that do not meet the criteria for registration but can be used to retrieve molecules out of a database using substructure searching (see below). An example of a structure that is valid for query but not registration is one with just one bond designated as aromatic.





Radical

An atom or a group of atoms with at least one unpaired electron (www.allwords.com).

- singlet – A radical with two unpaired electrons whose spins are opposite.
- doublet – A radical with a single unpaired electron.
- triplet – A radical with two unpaired electrons whose spins are aligned.

Registerable structure

A chemical structure that meets the criteria for inclusion in a repository. (Compare with '**Query structure**' above).

Similarity Search

A chemical data query in which a user seeks compounds that *resemble* a given structure without necessarily having a **substructure** (q.v.) match. The resemblance is often intuitive to a chemist but has a mathematical basis in terms of common features or properties. Similarity searches typically include a cutoff value *X* so the user sees only structures having *X*% or more similarity to the query structure.

The mathematics of similarity searching require:

1. A means of evaluating individual chemical structures. Generally, this involves computing **keys** (q.v.) or **fingerprints** (q.v.).
2. A metric for comparing keys or fingerprints from two structures. The most common of these is the **Tanimoto coefficient** (q.v.).

At search time, a user-supplied query structure is compared with every other structure in the database and those with a similarity metric greater than the cutoff are considered 'hits.' (Bit operations are typically fast enough to make large number of comparisons practicable.)

Spin state

A way of characterizing the angular momentum of electrons. Individual electrons may spin 'up' or 'down.' Two or more electrons may have their spins parallel or antiparallel.

Stereochemistry

Studying the effect of configuration of atoms around asymmetric atoms and bonds.

Structural keys

When compounds are registered into most chemical search software systems, the structures are scanned for the presence of predefined features, such as heteroatoms (q.v.) or 6-membered rings. Each key sets a bit within a string that may be hundreds of characters long. These keys are used for substructure and similarity searching.

Substructure

One chemical structure is said to be a **substructure** of another if the first structure can be located within the second. (The second is said to be the **superstructure** of the first.) All structures are substructures of themselves.

A **substructure search** scans a database for all substructural matches.

Tanimoto coefficient

Mathematical formula for evaluating the similarity of two structures

$$S_{A,B} = c/[a + b - c]$$

Where

$S_{A,B}$ = 'similarity of structures A and B'

c = number of features in common between the given property in the two structures. (In the case of structure keys or fingerprints, this means the number of ON bits when the two bit strings are ANDed.)

a = number of features ON in structure A

b = number of features ON in structure B

[From *J. Chem. Inf. Comput. Sci*. **1998,** *38*, 983-996]

Tautomer

'One of two or more structural isomers that exist in equilibrium and are readily converted from one isomeric form to another.' From http://www.bartleby.com/65/ta/tautomer.html. An illustration of the tautomers of ethyl acetoacetate is shown:

Valence

The number of bonds an atom has to other atoms.

Annex A: UML Use Cases

(normative)

Actor Catalogue

Actor	Definition
Client	This is a logical client that represents any given system that will interface with the provided API.

Master Use Case

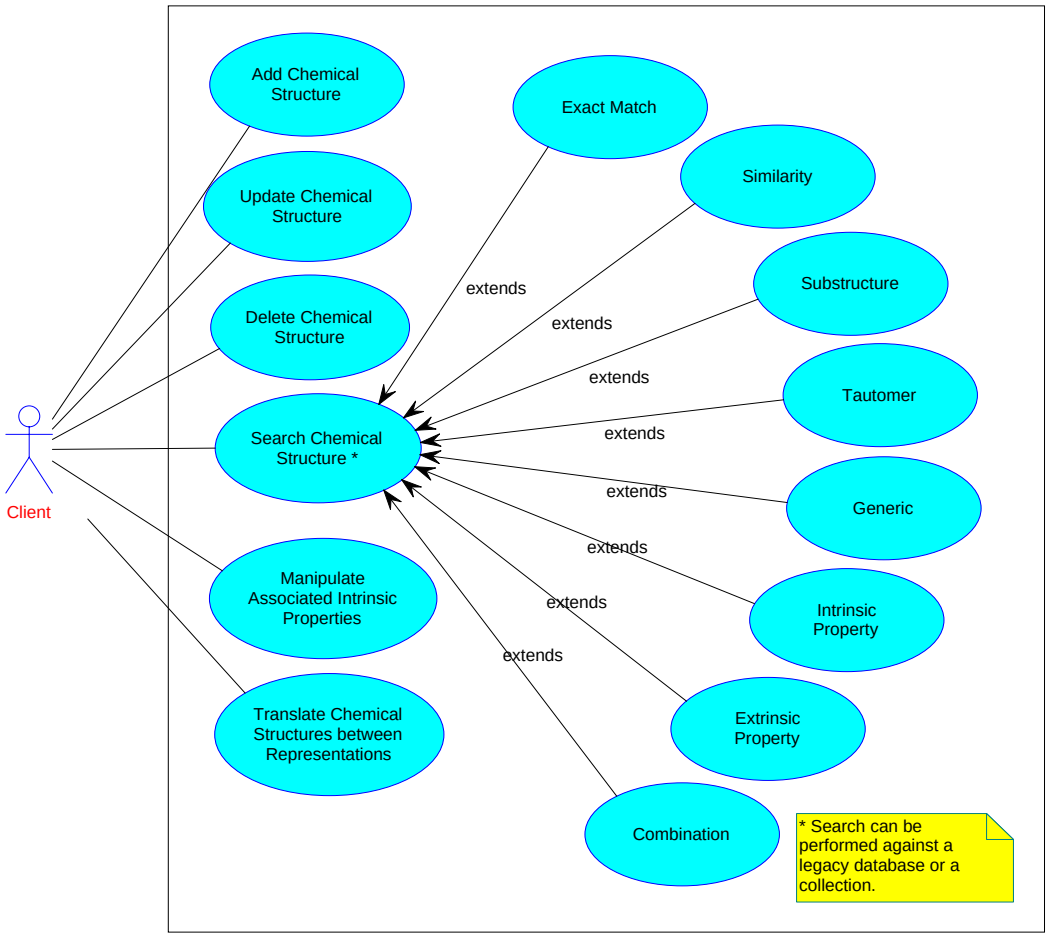


Figure A.1 - General Use Case Scenario

Detailed Use Case Scenarios

Add Chemical Structure

Desired Outcome:	A chemical structure is inserted into database.
Entered When:	Actor invokes the insert function.
Finished When:	A chemical structure is inserted into database and a confirmation is returned to the actor.
Description:	The purpose of this function is to insert a chemical structure into database. A chemical structure can be specified using CML.

Data Elements

Name	Description
Chemical structure	A chemical structure to be inserted into database.

Add Chemical Structure – Add Unique Chemical Structure

Desired Outcome:	A chemical structure is inserted into database only if it will not be a duplicate.
Entered When:	Actor invokes the insert_if_not_duplicate function.
Finished When:	A chemical structure is inserted into database and a confirmation is returned to the actor. A collection of duplicated chemical structures is returned if any duplicate is found.
Description:	The purpose of this function is to insert a unique chemical structure into database. An exact match search will be performed to ensure a uniqueness of the input chemical structure. A chemical structure can be specified using CML.

Data Elements

Name	Description
Chemical structure	A chemical structure to be inserted into database.

Update Chemical Structure

Desired Outcome:	An existing chemical structure data in database is updated.
Entered When:	Actor invokes the update function.
Finished When:	An existing chemical structure data in database is updated and a confirmation is returned to the actor.
Description:	The purpose of this function is to update an existing chemical structure data in database. A chemical structure can be specified using CML.

Data Elements

Name	Description
Chemical structure	A chemical structure to replace an existing data.

Delete Chemical Structure

Desired Outcome:	An existing chemical structure in database is removed.
Entered When:	Actor invokes the delete function.
Finished When:	An existing chemical structure in database is removed and a confirmation is returned to the actor.
Description:	The purpose of this function is to delete an existing chemical structure in database. A chemical structure to be deleted can be specified using CML/ID.

Data Elements

Name	Description
Chemical structure/ID	Identifier of chemical structure to be deleted.

Search Chemical Structure

Desired Outcome:	Collection of structures meeting the search criteria is returned.
Entered When:	Actor invokes the search function.
Finished When:	A collection of structures is returned. An empty collection is returned if no structure meets the specified criteria.
Description:	The purpose of this function is to identify a collection of structures meeting the specified criteria. Search criteria can be specified by a query string and/or structure (CML). A returned collection is in CML, which can be transformed into another format such as SMILES, MOL, SLN.

Data Elements

Name	Description
Query string	Specifies the search criteria.

Comments

The following are sample query strings used in the MDL ISIS/Host:

molstructure = [{mjk38smasd903kqlads90rmlw9masksoaskdoq}]

molstructure tautomer [{mjk38smasd903kqlads90rmlw9masksoaskdoq}]

pdnum = ‘0123456-0000’

pdnum like ‘0123456-%’

mol.weight < 500

The following are sample query strings used in the Daylight DayCard (Daylight Chemistry Cartridge for Oracle).

http://www.daylight.com/meetings/mug2000/Delany/cartridge.html

select * from medium where exact(smiles, 'O=c1ccocc1') = 1

select count(smiles) from large where contains(smiles, '>>O=c1c(C)cocc1') = 1;

Exact Match

molstructure = [{mjk38smasd903kqlads90rmlw9masksoaskdoq}]

select * from medium where exact(smiles, 'O=c1ccocc1') = 1

Similarity

molstructure = [{mjk38smasd903kqlads90rmlw9masksoaskdoq}] and factor = 0.7

Substructure

molstructure sss [{mjk38smasd903kqlads90rmlw9masksoaskdoq}]

select * from medium where contains(smiles, 'O=c1ccocc1') = 1

Tautomer

molstructure tautomer [{mjk38smasd903kqlads90rmlw9masksoaskdoq}]

select * from medium where tautomer(smiles, 'O=c1ccocc1') = 1

Comments:

(from Tripos' RFI) There are potentially different methods of working with tautomers, either recognizing the potential for rearrangement at registration into the chemical database with flags that highlight the areas where tautomerism can take place. An alternative method would be to register the tautomers as different compounds and then flag their tautomeric relatives. The latter is what PD does.

Generic

Comments:

i.e., Markush, R-group, etc.

Intrinsic Properties

mol.weight < 500

Combination

molstructure sss [{mjk38smasd903kqlads90rmlw9masksoaskdoq}] and mol.weight < 500

Get Intrinsic Properties

Desired Outcome:	An intrinsic property of a chemical structure is returned.
Entered When:	Actor invokes the access (get) function.
Finished When:	An intrinsic property of a chemical structure is returned to the actor.
Description:	The purpose of this function is to query an intrinsic property of a chemical structure.

Data Elements

Name	Description
Intrinsic property	Name of an intrinsic property to be queried.

Translate Chemical Structure from One to Another

Desired Outcome:	A type of chemical structure representation is transformed into another type.
Entered When:	Actor invokes the translate function.
Finished When:	A type of chemical structure representation is transformed into another type, which is then returned to the actor.
Description:	The purpose of this function is to transform the representation of one chemical structure into another representation.

Data Elements

Name	Description
Structure representation type	Source representation.
Structure representation	Chemical structure to be translated.
Structure representation type	Destination representation.

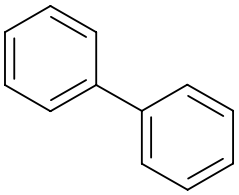
Annex B: Use Cases for Chemistry

(normative)

1. Structure search

A. using ISIS/Draw to sketch a molecule for a substructure search of a Daylight database

I. simple structure (e.g., biphenyl)

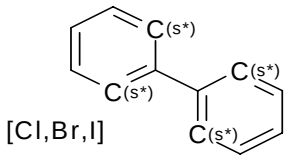


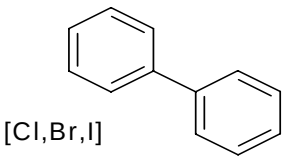
II. more complicated structure (disconnected fragments; atom lists; substitution counts; charges, etc.)

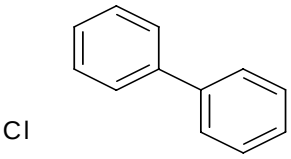
a. two fragments (twofrag.mol)

b. two fragments + atom list (tfraglist.mol)

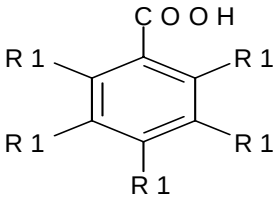
c. two fragments + atom list + substitution counts (tfsub.mol)

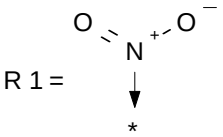




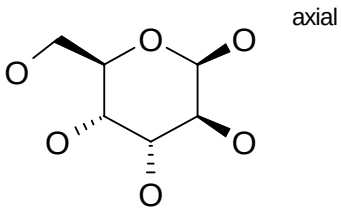


III. R Group Query





IV. S-Group data



B. using the Ertl Java editor to generate a SMILES to search an MDL database

C. browsing the hits

I. Browsing regular structures

II. Browsing structures with polymeric constructs

III. Browsing structures to which user does not have rights

2. Use ChemSymphony to search Web-based Available Chemical Directory (ACD) system for suppliers of a given set of structures.

A. superstructures of aromatic acid chlorides – very simple substructure to find a large class of compounds (aromacchlor.mol)

B. p-nitrobenzoic acid – search for a specific compound (pnitrobenz.mol)

3. Looking for similar compounds

A. ISIS/Draw front-end to Unity (or RS^3) database back end

B. ChemSymphony front-end to MDL database

4. Registration

   A. using ISIS/Draw to generate a molfile for registration into Daylight (convert to SMILES for direct chemical registration, as well as saving the molfile to an Oracle field).

      I. Simple structure; everything in molfile translates to SMILES

      II. Complex structure (charges, valence) but properties do translate

      III. Parts of the structure (brackets, S-Group data) do not translate to SMILES

   B. using ChemSymphony to generate structures for registration to a Unity database

5. Registration correction

   A. database is MDL; need to locate molecule by ID, replace structure. Drawing tool is ChemSymphony

      I. simple structure – no loss of information

      II. more complicated structure – all information can be translated

      III. very complicated structure – some information does not map

Annex C: UML Related Interface Documentation

(normative)

Submitted separately in a zip file. See OMG document: lifesci/05-08-02.

Annex D: Java Code Segments

(normative)

The Java code is included in a separate zip file. See OMG document: lifesci/05-08-03.

Annex E: The XMI

(normative)

The XMI is included in a separate file. See OMG document:lifesci/05-08-04.

INDEX