Macromolecular Structure RFP Response

Revised Submission

National Institute of Standards and Technology

Supported By:

Research Collaboratory for Structural Bioinformatics

San Diego Supercomputer Center

OMG Document Number: lifesci/2000-08-01

Table of Contents

1 Preface

1.1 Submission Contact Point

Douglas S. Greer
San Diego Supercomputer Center
University of California, San Diego
9500 Gilman Drive
La Jolla, CA 92093-0527
email: dsg@sdsc.edu

1.2 Acknowledgments

The author of this document wishes to express his appreciation to those listed below (in alphabetic order) for their contributions of ideas and experience. Ultimately, the ideas expressed in this document are those of the author and do not necessarily reflect the views or ideas of these individuals, nor does the inclusion of their names imply an endorsement of the final product.

John Badger	john_badger@stromix.com
Ira Baron	baron@mpi.com
David Benton	w_david_benton@sbphrd.com
Philip Bourne	bourne@sdsc.edu
Carol Burt	cburt@2ab.com
Steve Chervitz	sac@neomorphic.com
Robert DeWitte	rob@acdlabs.com
David Frankel	dfrankel@gendev.com
Gary Gilliland	Gary.gilliland@nist.gov
Peter Keller	keller@ebi.ac.uk
Karl Konnerth	konnerth@incyte.com
Scott Markel	smarkel@netgenics.com
Jeff Mischkinsky	jeff@persistence.com
Martin Senger	senger@ebi.ac.uk
Lynn TenEyck	teneyckl@sdsc.edu
Helge Weissig	helgew@sdsc.edu
John Westbrook	jwest@ndbdev.rutgers.edu

This work developing this specification was supported in part by the Research Collaboratory for Structural Bioinformatics, the Protein Data Bank and the San Diego Supercomputer Center.

2 Proof of Concept

Java code has been designed, written and tested at the San Diego Supercomputer Center which implements the IDL set forth in this proposal. The core ideas presented in this submission are based on the experience of SDSC, the Research Collaboratory for Structural Bioinformatics and other members of the OMG Life Sciences Research Task Force.

3 Response to RFP Requirements

The following are requirements listed in the Macromolecular Structure RFP (OMG lifesci/99-08-15).

3.1 Mandatory Requirements

- ***The proposed interfaces must provide retrieval functionality based on the macromolecular entry identifier.***

  This is provided by the Entry interface.

- ***The interfaces must provide uniform access to macromolecular structure data independent of underlying implementation or storage format.***

  The submission does this.

- ***Proposals shall insure that the interfaces and types specified and the values of fields or structure members, have precise meanings that have been generally agreed upon within the scientific research community.***

  Wherever possible, the precise meaning of all fields is in agreement with the terms used in the worldwide scientific community. This includes terminology adopted by the International Union of Crystallography, which has developed standardized definitions for the specification of macromolecular structure.

- ***The proposal shall clearly define the interfaces and types specified and the values of fields or structure members in terms of the scientific concepts they represent.***

  The submission does this.

- ***At a minimum, the interface shall provide information about: Atomic positions primary and secondary structure, interatomic bonds, date of deposition and author, available information about temperature factors.***

  All of these are provided.

3.2 Optional Requirements

- ***Where practical the proposed interface may supply additional information such as: raw experimental data, data collection and processing parameters, structure refinement, parameters, pointers to other relevant database entries, pointers to external references, crystallographic cell and symmetry information, NMR-specific parameters, depositor information, related publications, indicators of data structure and quality, structure classification information.***

  The proposed interface provides information about intrinsic chemical structure and external literature and database references. Information about the experiment itself such as NMR or X-ray Crystallography data is not specified in this proposal but may be obtained through the metamodel interface provided. This information may also be specified through a future RFP.

- ***The proposed interface may include a mechanism for retrieving a list of entry identifiers that includes a field indicating the date each entry was last modified.***

  This is provided.

- ***Interfaces may include a mechanism for identifying legacy format coding systems and for downloading serialized macromolecular entries, for example as a single octet sequence in the legacy format.***

  This is provided.

- ***Where practical, the proposed solution may partition the intrinsic macromolecular structure information and the supplemental information about laboratory parameters associated with the experimental data collection into separate modules.***

  The core DsLSRMacromolecularStructure module defines an interface for basic administrative and utility functions and for obtaining information about intrinsic macromolecular structure. Interfaces for other information has been partitioned into separate modules.

- ***Submissions may provide interfaces for the deposition of macromolecular structure. The interface for this capability, if present, shall be clearly partitioned in a separate module.***

  A deposition interface is not specified in this proposal but may be obtained through the metamodel interface provided. This interface may also be specified through a future RFP.

3.3 RFP Evaluation Criteria

Several of the evaluation criteria listed in the RFP influenced the overall design.

- ***Scalability in terms of being able to support both a growing number of users and a growing number of data sets.***

- ***Efficiency and the ability to support large, high performance applications.***

- ***Granularity in the ability to access only the data of interest.***

  Several elements of the IDL design were incorporated, to help meet these criteria. Some of the relevant aspects of these are discussed in section 4.2.7 "Presence Flags" on page 19, section 4.2.4 "Granularity" on page 17, section 4.2.6 "Indices vs. Object Embedding" on page 19 and section 4.2.8 "Distributed State" on page 20.

- ***Ability of the interface to support a client-side flyweight object design pattern for representing such concepts as atoms, bonds and monomers.***

  Elements of the design that support this feature are discussed in section 4.2.6 "Indices vs. Object Embedding" on page 19.

3.4 Issues to be Discussed

- ***The proposal shall discuss design decisions relating to the overall performance and efficiency of the interface.***

  Many elements of this specification were designed to allow the implementation of an efficient high-performance client-server interface. Performance issues must however be analyzed in the context of expected applications. The three general categories of applications that were under consideration for this design are discussed in Section 4.1, "Use Cases".

  Two major issues that can be expected to have a significant impact or performance are discussed in Section 4.2.7, "Presence Flags" and in Section 4.2.6, "Indices vs. Object Embedding". The issues discussed in Section 4.2.4, "Granularity" will have a large impact in applications where clients need only a small about of information about many different structures. Over the long term, the elements in Section 4.2.1, "Format Independence" will affect the implementors ability to modify and improve server performance.

- ***A appropriate, the proposal shall discuss any interface design decisions that would significantly affect a distributed system implementation.***

Most of the design issues affecting performance that are listed above are relevant because of distributed nature of the predicted use cases. In addition to those issues listed, the rational for the access method design, which will have a major impact on implementation, is discussed in Section 4.2.8, "Distributed State".

4 Overall Design Rational

This chapter reviews the rational that underlies many of the architectural design decisions made in this proposal.

4.1 Use Cases

In the early stages of the design, a broad spectrum of distributed macromolecular structure applications were analyzed. It was found that the use cases more or less fell into the groups discussed below. An overall goal of the design was to try to make the interface as general as possible but still optimized for these common usage patterns.

Three Dimensional Interactive Graphics

Three dimensional graphics applications may in general be characterized as having many clients applications making relatively infrequent requests for structural information. Although requests are infrequent, to maintain an interactive user interface, response time should be kept to a minimum. Also, because of the light client load, it may be expected that a single server could be providing structural information to many, perhaps thousands of user applications.

Scientific Computation

As Mms servers become available, increased use by mathematical software in discovery and analysis applications can be expected. These applications will likely use large multiprocessor systems and can be characterized as requiring optimized server performance in terms of low latency and high throughput.

Multiple Tier Search Engines

An important class of applications involves multiple tier designs where a middle tier is providing query or other similar services. This specification could be used to communicate between this search engine and a Mms server back end. Http or another CORBA protocol may be used between the middle tier and a front end client. These use cases may in general be characterized by the need for fine granularity of access and functionality provided by the presence flags discussed below.

4.2 Architectural Issues

Several general principles discussed below are central to the IDL design.

4.2.1 Format Independence

A primary goal in the design was to make it possible to implement an Mms server using any type of storage format, or storage mechanism e.g. flat files, a relational or other type of database, or serialized objects. However, without strong scientific definitions of the terms used, there is nothing concrete to tie these different types of implementations together and to insure correct results in applications.

To provide the scientific definitions needed, a dictionary of terms developed by the International Union of Crystallography (IUCr) was used to help define the structures and fields in the IDL. This collection of definitions has been extensively debated and agreed upon in the scientific community. Any duplicate effort to redefined these terms would be detrimental to the clear and unambiguous terminology required for scientific research. In order to achieve the goal of creating a pure Mms CORBA definition, every effort was made to extract the scientific definitions while removing any dependencies on a particular file format.

In discussing the scientific definitions set forth by the IUCr, it is important to distinguish between the central core IUCr dictionary and numerous dictionary extensions used by various groups and individuals. In practice, these extensions provide an analogous functionality to subclasses in object-oriented design. The central core dictionary has been agreed upon within the scientific community, and while there may be future additions, no deletions will be made except for minor corrections. In the proposed IDL, only scientific definitions present in the core dictionary have been used. Consequently, future additional definitions to the core dictionary can be easily accommodated by subclassing the existing value types. Implementations may of course also subclass the core value types to provide functionality for particular extensions. This approach is of course meant to help insure the correct operation of software written to the current specification while allowing for future additions and customizations.

4.2.2 Modules

The two modules in this submission are:

1. DsLSRMacromolecularStructure (Required)
2. DsLSRMmsReference (Optional)

A future RFP may provide experimental data specifications in modules for X-ray crystallography, nuclear magnetic resonance and the results of computational methods for predicted folding.

The decision criteria used for selection of the core module value types was that the module should only contain intrinsic chemical information, i.e. information inherent in the physical model independent of any experimental procedure, measurement technique or resulting publication. The size of the core module is mainly due to the fact that the underlying biochemistry of macromolecules is inherently complex. Leaving out parts of the proposed specification for simplicity, would not simplify the biochemistry, but merely make some parts of its description inaccessible.

Where possible, portions of the interface have been separated into optional modules. Modules for bibliographic reference, X-Ray crystallography and deposition have been defined and implemented although only the optional bibliographic reference module is included in this specification. In each case, these optional modules contain the definition of an Entry<Module_Name> interface object that can be obtained from the core Entry object. Each of these optional Entry<> objects contains its own set of presence flags and its own set of access methods for the data types it defines.

4.2.3 Metamodels

Upon the recommendation of the initial submission review committee, the OMG metamodel specification was examined and found to be of potential value in providing a runtime interface definition for optional modules.

Using the definitions provided in the MOF specification, an implementation may provide a list of optional modules supported along with their meta-object description. This list of optional interfaces is returned as an object of type `BaseIDL::ModuleDefSet` [See MOF99, Comp99].

4.2.4 Granularity

The granularity provided by the IDL specification is provided to enable high performance in the expected use cases. The granularity insures that only value types of interest need to be retrieved and that the data is returned in binary form as appropriate. This significantly reduces the amount of data that needs to be sent when compared to retrieving an entire flat file via ftp or http, a method commonly used in present applications.

4.2.5 Ease of Use

A primary requirement of the design was that it present a interface that was clearly defined and easy to use from the point of view of developing new applications. Since an ease-of-use evaluation for a new interface is often based on comparisons with the previously existing methodology, we briefly note the current state of art in this area.

To obtain quantitative macromolecular data, the vast majority of current applications parse a large text file that employs a legacy format developed over 25 years ago at the Brookhaven Protein Data Bank and was originally based on punched cards [Bernstein77]. An example of this data format, that will likely be familiar to many biochemists working in the field, is shown in the excerpt below. This excerpt lists several of the atom positions in a hemoglobin molecule (4hhb.ent). Despite the many problems with this format, to its credit it is simple to understand and in most cases easy to parse.

Excerpt of ATOM records from a legacy PDB format file

```
...
ATOM      6  CG1 VAL A   1      7.009  20.127   5.418  ...
ATOM      7  CG2 VAL A   1      5.246  18.533   5.681  ...
ATOM      8  N   LEU A   2      9.096  18.040   3.857  ...
ATOM      9  CA  LEU A   2     10.600  17.889   4.283  ...
ATOM     10  C   LEU A   2     11.265  19.184   5.297  ...
ATOM     11  O   LEU A   2     10.813  20.177   4.647  ...
ATOM     12  CB  LEU A   2     11.099  18.007   2.815  ...
ATOM     13  CG  LEU A   2     11.322  16.956   1.934  ...
...
```

A single instance of the AtomSite structure documented in section "AtomSite" on page 29 stores the cartesian position and other information about an atom just as a single ATOM record does in this legacy PDB format. The complete list (an IDL sequence) of all atoms in a macromolecular structure is returned by invoking the `get_atom_site_list` method on an instance of the `Entry` interface object.

As a simple example to illustrate the ease-of-use of the interface definition, the following Java code fragment would print out the atom identifier, atom type and the cartesian (x,y,z) position for all atoms in the macromolecule 4hhb.

```
Entry e = entryFactory.get_entry_from_id("3ebx");
AtomSite[] a = e.get_atom_site_list();
for (int i = 0; i < a.length; i++)  {
    System.out.println(a[i].id + " " + a[i].type_symbol
        + " (" +  a[i].cartn.x + ", " + a[i].cartn.y
        + ", " + a[i].cartn.z + ")");
    }
```

This code fragment produces the output:

```
...
6 C (7.002, 20.127, 5.418)
7 C (5.246, 18.533, 5.681)
8 N (9.096, 18.040, 3.857)
9 C (10.60, 17.889, 4.283)
...
```

Note that in the code fragment above, only the first two lines are required to retrieve a reference to an instance of a “4hhb” Entry object and to then retrieve its list of atomic positions.

4.2.6 Indices vs. Object Embedding

Most of the data available through the interface is returned in the form of sequences of value types. There are at least two ways to link value types between sequences 1) by specifying a index into the sequence or 2) by specifying an object as embedded in the value type. A number of technical factors outlined below entered into the design decision to use indices in most cases.

Object Graphs

Many of the value types in the proposed IDL contain index references to other value types. Many of these in turn, contain index references to yet other value types and in general there is a large interconnected graph of shared value types. Since the Objects-By-Value specification requires that the graph which is reconstructed in the receiving context is structurally isomorphic to the graph in the sending context, if embedded objects were used, this would require sending the entire graph that is referenced by an object any time that object was passed as an argument. This would not permit a fine granularity of data access and the result would be a significant loss of performance when a client needs only a small subset of data.

Multiple Sequences

In some cases the same index is used into more than one sequence. If indices were not used, multiple objects would need to be embedded, rather than a single index.

The Flyweight Design Pattern

For atoms and residues, the index provides the natural context parameter for the flyweight design pattern listed in the RFP optional requirements.

4.2.7 Presence Flags

Presence flags have been included in the specification to optimize application performance. For each value type, a single bit position is defined, and is set when that value type is present for a particular Entry. Similarly, each optional field within a value type also has a defined presence bit.

Altogether, this independent set of presence flags is less that 80 bytes for each entry and allows a client to determine if any particular value type or field is present. Due to its small size, a query server could easily store the flags for an entire dataset in main memory.

The flag names for value types are of the form `S_`*<ObjectName>*. The flag names for optional fields within a particular value type of the form `F_`*<ObjectName>*`_`*<FieldName>*. A request to retrieve a value type which does not have its presence flag set, results in a `DataAccessException`.

The design of the interfaces and the presence flags also make it a relatively easy to implement a very simple server that provides only a small subset of the data, e.g. the data available from the old format PDB files. A simple server implemented using this data format could provide the basic information about sequences and atomic positions required by many applications. Since this format provides a subset of what is defined in the IDL, most of the presence flags would simply be set to false. The key point here is that both rich and simple implementations can use an identical interface.

4.2.8 Distributed State

Distributed state is required when a server must maintain information about the state of objects in its clients. As a practical matter, because connections can be terminated at any time due to hardware, software or network problems, this often ends up requiring client polling with time-outs or some other mechanism to determine when to free up memory.

One example of distributed state is when an iterator is distributed between the client and server. The server must remember how many elements each client has received thus far so it can correctly supply the next elements in the sequence. In cases where there are expected to be relatively few clients or when some distributed state already exits between the client and server, the distribution of a small amount additional state may not be a major issue. The interface presented in this proposal however is designed to support thousands of clients from a single server, and in such cases keeping track of this distributed state would present an onerous burden to the server.

In cases where there is potentially a large list of elements to be returned, e.g. the list of atom positions in AtomSiteList, this specification provides a `..._block_n()` method that has several advantages in terms of simplified memory management, scalability, reliability and performance. As a mechanism to support client side iterators, the `block_n()` method takes two parameters, the last_element read and the requested size_n. If desired, it is a simple matter to create a client object that keeps track of the last element received and implements an iterator by calling the `block_n()` method provided. The important difference is that with the `block_n()` method, there is no distributed state. The client always keeps track of the last element read and supplies this count when needed.

5 Modules

5.1 Overview

The proposed interface comprises the two modules described in this section. The first and required module, DsLSRMacromolecularStructure, contains definitions, exceptions and simple structures used in both modules. It also contains methods in the EntryFactory and Entry interfaces for accessing other optional elements.

5.2 Notation

In several places, lengthy explanations apply equally to several attributes that vary by a single letter or digit. A comma separated list enclosed in parentheses is used to represent these alternatives where the repetition would otherwise reduce readability. For example, the description contains the text ChemCompBond.atom_id_(1,2) instead of writing out the longer ChemCompBond.atom_id_1, ChemCompBond.atom_id_2. However, in all cases the IDL definitions are written out in full and are not abbreviated.

5.3 The DsLSRMacromolecularStructure Module

5.3.1 Core Module Definitions

DataAccessException

A DataAccessException is thrown whenever requested data is not available. The reason for the exception is given in the description field. The string method_name is the name of the method that threw the exception.

```
exception DataAccessException
{
   string method_name;
   string description;
};
```

Identifier Strings

There is frequently a requirement for a simple data type to indicate an entry's identity. In most cases, this need is or can be addressed by using a string type. The advantages are that it is simple, lightweight, and ubiquitous throughout the realm of computing. However the risk of using strings is that they can be too flexible, both in terms of syntax and semantics. This easily results in the lack of interoperability. To allow strings, yet mitigate their potential for abuse, this standard uses a restricted version of the syntax convention of `CosNaming::StringName` as described in the Interoperable Naming service. This convention is mainly a syntactical one; in no way is the use of a naming service implementation required or implied (but it is not precluded either).

A brief description of `CosNaming::StringName` is as follows. `CosNaming::Name` is a list of `struct NameComponent`s. For the purpose of illustration, a `NameComponent` can be likened to a directory or filename, whereas `CosNaming::Name` constitutes a full path-name. The `struct NameComponent` has string members `id` and `kind`. To transform a `CosNaming::Name` into a string, all its `NameComponent`s are represented as strings "*`id.kind`*". If the `kind`-field is empty, this becomes simply "*`id`*". The full *stringified* `CosNaming::Name` is obtained by concatenating all the `NameComponent`s using "`/`" as a separator character.

This same syntax convention is used with additional constraints on the `Identifier` data type. These rules do not follow from, nor are implied by any semantics of the Naming Service. The additional constraints make this data type sufficiently different from CosNaming::StringName to warrant the dedicated `typedef string Identifier`.

In the remainder of this description, 'component' means: the sub-string of an `Identifier` that corresponds to one `CosNaming::NameComponent`; likewise, *id*-field and *kind*-field correspond to the equivalent fields of `NameComponent`.

The rules are as follows:

- Names can refer to entries or groups of entries. Names referring to entries within collections consist of at least two components.
- The first component represents the data source. It is up to the implementation to document the accepted names for the data source.
- The empty name is valid for the first component, and represents the 'local' or 'default' collection. It is up to the implementation to document what the default is.
- Names that refer to entries within collections may consist of two or more components. The second component of such names represents an identifier that is unique in the context of the data source. No empty `id`-fields are allowed in this or any further components.
- If two components are not enough to uniquely identify an entry, an `Identifier` can contain more than two components, but no more than necessary to make the identification unique. That is, an `Identifier` may not be used to freely attach textual information.
- The only characters valid in a name are "a" through "z", "0" through "9" and "_" (underscore).
- String comparisons must be done in a case-insensitive manner.

The *id* and *kind* parts of the string components of `Identifier` are used as follows:

- The *id*-field of a component contains the principal value that makes it unique in the scope provided by the preceding component. It may only be empty in the case of the first component of an `Identifier` (see above).
- The *kind*-field of a component is used to represent information indicating the release, version or mutation of an entry, and can be empty. An empty *kind*-field is synonymous with the 'standard' or most widely accepted version. It is up to the implementation to document the syntax and semantics of the version information.

The adoption of this convention has the following advantages:

- it is simple and lightweight,
- it has a well-defined and 're-used' syntax,
- it is compatible with existing practice,
- it is sufficiently flexible to allow for *sub*-IDs if necessary.

Except for the assumption that an empty *kind*-field signifies a standardized version, the revised submission on Biomolecular Sequence Analysis [BSA99] and the revised submission on Genomic Maps[GM99] use the same `Identifier` type and semantics. In these specification an empty *kind*-field signifies the most recent version. The default was made signify the standard version based on the general software engineering principle that applications (in particular end-user demonstrations) rely on fixed names to generate predicable results. The potential harm to system reliability caused by the substitution of new data is seen to override the potential benefit of using the modified data. This is reinforced by the fact that the newer data can be accessed from applications by using a non-empty *kind*-field.

typedef string Identifier;

Vector3

Representation of a 3 element tensor or translation vector.

typedef float Vector3[3];

Matrix3

Representation of a 3x3 rotation matrix in 3D Euclidean space.

typedef Vector3 Matrix3[3];

FormatTypeList

List of native formats supported for updates and deposition

```
typedef sequence<string> FormatTypeList;
```

EntryRepresentation

Representation of an entry in a native server format

```
typedef sequence<octet> EntryRepresentation;
```

IndexID

An struct used to reference a single element in an array of structures. The string id contains the referenced string value and the numerical long index can be used as an index into the array. An index value of -1 indicates the element referred to is not present in this Entry.

```
struct IndexId
{
   string id;
   long index;
};
```

VectorXYZ

Struct for a 3D spatial position when the most natural representation is to store the X, Y and Z positions as attributes.

```
struct VectorXYZ
{
   float x;
   float y;
   float z;
};
```

SeqIndex

A commonly used collection of 4 indices that uniquely identifies a sequence, with its component, asymmetric unit and alternate identifier.

```
struct SeqIndex
{
   IndexId seq;
   IndexId comp;
   IndexId asym;
   IndexId alt;
};
```

AtomIndex

A commonly used collection of 5 indices that uniquely identifies an atom, with its sequence, component, asymmetric unit and alternate identifier.

```
struct AtomIndex
{
   IndexId atom;
   IndexId seq;
   IndexId comp;
   IndexId asym;
   IndexId alt;
};
```

EntryID

Unique string identifier for an entry.

```
typedef  Identifier EntryId;
typedef sequence<EntryId> EntryIdList;
```

Entry Groups

Entry groups form a traditional two-level hierarchy for entry lists.

```
typedef Identifier EntryGroupId;
typedef sequence<EntryGroupId> EntryGroupIdList;
```

Modification Date

Date the entry was last modified. The TimeT date is specified in coordinated universal time (UTC) defined by the OMG TimeBase IDL.

```
struct ModificationDate
   {
      EntryId entry_id;
      TimeT date;
   };
typedef sequence<ModificationDate> ModificationDateList;
```

5.3.2 The EntryFactory Interface

The EntryFactory interface contains methods for returning lists of Entry identifiers, obtaining a single Entry object reference, and methods for efficiently updating mirror servers.

The `get_version()` method retrieves a string identifying the type and version number of the server.

Retrieving Lists of Entries

`get_entry_id_list()` retrieves a list of all entries.

The `get_entry_modification_dates()` method retrieves a list of all entries along with the date they were last modified. The time information provided by this method allows mirror servers to find new or modified entries and to incrementally bring the mirror server up to date.

Entry Groups

A server may optionally partition the complete set of entries into smaller more manageable groups. The manner in which the server divides the entries into groups is not defined by this specification.

A list of the entry groups is retrieved with `get_entry_group_list()`. All entries in a specified entry group are retrieved with `get_entries_in_group()`.

Obtaining a n Entry Object.

To retrieve a reference to the Entry interface object for a specified EntryId string the method `get_entry_from_id()` is used.

Native Format Methods

The `native_formats_supported()` method retrieves a list of native formats a server supports. The data representing an entry is retrieved with `get_native_entry_representation().`

BaseIDL

The `get_extension_modules` method returns a list of metamodels that describe optional services provided by an implementation. The returned metamodel representation type, `BaseIDL::ModuleDefSet`, is defined in the OMG Components Model and Component Descriptors specification [orbos/99-07-02] which is based on the Meta-Object Facility [ad/99-09-05].

```
interface EntryFactory
{
	string get_version();
	BaseIDL::ModuleDefSet get_extension_modules();
	EntryIdList get_entry_id_list()
		raises (DataAccessException);
	EntryIdList get_entry_id_list_block_n(
		in long from,
		in long to)
		raises (DataAccessException);
	ModificationDateList get_entry_modification_dates()
		raises (DataAccessException);
	ModificationDateList get_entry_modification_dates_block_n(
		in long from,
		in long to)
		raises (DataAccessException);
	EntryGroupIdList get_entry_group_list()
		raises (DataAccessException);
	EntryIdList get_entries_in_group(in EntryGroupId group)
		raises (DataAccessException);
	Entry get_entry_from_id(in EntryId entry_id)
		raises (DataAccessException);
	FormatTypeList native_formats_supported()
		raises (DataAccessException);
	EntryRepresentation get_native_entry_representation(
		in FormatType format,
		in EntryId entry_id)
		raises (DataAccessException);
}
```

5.3.3 The Entry Interface

Central to the design, is the Entry interface object. All the data structures are retrieved using methods defined on an Entry object.

Presence Flags

A Flags vector returned by `get_presences_flags()` is used to efficiently determine those value types that are present for a given entry, and which fields in each valuetype are valid. The Flag vector represents a sequence of bits with a bit set to "1" indicating a particular valuetype or field is present and valid.

The index of the octet within the sequence is determined by integer division of the flags numeric value by eight (`flag/8`). The bit within the octet is specified by the low 3 order bits of the flags numeric value (`1<<(flag&7)`).

The `get_presence_flag()` method retrieves the present/valid flags for an entry. Flags that indicate if a valuetype or struct is present are indicated with a "`S_`" prefix. Flags indicating the validity of optional fields within a valuetype are indicated with a "`F_`" prefix followed by the name of the valuetype and the field name. Flags are not provided for the mandatory fields that are always present and valid.

In cases where a sequence of value types contains a string field which is sometimes but not always valid, the Flag bit is set to true and the string data values that are undefined are represented by a period “.”; data values that are unknown are represented by a question mark “?”. Integer fields that are undefined or unknown shall be assigned the maximum negative value for that type. Floating point fields that are undefined or unknown are assigned a NaN (Not-a-Number) value.

Subentries

Subentries provides a well defined mechanism for obtaining optional, supplemental information about a macromolecular structure in addition to that available from the core Entry object.

The optional module DsLSRMmsReference defines a subentry interface `MmsReferenceEntry` that functions analogously to the `Entry` interface in the core DsLSRMacromolecularStructure module. Like the `Entry` interface this subentry interface defines its own set of presence flags and its own set of access methods for the data structures defined in the module.

Extension modules described using the MOF and returned by the `get_extension_modules` method in the `EntryFactory` interface are also expected to define analogous subentries.

Once a reference to an `Entry` objects is obtained, the list of available subentries may be retrieved using the `get_subentry_list` method. This returned list is represented as a `CosPropertyService::Properties` struct. To help insure the correct operation of programs it is required that each property name be unique and the `property_name` attribute correctly identifies the type of the object stored in the `property_value`, i.e. a “narrow” operation to the type specified by `property_name` would be successful.

The data retrieval methods

Each of the “data” value types and structs defined in the modules has two corresponding methods in the entry or subentry interface. One to retrieve the actual list of structures and another that simply returns the size of the list.

```
typedef sequence<octet> Flags;

interface Entry
{
    Flags get_presence_flags()
      raises (DataAccessException);
    CosPropertyService::Properties get_subentry_list()
      raises (DataAccessException);

    ...
}
```

5.3.4 DsLSRMacromolecularStructure Summary

The following structures and value type make up the core DsLSRMacromolecularStructure module. They have been placed together here in categories according to their content.

ATOM

AtomSite
Details of each atomic position

AtomSiteExt
Fundamental type and position information

AtomSiteAnisotrop
Anisotropic thermal displacement

AtomType
Properties of an atom at a particular atom site

CHEM COMP

ChemComp
Details of the chemical components

ChemCompAngle
Bond angles in a chemical component

ChemCompAtom
Atoms defining a chemical component

ChemCompBond
Characteristics of bonds in a chemical component

ChemCompChir
Details of the chiral centers in a chemical component

ChemCompChirAtom
Atoms comprising a chiral center in a chemical component

ChemCompLink
Linkages between chemical Categories

ChemCompPlane
Planes found in a chemical component

ChemCompPlaneAtom
Atoms comprising a plane in a chemical component

ChemCompTor
Details of the torsion angles in a chemical component

ChemCompTorValue
Target values for the torsion angles in a chemical component

CHEM LINK

ChemLink

Details of the linkages between chemical components

ChemLinkAngle

Details of the angles in the chemical component linkage

ChemLinkBond

Details of the bonds in the chemical component linkage

ChemLinkChir

Chiral centers in a link between two chemical components

ChemLinkChirAtom

Atoms bonded to a chiral atom in a linkage between two chemical components

ChemLinkPlane

Planes in a linkage between two chemical components

ChemLinkPlaneAtom

Atoms in the plane forming a linkage between two chemical components

ChemLinkTor

Torsion angles in a linkage between two chemical components

ChemLinkTorValue

Target values for torsion angles enumerated in a linkage between two chemical components

ENTITY

Entity

Details pertaining to each unique chemical component of the structure

EntityKeywords

Keywords describing each entity

EntityLink

Details of the links between entities

EntityNameCom

Common name for the entity

EntityNameSys

Systematic name for the entity

EntityPoly

Characteristics of a polymer

EntityPolySeq

Sequence of monomers in a polymer

EntitySrcGen

Source of the entity

EntitySrcNat

Details of the natural source of the entity

GEOM

Geom

Derived geometry information

GeomAngle

Derived bond angles

GeomBond

Derived bonds

GeomContact

Derived intermolecular contacts

GeomTorsion

Derived torsion angles

STRUCT

Struct

Details pertaining to a description of the structure

StructAsym

Details pertaining to structure components within the asymmetric unit

StructBiol

Details pertaining to components of the structure that have biological significance

StructBiolGen

Details pertaining to generating biological components

StructBiolKeywords

Keywords for describing biological components

StructBiolView

Description of views of the structure with biological significance

StructConf

Conformations of the backbone

StructConfType

Details of each backbone conformation

StructConn

Details pertaining to intermolecular contacts

StructConnType

Details of each type of intermolecular contact

StructKeywords

Description of the chemical structure

StructMonDetails

Calculation summaries at the monomer level

StructMonNucl

Calculation summaries specific to nucleic acid monomers

StructMonProt

Calculation summaries specific to protein monomers

StructMonProtCis

Calculation summaries specific to cis peptides

StructNcsDom

Details of domains within an ensemble of domains

StructNcsDomLim

Beginning and end points within polypeptide chains forming a specific domain

StructNcsEns

Description of ensembles

StructNcsEnsGen

Description of domains related by non-crystallographic symmetry

StructNcsOper

Operations required to superimpose individual members of an ensemble

StructRef

External database references to biological units within the structure

StructRefSeq

Describes the alignment of the external database sequence with that found in the structure

StructRefSeqDif

Describes differences in the external database sequence with that found in the structure

StructSheet

Beta sheet description

StructSheetHbond

Hydrogen bond description in beta sheets

StructSheetOrder

Order of residue ranges in beta sheets

StructSheetRange

Residue ranges in beta sheets

StructSheetTopology

Topology of residue ranges in beta sheets

StructSite

Details pertaining to specific sites within the structure

StructSiteGen

Details pertaining to how the site is generated

StructSiteKeywords

Keywords describing the site

StructSiteView

Description of views of the specified site

5.3.5 DsLSRMacromolecularStructure Valuetypes and Structs

AtomSite

Data fields in the AtomSite valuetype record details about the atom sites in a macromolecular structure, such as the positional coordinates, atomic displacement parameters, magnetic moments and directions, and so on.

The data fields for describing anisotropic temperature or thermal displacement factors are only used if the corresponding fields are not given in the AtomSiteAnisotrop valuetype.

The existence of the AtomSite valuetype in an Entry is optional. Its presence can be determined using the S_ATOM_SITE flag.

```
struct AtomSite
  {
   ...
  };

typedef sequence<AtomSite> AtomSiteList;
```

AtomSite.id

The value of AtomSite.id must uniquely identify a record in the AtomSite list.

AtomSite.id is a mandatory field and will always be set to a valid value.

```
string id;
```

AtomSite.type_symbol

Type_symbol is a pointer to AtomType.symbol in the AtomType valuetype.

AtomSite.type_symbol is a mandatory field and will always be set to a valid value. Type_symbol is an index into the AtomType list such that the id field (type_symbol) is equal to AtomType.symbol.

```
IndexId type_symbol;
```

AtomSite.label

Components of the macromolecular identifier for this atom site.

Label.atom is an index into the ChemCompAtom list such that the id field (label_atom.id) is equal to ChemCompAtom.atom_id. AtomSite.label.atom is an optional field. The flag F_ATOM_SITE_LABEL_ATOM_ID can be used to determine if its value has been set.

Label.comp is an index into the ChemComp list such that the id field (label_comp.id) is equal to ChemComp.id. AtomSite.label.comp is an optional field. The flag F_ATOM_SITE_LABEL_COMP_ID can be used to determine if its value has been set.

Label.asym is an index into the StructAsym list such that the id field (label_asym.id) is equal to StructAsym.id. AtomSite.label.asym is an optional field. The flag F_ATOM_SITE_LABEL_ASYM_ID can be used to determine if its value has been set.

Label.seq is an index into the EntityPolySeq list such that the id field (label_seq.id) is equal to EntityPolySeq.num. AtomSite.label.seq is an optional field. The flag F_ATOM_SITE_LABEL_SEQ_ID can be used to determine if its value has been set.

Label.alt is an index into the AtomSitesAlt list such that the id field (label_alt.id) is equal to AtomSitesAlt.id. AtomSite.label_alt_id is an optional field. The flag F_ATOM_SITE_LABEL_ALT_ID can be used to determine if its value has been set.

```
AtomIndex label;
```

AtomSite.label_entity

Label_entity is an index into the Entity list such that the id field (label_entity.id) is equal to Entity.id. AtomSite.label_entity_id is an optional field. The flag F_ATOM_SITE_LABEL_ENTITY_ID can be used to determine if its value has been set.

```
IndexId label_entity;
```

AtomSite.cartn

The x, y and z atom site coordinates in angstroms specified according to a set of orthogonal Cartesian axes related to the cell axes as specified by the description given in AtomSites.cartn_transform_axes.

AtomSite.cartn.(x,y,z) are optional fields. The flags F_ATOM_SITE_CARTN_(X,Y,Z) can be used to determine if their value has been set.

```
VectorXYZ cartn;
```

AtomSite.occupancy

The fraction of the atom type present at this site. The sum of the occupancies of all the atom types at this site may not significantly exceed 1.0 unless it is a dummy site.

AtomSite.occupancy is an optional field. The flag F_ATOM_SITE_OCCUPANCY can be used to determine if its value has been set.

```
float occupancy;
```

AtomSite.b_iso_or_equiv

Isotropic temperature factor parameter, or equivalent isotropic temperature factor, B_{equiv} calculated from anisotropic temperature factor parameters.

$$B_{equiv} = \frac{1}{3}\sum_{i}\sum_{j} B_{ij} A_i A_j a_i^* a_j^*$$

Where:

A = the real space cell lengths

a^* = the reciprocal space cell lengths

$B_{ij} = 8\pi^2 U_{ij}$

Ref: Fischer, R. X. & Tillmanns, E. (1988). Acta Cryst. C44, 775-776.

AtomSite.b_iso_or_equiv is an optional field. The flag F_ATOM_SITE_B_ISO_OR_EQUIV can be used to determine if its value has been set.

```
float b_iso_or_equiv;
```

AtomSiteExt

Data fields in the AtomSiteExt valuetype record details about the atom sites in a macromolecular structure, such as the positional coordinates, atomic displacement parameters, magnetic moments and directions, and so on.

The data fields for describing anisotropic temperature or thermal displacement factors are only used if the corresponding fields are not given in the AtomSiteAnisotrop valuetype. The existence of the AtomSiteExt valuetype in an Entry is optional. Its presence can be determined using the S_ATOM_SITE_EXT flag.

```
struct AtomSiteExt
  {
   ...
  };

typedef sequence<AtomSiteExt> AtomSiteExtList;
```

AtomSitesExt.aniso_b[i][j]

The elements of the anisotropic thermal displacement matrix B, which appears in the structure factor term as:

$$T = \exp\left\{-\frac{1}{4}\sum_i \sum_j B_{ij} h_i h_j a^*_i a^*_j\right\}$$

Where:

h = the Miller indices

a^* = the reciprocal space cell lenghts

These matrix elements may appear with atomic coordinates in the AtomSiteExt valuetype, or they may appear in the separate AtomSiteAnisotrop valuetype, but they do not appear in both places. Similarly, anisotropic displacements may appear as either B's or U's, but not as both.

The IUCr Commission on Nomenclature recommends against the use of B for reporting atomic displacement parameters. U, being directly proportional to B, is preferred.

AtomSiteExt.aniso_b is an optional field. The flag F_ATOM_SITE_EXT_ANISO_B can be used to determine if its value has been set.

Matrix3 aniso_b;

AtomSiteExt.aniso_b_esd[i][j]

The estimated standard deviation of AtomSiteExt.aniso_b[i][j].

AtomSiteExt.aniso_b_esd is an optional field. The flag F_ATOM_SITE_EXT_ANISO_B_ESD can be used to determine if its value has been set.

Matrix3 aniso_b_esd;

AtomSiteExt.aniso_ratio

Ratio of the maximum to minimum principal axes of displacement (thermal) ellipsoids.

AtomSiteExt.aniso_ratio is an optional field. The flag F_ATOM_SITE_EXT_ANISO_RATIO can be used to determine if its value has been set.

float aniso_ratio;

AtomSiteExt.aniso_u[i][j]

The elements of the standard anisotropic atomic displacement matrix U, which appears in the structure factor term:

$$T = \exp\left\{-2\pi^2 \sum_i \sum_j U_{ij} h_i h_j a^*_i a^*_j\right\}$$

Where:

h = the Miller indices

a^* = the reciprocal space cell lenghts

These matrix elements may appear with atomic coordinates in the AtomSiteExt valuetype, or they may appear in the separate AtomSiteAnisotrop valuetype, but they do not appear in both places. Similarly, anisotropic displacements may appear as either B’s or U’s, but not as both.

AtomSiteExt.aniso_u is an optional field. The flag F_ATOM_SITE_EXT_ANISO_U can be used to determine if its value has been set.

Matrix3 aniso_u;

AtomSiteExt.aniso_u[i][j]

The estimated standard deviation of AtomSiteExt.aniso_u[i][j].

AtomSiteExt.aniso_u_esd is an optional field. The flag F_ATOM_SITE_EXT_ANISO_U_ESD can be used to determine if its value has been set.

Matrix3 aniso_u_esd;

AtomSiteExt.attached_hydrogens;

The number of hydrogen atoms attached to the atom at this site excluding any H atoms for which coordinates (measured or calculated) are given.

AtomSiteExt.attached_hydrogens is an optional field. The flag F_ATOM_SITE_EXT_ATTACHED_HYDROGENS can be used to determine if its value has been set.

long attached_hydrogens;

AtomSiteExt.auth_asym_id

An alternative identifier for AtomSite.label.asym.id that may be provided by an author in order to match the identification used in the publication that describes the structure.

AtomSiteExt.auth_asym_id is an optional field. The flag F_ATOM_SITE_EXT_AUTH_ASYM_ID can be used to determine if its value has been set.

string auth_asym_id;

AtomSiteExt.auth_atom_id

An alternative identifier for AtomSite.label.atom.id that may be provided by an author in order to match the identification used in the publication that describes the structure.

AtomSiteExt.auth_atom_id is an optional field. The flag F_ATOM_SITE_EXT_AUTH_ATOM_ID can be used to determine if its value has been set.

string auth_atom_id;

AtomSiteExt.auth_comp_id

An alternative identifier for AtomSite.label.comp.id that may be provided by an author in order to match the identification used in the publication that describes the structure.

AtomSiteExt.auth_comp_id is an optional field. The flag F_ATOM_SITE_EXT_AUTH_COMP_ID can be used to determine if its value has been set.

string auth_comp_id;

AtomSiteExt.auth_seq_id

An alternative identifier for AtomSite.label.seq.id that may be provided by an author in order to match the identification used in the publication that describes the structure.

Note that this is not necessarily a number, that the values do not have to be positive, and that the value does not have to correspond to the value of AtomSite.label.seq.id. The value of AtomSiteExt.label_seq_id is required to be a sequential list of positive integers.

The deposition author may assign values to AtomSiteExt.auth_seq_id in any way they choose. For instance, the values may be used to relate this structure to a numbering scheme in a homologous structure, including sequence gaps or insertion codes. Alternatively, a scheme may to used for a truncated polymer that maintains the numbering scheme of the full length polymer. In all cases, the scheme used here must match the scheme used in the publication that describes the structure.

AtomSiteExt.auth_seq_id is an optional field. The flag F_ATOM_SITE_AUTH_SEQ_ID can be used to determine if its value has been set.

```
string auth_seq_id;
```

AtomSiteExt.b_equiv_geom_mean

Equivalent isotropic atomic displacement parameter, B_{equiv} in angstroms squared, calculated as the geometric mean of the anisotropic atomic displacement parameters.

$$B_{equiv} = (B_i B_j B_k)^{1/3}$$

where:

B_n = the principal components of the orthogonalised B_{ij}

AtomSiteExt.b_equiv_geom_mean is an optional field. The flag F_ATOM_SITE_EXT_B_EQUIV_GEOM_MEAN can be used to determine if its value has been set.

```
float b_equiv_geom_mean;
```

AtomSiteExt.b_equiv_geom_mean_esd

The estimated standard deviation of AtomSiteExt.b_equiv_geom_mean.

AtomSiteExt.b_equiv_geom_mean_esd is an optional field. The flag F_ATOM_SITE_EXT_B_EQUIV_GEOM_MEAN_ESD can be used to determine if its value has been set.

```
float b_equiv_geom_mean_esd;
```

AtomSiteExt.b_iso_or_equiv_esd

The estimated standard deviation of AtomSiteExt.b_iso_or_equiv.

AtomSiteExt.b_iso_or_equiv_esd is an optional field. The flag F_ATOM_SITE_EXT_B_ISO_OR_EQUIV_ESD can be used to determine if its value has been set.

float b_iso_or_equiv_esd;

AtomSiteExt.calc_attached_atom

The AtomSiteExt.id of the atom site to which the 'geometry-' calculated' atom site is attached.'

AtomSiteExt.calc_attached_atom is an optional field. The flag F_ATOM_SITE_CALC_ATTACHED_ATOM can be used to determine if its value has been set.

string calc_attached_atom;

AtomSiteExt.calc_flag

A standard code to signal if the site data have been determined from the intensities or calculated from the geometry of surrounding sites, or have been assigned dummy coordinates. The abbreviation 'c' may be used in place of 'calc'.

AtomSiteExt.calc_flag is an optional field. The flag F_ATOM_SITE_CALC_FLAG can be used to determine if its value has been set.

string calc_flag;

AtomSiteExt.cartn_esd

The estimated standard deviation of AtomSite.cartn.(x,y,z). AtomSite.cartn_esd.(x,y,z) are optional fields. The flags F_ATOM_SITE_CARTN_EXT_ESD_(X,Y,Z) can be used to determine if their value has been set.

VectorXYZ cartn_esd;

AtomSiteExt.constraints

A description of the constraints applied to parameters at this site during refinement. See also AtomSiteExt.refinement_flags and Refine.ls_number_constraints.

AtomSiteExt.constraints is an optional field. The flag F_ATOM_SITE_EXT_CONSTRAINTS can be used to determine if its value has been set.

string constraints;

AtomSiteExt.details

A description of special aspects of this site. See also AtomSiteExt.refinement_flags.

AtomSiteExt.details is an optional field. The flag F_ATOM_SITE_EXT_DETAILS can be used to determine if its value has been set.

string details;

AtomSiteExt.disorder_group

A code that identifies a group of positionally disordered atom sites that are locally simultaneously occupied. Atoms that are positionally disordered over two or more sites (e.g. the H atoms of a methyl group that exists in two orientations) can be assigned to two or more groups. Sites belonging to the same group are simultaneously occupied, but those belonging to different groups are not. A minus prefix (e.g. "-1") is used to indicate sites disordered about a special position.

AtomSiteExt.disorder_group is an optional field. The flag F_ATOM_SITE_DISORDER_GROUP can be used to determine if its value has been set.

string disorder_group;

AtomSiteExt.footnote

The value of AtomSiteExt.footnote_id must match an id specified by AtomSiteExtsFootnote.id in the AtomSiteExtsFootnote list.

AtomSiteExt.footnote_id is an optional field. The flag F_ATOM_SITE_FOOTNOTE_ID can be used to determine if its value has been set. Footnote is an index into the AtomSitesFootnote list such that the id field (footnote.id) is equal to AtomSitesFootnote.id.

IndexId footnote;

AtomSite.fract

The x, y and z coordinates of the atom site position specified as a fraction of Cell. length.

AtomSiteExt.fract.(x,y,z) are optional fields. The flags F_ATOM_SITE_EXT_FRACT(X,Y,Z) can be used to determine if their value has been set.

VectorXYZ fract;

AtomSite.fract_esd

The estimated standard deviation of AtomSiteExt.fract. AtomSiteExt.fract_esd.(x,y,z) are optional fields. The flags F_ATOM_SITE_EXT_FRACT_ESD(X,Y,Z) can be used to determine if their value has been set.

VectorXYZ fract_esd;

AtomSiteExt.occupancy_esd

The estimated standard deviation of AtomSiteExt.occupancy.

AtomSiteExt.occupancy_esd is an optional field. The flag F_ATOM_SITE_EXT_OCCUPANCY_ESD can be used to determine if its value has been set.

```
float occupancy_esd;
```

AtomSiteExt.occupancy_esd

A concatenated series of single-letter codes which indicate the refinement restraints or constraints applied to this site.

AtomSiteExt.refinement_flags is an optional field. The flag F_ATOM_SITE_EXT_REFINEMENT_FLAGS can be used to determine if its value has been set.

```
string refinement_flags;
```

AtomSiteExt.restraints

A description of restraints applied to specific parameters at this site during refinement. See also AtomSiteExt.refinement_flags and Refine.ls_number_restraints.

AtomSiteExt.restraints is an optional field. The flag F_ATOM_SITE_EXT_RESTRAINTS can be used to determine if its value has been set.

```
string restraints;
```

AtomSiteExt.symmetry_multiplicity

The multiplicity of a site due to the space-group symmetry as is given in International Tables for Crystallography, Vol. A (1987). AtomSiteExt.symmetry_multiplicity is an optional field. The flag F_ATOM_SITE_EXT_SYMMETRY_MULTIPLICITY can be used to determine if its value has been set.

```
long symmetry_multiplicity;
```

AtomSiteExt.thermal_displace_type

A standard code used to describe the type of atomic displacement parameters used for the site.

AtomSiteExt.thermal_displace_type is an optional field. The flag F_ATOM_SITE_EXT_THERMAL_DISPLACE_TYPE can be used to determine if its value has been set.

```
string thermal_displace_type;
```

AtomSiteExt.u_equiv_geom_mean

Equivalent isotropic atomic displacement parameter, U~equiv~, in angstroms squared, calculated as the geometric mean of the anisotropic atomic displacement parameters.

Equivalent isotropic atomic displacement parameter, U_{equiv} in angstroms squared, calculated as the geometric mean of the anisotropic atomic displacement parameters.

$$U_{equiv} = (U_i U_j U_k)^{1/3}$$

where:

$_n$ = the principal components of the orthogonalised U

AtomSiteExt.u_equiv_geom_mean is an optional field. The flag F_ATOM_SITE_EXT_U_EQUIV_GEOM_MEAN can be used to determine if its value has been set.

```
float u_equiv_geom_mean;
```

AtomSiteExt.u_equiv_geom_mean_esd

The estimated standard deviation of AtomSiteExt.u_equiv_geom_mean.

AtomSiteExt.u_equiv_geom_mean_esd is an optional field. The flag F_ATOM_SITE_EXT_U_EQUIV_GEOM_MEAN_ESD can be used to determine if its value has been set.

```
float u_equiv_geom_mean_esd;
```

AtomSiteExt.u_iso_or_equiv

Isotropic atomic displacement parameter, or equivalent isotropic atomic displacement parameter, U_{equiv} calculated from anisotropic atomic displacement parameters.

$$U_{equiv} = \frac{1}{3}\sum_i \sum_j U_{ij} A_i A_j a_i^* a_j^*$$

Where:

A = the real space cell lengths

a^* = the reciprocal space cell lengths

Ref: Fischer, R. X. & Tillmanns, E. (1988). Acta Cryst. C44, 775-776.

AtomSiteExt.u_iso_or_equiv is an optional field. The flag F_ATOM_SITE_EXT_U_ISO_OR_EQUIV can be used to determine if its value has been set.

```
float u_iso_or_equiv;
```

AtomSiteExt.u_iso_or_equiv_esd

The estimated standard deviation of AtomSiteExt.u_iso_or_equiv.

AtomSiteExt.u_iso_or_equiv_esd is an optional field. The flag F_ATOM_SITE_EXT_U_ISO_OR_EQUIV_ESD can be used to determine if its value has been set.

```
float u_iso_or_equiv_esd;
```

AtomSiteExt.wyckoff_symbol

The Wyckoff symbol (letter) as listed in the space-group section of International Tables for Crystallography, Vol. A (1987).

AtomSiteExt.wyckoff_symbol is an optional field. The flag F_ATOM_SITE_WYCKOFF_SYMBOL can be used to determine if its value has been set.

```
string wyckoff_symbol;
```

AtomSiteAnisotrop

Data fields in the AtomSiteAnisotrop valuetype record details about temperature or thermal displacement factors, if those data fields are contained in a separate list from the AtomSite list. If the AtomSiteAnisotrop valuetype is used for storing these data, the corresponding AtomSite data fields are not used.

The existence of the AtomSiteAnisotrop valuetype in an Entry is optional. Its presence can be determined using the S_ATOM_SITE_ANISOTROP flag.

```
valuetype AtomSiteAnisotrop
{
  ...
};

typedef sequence<AtomSiteAnisotrop> AtomSiteAnisotropList;
```

AtomSiteAnisotrop.b

The elements of the anisotropic thermal displacement matrix B, which appears in the structure factor term as:

$$T = \exp\left\{-\frac{1}{4}\sum_i \sum_j B_{ij} h_i h_j a^*_i a^*_j\right\}$$

Where:

h = the Miller indices

a^* = the reciprocal space cell lenghts

These matrix elements may appear with atomic coordinates in the AtomSite valuetype, or they may appear in the separate AtomSiteAnisotrop valuetype, but they may not appear in both places. Similarly, anisotropic displacements may appear as either B's or U's, but not as both.

The IUCr Commission on Nomenclature recommends against the use of B for reporting atomic displacement parameters. U, being directly proportional to B, is preferred.

AtomSiteAnisotrop.b is an optional field. The flag F_ATOM_SITE_ANISOTROP_B can be used to determine if its value has been set.

```
Matrix3 b;
```

AtomSiteAnisotrop.b_esd

The estimated standard deviation of AtomSiteAnisotrop.b[i][j]. AtomSiteAnisotrop.b_esd is an optional field. The flag F_ATOM_SITE_ANISOTROP_B_ESD can be used to determine if its value has been set.

```
Matrix3 b_esd;
```

AtomSiteAnisotrop.ratio

Ratio of the maximum to minimum principal axes of displacement (thermal) ellipsoids.

AtomSiteAnisotrop.ratio is an optional field. The flag F_ATOM_SITE_ANISOTROP_RATIO can be used to determine if its value has been set.

```
float ratio;
```

AtomSiteAnisotrop.id

Id is a pointer to AtomSite.id in the AtomSite valuetype.

AtomSiteAnisotrop.id is a mandatory field and will always be set to a valid value. Id is an index into the AtomSite list such that the id field (id) is equal to AtomSite.id.

```
IndexId id;
```

AtomSiteAnisotrop.type_symbol

Type_symbol is a pointer to AtomType.symbol in the AtomType valuetype.

AtomSiteAnisotrop.type_symbol is a mandatory field and will always be set to a valid value. Type_symbol is an index into the AtomType list such that the id field (type_symbol) is equal to AtomType.symbol.

```
IndexId type_symbol;
```

AtomSiteAnisotrop.u

The elements of the standard anisotropic atomic displacement matrix U, which appears in the structure factor term:

$$T = \exp\left\{-2\pi^2 \sum_i \sum_j U_{ij} h_i h_j a^*_i a^*_j\right\}$$

Where:

h = the Miller indices

a^* = the reciprocal space cell lenghts

These matrix elements may appear with atomic coordinates in the AtomSite valuetype, or they may appear in the separate AtomSiteAnisotrop valuetype, but they may not appear in both places. Similarly, anisotropic displacements may appear as either B's or U's, but not as both.

AtomSiteAnisotrop.u is an optional field. The flag F_ATOM_SITE_ANISOTROP_U can be used to determine if its value has been set.

```
Matrix3 u;
```

AtomSiteAnisotrop.u_esd

The estimated standard deviation of AtomSiteAnisotrop.u[i][j].

AtomSiteAnisotrop.u_esd is an optional field. The flag F_ATOM_SITE_ANISOTROP_U_ESD can be used to determine if its value has been set.

```
Matrix3 u_esd;
```

AtomType

Data fields in the AtomType valuetype record details about properties of the atoms that occupy the atom sites, such as the atomic scattering factors.

The existence of the AtomType valuetype in an Entry is optional. Its presence can be determined using the S_ATOM_TYPE flag.

```
valuetype AtomType
{
  ...
};

typedef sequence<AtomType> AtomTypeList;
```

AtomType.analytical_mass_percent

Mass percentage of this atom type derived from chemical analysis. AtomType.analytical_mass_percent is an optional field. The flag F_ATOM_TYPE_ANALYTICAL_MASS_PERCENT can be used to determine if its value has been set.

```
float analytical_mass_percent;
```

AtomType.description

A description of the atom(s) designated by this atom type. In most cases this is the element name and oxidation state of a single atom species. For disordered or nonstoichiometric structures it will describe a combination of atom species. AtomType.description is an optional field. The flag F_ATOM_TYPE_DESCRIPTION can be used to determine if its value has been set.

```
string description;
```

AtomType.number_in_cell

Total number of atoms of this atom type in the unit cell. AtomType.number_in_cell is an optional field. The flag F_ATOM_TYPE_NUMBER_IN_CELL can be used to determine if its value has been set.

```
long number_in_cell;
```

AtomType.oxidation_number

Formal oxidation state of this atom type in the structure.

AtomType.oxidation_number is an optional field. The flag F_ATOM_TYPE_OXIDATION_NUMBER can be used to determine if its value has been set.

```
long oxidation_number;
```

AtomType.radius_bond

The effective intramolecular bonding radius in angstroms of this atom type.

AtomType.radius_bond is an optional field. The flag F_ATOM_TYPE_RADIUS_BOND can be used to determine if its value has been set.

```
float radius_bond;
```

AtomType.radius_contact

The effective intermolecular bonding radius in angstroms of this atom type.

AtomType.radius_contact is an optional field. The flag F_ATOM_TYPE_RADIUS_CONTACT can be used to determine if its value has been set.

```
float radius_contact;
```

AtomType.scat_cromer_mann_(a1,a2,a3,a4,b1,b2,b3,b4,c)

The Cromer-Mann scattering-factor coefficients used to calculate the scattering factors for this atom type.

Ref: International Tables for X-ray Crystallography, Vol. Iv, (1974). Table 2.2B. or: International Tables for Crystallography, Vol. C, (1991). Tables 6.1.1.4 and 6.1.1.5.

AtomType.scat_cromer_mann_(a1,a2,a3,a4,b1,b2,b3,b4,c) are optional fields. The flags F_ATOM_TYPE_SCAT_CROMER_MANN_(A1,A2,A3,A4,B1,B2,B3,B4,C) can be used to determine if their value has been set.

```
float scat_cromer_mann_a1;
float scat_cromer_mann_a2;
float scat_cromer_mann_a3;
float scat_cromer_mann_a4;
float scat_cromer_mann_b1;
float scat_cromer_mann_b2;
float scat_cromer_mann_b3;
float scat_cromer_mann_b4;
float scat_cromer_mann_c;
```

AtomType.scat_dispersion_imag

The imaginary component of the anomalous dispersion scattering factors, f'' (in electrons) for this atom type.

AtomType.scat_dispersion_imag is an optional field. The flag F_ATOM_TYPE_SCAT_DISPERSION_IMAG can be used to determine if its value has been set.

```
float scat_dispersion_imag;
```

AtomType.scat_dispersion_real

The real component of the anomalous dispersion scattering factors, and f' (in electrons) for this atom type.

AtomType.scat_dispersion_real is an optional field. The flag F_ATOM_TYPE_SCAT_DISPERSION_REAL can be used to determine if its value has been set.

```
float scat_dispersion_real;
```

AtomType.scat_length_neutron

The bound coherent scattering length in femtometres for the atom type at the isotopic composition used for the diffraction experiment.

AtomType.scat_length_neutron is an optional field. The flag F_ATOM_TYPE_SCAT_LENGTH_NEUTRON can be used to determine if its value has been set.

```
string scat_length_neutron;
```

AtomType.scat_source

Reference to source of scattering factors used for this atom type.

AtomType.scat_source is an optional field. The flag F_ATOM_TYPE_SCAT_SOURCE can be used to determine if its value has been set.

```
string scat_source;
```

AtomType.scat_versus_stol_list

A table of scattering factors as a function of sin theta over lambda.

AtomType.scat_versus_stol_list is an optional field. The flag F_ATOM_TYPE_SCAT_VERSUS_STOL_LIST can be used to determine if its value has been set.

```
string scat_versus_stol_list;
```

AtomType.symbol

The code used to identify the atom specie(s) representing this atom type. Normally this code is the element symbol. The code may be composed of any character except an underline with the additional proviso that digits designate an oxidation state and must be followed by a + or - character.

AtomType.symbol is a mandatory field and will always be set to a valid value.

```
string symbol;
```

ChemComp

Data fields in the ChemComp valuetype give details (such as name, mass, charge, etc.) about each of the chemical components from which the relevant chemical structures can be constructed.

The related ChemCompAtom, ChemCompBond, ChemCompAngle, etc. valuetypes describe the detailed geometry of these chemical components.

The existence of the ChemComp valuetype in an Entry is optional. Its presence can be determined using the S_CHEM_COMP flag.

```
valuetype ChemComp
{
  ...
};

typedef sequence<ChemComp> ChemCompList;
```

ChemComp.formula

The formula for the chemical component. Formulae are written according to the rules:

1. Only recognised element symbols may be used.

2. Each element symbol is followed by a 'count' number. A count of '1' may be omitted.

3. A space or parenthesis must separate each element symbol and its count, but in general parentheses are not used.

4. The order of elements depends on whether or not carbon is present. If carbon is present, the order should be: C, then H, then the other elements in alphabetical order of their symbol. If carbon is not present, the elements are listed purely in alphabetic order of their symbol. This is the 'Hill' system used by Chemical Abstracts.

ChemComp.formula is an optional field. The flag F_CHEM_COMP_FORMULA can be used to determine if its value has been set.

```
string formula;
```

ChemComp.formula_weight

Formula mass in daltons of the chemical component.

ChemComp.formula_weight is an optional field. The flag F_CHEM_COMP_FORMULA_WEIGHT can be used to determine if its value has been set.

```
float formula_weight;
```

ChemComp.id

The value of ChemComp.id must uniquely identify each field in the ChemComp list. For protein polymer entities, this is the three-letter code for amino acids. For nucleic acid polymer entities, this is the one-letter code for the bases.

ChemComp.id is a mandatory field and will always be set to a valid value.

```
string id;
```

ChemComp.model_details

A description of special aspects of the generation of the coordinates for the model of the component.

ChemComp.model_details is an optional field. The flag F_CHEM_COMP_MODEL_DETAILS can be used to determine if its value has been set.

string model_details;

ChemComp.model_ext_reference_file

A pointer to an 'external reference file', if the atomic description of the component is taken from such a file.

ChemComp.model_ext_reference_file is an optional field. The flag F_CHEM_COMP_MODEL_EXT_REFERENCE_FILE can be used to determine if its value has been set.

string model_ext_reference_file;

ChemComp.model_source

The source of the coordinates for the model of the component.

ChemComp.model_source is an optional field. The flag F_CHEM_COMP_MODEL_SOURCE can be used to determine if its value has been set.

string model_source;

ChemComp.mon_nstd_class

A description of the class of a non-standard monomer, if the group represents a modification of a standard monomer. ChemComp.mon_nstd_class is an optional field. The flag F_CHEM_COMP_MON_NSTD_CLASS can be used to determine if its value has been set.

string mon_nstd_class;

ChemComp.mon_nstd_details

A description of special details of a non-standard monomer.

ChemComp.mon_nstd_details is an optional field. The flag F_CHEM_COMP_MON_NSTD_DETAILS can be used to determine if its value has been set.

string mon_nstd_details;

ChemComp.mon_nstd_flag

A 'yes' value indicates that this is a "standard" monomer, a 'no' value that it is "non-standard." Non-standard monomers should be further described using the ChemComp.mon_nstd_parent, ChemComp.mon_nstd_class, and

ChemComp.mon_nstd_details data fields. ChemComp.mon_nstd_flag is an optional field. The flag F_CHEM_COMP_MON_NSTD_FLAG can be used to determine if its value has been set.

string mon_nstd_flag;

ChemComp.mon_nstd_parent

A name of the parent monomer of the non-standard monomer, if this group represents a modification of a standard monomer.

ChemComp.mon_nstd_parent is an optional field. The flag F_CHEM_COMP_MON_NSTD_PARENT can be used to determine if its value has been set.

string mon_nstd_parent;

ChemComp.mon_nstd_parent_comp_id

The identifier for the parent component of the non-standard component. ChemComp.mon_nstd_parent_comp_id is an optional field. The flag F_CHEM_COMP_MON_NSTD_PARENT_COMP_ID can be used to determine if its value has been set. Mon_nstd_parent_comp is an index into the ChemComp list such that the id field (mon_nstd_parent_comp.id) is equal to ChemComp.id.

IndexId mon_nstd_parent_comp;

ChemComp.name

The full name of the component. ChemComp.name is an optional field. The flag F_CHEM_COMP_NAME can be used to determine if its value has been set.

string name;

ChemComp.number_atoms_all

The total number of atoms in the component. ChemComp.number_atoms_all is an optional field. The flag F_CHEM_COMP_NUMBER_ATOMS_ALL can be used to determine if its value has been set.

long number_atoms_all;

ChemComp.number_atoms_nh

The number of non-hydrogen atoms in the component. ChemComp.number_atoms_nh is an optional field. The flag F_CHEM_COMP_NUMBER_ATOMS_NH can be used to determine if its value has been set.

long number_atoms_nh;

ChemComp.one_letter_code

For standard polymer components, the one-letter code for the component. If there is not a standard one letter code for this component, or if this is a non-polymer component, the one-letter code should be given as 'X'. This code may be preceded by a '+' character to indicate that the component is a modification of a standard component.

ChemComp.one_letter_code is an optional field. The flag F_CHEM_COMP_ONE_LETTER_CODE can be used to determine if its value has been set.

```
string one_letter_code;
```

ChemComp.three_letter_code

For standard polymer components, the three-letter code for the component. If there is not a standard three letter code for this component, or if this is a non-polymer component, the three-letter code should be given as 'unk'. This code may be preceded by a '+' character to indicate that the component is a modification of a standard component.

ChemComp.three_letter_code is an optional field. The flag F_CHEM_COMP_THREE_LETTER_CODE can be used to determine if its value has been set.

```
string three_letter_code;
```

ChemComp.type

For standard polymer components, the type of the monomer. Note that monomers that will form polymers are of three types: linking monomers, monomers with some type of N-terminal (or 5')' cap, and monomers with some type of C-terminal (or 3') cap.'

ChemComp.type is a mandatory field and will always be set to a valid value.

```
string type;
```

ChemCompAngle

Data fields in the ChemCompAngle valuetype record details about angles in a chemical component. Angles are designated by three atoms, with the second atom forming the vertex of the angle. Target values may be specified as angles in degrees, as a distance between the first and third atoms, or both.

The existence of the ChemCompAngle valuetype in an Entry is optional. Its presence can be determined using the S_CHEM_COMP_ANGLE flag.

```
valuetype ChemCompAngle
{
  ...
};

typedef sequence<ChemCompAngle> ChemCompAngleList;
```

ChemCompAngle.atom_id_(1,2,3)

The ids of the three atoms that define the angle. The second atom is taken to be the apex of the angle.

ChemCompAngle.atom_id_(1,2,3) are mandatory fields and will always be set to valid values. atom_id_(1,2,3) are indices into the ChemCompAtom list such that the id field (atom_id_(1,2,3)) is equal to ChemCompAtom.atom_id.

```
IndexId atom_id_1;
IndexId atom_id_2;
IndexId atom_id_3;
```

ChemCompAngle.comp

Comp_id is a pointer to ChemComp.id in the ChemComp valuetype

ChemCompAngle.comp is a mandatory field and will always be set to a valid value. Comp is an index into the ChemComp list such that the id field (comp.id) is equal to ChemComp.id.

```
IndexId comp;
```

ChemCompAngle.value_angle

The value that should be taken as the target value for the angle associated with the specified atoms, expressed in degrees. ChemCompAngle.value_angle is an optional field. The flag F_CHEM_COMP_ANGLE_VALUE_ANGLE can be used to determine if its value has been set.

```
float value_angle;
```

ChemCompAngle.value_angle_esd

The estimated standard deviation of ChemCompAngle.value_angle. ChemCompAngle.value_angle_esd is an optional field. The flag F_CHEM_COMP_ANGLE_VALUE_ANGLE_ESD can be used to determine if its value has been set.

```
float value_angle_esd;
```

ChemCompAngle.value_dist

The value that should be taken as the target value for the angle associated with the specified atoms, expressed as the distance between the atoms specified by ChemCompAngle.atom_id_1 and ChemCompAngle.atom_id_3.

ChemCompAngle.value_dist is an optional field. The flag F_CHEM_COMP_ANGLE_VALUE_DIST can be used to determine if its value has been set.

```
float value_dist;
```

ChemCompAngle.value_dist_esd

The estimated standard deviation of ChemCompAngle.value_dist_esd. ChemCompAngle.value_dist_esd is an optional field.

The flag F_CHEM_COMP_ANGLE_VALUE_DIST_ESD can be used to determine if its value has been set.

```
float value_dist_esd;
```

ChemCompAtom

Data fields in the ChemCompAtom valuetype record details about the atoms in a chemical component. Atomic coordinates can be given for the components;

Specifying coordinates is an alternative to specifying the structure of the component via bonds, angles, planes, etc., in the appropriate ChemComp subcategories.

The existence of the ChemCompAtom valuetype in an Entry is optional. Its presence can be determined using the S_CHEM_COMP_ATOM flag.

```
valuetype ChemCompAtom
{
 ...
};

typedef sequence<ChemCompAtom> ChemCompAtomList;
```

ChemCompAtom.alt_atom_id

An alternative identifier for the atom. alt_atom_id would be used in cases where alternative nomenclatures exist for labeling atoms in a group.

ChemCompAtom.alt_atom_id is an optional field. The flag F_CHEM_COMP_ATOM_ALT_ATOM_ID can be used to determine if its value has been set.

```
string alt_atom_id;
```

ChemCompAtom.atom_id

The value of ChemCompAtom.atom_id must uniquely identify each atom in each monomer in the ChemCompAtom list.

The atom identifiers need not be unique over all atoms in the entry; they need only be unique for each atom in a component. Note that this field need not be a number; it can be any unique identifier.

ChemCompAtom.atom_id is a mandatory field and will always be set to a valid value.

string atom_id;

ChemCompAtom.charge

The net integer charge assigned to this atom. This is the formal charge assignment normally found in chemical diagrams.

ChemCompAtom.charge is an optional field. The flag F_CHEM_COMP_ATOM_CHARGE can be used to determine if its value has been set.

long charge;

ChemCompAtom.model_cartn

The x, y and z coordinates for this atom in this component specified as orthogonal angstroms. The choice of reference axis frame for the coordinates is arbitrary.

The set of coordinates input for the entity here is intended to correspond to the atomic model used to generate restraints for structure refinement, and not to atom sites in the AtomSite list.

ChemCompAtom.model_cartn.(x,y,z) are optional fields. The flags F_CHEM_COMP_ATOM_MODEL_CARTN_(X,Y,Z) can be used to determine if their value has been set.

VectorXYZ model_cartn;

ChemCompAtom.model_cartn_esd

The estimated standard deviation of ChemCompAtom.model_cartn..

ChemCompAtom.model_cartn_esd.(x,y,z) are optional fields. The flags F_CHEM_COMP_ATOM_MODEL_CARTN_ESD_(X,Y,Z) can be used to determine if their value has been set.

VectorXYZ model_cartn_esd;

ChemCompAtom.comp

Comp is a pointer to ChemComp.id in the ChemComp valuetype.

ChemCompAtom.comp is a mandatory field and will always be set to a valid value. Comp is an index into the ChemComp list such that the id field (comp.id) is equal to ChemComp.id.

IndexId comp;

ChemCompAtom.partial_charge

The partial charge assigned to this atom.

ChemCompAtom.partial_charge is an optional field. The flag F_CHEM_COMP_ATOM_PARTIAL_CHARGE can be used to determine if its value has been set.

float partial_charge;

ChemCompAtom.substruct_code

Substruct_code assigns the atom to a substructure of the component, if appropriate.

ChemCompAtom.substruct_code is an optional field. The flag F_CHEM_COMP_ATOM_SUBSTRUCT_CODE can be used to determine if its value has been set.

string substruct_code;

ChemCompAtom.type_symbol

Type_symbol is a pointer to AtomType.symbol in the AtomType valuetype.

ChemCompAtom.type_symbol is a mandatory field and will always be set to a valid value. Type_symbol is an index into the AtomType list such that the id field (type_symbol) is equal to AtomType.symbol.

IndexId type_symbol;

ChemCompBond

Data fields in the ChemCompBond valuetype record details about the bonds between atoms in a chemical component. Target values may be specified as bond orders, as a distance between the two atoms, or both.

The existence of the ChemCompBond valuetype in an Entry is optional. Its presence can be determined using the S_CHEM_COMP_BOND flag.

```
valuetype ChemCompBond
{
  ...
};

typedef sequence<ChemCompBond> ChemCompBondList;
```

ChemCompBond.atom_id_(1,2)

The ids of the atoms that define the bond.

Atom_id_(1,2) are pointers to ChemCompAtom.atom_id in the ChemCompAtom valuetype.

ChemCompBond.atom_id_(1,2) are mandatory fields and will always be set to a valid value. Atom_id_(1,2) is an index into the ChemCompAtom list such that the id field (atom_id_(1,2)) is equal to ChemCompAtom.atom_id.

```
IndexId atom_id_1;
IndexId atom_id_2;
```

ChemCompBond.comp

Comp is a pointer to ChemComp.id in the ChemComp valuetype.

ChemCompBond.comp is a mandatory field and will always be set to a valid value. Comp is an index into the ChemComp list such that the id field (comp.id) is equal to ChemComp.id.

```
IndexId comp;
```

ChemCompBond.value_order

The value that should be taken as the target for the chemical bond associated with the specified atoms, expressed as a bond order.

ChemCompBond.value_order is an optional field. The flag F_CHEM_COMP_BOND_VALUE_ORDER can be used to determine if its value has been set.

```
string value_order;
```

ChemCompBond.value_dist

The value that should be taken as the target for the chemical bond associated with the specified atoms, expressed as a distance.

ChemCompBond.value_dist is an optional field. The flag F_CHEM_COMP_BOND_VALUE_DIST can be used to determine if its value has been set.

```
float value_dist;
```

ChemCompBond.value_dist_esd

The estimated standard deviation of ChemCompBond.value_dist.

ChemCompBond.value_dist_esd is an optional field. The flag F_CHEM_COMP_BOND_VALUE_DIST_ESD can be used to determine if its value has been set.

```
float value_dist_esd;
```

ChemCompChir

Data fields in the ChemCompChir valuetype provide detail about the chiral centers in a chemical component. The atoms bonded to the chiral atom are specified in the ChemCompChirAtom valuetype.

The existence of the ChemCompChir valuetype in an Entry is optional. Its presence can be determined using the S_CHEM_COMP_CHIR flag.

```
valuetype ChemCompChir
{
  ...
};

typedef sequence<ChemCompChir> ChemCompChirList;
```

ChemCompChir.atom

The id of the atom that is a chiral center.

Atom is a pointer to ChemCompAtom.atom_id in the ChemCompAtom valuetype.

ChemCompChir.atom is a mandatory field and will always be set to a valid value. Atom is an index into the ChemCompAtom list such that the id field (atom.id) is equal to ChemCompAtom.atom_id.

```
IndexId atom;
```

ChemCompChir.atom_config

The chiral configuration of the atom that is a chiral center.

ChemCompChir.atom_config is an optional field. The flag F_CHEM_COMP_CHIR_ATOM_CONFIG can be used to determine if its value has been set.

```
string atom_config;
```

ChemCompChir.id

The value of ChemCompChir.id must uniquely identify a record in the ChemCompChir list.

ChemCompChir.id is a mandatory field and will always be set to a valid value.

```
string id;
```

ChemCompChir.comp

Comp is a pointer to ChemComp.id in the ChemComp valuetype.

ChemCompChir.comp is a mandatory field and will always be set to a valid value. Comp is an index into the ChemComp list such that the id field (comp.id) is equal to ChemComp.id.

IndexId comp;

ChemCompChir.number_atoms_all

The total number of atoms bonded to the atom specified by ChemCompChir.atom.id.

ChemCompChir.number_atoms_all is an optional field. The flag F_CHEM_COMP_CHIR_NUMBER_ATOMS_ALL can be used to determine if its value has been set.

long number_atoms_all;

ChemCompChir.number_atoms_nh

The number of non-hydrogen atoms bonded to the atom specified by ChemCompChir.atom.id.

ChemCompChir.number_atoms_nh is an optional field. The flag F_CHEM_COMP_CHIR_NUMBER_ATOMS_NH can be used to determine if its value has been set.

long number_atoms_nh;

string volume_flag

A flag to indicate whether a chiral volume should match the standard value in both magnitude and sign, or in magnitude only.

ChemCompChir.volume_flag is an optional field. The flag F_CHEM_COMP_CHIR_VOLUME_FLAG can be used to determine if its value has been set.

string volume_flag;

ChemCompChir.volume_three

The chiral volume V_c for chiral centers that involve a chiral atom bonded to three non-hydrogen atoms and one hydrogen atom.

$$V_c = V_1 \bullet (V_2 \times V_3)$$

Where:

$\bullet$ = the vector dot product

$\times$ = the vector cross product

V1 = the vector distance from the atom specified by ChemCompChir.atom.id to the first atom in the ChemCompChirAtom list

V2 = the vector distance from the atom specified by ChemCompChir.atom.id to the second atom in the ChemCompChirAtom list

V3 = the vector distance from the atom specified by ChemCompChir.atom.id to the third atom in the ChemCompChirAtom list

ChemCompChir.volume_three is an optional field. The flag F_CHEM_COMP_CHIR_VOLUME_THREE can be used to determine if its value has been set.

```
float volume_three;
```

The estimated standard deviation of ChemCompChir.volume_three.

ChemCompChir.volume_three_esd is an optional field. The flag F_CHEM_COMP_CHIR_VOLUME_THREE_ESD can be used to determine if its value has been set.

```
float volume_three_esd;
```

ChemCompChirAtom

Data fields in the ChemCompChirAtom valuetype enumerate the atoms bonded to a chiral atom within a chemical component.

The existence of the ChemCompChirAtom valuetype in an Entry is optional. Its presence can be determined using the S_CHEM_COMP_CHIR_ATOM flag.

```
valuetype ChemCompChirAtom
{
   ...
};

typedef sequence<ChemCompChirAtom> ChemCompChirAtomList;
```

ChemCompChirAtom.atom

The id of an atom bonded to the chiral atom.

Atom is a pointer to ChemCompAtom.atom_id in the ChemCompAtom valuetype.

ChemCompChirAtom.atom is a mandatory field and will always be set to a valid value. Atom is an index into the ChemCompAtom list such that the id field (atom.id) is equal to ChemCompAtom.atom_id.

```
IndexId atom;
```

ChemCompChirAtom.chir

Chir is a pointer to ChemCompChir.id in the ChemCompChir valuetype.

ChemCompChirAtom.chir_id is a mandatory field and will always be set to a valid value. Chir is an index into the ChemCompChir list such that the id field (chir.id) is equal to ChemCompChir.id.

```
IndexId chir;
```

ChemCompChirAtom.comp

Comp is a pointer to ChemComp.id in the ChemComp valuetype.

ChemCompChirAtom.comp is a mandatory field and will always be set to a valid value. Comp is an index into the ChemComp list such that the id field (comp.id) is equal to ChemComp.id.

```
IndexId comp;
```

ChemCompChirAtom.dev

The estimated standard deviation of the position of this atom from the plane defined by all of the atoms in the plane.

ChemCompChirAtom.dev is an optional field. The flag F_CHEM_COMP_CHIR_ATOM_DEV can be used to determine if its value has been set.

```
float dev;
```

ChemCompLink

Data fields in the ChemCompLink valuetype give details about the linkages between chemical components.

The existence of the ChemCompLink valuetype in an Entry is optional. Its presence can be determined using the S_CHEM_COMP_LINK flag.

```
valuetype ChemCompLink
{
  ...
};

typedef sequence<ChemCompLink> ChemCompLinkList;
```

ChemCompLink.link

Link is a pointer to ChemLink.id in the ChemLink valuetype.

ChemCompLink.link is a mandatory field and will always be set to a valid value. Link is an index into the ChemLink list such that the id field (link.id) is equal to ChemLink.id.

```
IndexId link;
```

ChemCompLink.details

A description of special aspects of a linkage between chemical components in the structure.

ChemCompLink.details is an optional field. The flag F_CHEM_COMP_LINK_DETAILS can be used to determine if its value has been set.

```
string details;
```

ChemCompLink.type_comp_(1,2)

The type of the components joined by the linkage.

Type_comp_(1,2) are pointers to ChemComp.type in the ChemComp valuetype.

ChemCompLink.type_comp_(1,2) are mandatory fields and will always be set to a valid value. Type_comp_(1,2) are indices into the ChemComp list such that the id field (type_comp_(1,2).id) is equal to ChemComp.type.

```
IndexId type_comp_1;
IndexId type_comp_2;
```

ChemCompPlane

Data fields in the ChemCompPlane valuetype provide identifiers for the planes in a chemical component. The atoms in the plane are specified in the ChemCompPlaneAtom valuetype.

The existence of the ChemCompPlane valuetype in an Entry is optional. Its presence can be determined using the S_CHEM_COMP_PLANE flag.

```
valuetype ChemCompPlane
{
  ...
};

typedef sequence<ChemCompPlane> ChemCompPlaneList;
```

ChemCompPlane.id

The value of ChemCompPlane.id must uniquely identify a record in the ChemCompPlane list.

ChemCompPlane.id is a mandatory field and will always be set to a valid value.

```
string id;
```

ChemCompPlane.comp

Comp is a pointer to ChemComp.id in the ChemComp valuetype.

ChemCompPlane.comp is a mandatory field and will always be set to a valid value. Comp is an index into the ChemComp list such that the id field (comp.id) is equal to ChemComp.id.

IndexId comp;

ChemCompPlane.number_atoms_all

The total number of atoms in the plane.

ChemCompPlane.number_atoms_all is an optional field. The flag F_CHEM_COMP_PLANE_NUMBER_ATOMS_ALL can be used to determine if its value has been set.

long number_atoms_all;

ChemCompPlane.number_atoms_nh

The number of non-hydrogen atoms in the plane.

ChemCompPlane.number_atoms_nh is an optional field. The flag F_CHEM_COMP_PLANE_NUMBER_ATOMS_NH can be used to determine if its value has been set.

long number_atoms_nh;

ChemCompPlaneAtom

Data fields in the ChemCompPlaneAtom valuetype enumerate the atoms in a plane within a chemical component.

The existence of the ChemCompPlaneAtom valuetype in an Entry is optional. Its presence can be determined using the S_CHEM_COMP_PLANE_ATOM flag.

```
valuetype ChemCompPlaneAtom
{
  ...
};
```

```
typedef sequence<ChemCompPlaneAtom> ChemCompPlaneAtomList;
```

ChemCompPlaneAtom.atom

The id of an atom involved in the plane.

Atom is a pointer to ChemCompAtom.atom_id in the ChemCompAtom valuetype.

ChemCompPlaneAtom.atom is a mandatory field and will always be set to a valid value. Atom is an index into the ChemCompAtom list such that the id field (atom.id) is equal to ChemCompAtom.atom_id.

IndexId atom;

ChemCompPlaneAtom.comp

Comp is a pointer to ChemComp.id in the ChemComp valuetype.

ChemCompPlaneAtom.comp is a mandatory field and will always be set to a valid value. Comp is an index into the ChemComp list such that the id field (comp.id) is equal to ChemComp.id.

```
IndexId comp;
```

ChemCompPlaneAtom.plane

Plane is a pointer to ChemCompPlane.id in the ChemCompPlane valuetype.

ChemCompPlaneAtom.plane is a mandatory field and will always be set to a valid value. Plane is an index into the ChemCompPlane list such that the id field (plane.id) is equal to ChemCompPlane.id.

```
IndexId plane;
```

ChemCompPlaneAtom.dist_esd

Dist_esd is the standard deviation of the out of plane distance for this atom.

ChemCompPlaneAtom.dist_esd is an optional field. The flag F_CHEM_COMP_PLANE_ATOM_DIST_ESD can be used to determine if its value has been set.

```
float dist_esd;
```

ChemCompTor

Data fields in the ChemCompTor valuetype record details about the torsion angles in a chemical component. As torsion angles can have more than one target value, the target values are specified in the ChemCompTorValue valuetype.

The existence of the ChemCompTor valuetype in an Entry is optional. Its presence can be determined using the S_CHEM_COMP_TOR flag.

```
valuetype ChemCompTor
{
 ...
};

typedef sequence<ChemCompTor> ChemCompTorList;
```

ChemCompTor.atom_id_(1,2,3,4)

The id of the four atoms that define the torsion angle.

Atom_id_(1,2,3,4) are pointers to ChemCompAtom.atom_id in the ChemCompAtom valuetype.

ChemCompTor.atom_id_(1,2,3,4) are mandatory fields and will always be set to a valid value. Atom_id_(1,2,3,4) are indices into the ChemCompAtom list such that the id field (atom_id_(1,2,3,4).id) is equal to ChemCompAtom.atom_id.

```
IndexId atom_id_1;
IndexId atom_id_2;
IndexId atom_id_3;
IndexId atom_id_4;
```

ChemCompTor.id

The value of ChemCompTor.id must uniquely identify a record in the ChemCompTor list.

ChemCompTor.id is a mandatory field and will always be set to a valid value.

```
string id;
```

ChemCompTor.comp

Comp is a pointer to ChemComp.id in the ChemComp valuetype.

ChemCompTor.comp is a mandatory field and will always be set to a valid value. Comp is an index into the ChemComp list such that the id field (comp.id) is equal to ChemComp.id.

```
IndexId comp;
```

ChemCompTorValue

Data fields in the ChemCompTorValue valuetype record details about the target values for the torsion angles enumerated in the ChemCompTor list. Target values may be specified as angles in degrees, as a distance between the first and fourth atoms, or both.

The existence of the ChemCompTorValue valuetype in an Entry is optional. Its presence can be determined using the S_CHEM_COMP_TOR_VALUE flag.

```
valuetype ChemCompTorValue
{
   ...
};

typedef sequence<ChemCompTorValue> ChemCompTorValueList;
```

ChemCompTorValue.comp

Comp is a pointer to ChemCompAtom.comp_id in the ChemCompAtom valuetype.

ChemCompTorValue.comp is a mandatory field and will always be set to a valid value. Comp is an index into the ChemComp list such that the id field (comp.id) is equal to ChemComp.id.

IndexId comp;

ChemCompTorValue.tor

Tor is a pointer to ChemCompTor.id in the ChemCompTor valuetype.

ChemCompTorValue.tor is a mandatory field and will always be set to a valid value. Tor is an index into the ChemCompTor list such that the id field (tor.id) is equal to ChemCompTor.id.

IndexId tor;

ChemCompTorValue.angle

A value that should be taken as a potential target value for the torsion angle associated with the specified atoms, expressed in degrees.

ChemCompTorValue.angle is a mandatory field and will always be set to a valid value.

float angle;

ChemCompTorValue.angle_esd

The estimated standard deviation of ChemCompTorValue.angle.

ChemCompTorValue.angle_esd is a mandatory field and will always be set to a valid value.

float angle_esd;

ChemCompTorValue.dist

A value that should be taken as a potential target value for the torsion angle associated with the specified atoms, expressed as the distance between the atoms specified by ChemCompTor.atom_id_1 and ChemCompTor.atom_id_4 in the referenced record in the ChemCompTor list. Note that the torsion angle cannot be fully specified by a distance (for instance, a torsion angle of -60 will yield the same distance as a 60 degree angle). However the distance specification can be useful for refinement in situations in which the angle is already close to the desired value.

ChemCompTorValue.dist is an optional field. The flag F_CHEM_COMP_TOR_VALUE_DIST can be used to determine if its value has been set.

float dist;

ChemCompTorValue.dist_esd

The estimated standard deviation of ChemCompTorValue.dist_esd.

ChemCompTorValue.dist_esd is an optional field. The flag F_CHEM_COMP_TOR_VALUE_DIST_ESD can be used to determine if its value has been set.

```
float dist_esd;
```

ChemLink

Data fields in the ChemLink valuetype give details about the linkages between chemical groups.

The existence of the ChemLink valuetype in an Entry is optional. Its presence can be determined using the S_CHEM_LINK flag.

```
valuetype ChemLink
{
 ...
};

typedef sequence<ChemLink> ChemLinkList;
```

ChemLink.id

The value of ChemLink.id must uniquely identify each field in the ChemLink list.

ChemLink.id is a mandatory field and will always be set to a valid value.

```
string id;
```

ChemLink.details

A description of special aspects of a linkage between chemical components in the structure.

ChemLink.details is an optional field. The flag F_CHEM_LINK_DETAILS can be used to determine if its value has been set.

```
string details;
```

ChemLinkAngle

Data fields in the ChemLinkAngle valuetype record details about angles in a linkage between chemical groups.

The existence of the ChemLinkAngle valuetype in an Entry is optional. Its presence can be determined using the S_CHEM_LINK_ANGLE flag.

```
valuetype ChemLinkAngle
{
  ...
};

typedef sequence<ChemLinkAngle> ChemLinkAngleList;
```

ChemLinkAngle.atom_(1,2,3)_comp_id

Atom_(1,2,3)_comp_id indicates whether an atom is found in the first or the second of the two component connected by the linkage.

ChemLinkAngle.atom_(1,2,3)_comp_id are optional fields. The flags F_CHEM_LINK_ANGLE_ATOM_(1,2,3)_COMP_ID can be used to determine if its value has been set.

```
string atom_1_comp_id;
string atom_2_comp_id;
string atom_3_comp_id;
```

ChemLinkAngle.atom_id_(1,2,3)

The ids of the three atoms that define the angle.

As these data fields do not point to a specific atom in a specific component, they are not indices in the linkage sense.

ChemLinkAngle.atom_id_1 is a mandatory field and will always be set to a valid value.

```
string atom_id_1;
string atom_id_2;
string atom_id_3;
```

ChemLinkAngle.link

Link is a pointer to ChemLink.id in the ChemLink valuetype.

ChemLinkAngle.link is a mandatory field and will always be set to a valid value. Link is an index into the ChemLink list such that the id field (link.id) is equal to ChemLink.id.

```
IndexId link;
```

ChemLinkAngle.value_angle

The value that should be taken as the target value for the angle associated with the specified atoms, expressed in degrees.

ChemLinkAngle.value_angle is an optional field. The flag F_CHEM_LINK_ANGLE_VALUE_ANGLE can be used to determine if its value has been set.

```
float value_angle;
```

ChemLinkAngle.value_angle_esd

The estimated standard deviation of ChemLinkAngle.value_angle.

ChemLinkAngle.value_angle_esd is an optional field. The flag F_CHEM_LINK_ANGLE_VALUE_ANGLE_ESD can be used to determine if its value has been set.

```
float value_angle_esd;
```

ChemLinkAngle.value_dist

The value that should be taken as the target value for the angle associated with the specified atoms, expressed as the distance between the atoms specified by ChemCompAngle.atom_id_1 and ChemCompAngle.atom_id_3.

ChemLinkAngle.value_dist is an optional field. The flag F_CHEM_LINK_ANGLE_VALUE_DIST can be used to determine if its value has been set.

```
float value_dist;
```

ChemLinkAngle.value_dist_esd

The estimated standard deviation of ChemCompAngle.value_dist_esd.

ChemLinkAngle.value_dist_esd is an optional field. The flag F_CHEM_LINK_ANGLE_VALUE_DIST_ESD can be used to determine if its value has been set.

```
float value_dist_esd;
```

ChemLinkBond

Data fields in the ChemLinkBond valuetype record details about bonds in a linkage between components in the chemical structure.

The existence of the ChemLinkBond valuetype in an Entry is optional. Its presence can be determined using the S_CHEM_LINK_BOND flag.

```
valuetype ChemLinkBond
{
 ...
};

typedef sequence<ChemLinkBond> ChemLinkBondList;
```

ChemLinkBond.atom_(1,2)_comp_id

Atom_(1,2)_comp_id indicates whether an atom is found in the first or the second of the two components connected by the linkage.

ChemLinkBond.atom_(1,2)_comp_id are optional fields. The flags F_CHEM_LINK_BOND_ATOM_(1,2)_COMP_ID can be used to determine if its value has been set.

```
string atom_1_comp_id;
string atom_2_comp_id;
```

ChemLinkBond.atom_id_(1,2)

The ids the two atoms that define the bond. As these data fields do not point to a specific atom in a specific chemical component, they are not indices in the linkage sense.

ChemLinkBond.atom_id_(1,2) are mandatory fields and will always be set to a valid value.

```
string atom_id_1;
string atom_id_2;
```

ChemLinkBond.link

Link is a pointer to ChemLink.id in the ChemLink valuetype.

ChemLinkBond.link is a mandatory field and will always be set to a valid value. Link is an index into the ChemLink list such that the id field (link.id) is equal to ChemLink.id.

```
IndexId link;
```

ChemLinkBond.value_dist

The value that should be taken as the target for the chemical bond associated with the specified atoms, expressed as a distance.

ChemLinkBond.value_dist is an optional field. The flag F_CHEM_LINK_BOND_VALUE_DIST can be used to determine if its value has been set.

```
float value_dist;
```

ChemLinkBond.value_dist_esd

The estimated standard deviation of ChemLinkBond.value_dist_esd.

ChemLinkBond.value_dist_esd is an optional field. The flag F_CHEM_LINK_BOND_VALUE_DIST_ESD can be used to determine if its value has been set.

```
float value_dist_esd;
```

ChemLinkBond.value_order

The value that should be taken as the target for the chemical bond associated with the specified atoms, expressed as a bond order.

ChemLinkBond.value_order is an optional field. The flag F_CHEM_LINK_BOND_VALUE_ORDER can be used to determine if its value has been set.

```
string value_order;
```

ChemLinkChir

Data fields in the ChemLinkChir valuetype provide detail about the chiral centers in a linkage between two chemical components. The atoms bonded to the chiral atom are specified in the ChemLinkChirAtom valuetype.

The existence of the ChemLinkChir valuetype in an Entry is optional. Its presence can be determined using the S_CHEM_LINK_CHIR flag.

```
valuetype ChemLinkChir
{
  ...
};

typedef sequence<ChemLinkChir> ChemLinkChirList;
```

ChemLinkChir.atom_comp_id

Atom_comp_id indicates whether the chiral atom is found in the first or the second of the two component connected by the linkage.

ChemLinkChir.atom_comp_id is an optional field. The flag F_CHEM_LINK_CHIR_ATOM_COMP_ID can be used to determine if its value has been set.

```
string atom_comp_id;
```

ChemLinkChir.atom_id

The id of the atom that is a chiral center.

As this data field does not point to a specific atom in a specific chemical component, it is not a child in the linkage sense.

ChemLinkChir.atom_id is a mandatory field and will always be set to a valid value.

```
string atom_id;
```

ChemLinkChir.atom_config

The chiral configuration of the atom that is a chiral center.

ChemLinkChir.atom_config is an optional field. The flag F_CHEM_LINK_CHIR_ATOM_CONFIG can be used to determine if its value has been set.

string atom_config;

ChemLinkChir.id

The value of ChemLinkChir.id must uniquely identify a record in the ChemLinkChir list.

ChemLinkChir.id is a mandatory field and will always be set to a valid value.

string id;

ChemLinkChir.link

Link is a pointer to ChemLink.id in the ChemLink valuetype.

ChemLinkChir.link is a mandatory field and will always be set to a valid value.

link is an index into the ChemLink list such that the id field (link.id) is equal to ChemLink.id.

IndexId link;

ChemLinkChir.number_atoms_all

The total number of atoms bonded to the atom specified by ChemLinkChir.atom_id.

ChemLinkChir.number_atoms_all is an optional field. The flag F_CHEM_LINK_CHIR_NUMBER_ATOMS_ALL can be used to determine if its value has been set.

long number_atoms_all;

ChemLinkChir.number_atoms_nh

The number of non-hydrogen atoms bonded to the atom specified by ChemLinkChir.atom_id.

ChemLinkChir.number_atoms_nh is an optional field. The flag F_CHEM_LINK_CHIR_NUMBER_ATOMS_NH can be used to determine if its value has been set.

long number_atoms_nh;

ChemLinkChir.volume_flag

A flag to indicate whether a chiral volume should match the standard value in both magnitude and sign, or in magnitude only.

ChemLinkChir.volume_flag is an optional field. The flag F_CHEM_LINK_CHIR_VOLUME_FLAG can be used to determine if its value has been set.

string volume_flag;

ChemLinkChir.volume_three

The chiral volume V_c for chiral centers that involve a chiral atom bonded to three non-hydrogen atoms and one hydrogen atom.

$$V_c = V_1 \bullet (V_2 \times V_3)$$

Where:

$\bullet$ = the vector dot product

$\times$ = the vector cross product

V1 = the vector distance from the atom specified by ChemLinkChir.atom.id to the first atom in the ChemCompChirAtom list

V2 = the vector distance from the atom specified by ChemLinkChir.atom.id to the second atom in the ChemCompChirAtom list

V3 = the vector distance from the atom specified by ChemLinkChir.atom.id to the third atom in the ChemCompChirAtom list

ChemLinkChir.volume_three is an optional field. The flag F_CHEM_LINK_CHIR_VOLUME_THREE can be used to determine if its value has been set.

float volume_three;

ChemLinkChir.volume_three_esd

The estimated standard deviation of ChemLinkChir.volume_three.

ChemLinkChir.volume_three_esd is an optional field. The flag F_CHEM_LINK_CHIR_VOLUME_THREE_ESD can be used to determine if its value has been set.

float volume_three_esd;

ChemLinkChirAtom

Data fields in the ChemLinkChirAtom valuetype enumerate the atoms bonded to a chiral atom in a linkage between two chemical components.

The existence of the ChemLinkChirAtom valuetype in an Entry is optional. Its presence can be determined using the S__CHEM_LINK_CHIR_ATOM flag.

```
valuetype ChemLinkChirAtom
{
  ...
};

typedef sequence<ChemLinkChirAtom> ChemLinkChirAtomList;
```

ChemLinkChirAtom.atom_comp_id

Atom_comp_id indicates whether the atom bonded to a chiral atom is found in the first or the second of the two components connected by the linkage.

ChemLinkChirAtom.atom_comp_id is an optional field. The flag F_CHEM_LINK_CHIR_ATOM_ATOM_COMP_ID can be used to determine if its value has been set.

```
string atom_comp_id;
```

ChemLinkChirAtom.atom_id

The id of an atom bonded to the chiral atom.

As this data field does not point to a specific atom in a specific chemical component, it is not an index in the linkage sense.

ChemLinkChirAtom.atom_id is a mandatory field and will always be set to a valid value.

```
string atom_id;
```

ChemLinkChirAtom.chir

Chir is a pointer to ChemLinkChir.id in the ChemLinkChir valuetype.

ChemLinkChirAtom.chir is a mandatory field and will always be set to a valid value. Chir is an index into the ChemLinkChir list such that the id field (chir.id) is equal to ChemLinkChir.id.

```
IndexId chir;
```

ChemLinkChirAtom.dev

The estimated standard deviation of the position of this atom from the plane defined by all of the atoms in the plane.

ChemLinkChirAtom.dev is an optional field. The flag F_CHEM_LINK_CHIR_ATOM_DEV can be used to determine if its value has been set.

```
float dev;
```

ChemLinkPlane

Data fields in the ChemLinkPlane valuetype provide identifiers for the planes in a linkage between two chemical components. The atoms in the plane are specified in the ChemLinkPlaneAtom valuetype.

The existence of the ChemLinkPlane valuetype in an Entry is optional. Its presence can be determined using the S_CHEM_LINK_PLANE flag.

```
valuetype ChemLinkPlane
{
  ...
};

typedef sequence<ChemLinkPlane> ChemLinkPlaneList;
```

ChemLinkPlane.id

The value of ChemLinkPlane.id must uniquely identify a record in the ChemLinkPlane list.

ChemLinkPlane.id is a mandatory field and will always be set to a valid value.

```
string id;
```

ChemLinkPlane.link

Link is a pointer to ChemLink.id in the ChemLink valuetype.

ChemLinkPlane.link is a mandatory field and will always be set to a valid value. Llink is an index into the ChemLink list such that the id field (link.id) is equal to ChemLink.id.

```
IndexId link;
```

ChemLinkPlane.number_atoms_all

The total number of atoms in the plane.

ChemLinkPlane.number_atoms_all is an optional field. The flag F_CHEM_LINK_PLANE_NUMBER_ATOMS_ALL can be used to determine if its value has been set.

```
long number_atoms_all;
```

ChemLinkPlane.number_atoms_nh

The number of non-hydrogen atoms in the plane.

ChemLinkPlane.number_atoms_nh is an optional field. The flag F_CHEM_LINK_PLANE_NUMBER_ATOMS_NH can be used to determine if its value has been set.

```
long number_atoms_nh;
```

ChemLinkPlaneAtom

Data fields in the ChemLinkPlaneAtom valuetype enumerate the atoms in a plane in a linkage between two chemical components.

The existence of the ChemLinkPlaneAtom valuetype in an Entry is optional. Its presence can be determined using the S_CHEM_LINK_PLANE_ATOM flag.

```
valuetype ChemLinkPlaneAtom
{
  ...
};

typedef sequence<ChemLinkPlaneAtom> ChemLinkPlaneAtomList;
```

ChemLinkPlaneAtom.atom_comp_id

Atom_comp_id indicates whether the atom in a plane is found in the first or the second of the two components connected by the linkage.

ChemLinkPlaneAtom.atom_comp_id is an optional field. The flag F_CHEM_LINK_PLANE_ATOM_ATOM_COMP_ID can be used to determine if its value has been set.

```
string atom_comp_id;
```

ChemLinkPlaneAtom.atom_id

The id of an atom involved in the plane.

As this data field does not point to a specific atom in a specific chemical component, it is not an index in the linkage sense.

ChemLinkPlaneAtom.atom_id is a mandatory field and will always be set to a valid value.

```
string atom_id;
```

ChemLinkPlaneAtom.plane

Plane is a pointer to ChemLinkPlane.id in the ChemLinkPlane valuetype.

ChemLinkPlaneAtom.plane is a mandatory field and will always be set to a valid value. Plane is an index into the ChemLinkPlane list such that the id field (plane.id) is equal to ChemLinkPlane.id.

```
IndexId plane;
```

ChemLinkTor

Data fields in the ChemLinkTor valuetype record details about the torsion angles in a linkage between two chemical components. As torsion angles can have more than one target value, the target values are specified in the ChemLinkTorValue valuetype.

The existence of the ChemLinkTor valuetype in an Entry is optional. Its presence can be determined using the S__CHEM_LINK_TOR flag.

```
valuetype ChemLinkTor
{
  ...
};

typedef sequence<ChemLinkTor> ChemLinkTorList;
```

ChemLinkTor.atom_(1,2,3,4)_comp_id

Atom_(1,2,3,4)_comp_id indicates whether an atom is found in the first or the second of the two components connected by the linkage.

ChemLinkTor.atom_(1,2,3,4)_comp_id are optional fields. The flag F_CHEM_LINK_TOR_ATOM_(1,2,3,4)_COMP_ID can be used to determine if their value has been set.

```
string atom_1_comp_id;
string atom_2_comp_id;
string atom_3_comp_id;
string atom_4_comp_id;
```

ChemLinkTor.atom_id_(1,2,3,4)

The ids of the four atoms that define the torsion angle.

As these data fields do not point to a specific atom in a specific chemical component, they are not indices in the linkage sense.

ChemLinkTor.atom_id_(1,2,3,4) is a mandatory field and will always be set to a valid value.

```
string atom_id_1;
string atom_id_2;
string atom_id_3;
string atom_id_4;
```

ChemLinkTor.id

The value of ChemLinkTor.id must uniquely identify a record in the ChemLinkTor list.

ChemLinkTor.id is a mandatory field and will always be set to a valid value.

```
string id;
```

ChemLinkTor.link

Link is a pointer to ChemLink.id in the ChemLink valuetype.

ChemLinkTor.link is a mandatory field and will always be set to a valid value. Link is an index into the ChemLink list such that the id field (link.id) is equal to ChemLink.id.

```
IndexId link;
```

ChemLinkTorValue

Data fields in the ChemLinkTorValue valuetype record details about the target values for the torsion angles enumerated in the ChemLinkTor list. Target values may be specified as angles in degrees, as a distance between the first and fourth atoms, or both.

The existence of the ChemLinkTorValue valuetype in an Entry is optional. Its presence can be determined using the S_CHEM_LINK_TOR_VALUE flag.

```
valuetype ChemLinkTorValue
{
  ...
};

typedef sequence<ChemLinkTorValue> ChemLinkTorValueList;
```

ChemLinkTorValue.tor

Tor is a pointer to ChemLinkTor.id in the ChemLinkTor valuetype.

ChemLinkTorValue.tor is a mandatory field and will always be set to a valid value. Tor is an index into the ChemLinkTor list such that the id field (tor.id) is equal to ChemLinkTor.id.

```
IndexId tor;
```

ChemLinkTorValue.angle

A value that should be taken as a potential target value for the torsion angle associated with the specified atoms, expressed in degrees.

ChemLinkTorValue.angle is a mandatory field and will always be set to a valid value.

```
float angle;
```

ChemLinkTorValue.angle_esd

The estimated standard deviation of ChemLinkTorValue.angle.

ChemLinkTorValue.angle_esd is a mandatory field and will always be set to a valid value.

float angle_esd;

ChemLinkTorValue.dist

A value that should be taken as a potential target value for the torsion angle associated with the specified atoms, expressed as the distance between the atoms specified by ChemLinkTor.atom_id_1 and ChemLinkTor.atom_id_4 in the referenced record in the ChemLinkTor list. Note that the torsion angle cannot be fully specified by a distance (for instance, a torsion angle of -60 will yield the same distance as a 60 degree angle). However the distance specification can be useful for refinement in situations in which the angle is already close to the desired value.

ChemLinkTorValue.dist is an optional field. The flag F_CHEM_LINK_TOR_VALUE_DIST can be used to determine if its value has been set.

float dist;

ChemLinkTorValue.dist_esd

The estimated standard deviation of ChemLinkTorValue.dist_esd.

ChemLinkTorValue.dist_esd is an optional field. The flag F_CHEM_LINK_TOR_VALUE_DIST_ESD can be used to determine if its value has been set.

float dist_esd;

Entity

Data fields in the Entity valuetype record details (such as chemical composition, name, and source) about the molecular entities that are present in the structure. Fields in the various Entity valuetypes provide a full chemical description of these molecular entities.

Entities are of three types: polymer, non-polymer and water. Note that the water type includes only water; ordered solvent such as sulfate ion or acetone would be described as individual non-polymer entities.

Entity data are not the result of an experiment; those results are represented in the AtomSite data fields. Entity data fields describe the chemistry of the molecules under investigation, and can most usefully be though of as the ideal groups to which the structure is restrained or constrained during refinement.

Entities do not correspond directly to the enumeration of the contents of the asymmetric unit. Entities are described only once, even in those structures that contain multiple observations of an entity. The StructAsym data fields, which reference the entity list, describe and label the contents of the asymmetric unit.

The existence of the Entity valuetype in an Entry is optional. Its presence can be determined using the S_ENTITY flag.

```
valuetype Entity
{
  ...
};

typedef sequence<Entity> EntityList;
```

Entity.details

A description of special aspects of the entity.

Entity.details is an optional field. The flag F_ENTITY_DETAILS can be used to determine if its value has been set.

```
string details;
```

Entity.formula_weight

Formula mass in daltons of the entity.

Entity.formula_weight is an optional field. The flag F_ENTITY_FORMULA_WEIGHT can be used to determine if its value has been set.

```
float formula_weight;
```

Entity.id

The value of Entity.id must uniquely identify a record in the Entity list. Note that this field need not be a number; it can be any unique identifier.

Entity.id is a mandatory field and will always be set to a valid value.

```
string id;
```

Entity.src_method

The method by which the sample for the entity was produced. Entities isolated directly from natural sources (tissues, soil samples, etc.) are expected to have further information in the EntitySrcNat valuetype. Entities isolated from genetically manipulated sources are expected to have further information in the EntitySrcGen valuetype.

Entity.src_method is an optional field. The flag F_ENTITY_SRC_METHOD can be used to determine if its value has been set.

```
string src_method;
```

Entity.type

Defines the type of the entity.

Polymer entities are expected to have corresponding EntityPoly and associated entries. Non-polymer entities are expected to have corresponding ChemComp and associated entries. Water entities are not expected to have corresponding entries in the Entity valuetype.

Entity.type is an optional field. The flag F_ENTITY_TYPE can be used to determine if its value has been set.

```
string type;
```

Data fields in the EntityKeywords valuetype specify keywords relevant to the molecular entities. Note that this list of keywords is separate from the list that is used to keyword the StructBiol data fields, and is intended to provide only the information that one would know about the molecular entity *if one did not know its structure*. Hence polypeptides are simply polypeptides, and not cytokines or beta-alpha-barrels, and polyribonucleic acids are simply poly-RNA, and not transfer- Rna.

The existence of the EntityKeywords valuetype in an Entry is optional. Its presence can be determined using the S_ENTITY_KEYWORDS flag.

```
valuetype EntityKeywords
{
  ...
};

typedef sequence<EntityKeywords> EntityKeywordsList;
```

EntityKeywords.entity

Entity is a pointer to Entity.id in the Entity valuetype.

EntityKeywords.entity is a mandatory field and will always be set to a valid value. Entity is an index into the Entity list such that the id field (entity.id) is equal to Entity.id.

```
IndexId entity;
```

EntityKeywords.text

Keywords describing this entity.

EntityKeywords.text is a mandatory field and will always be set to a valid value.

```
string text;
```

EntityLink

Data fields in the EntityLink valuetype give details about the linkages between entities.

The existence of the EntityLink valuetype in an Entry is optional. Its presence can be determined using the S_ENTITY_LINK flag.

```
valuetype EntityLink
{
  ...
};

typedef sequence<EntityLink> EntityLinkList;
```

EntityLink.link

Link is a pointer to ChemLink.id in the ChemLink valuetype.

EntityLink.link is a mandatory field and will always be set to a valid value. Link is an index into the ChemLink list such that the id field (link.id) is equal to ChemLink.id.

```
IndexId link;
```

EntityLink.details

A description of special aspects of a linkage between chemical components in the structure.

EntityLink.details is an optional field. The flag F_ENTITY_LINK_DETAILS can be used to determine if its value has been set.

```
string details;
```

EntityLink.entity_id_(1,2)

The entity ids of the two entities joined by the linkage.

EntityLink.entity_id_(1,2) are mandatory fields and will always be set to a valid value. Entity_id_(1,2) are indices into the Entity list such that the id field (entity_id_(1,2).id) is equal to Entity.id.

```
IndexId entity_id_1;
IndexId entity_id_2;
```

EntityLink.entity_seq_num_(1,2)

For a polymer entity, the sequence numbers in the two entities containing the linkage.

EntityLink.entity_seq_num_(1,2) are optionals fields. The flags F_ENTITY_LINK_ENTITY_SEQ_NUM_(1,2) can be used to determine if their value has been set. Entity_seq_num_(1,2) are indices into the EntityPolySeq list such that the id field (entity_seq_num_(1,2).id) is equal to EntityPolySeq.num.

```
IndexId entity_seq_num_1;
IndexId entity_seq_num_2;
```

EntityNameCom

Data fields in the EntityNameCom valuetype record the common name or names associated with the entity. In some case, the entity name may not be the same as the name of the biological structure. For instance, hemoglobin alpha chain would be the entity common name, not hemoglobin.

The existence of the EntityNameCom valuetype in an Entry is optional. Its presence can be determined using the S_ENTITY_NAME_COM flag.

```
valuetype EntityNameCom
{
  ...
};

typedef sequence<EntityNameCom> EntityNameComList;
```

EntityNameCom.entity

Entity is a pointer to Entity.id in the Entity valuetype.

EntityNameCom.entity is a mandatory field and will always be set to a valid value. entity is an index into the Entity list such that the id field (entity.id) is equal to Entity.id.

```
IndexId entity;
```

EntityNameCom.name

A common name for the entity.

EntityNameCom.name is a mandatory field and will always be set to a valid value.

```
string name;
```

EntityNameSys

Data fields in the EntityNameSys valuetype record the systematic name or names associated with the entity, and tell which system was the source of the systematic name. In some case, the entity name may not be the same as the name of the biological structure. For instance, hemoglobin alpha chain would be the entity common name, not hemoglobin.

The existence of the EntityNameSys valuetype in an Entry is optional. Its presence can be determined using the S_ENTITY_NAME_SYS flag.

```
valuetype EntityNameSys
{
  ...
};
```

```
typedef sequence<EntityNameSys> EntityNameSysList;
```

EntityNameSys.entity

Entity is a pointer to Entity.id in the Entity valuetype.

EntityNameSys.entity is a mandatory field and will always be set to a valid value. Entity is an index into the Entity list such that the id field (entity.id) is equal to Entity.id.

```
IndexId entity;
```

EntityNameSys.name

The systematic name for the entity.

EntityNameSys.name is a mandatory field and will always be set to a valid value.

```
string name;
```

EntityNameSys.system

The system used to generate the systematic name of the entity.

EntityNameSys.system is an optional field. The flag F_ENTITY_NAME_SYS_SYSTEM can be used to determine if its value has been set.

```
string system;
```

EntityPoly

Data fields in the EntityPoly valuetype record characteristics of the polymer.

The existence of the EntityPoly valuetype in an Entry is optional. Its presence can be determined using the S_ENTITY_POLY flag.

```
valuetype EntityPoly
{
  ...
};
```

```
typedef sequence<EntityPoly> EntityPolyList;
```

EntityPoly.entity

Entity is a pointer to Entity.id in the Entity valuetype.

EntityPoly.entity is a mandatory field and will always be set to a valid value. Entity is an index into the Entity list such that the id field (entity.id) is equal to Entity.id.

IndexId entity;

EntityPoly.nstd_chirality

A flag to indicate whether or not the polymer contains at least one monomer unit with chirality different from that specified in EntityPoly.type.

EntityPoly.nstd_chirality is an optional field. The flag F_ENTITY_POLY_NSTD_CHIRALITY can be used to determine if its value has been set.

string nstd_chirality;

EntityPoly.nstd_linkage

A flag to indicate whether or not the polymer contains at least one monomer-to-monomer linkage different from that implied by EntityPoly.type.

EntityPoly.nstd_linkage is an optional field. The flag F_ENTITY_POLY_NSTD_LINKAGE can be used to determine if its value has been set.

string nstd_linkage;

EntityPoly.nstd_monomer

A flag to indicate whether or not the polymer contains at least one monomer that is not considered standard.

EntityPoly.nstd_monomer is an optional field. The flag F_ENTITY_POLY_NSTD_MONOMER can be used to determine if its value has been set.

string nstd_monomer;

EntityPoly.number_of_monomer

The number of monomers in the polymer.

EntityPoly.number_of_monomers is an optional field. The flag F_ENTITY_POLY_NUMBER_OF_MONOMERS can be used to determine if its value has been set.

long number_of_monomers;

EntityPoly.type

The type of the polymer.

EntityPoly.type is an optional field. The flag F_ENTITY_POLY_TYPE can be used to determine if its value has been set.

```
string type;
```

EntityPoly.type_details

A description of special aspects of the polymer type.

EntityPoly.type_details is an optional field. The flag F_ENTITY_POLY_TYPE_DETAILS can be used to determine if its value has been set.

```
string type_details;
```

EntityPolySeq

Data fields in the EntityPolySeq struct specify the sequence of monomers in a polymer. Allowance is made for the possibility of microheterogeneity in a sample by allowing a given sequence number to be correlated with more than one monomer id - the corresponding AtomSite entries should reflect this heterogeneity.

The existence of the EntityPolySeq valuetype in an Entry is optional. Its presence can be determined using the S_ENTITY_POLY_SEQ flag.

```
struct EntityPolySeq
{
  ...
};

typedef sequence<EntityPolySeq> EntityPolySeqList;
```

EntityPolySeq.entity

entity is a pointer to Entity.id in the Entity valuetype.

EntityPolySeq.entity is a mandatory field and will always be set to a valid value. Entity is an index into the Entity list such that the id field (entity.id) is equal to Entity.id.

```
IndexId entity;
```

EntityPolySeq.hetero

A flag to indicate whether or not this monomer in the polymer is heterogeneous in sequence. This would be a rare phenomenon.

EntityPolySeq.hetero is an optional field. The flag F_ENTITY_POLY_SEQ_HETERO can be used to determine if its value has been set.

```
string hetero;
```

EntityPolySeq.mon

Mon is a pointer to ChemComp.id in the ChemComp valuetype.

EntityPolySeq.mon is a mandatory field and will always be set to a valid value. Mon is an index into the ChemComp list such that the id field (mon.id) is equal to ChemComp.id.

```
IndexId mon;
```

EntityPolySeq.num

The value of EntityPolySeq.num must uniquely and sequentially identify a record in the EntityPolySeq list.

Note that this field must be a number, and that the sequence numbers must progress in increasing numerical order.

EntityPolySeq.num is a mandatory field and will always be set to a valid value.

```
long num;
```

EntitySrcGen

Data fields in the EntitySrcGen valuetype records details of the source from which the entity was obtained, in those cases where the source was a genetically manipulated one. The following are treated separately: Fields pertaining to the tissue from which the gene was obtained, fields pertaining to the host organism for gene expression and fields pertaining to the actual producting organism (plasmid).

The existence of the EntitySrcGen valuetype in an Entry is optional. Its presence can be determined using the S_ENTITY_SRC_GEN flag.

```
valuetype EntitySrcGen
{
  ...
};

typedef sequence<EntitySrcGen> EntitySrcGenList;
```

EntitySrcGen.entity

Entity is a pointer to Entity.id in the Entity valuetype.

EntitySrcGen.entity is a mandatory field and will always be set to a valid value. Entity is an index into the Entity list such that the id field (entity.id) is equal to Entity.id.

```
IndexId entity;
```

EntitySrcGen.gene_src_common_name

The common name of the natural organism from which the gene was obtained.

EntitySrcGen.gene_src_common_name is an optional field. The flag F_ENTITY_SRC_GEN_GENE_SRC_COMMON_NAME can be used to determine if its value has been set.

```
string gene_src_common_name;
```

EntitySrcGen.gene_src_details

A description of special aspects of the natural organism from which the gene was obtained.

EntitySrcGen.gene_src_details is an optional field. The flag F_ENTITY_SRC_GEN_GENE_SRC_DETAILS can be used to determine if its value has been set.

```
string gene_src_details;
```

EntitySrcGen.gene_src_genus

The genus of the natural organism from which the gene was obtained.

EntitySrcGen.gene_src_genus is an optional field. The flag F_ENTITY_SRC_GEN_GENE_SRC_GENUS can be used to determine if its value has been set.

```
string gene_src_genus;
```

EntitySrcGen.gene_src_species

The species of the natural organism from which the gene was obtained.

EntitySrcGen.gene_src_species is an optional field. The flag F_ENTITY_SRC_GEN_GENE_SRC_SPECIES can be used to determine if its value has been set.

```
string gene_src_species;
```

EntitySrcGen.gene_src_strain

The strain of the natural organism from which the gene was obtained, if relevant.

EntitySrcGen.gene_src_strain is an optional field. The flag F_ENTITY_SRC_GEN_GENE_SRC_STRAIN can be used to determine if its value has been set.

string gene_src_strain;

EntitySrcGen.gene_src_tissue

The tissue of the natural organism from which the gene was obtained.

EntitySrcGen.gene_src_tissue is an optional field. The flag F_ENTITY_SRC_GEN_GENE_SRC_TISSUE can be used to determine if its value has been set.

string gene_src_tissue;

EntitySrcGen.gene_src_tissue_fraction

The sub-cellular fraction of the tissue of the natural organism from which the gene was obtained.

EntitySrcGen.gene_src_tissue_fraction is an optional field. The flag F_ENTITY_SRC_GEN_GENE_SRC_TISSUE_FRACTION can be used to determine if its value has been set.

string gene_src_tissue_fraction;

EntitySrcGen.host_org_common_name

The common name of the organism that served as host for the production of the entity.

EntitySrcGen.host_org_common_name is an optional field. The flag F_ENTITY_SRC_GEN_HOST_ORG_COMMON_NAME can be used to determine if its value has been set.

string host_org_common_name;

EntitySrcGen.host_org_details

A description of special aspects of the organism that served as host for the production of the entity.

EntitySrcGen.host_org_details is an optional field. The flag F_ENTITY_SRC_GEN_HOST_ORG_DETAILS can be used to determine if its value has been set.

string host_org_details;

EntitySrcGen.host_org_genus

The genus of the organism that served as host for the production of the entity.

EntitySrcGen.host_org_genus is an optional field. The flag F_ENTITY_SRC_GEN_HOST_ORG_GENUS can be used to determine if its value has been set.

string host_org_genus;

EntitySrcGen.host_org_species

The species of the organism that served as host for the production of the entity.

EntitySrcGen.host_org_species is an optional field. The flag F_ENTITY_SRC_GEN_HOST_ORG_SPECIES can be used to determine if its value has been set.

string host_org_species;

EntitySrcGen.host_org_strain

The strain of the organism that served as host for the production of the entity.

EntitySrcGen.host_org_strain is an optional field. The flag F_ENTITY_SRC_GEN_HOST_ORG_STRAIN can be used to determine if its value has been set.

string host_org_strain;

EntitySrcGen.plasmid_details

A description of special aspects of the plasmid that produced the entity in the host organism.

EntitySrcGen.plasmid_details is an optional field. The flag F_ENTITY_SRC_GEN_PLASMID_DETAILS can be used to determine if its value has been set.

string plasmid_details;

EntitySrcGen.plasmid_name

The name of the plasmid that produced the entity in the host organism.

EntitySrcGen.plasmid_name is an optional field. The flag F_ENTITY_SRC_GEN_PLASMID_NAME can be used to determine if its value has been set.

string plasmid_name;

EntitySrcNat

Data fields in the EntitySrcNat valuetype records details of the source from which the entity was obtained, in those cases where the entity was isolated directly from a natural tissue.

The existence of the EntitySrcNat valuetype in an Entry is optional. Its presence can be determined using the S_ENTITY_SRC_NAT flag.

```
valuetype EntitySrcNat
{
 ...
};

typedef sequence<EntitySrcNat> EntitySrcNatList;
```

EntitySrcNat.common_name

The genus of the organism from which the entity was isolated.

EntitySrcNat.common_name is a mandatory field and will always be set to a valid value.

```
string common_name;
```

EntitySrcNat.details

A description of special aspects of the organism from which the entity was isolated.

EntitySrcNat.details is an optional field. The flag F_ENTITY_SRC_NAT_DETAILS can be used to determine if its value has been set.

```
string details;
```

EntitySrcNat.entity

Entity is a pointer to Entity.id in the Entity valuetype.

EntitySrcNat.entity is a mandatory field and will always be set to a valid value. Entity is an index into the Entity list such that the id field (entity.id) is equal to Entity.id.

```
IndexId entity;
```

EntitySrcNat.genus

The genus of the organism from which the entity was isolated.

EntitySrcNat.genus is a mandatory field and will always be set to a valid value.

```
string genus;
```

EntitySrcNat.species

The species of the organism from which the entity was isolated.

EntitySrcNat.species is a mandatory field and will always be set to a valid value.

```
string species;
```

EntitySrcNat.strain

The strain of the organism from which the entity was isolated.

EntitySrcNat.strain is a mandatory field and will always be set to a valid value.

```
string strain;
```

EntitySrcNat.tissue

The tissue of the organism from which the entity was isolated.

EntitySrcNat.tissue is a mandatory field and will always be set to a valid value.

```
string tissue;
```

EntitySrcNat.tissue_fraction

The sub-cellular fraction of the tissue of the organism from which the entity was isolated.

EntitySrcNat.tissue_fraction is a mandatory field and will always be set to a valid value.

```
string tissue_fraction;
```

EntryLink

Data fields in the EntryLink valuetype record the relationships between the current entry identified by Entry.id and other entries.

The existence of the EntryLink valuetype in an Entry is optional. Its presence can be determined using the S_ENTRY_LINK flag.

```
valuetype EntryLink
{
  ...
};

typedef sequence<EntryLink> EntryLinkList;
```

EntryLink.entry_id

Entry_id is a pointer to another entry.

EntryLink.entry_id is a mandatory field and will always be set to a valid value.

```
EntryId entry_id;
```

EntryLink.id

The value of EntryLink.id identifies an entry related the entry identified by EntryLink.entry_id.

EntryLink.id is a mandatory field and will always be set to a valid value.

```
string id;
```

EntryLink.details

The description of the relationship between the entries identified by EntryLink.id and EntryLink.entry_id.

EntryLink.details is an optional field. The flag F_ENTRY_LINK_DETAILS can be used to determine if its value has been set.

```
string details;
```

Geom

Data fields in the Geom and related (GeomAngle, GeomBond, GeomContact, GeomHbond and GeomTorsion) structures record details about the molecular geometry, as calculated from the contents of the atom, cell, and symmetry data.

The existence of the Geom valuetype in an Entry is optional. Its presence can be determined using the S_GEOM flag.

```
valuetype Geom
{
  ...
};

typedef sequence<Geom> GeomList;
```

Geom.entry_id

Entry_id is a pointer to Entry.id in the Entry valuetype.

Geom.entry_id is a mandatory field and will always be set to a valid value.

```
EntryId entry_id;
```

Geom.details

The description of geometrical information not covered by the existing data names in the Geom valuetype, such as least-squares planes.

Geom.details is an optional field. The flag F_GEOM_DETAILS can be used to determine if its value has been set.

```
string details;
```

GeomAngle

Data fields in the GeomAngle valuetype record details about the molecular angles, as calculated from the atom and symmetry data.

The existence of the GeomAngle valuetype in an Entry is optional. Its presence can be determined using the S_GEOM_ANGLE flag.

```
valuetype GeomAngle
{
  ...
};

typedef sequence<GeomAngle> GeomAngleList;
```

GeomAngle.atom_site_id_(1,2,3)

The identifiers of the three atom sites that define the angle specified by GeomAngle.value.

GeomAngle.atom_site_id_(1,2,3) are mandatory fields and will always be set to a valid values. Atom_site_id_(1,2,3) are indices into the AtomSite list such that the id field (atom_site_id_(1,2,3).id) is equal to AtomSite.id.

```
IndexId atom_site_id_1;
IndexId atom_site_id_2;
IndexId atom_site_id_3;
```

GeomAngle.atom_site_label_(1,2,3)

An optional identifier of the three atom sites that define the angle specified by GeomAngle.value.

GeomAngle.atom_site_label_(1,2,3).atom are optional fields. The flags F_GEOM_ANGLE_ATOM_SITE_LABEL_(1,2,3)_ATOM_ID can be used to determine if their value has been set. Atom_site_label_(1,2,3).atom is an index into the ChemCompAtom list such that the id field (atom_site_label_(1,2,3).atom.id) is equal to ChemCompAtom.id.

GeomAngle.atom_site_label_(1,2,3).comp are optional fields. The flags F_GEOM_ANGLE_ATOM_SITE_LABEL_(1,2,3)_COMP_ID can be used to determine if their value has been set. Atom_site_label_(1,2,3).comp is an index into the ChemComp list such that the id field (atom_site_label_(1,2,3).comp.id) is equal to ChemComp.id.

GeomAngle.atom_site_label_(1,2,3).seq are optional fields. The flags F_GEOM_ANGLE_ATOM_SITE_LABEL_(1,2,3)_SEQ_ID can be used to determine if their value has been set. Atom_site_label_(1,2,3).seq is an index into the EntityPolySeq list such that the id field (atom_site_label_(1,2,3).seq.id) is equal to EntityPolySeq.num.

GeomAngle.atom_site_label_(1,2,3).asym are optional fields. The flags F_GEOM_ANGLE_ATOM_SITE_LABEL_(1,2,3)_ASYM_ID can be used to determine if their value has been set. Atom_site_label_(1,2,3).asym is an index into the StructAsym list such that the id field (atom_site_label_(1,2,3).asym.id) is equal to StructAsym.id.

GeomAngle.atom_site_label_(1,2,3).alt is an optional field. The flags F_GEOM_ANGLE_ATOM_SITE_LABEL_(1,2,3)_ALT_ID can be used to determine if their value has been set. Atom_site_label_(1,2,3).alt is an index into the AtomSite list such that the id field (atom_site.label_(1,2,3).alt.id) is equal to AtomSite.id.

```
AtomIndex atom_site_label_1;
AtomIndex atom_site_label_2;
AtomIndex atom_site_label_3;
```

GeomAngle.atom_site_auth_(1,2,3)

An optional identifier of the three atom sites that define the angle specified by GeomAngle.value.

GeomAngle.atom_site_auth_(1,2,3).atom are optional fields. The flags F_GEOM_ANGLE_ATOM_SITE_AUTH_(1,2,3)_ATOM_ID can be used to determine if their value has been set. Atom_site_auth_(1,2,3).atom is an index into the AtomSiteExt list such that the id field (atom_site_auth_(1,2,3).atom.id) is equal to AtomSiteExt.auth_atom_id.

GeomAngle.atom_site_auth_(1,2,3).comp are optional fields. The flags F_GEOM_ANGLE_ATOM_SITE_AUTH_(1,2,3)_COMP_ID can be used to determine if their value has been set. Atom_site_auth_(1,2,3).comp is an index into the AtomSiteExt list such that the id field (atom_site_auth_(1,2,3).comp.id) is equal to AtomSiteExt.auth_comp_id.

GeomAngle.atom_site_auth_(1,2,3).seq are optional fields. The flags F_GEOM_ANGLE_ATOM_SITE_AUTH_(1,2,3)_SEQ_ID can be used to determine if their value has been set. Atom_site_auth_(1,2,3).seq is an index into the AtomSiteExt list such that the id field (atom_site_auth_(1,2,3).seq.id) is equal to AtomSiteExt.auth_seq_id.

GeomAngle.atom_site_auth_(1,2,3).asym are optional fields. The flags F_GEOM_ANGLE_ATOM_SITE_AUTH_(1,2,3)_ASYM_ID can be used to determine if their value has been set. Atom_site_auth_(1,2,3).asym is an index into the AtomSiteExt list such that the id field (atom_site_auth_(1,2,3).asym.id) is equal to AtomSiteExt.auth_asym_id

```
AtomIndex atom_site_auth_1;
AtomIndex atom_site_auth_2;
AtomIndex atom_site_auth_3;
```

GeomAngle.publ_flag

This code signals if the angle is referred to in a publication or should be placed in a table of significant angles.

GeomAngle.publ_flag is an optional field. The flag F_GEOM_ANGLE_PUBL_FLAG can be used to determine if its value has been set.

```
string publ_flag;
```

GeomAngle.site_symmetry_(1,2,3)

The symmetry code of the three atom sites that define the angle specified by GeomAngle.

GeomAngle.site_symmetry_(1,2,3) are mandatory fields and will always be set to a valid value.

```
string site_symmetry_1;
string site_symmetry_2;
string site_symmetry_3;
```

GeomAngle.value

Angle in degrees bounded by the three sites GeomAngle.atom_site_id_1, GeomAngle.atom_site_id_2 and GeomAngle.atom_site_id_3.

GeomAngle.value is an optional field. The flag F_GEOM_ANGLE_VALUE can be used to determine if its value has been set.

```
float value;
```

GeomAngle.value_esd

The estimated standard deviation of GeomAngle.value.

GeomAngle.value_esd is an optional field. The flag F_GEOM_ANGLE_VALUE_ESD can be used to determine if its value has been set.

```
float value_esd;
```

GeomBond

Data fields in the GeomBond valuetype record details about molecular bonds, as calculated from the contents of the Atom, Cell, and Symmetry data.

The existence of the GeomBond valuetype in an Entry is optional. Its presence can be determined using the S_GEOM_BOND flag.

```
valuetype GeomBond
{
  ...
};

typedef sequence<GeomBond> GeomBondList;
```

GeomBond.atom_site_id_(1,2)

The identifiers of the two atom sites that define the bond specified by GeomBond.dist.

GeomBond.atom_site_id_(1,2) are mandatory fields and will always be set to a valid value. Atom_site_id_(1,2) are indices into the AtomSite list such that the id field (atom_site_id_(1,2)) is equal to AtomSite.id.

```
IndexId atom_site_id_1;
IndexId atom_site_id_1;
```

GeomBond.atom_site_label_(1,2)

An optional identifier of the two atom sites that define the bond specified by GeomBond.dist.

GeomBond.atom_site_label_(1,2).atom are optional fields. The flags F_GEOM_BOND_ATOM_SITE_LABEL_(1,2)_ATOM_ID can be used to determine if their value has been set. Atom_site_label_(1,2).atom is an index into the ChemCompAtom list such that the id field (atom_site_label_(1,2).atom.id) is equal to ChemCompAtom.id.

GeomBond.atom_site_label_(1,2).comp are optional fields. The flags F_GEOM_BOND_ATOM_SITE_LABEL_(1,2)_COMP_ID can be used to determine if their value has been set. Atom_site_label_(1,2).comp is an index into the ChemComp list such that the id field (atom_site_label_(1,2).comp.id) is equal to ChemComp.id.

GeomBond.atom_site_label_(1,2).seq are optional fields. The flags F_GEOM_BOND_ATOM_SITE_LABEL_(1,2)_SEQ_ID can be used to determine if their value has been set. Atom_site_label_(1,2).seq is an index into the EntityPolySeq list such that the id field (atom_site_label_(1,2).seq.id) is equal to EntityPolySeq.num.

GeomBond.atom_site_label_(1,2).asym are optional fields. The flags F_GEOM_BOND_ATOM_SITE_LABEL_(1,2)_ASYM_ID can be used to determine if their value has been set. Atom_site_label_(1,2).asym is an index into the StructAsym list such that the id field (atom_site_label_(1,2).asym.id) is equal to StructAsym.id.

GeomBond.atom_site_label_(1,2)_alt is an optional field. The flags F_GEOM_BOND_ATOM_SITE_LABEL_(1,2)_ALT_ID can be used to determine if their value has been set. Atom_site_label_(1,2).alt is an index into the AtomSite list such that the id field (atom_site.label_(1,2).alt.id) is equal to AtomSite.label.alt.id.

```
AtomIndex atom_site_label_1;
AtomIndex atom_site_label_2;
```

GeomBond.atom_site_auth_(1,2)

An optional identifier of the two atom sites that define the bond specified by GeomBond.dist.

GeomBond.atom_site_auth_(1,2).atom are optional fields. The flags F_GEOM_BOND_ATOM_SITE_AUTH_(1,2)_ATOM_ID can be used to determine if their value has been set. Atom_site_auth_(1,2).atom is an index into the AtomSiteExt list such that the id field (atom_site_auth_(1,2).atom.id) is equal to AtomSiteExt.auth_atom_id.

GeomBond.atom_site_auth_(1,2).comp are optional fields. The flags F_GEOM_BOND_ATOM_SITE_AUTH_(1,2)_COMP_ID can be used to determine if their value has been set. Atom_site_auth_(1,2).comp is an index into the AtomSiteExt list such that the id field (atom_site_auth_(1,2).comp.id) is equal to AtomSiteExt.auth_comp_id.

GeomBond.atom_site_auth_(1,2).seq are optional fields. The flags F_GEOM_BOND_ATOM_SITE_AUTH_(1,2)_SEQ_ID can be used to determine if their value has been set. Atom_site_auth_(1,2).seq is an index into the AtomSiteExt list such that the id field (atom_site_auth_(1,2).seq.id) is equal to AtomSiteExt.auth_seq_id.

GeomBond.atom_site_auth_(1,2).asym are optional fields. The flags F_GEOM_BOND_ATOM_SITE_AUTH_(1,2)_ASYM_ID can be used to determine if their value has been set. Atom_site_auth_(1,2).asym is an index into the AtomSiteExt list such that the id field (atom_site_auth_(1,2).asym.id) is equal to AtomSiteExt.auth_asym_id

```
AtomIndex atom_site_auth_1;
AtomIndex atom_site_auth_2;
```

GeomBond.dist

The intramolecular bond distance in angstroms.

GeomBond.dist is an optional field. The flag F_GEOM_BOND_DIST can be used to determine if its value has been set.

```
float dist;
```

GeomBond.dist_esd

The estimated standard deviation of GeomBond.dist.

GeomBond.dist_esd is an optional field. The flag F_GEOM_BOND_DIST_ESD can be used to determine if its value has been set.

```
float dist_esd;
```

GeomBond.publ_flag

This code signals if the bond distance is referred to in a publication or should be placed in a list of significant bond distances.

GeomBond.publ_flag is an optional field. The flag F_GEOM_BOND_PUBL_FLAG can be used to determine if its value has been set.

```
string publ_flag;
```

GeomBond.site_symmetry_(1,2)

The symmetry codes of the two atom sites that define the bond specified by GeomBond.dist.

GeomBond.site_symmetry_(1,2) is a mandatory field and will always be set to a valid value.

```
string site_symmetry_1;
string site_symmetry_2;
```

GeomContact

Data fields in the GeomContact valuetype record details about molecular contacts, as calculated from the contents of the Atom, Cell, and Symmetry data.

The existence of the GeomContact valuetype in an Entry is optional. Its presence can be determined using the S_GEOM_CONTACT flag.

```
valuetype GeomContact
{
  ...
};

typedef sequence<GeomContact> GeomContactList;
```

GeomContact.atom_site_id_(1,2)

The identifiers of the two atom sites that define the contact specified by GeomContact.dist.

GeomContact.atom_site_id_(1,2) are mandatory fields and will always be set to a valid value. Atom_site_id_(1,2) are indices into the AtomSite list such that the id field (atom_site_id_(1,2)) is equal to AtomSite.id.

```
IndexId atom_site_id_1;
IndexId atom_site_id_1;
```

GeomContact.atom_site_label_(1,2)

An optional identifier of the two atom sites that define the contact specified by GeomContact.dist.

GeomContact.atom_site_label_(1,2).atom are optional fields. The flags F_GEOM_CONTACT_ATOM_SITE_LABEL_(1,2)_ATOM_ID can be used to determine if their value has been set. Atom_site_label_(1,2).atom is an index into the ChemCompAtom list such that the id field (atom_site_label_(1,2).atom.id) is equal to ChemCompAtom.id.

GeomContact.atom_site_label_(1,2).comp are optional fields. The flags F_GEOM_CONTACT_ATOM_SITE_LABEL_(1,2)_COMP_ID can be used to determine if their value has been set. Atom_site_label_(1,2).comp is an index into the ChemComp list such that the id field (atom_site_label_(1,2).comp.id) is equal to ChemComp.id.

GeomContact.atom_site_label_(1,2).seq are optional fields. The flags F_GEOM_CONTACT_ATOM_SITE_LABEL_(1,2)_SEQ_ID can be used to determine if their value has been set. Atom_site_label_(1,2).seq is an index into the EntityPolySeq list such that the id field (atom_site_label_(1,2).seq.id) is equal to EntityPolySeq.num.

GeomContact.atom_site_label_(1,2).asym are optional fields. The flags F_GEOM_CONTACT_ATOM_SITE_LABEL_(1,2)_ASYM_ID can be used to determine if their value has been set. Atom_site_label_(1,2).asym is an index into the StructAsym list such that the id field (atom_site_label_(1,2).asym.id) is equal to StructAsym.id.

GeomContact.atom_site_label_(1,2)_alt is an optional field. The flags F_GEOM_CONTACT_ATOM_SITE_LABEL_(1,2)_ALT_ID can be used to determine if their value has been set. Atom_site_label_(1,2).alt is an index into the AtomSite list such that the id field (atom_site.label_(1,2).alt.id) is equal to AtomSite.id.

```
AtomIndex atom_site_label_1;
AtomIndex atom_site_label_2;
```

GeomContact.atom_site_auth_(1,2)

An optional identifier of the two atom sites that define the contact specified by GeomContact.dist.

GeomContact.atom_site_auth_(1,2).atom are optional fields. The flags F_GEOM_CONTACT_ATOM_SITE_AUTH_(1,2)_ATOM_ID can be used to determine if their value has been set. Atom_site_auth_(1,2).atom is an index into the AtomSiteExt list such that the id field (atom_site_auth_(1,2).atom.id) is equal to AtomSiteExt.auth_atom_id.

GeomContact.atom_site_auth_(1,2).comp are optional fields. The flags F_GEOM_CONTACT_ATOM_SITE_AUTH_(1,2)_COMP_ID can be used to determine if their value has been set. Atom_site_auth_(1,2).comp is an index into the AtomSiteExt list such that the id field (atom_site_auth_(1,2).comp.id) is equal to AtomSiteExt.auth_comp_id.

GeomContact.atom_site_auth_(1,2).seq are optional fields. The flags F_GEOM_CONTACT_ATOM_SITE_AUTH_(1,2)_SEQ_ID can be used to determine if their value has been set. Atom_site_auth_(1,2).seq is an index into the AtomSiteExt list such that the id field (atom_site_auth_(1,2).seq.id) is equal to AtomSiteExt.auth_seq_id.

GeomContact.atom_site_auth_(1,2).asym are optional fields. The flags F_GEOM_CONTACT_ATOM_SITE_AUTH_(1,2)_ASYM_ID can be used to determine if their value has been set. Atom_site_auth_(1,2).asym is an index into the AtomSiteExt list such that the id field (atom_site_auth_(1,2).asym.id) is equal to AtomSiteExt.auth_asym_id

```
AtomIndex atom_site_auth_1;
AtomIndex atom_site_auth_2;
```

GeomContact.dist

The interatomic contact distance in angstroms.

GeomContact.dist is an optional field. The flag F_GEOM_CONTACT_DIST can be used to determine if its value has been set.

```
float dist;
```

GeomContact.dist_esd

The estimated standard deviation of GeomContact.dist.

GeomContact.dist_esd is an optional field. The flag F_GEOM_CONTACT_DIST_ESD can be used to determine if its value has been set.

```
float dist_esd;
```

GeomContact.publ_flag

This code signals if the contact distance is referred to in a publication or should be placed in a list of significant contact distances.

GeomContact.publ_flag is an optional field. The flag F_GEOM_CONTACT_PUBL_FLAG can be used to determine if its value has been set.

```
string publ_flag;
```

GeomContact.site_symmetry_(1,2)

The symmetry codes of the two atom sites that define the contact specified by GeomContact.dist.

GeomContact.site_symmetry_(1,2) are mandatory fields and will always be set to a valid value.

```
string site_symmetry_1;
string site_symmetry_2;
```

GeomHbond

Data fields in the GeomHbond valuetype record details about hydrogen bonds, as calculated from the contents of the Atom, Cell, and Symmetry data.

The existence of the GeomHbond valuetype in an Entry is optional. Its presence can be determined using the S_GEOM_HBOND flag.

```
valuetype GeomHbond
{
  ...
};

typedef sequence<GeomHbond> GeomHbondList;
```

GeomHbond.angle_dha

The angle in degrees defined by the donor, hydrogen and acceptor atoms sites in a hydrogen bond.

GeomHbond.angle_dha is an optional field. The flag F_GEOM_HBOND_ANGLE_DHA can be used to determine if its value has been set.

```
float angle_dha;
```

GeomHbond.angle_dha_esd

The standard undercertainty (e.s.d) of GeomHbond.angle_dha.

GeomHbond.angle_dha_esd is an optional field. The flag F_GEOM_HBOND_ANGLE_DHA_ESD can be used to determine if its value has been set.

```
float angle_dha_esd;
```

GeomHbond.atom_site_id_(a,d,h)

The identifiers of the three atom sites that define the hydrogen bond. "_a" refers to the acceptor atom site that defines the hydrogen bond. "_d" refers to the donor atom site and "_h" refers to the hydrogen atom site.

GeomHbond.atom_site_id_(a,d,h) are mandatory fields and will always be set to a valid values. Atom_site_id_(a,d,h) are indices into the AtomSite list such that the id field (atom_site_id_(a,d,h).id) is equal to AtomSite.id.

IndexId atom_site_id_a;
IndexId atom_site_id_d;
IndexId atom_site_id_h;

GeomHbond.atom_site_label_(a,d,h)

Optional identifiers of the three atom sites that define the hydrogen bond.

GeomHbond.atom_site_label_(a,d,h).atom are optional fields. The flags F_GEOM_HBOND_ATOM_SITE_LABEL_(A,D,H)_ATOM_ID can be used to determine if their value has been set. Atom_site_label_(a,d,h).atom is an index into the ChemCompAtom list such that the id field (atom_site_label_(a,d,h).atom.id) is equal to ChemCompAtom.id.

GeomHbond.atom_site_label_(a,d,h).comp are optional fields. The flags F_GEOM_HBOND_ATOM_SITE_LABEL_(A,D,H)_COMP_ID can be used to determine if their value has been set. Atom_site_label_(a,d,h).comp is an index into the ChemComp list such that the id field (atom_site_label_(a,d,h).comp.id) is equal to ChemComp.id.

GeomHbond.atom_site_label_(a,d,h).seq are optional fields. The flags F_GEOM_HBOND_ATOM_SITE_LABEL_(A,D,H)_SEQ_ID can be used to determine if their value has been set. Atom_site_label_(a,d,h).seq is an index into the EntityPolySeq list such that the id field (atom_site_label_(a,d,h).seq.id) is equal to EntityPolySeq.num.

GeomHbond.atom_site_label_(a,d,h).asym are optional fields. The flags F_GEOM_HBOND_ATOM_SITE_LABEL_(A,D,H)_ASYM_ID can be used to determine if their value has been set. Atom_site_label_(a,d,h).asym is an index into the StructAsym list such that the id field (atom_site_label_(a,d,h).asym.id) is equal to StructAsym.id.

GeomHbond.atom_site_label_(a,d,h).alt is an optional field. The flags F_GEOM_HBOND_ATOM_SITE_LABEL_(A,D,H)_ALT_ID can be used to determine if their value has been set. Atom_site_label_(a,d,h).alt is an index into the AtomSite list such that the id field (atom_site.label_(a,d,h).alt.id) is equal to AtomSite.id.

AtomIndex atom_site_label_a;
AtomIndex atom_site_label_d;
AtomIndex atom_site_label_h;

GeomHbond.atom_site_auth_(a,d,h)

Optional identifiers of the three atom sites that define the hydrogen bond.

GeomHbond.atom_site_auth_(a,d,h).atom are optional fields. The flags F_GEOM_HBOND_ATOM_SITE_AUTH_(A,D,H)_ATOM_ID can be used to determine if their value has been set. Atom_site_auth_(a,d,h).atom is an index into the AtomSiteExt list such that the id field (atom_site_auth_(a,d,h).atom.id) is equal to AtomSiteExt.auth_atom_id.

GeomHbond.atom_site_auth_(a,d,h).comp are optional fields. The flags F_GEOM_HBOND_ATOM_SITE_AUTH_(A,D,H)_COMP_ID can be used to determine if their value has been set. Atom_site_auth_(a,d,h).comp is an index into the AtomSiteExt list such that the id field (atom_site_auth_(a,d,h).comp.id) is equal to AtomSiteExt.auth_comp_id.

GeomHbond.atom_site_auth_(a,d,h).seq are optional fields. The flags F_GEOM_HBOND_ATOM_SITE_AUTH_(A,D,H)_SEQ_ID can be used to determine if their value has been set. Atom_site_auth_(a,d,h).seq is an index into the AtomSiteExt list such that the id field (atom_site_auth_(a,d,h).seq.id) is equal to AtomSiteExt.auth_seq_id.

GeomHbond.atom_site_auth_(a,d,h).asym are optional fields. The flags F_GEOM_ANGLE_ATOM_SITE_AUTH_(A,D,H)_ASYM_ID can be used to determine if their value has been set. Atom_site_auth_(a,d,h).asym is an index into the AtomSiteExt list such that the id field (atom_site_auth_(a,d,h).asym.id) is equal to AtomSiteExt.auth_asym_id

```
AtomIndex atom_site_auth_a;
AtomIndex atom_site_auth_d;
AtomIndex atom_site_auth_h;
```

GeomHbond.dist_da

The distance in angstroms between the donor and acceptor atom sites in a hydrogen bond.

GeomHbond.dist_da is an optional field. The flag F_GEOM_HBOND_DIST_DA can be used to determine if its value has been set.

```
float dist_da;
```

GeomHbond.dist_da_esd

The standard undercertainty (e.s.d) in angstroms of GeomHbond.dist_da.

GeomHbond.dist_da_esd is an optional field. The flag F_GEOM_HBOND_DIST_DA_ESD can be used to determine if its value has been set.

```
float dist_da_esd;
```

GeomHbond.dist_dh

The distance in angstroms between the donor and hydrogen atom sites in a hydrogen bond.

GeomHbond.dist_dh is an optional field. The flag F_GEOM_HBOND_DIST_DH can be used to determine if its value has been set.

```
float dist_dh;
```

GeomHbond.dist_dh_esd

The standard undercertainty (e.s.d) in angstroms of GeomHbond.dist_dh.

GeomHbond.dist_dh_esd is an optional field. The flag F_GEOM_HBOND_DIST_DH_ESD can be used to determine if its value has been set.

```
float dist_dh_esd;
```

GeomHbond.dist_ha

The distance in angstroms between the hydrogen and acceptor atom sites in a hydrogen bond.

GeomHbond.dist_ha is an optional field. The flag F_GEOM_HBOND_DIST_HA can be used to determine if its value has been set.

```
float dist_ha;
```

GeomHbond.dist_ha_esd

The standard undercertainty (e.s.d) in angstroms of GeomHbond.dist_ha.

GeomHbond.dist_ha_esd is an optional field. The flag F_GEOM_HBOND_DIST_HA_ESD can be used to determine if its value has been set.

```
float dist_ha_esd;
```

GeomHbond.publ_flag

This code signals if the hydrogen bond distance is referred to in a publication or should be placed in a table of signficiant hydrogen-bond geometry.

GeomHbond.publ_flag is an optional field. The flag F_GEOM_HBOND_PUBL_FLAG can be used to determine if its value has been set.

```
string publ_flag;
```

GeomHbond.site_symmetry_(a,d,h)

The symmetry code of the (acceptor, donor, hydrogen) atom site that defines the hydrogen bond.

GeomHbond.site_symmetry_(a,d,h) are mandatory fields and will always be set to a valid value.

```
string site_symmetry_a;
string site_symmetry_d;
string site_symmetry_h;
```

GeomTorsion

Data fields in the GeomTorsion valuetype record details about molecular torsion angles, as calculated from the contents of the atom, cell, and symmetry data.

The vector direction GeomTorsion.atom_site_id_2 to GeomTorsion.atom_site_id_3 is the viewing direction, and the torsion angle is the angle of twist required to superimpose the projection of the vector site2-site1 onto the projection of the vector site3-site4. Clockwise torsions are positive, anticlockwise torsions are negative.

Ref: Klyne, W. & Prelog, V. (1960). Experientia, 16, 521-523.

The existence of the GeomTorsion valuetype in an Entry is optional. Its presence can be determined using the S_GEOM_TORSION flag.

```
valuetype GeomTorsion
{
  ...
};

typedef sequence<GeomTorsion> GeomTorsionList;
```

GeomTorsion.atom_site_id_(1,2,3,4)

The identifiers of the four atom sites that define the torsion angle specified by GeomTorsion.value.

GeomTorsion.atom_site_id_(1,2,3,4) are mandatory fields and will always be set to a valid values. Atom_site_id_(1,2,3,4) are indices into the AtomSite list such that the id field (atom_site_id_(1,2,3,4).id) is equal to AtomSite.id.

```
IndexId atom_site_id_1;
IndexId atom_site_id_2;
IndexId atom_site_id_3;
IndexId atom_site_id_4;
```

GeomTorsion.atom_site_label_(1,2,3,4)

Optional identifiers of the four atom sites that define the torsion angle specified by GeomTorsion.value.

GeomTorsion.atom_site_label_(1,2,3,4).atom are optional fields. The flags F_GEOM_TORSION_ATOM_SITE_LABEL_(1,2,3,4)_ATOM_ID can be used to determine if their value has been set. Atom_site_label_(1,2,3,4).atom is an index into the ChemCompAtom list such that the id field (atom_site_label_(1,2,3,4).atom.id) is equal to ChemCompAtom.id.

GeomTorsion.atom_site_label_(1,2,3,4).comp are optional fields. The flags F_GEOM_TORSION_ATOM_SITE_LABEL_(1,2,3,4)_COMP_ID can be used to determine if their value has been set. Atom_site_label_(1,2,3,4).comp is an index into the ChemComp list such that the id field (atom_site_label_(1,2,3,4).comp.id) is equal to ChemComp.id.

GeomTorsion.atom_site_label_(1,2,3,4).seq are optional fields. The flags F_GEOM_TORSION_ATOM_SITE_LABEL_(1,2,3,4)_SEQ_ID can be used to determine if their value has been set. Atom_site_label_(1,2,3,4).seq is an index into the EntityPolySeq list such that the id field (atom_site_label_(1,2,3,4).seq.id) is equal to EntityPolySeq.num.

GeomTorsion.atom_site_label_(1,2,3,4).asym are optional fields. The flags F_GEOM_TORSION_ATOM_SITE_LABEL_(1,2,3,4)_ASYM_ID can be used to determine if their value has been set. Atom_site_label_(1,2,3,4).asym is an index into the StructAsym list such that the id field (atom_site_label_(1,2,3,4).asym.id) is equal to StructAsym.id.

GeomTorsion.atom_site_label_(1,2,3,4).alt is an optional field. The flags F_GEOM_TORSION_ATOM_SITE_LABEL_(1,2,3,4)_ALT_ID can be used to determine if their value has been set. Atom_site_label_(1,2,3,4).alt is an index into the AtomSite list such that the id field (atom_site.label_(1,2,3,4).alt.id) is equal to AtomSite.id.

```
AtomIndex atom_site_label_1;
AtomIndex atom_site_label_2;
AtomIndex atom_site_label_3;
AtomIndex atom_site_label_4;
```

GeomTorsion.atom_site_auth_(1,2,3,4)

Optional identifiers of the four atom sites that define the torsion angle specified by GeomTorsion.value.

GeomTorsion.atom_site_auth_(1,2,3,4).atom are optional fields. The flags F_GEOM_TORSION_ATOM_SITE_AUTH_(1,2,3,4)_ATOM_ID can be used to determine if their value has been set. Atom_site_auth_(1,2,3,4).atom is an index into the AtomSiteExt list such that the id field (atom_site_auth_(1,2,3,4).atom.id) is equal to AtomSiteExt.auth_atom_id.

GeomTorsion.atom_site_auth_(1,2,3,4).comp are optional fields. The flags F_GEOM_TORSION_ATOM_SITE_AUTH_(1,2,3,4)_COMP_ID can be used to determine if their value has been set. Atom_site_auth_(1,2,3,4).comp is an index into the AtomSiteExt list such that the id field (atom_site_auth_(1,2,3,4).comp.id) is equal to AtomSiteExt.auth_comp_id.

GeomTorsion.atom_site_auth_(1,2,3,4).seq are optional fields. The flags F_GEOM_TORSION_ATOM_SITE_AUTH_(1,2,3,4)_SEQ_ID can be used to determine if their value has been set. Atom_site_auth_(1,2,3,4).seq is an index into the AtomSiteExt list such that the id field (atom_site_auth_(1,2,3,4).seq.id) is equal to AtomSiteExt.auth_seq_id.

GeomTorsion.atom_site_auth_(1,2,3,4).asym are optional fields. The flags F_GEOM_TORSION_ATOM_SITE_AUTH_(1,2,3,4)_ASYM_ID can be used to determine if their value has been set. Atom_site_auth_(1,2,3,4).asym is an index into the AtomSiteExt list such that the id field (atom_site_auth_(1,2,3,4).asym.id) is equal to AtomSiteExt.auth_asym_id

```
AtomIndex atom_site_auth_1;
AtomIndex atom_site_auth_2;
AtomIndex atom_site_auth_3;
AtomIndex atom_site_auth_4;
```

GeomTorsion.publ_flag

This code signals if the torsion angle is referred to in a publication or should be placed in a table of significant torsion angles.

GeomTorsion.publ_flag is an optional field. The flag F_GEOM_TORSION_PUBL_FLAG can be used to determine if its value has been set.

string publ_flag;

The symmetry codes of the four atom sites that define the torsion angle specified by GeomTorsion.

GeomTorsion.site_symmetry_(1,2,3,4) are mandatory fields and will always be set to valid values.

```
string site_symmetry_1;
string site_symmetry_2;
string site_symmetry_3;
string site_symmetry_4;
```

GeomTorsion.value

The value of the torsion angle in degrees.

GeomTorsion.value is an optional field. The flag F_GEOM_TORSION_VALUE can be used to determine if its value has been set.

float value;

GeomTorsion.value_esd

The estimated standard deviation of GeomTorsion.value.

GeomTorsion.value_esd is an optional field. The flag F_GEOM_TORSION_VALUE_ESD can be used to determine if its value has been set.

```
float value_esd;
```

Structure

Data fields in the Structure valuetype record details about the description of the structure.

The existence of the Structure valuetype in an Entry is optional. Its presence can be determined using the S_STRUCTURE flag.

```
valuetype Structure
{
  ...
};

typedef sequence<Structure> StructureList;
```

Structure.entry_id

Entry_id is a pointer to the entry identifier.

Structure.entry_id is a mandatory field and will always be set to a valid value.

```
EntryId entry_id;
```

Structure.title

A title for the structure. The author should attempt to convey the essence of the structure archived in the CIF in the title, and to distinguish this structural result from others.

Structure.title is an optional field. The flag F_STRUCTURE_TITLE can be used to determine if its value has been set.

```
string title;
```

StructAsym

Data fields in the StructAsym valuetype record details about the structural elements in the asymmetric unit.

The existence of the StructAsym valuetype in an Entry is optional. Its presence can be determined using the S_STRUCT_ASYM flag.

```
valuetype StructAsym
{
  ...
};

typedef sequence<StructAsym> StructAsymList;
```

StructAsym.details

A description of special aspects of this portion of the contents of the asymmetric unit.

StructAsym.details is an optional field. The flag F_STRUCT_ASYM_DETAILS can be used to determine if its value has been set.

```
string details;
```

StructAsym.entity

Entity is a pointer to Enity.id.

StructAsym.entity is a mandatory field and will always be set to a valid value. Entity is an index into the Entity list such that the id field (entity.id) is equal to Entity.id.

```
IndexId entity;
```

StructAsym.id

The value of StructAsym.id must uniquely identify a record in the StructAsym list. Note that this field need not be a number; it can be any unique identifier.

StructAsym.id is a mandatory field and will always be set to a valid value.

```
string id;
```

StructBiol

Data fields in the StructBiol valuetype record details about the structural elements that form each structure of biological significance.

A given crystal structure may contain many different biological structures. A given structural component in the asymmetric unit may be part of more than one biological unit. A given biological structure may involve crystallographic symmetry.

For instance, in a structure of a lysozyme-FAB structure, the light and heavy chain components of the Fab could be one biological unit, while the two chains of the Fab and the lysozyme could constitute a second biological unit.

The existence of the StructBiol valuetype in an Entry is optional. Its presence can be determined using the S_STRUCT_BIOL flag.

```
valuetype StructBiol
{
  ...
};

typedef sequence<StructBiol> StructBiolList;
```

StructBiol.details

A description of special aspects of the biological unit.

StructBiol.details is an optional field. The flag F_STRUCT_BIOL_DETAILS can be used to determine if its value has been set.

```
string details;
```

StructBiol.id

The value of StructBiol.id must uniquely identify a record in the StructBiol list. Note that this field need not be a number; it can be any unique identifier.

StructBiol.id is a mandatory field and will always be set to a valid value.

```
string id;
```

StructBiolGen

Data fields in the StructBiolGen valuetype record details about the generation of each biological unit. The StructBiolGen data fields provide the specifications of the components that constitute that biological unit, which may include symmetry elements.

The existence of the StructBiolGen valuetype in an Entry is optional. Its presence can be determined using the S_STRUCT_BIOL_GEN flag.

```
valuetype StructBiolGen
{
  ...
};

typedef sequence<StructBiolGen> StructBiolGenList;
```

StructBiolGen.asym

Asym is a pointer to StructAsym.id in the StructAsym valuetype.

StructBiolGen.asym is a mandatory field and will always be set to a valid value. Asym is an index into the StructAsym list such that the id field (asym.id) is equal to StructAsym.id.

```
IndexId asym;
```

StructBiolGen.biol

Biol is a pointer to StructBiol.id in the StructBiol valuetype.

StructBiolGen.biol is a mandatory field and will always be set to a valid value. Biol is an index into the StructBiol list such that the id field (biol.id) is equal to StructBiol.id.

```
IndexId biol;
```

StructBiolGen.details

A description of special aspects of the symmetry generation of this portion of the biological structure.

StructBiolGen.details is an optional field. The flag F_STRUCT_BIOL_GEN_DETAILS can be used to determine if its value has been set.

```
string details;
```

StructBiolGen.symmetry

Describes the symmetry operation that should be applied to the atom set specified by StructBiolGen.asym_id to generate a portion of the biological structure.

StructBiolGen.symmetry is a mandatory field and will always be set to a valid value.

```
string symmetry;
```

StructBiolKeywords

Data fields in the StructBiolKeywords valuetype record details about keywords that describe each biological unit.

The existence of the StructBiolKeywords valuetype in an Entry is optional. Its presence can be determined using the S_STRUCT_BIOL_KEYWORDS flag.

```
valuetype StructBiolKeywords
{
  ...
};

typedef sequence<StructBiolKeywords> StructBiolKeywordsList;
```

StructBiolKeywords.biol

Biol is a pointer to StructBiol.id in the StructBiol valuetype.

StructBiolKeywords.biol is a mandatory field and will always be set to a valid value. Biol is an index into the StructBiol list such that the id field (biol.id) is equal to StructBiol.id.

```
IndexId biol;
```

StructBiolKeywords.text

Keywords describing this biological entity.

StructBiolKeywords.text is a mandatory field and will always be set to a valid value.

```
string text;
```

StructBiolView

Data fields in the StructBiolView valuetype record details about how to draw and annotate a useful didactic view of the biological structure.

The existence of the StructBiolView valuetype in an Entry is optional. Its presence can be determined using the S_STRUCT_BIOL_VIEW flag.

```
valuetype StructBiolView
{
  ...
};

typedef sequence<StructBiolView> StructBiolViewList;
```

StructBiolView.biol

Biol is a pointer to StructBiol.id in the StructBiol valuetype.

StructBiolView.biol is a mandatory field and will always be set to a valid value. Biol is an index into the StructBiol list such that the id field (biol.id) is equal to StructBiol.id.

```
IndexId biol;
```

StructBiolView.details

A description of special aspects of this view of the biological structure. Details can be used as a figure legend, if desired.

StructBiolView.details is an optional field. The flag F_STRUCT_BIOL_VIEW_DETAILS can be used to determine if its value has been set.

```
string details;
```

StructBiolView.id

The value of StructBiolView.id must uniquely identify a record in the StructBiolView list. Note that this field need not be a number; it can be any unique identifier.

StructBiolView.id is a mandatory field and will always be set to a valid value.

```
string id;
```

StructBiolView.rot_matrix

The elements of the matrix used to rotate the subset of the Cartesian coordinates in the AtomSite valuetype identified in the StructBiolViewGen valuetype to a view useful for describing the structure. The conventions used in the rotation are described in StructBiolView.details.

$$\begin{bmatrix} x' \\ y' \\ z' \end{bmatrix}_{\text{reoriented Cartesian}} = \begin{bmatrix} r_{11} & r_{12} & r_{13} \\ r_{21} & r_{22} & r_{23} \\ r_{31} & r_{32} & r_{33} \end{bmatrix} \cdot \begin{bmatrix} x \\ y \\ z \end{bmatrix}_{\text{Cartesian}}$$

StructBiolView.rot_matrix is an optional field. The flag F_STRUCT_BIOL_VIEW_ROT_MATRIX can be used to determine if its value has been set.

```
Matrix3 rot_matrix;
```

StructConf

Data fields in the StructConf valuetype record details about the backbone conformation of a segment of polymer.

The StructConfType records define the criteria used to identify the backbone conformations.

The existence of the StructConf valuetype in an Entry is optional. Its presence can be determined using the S__STRUCT_CONF flag.

```
valuetype StructConf
{
  ...
};

typedef sequence<StructConf> StructConfList;
```

StructConf.(beg,end)_label

The identifiers for the residues at which the conformation segment begins and ends.

StructConf.(beg,end)_label.comp are mandatory fields and will always be set to a valid value. (Beg,end)_label.comp is an index into the ChemComp list such that the id field ((beg,end)_label.comp.id) is equal to ChemComp.id.

StructConf.(beg,end)_label.seq are mandatory fields and will always be set to a valid value. (Beg,end)_label.seq is an index into the EntityPolySeq list such that the id field ((beg,end)_label.seq.id) is equal to EntityPolySeq.num.

StructConf.(beg,end)_label.asym are mandatory fields and will always be set to a valid value. (Beg,end)_label.asym is an index into the StructAsym list such that the id field ((beg,end)_label.asym.id) is equal to StructAsym.id.

SeqIndex beg_label;
SeqIndex end_label;

StructConf.(beg,end)_auth

Identifiers provided by the author for the residue at which the conformation segment begins and ends.

StructConf.(beg,end)_auth.comp is an optional field. The flag F_STRUCT_CONF_(BEG,END)_AUTH_COMP_ID can be used to determine if its value has been set. (Beg,end)_auth_(1,2).comp is an index into the AtomSiteExt list such that the id field ((beg,end)_auth_(1,2).comp.id) is equal to AtomSiteExt.auth_comp_id.

StructConf.(beg,end)_auth.seq is an optional field. The flag F_STRUCT_CONF_(BEG,END)_AUTH_SEQ_ID can be used to determine if its value has been set. (Beg,end)_auth_(1,2).seq is an index into the AtomSiteExt list such that the id field ((beg,end)_auth_(1,2).seq.id) is equal to AtomSiteExt.auth_seq_id.

StructConf.(beg,end)_auth.asym is an optional field. The flag F_STRUCT_CONF_(BEG,END)_AUTH_ASYM_ID can be used to determine if its value has been set. (Beg,end)_auth_(1,2).asym is an index into the AtomSiteExt list such that the id field ((beg,end)_auth_(1,2).asym.id) is equal to AtomSiteExt.auth_asym_id

SeqIndex beg_auth;
SeqIndex end_auth;

StructConf.details

A description of special aspects of the conformation assignment.

StructConf.details is an optional field. The flag F_STRUCT_CONF_DETAILS can be used to determine if its value has been set.

string details;

StructConf.id

The value of StructConf.id must uniquely identify a record in the StructConf list. Note that this field need not be a number; it can be any unique identifier.

StructConf.id is a mandatory field and will always be set to a valid value.

string id;

StructConfType

Data fields in the StructConfType valuetype record details about the criteria used to identify backbone conformations of a segment of polymer.

The existence of the StructConfType valuetype in an Entry is optional. Its presence can be determined using the S_STRUCT_CONF_TYPE flag.

```
valuetype StructConfType
{
  ...
};

typedef sequence<StructConfType> StructConfTypeList;
```

StructConfType.criteria

The criteria used to assign this conformation type.

StructConfType.criteria is an optional field. The flag F_STRUCT_CONF_TYPE_CRITERIA can be used to determine if its value has been set.

```
string criteria;
```

StructConfType.id

The descriptor that categorizes type of the conformation of the backbone of the polymer (whether protein or nucleic acid). Explicit values for the torsions angles that define each conformation are not given here, but it is expected that the author would provide such information in either the StructConfType.criteria or StructConfType.reference data fields, or both.

StructConfType.id is a mandatory field and will always be set to a valid value.

```
string id;
```

StructConfType.reference

A literature reference that defines the criteria used to assign this conformation type and subtype.

StructConfType.reference is an optional field. The flag F_STRUCT_CONF_TYPE_REFERENCE can be used to determine if its value has been set.

```
string reference;
```

StructConn

Data fields in the StructConn valuetype record details about the interactions between portions of structure. These can be hydrogen bonds, salt bridges, disulfide bridges, and so on.

The StructConnType records define the criteria used to identify these contacts.

The existence of the StructConn valuetype in an Entry is optional. Its presence can be determined using the S_STRUCT_CONN flag.

```
struct StructConn
{
  ...
};

typedef sequence<StructConn> StructConnList;
```

StructConn.conn_type_id

Conn_type_id is a pointer to StructConnType.id in the StructConnType valuetype.

StructConn.conn_type_id is a mandatory field and will always be set to a valid value. Conn_type is an index into the StructConnType list such that the id field (conn_type.id) is equal to StructConnType.id.

```
IndexId conn_type;
```

StructConn.details

A description of special aspects of the connect field.

StructConn.details is an optional field. The flag F_STRUCT_CONN_DETAILS can be used to determine if its value has been set.

```
string details;
```

StructConn.id

The value of StructConn.id must uniquely identify a record in the StructConn list. Note that this field need not be a number; it can be any unique identifier.

StructConn.id is a mandatory field and will always be set to a valid value.

```
string id;
```

StructConn.ptnr(1,2)_label

The identifiers for the two atom site partners that define the structure connection.

StructConn.ptnr(1,2)_label.atom are optional fields. The flags F_STRUCT_CONN_PTNR(1,2)_LABEL_ATOM_ID can be used to determine if their value has been set. Ptnr(1,2)_label.atom is an index into the ChemCompAtom list such that the id field (ptnr(1,2)_label.atom.id) is equal to ChemCompAtom.id.

StructConn.ptnr(1,2)_label.comp are optional fields. The flags F_STRUCT_CONN_PTNR(1,2)_LABEL_COMP_ID can be used to determine if their value has been set. Ptnr(1,2)_label.comp is an index into the ChemComp list such that the id field (ptnr(1,2)_label.comp.id) is equal to ChemComp.id.

StructConn.ptnr(1,2)_label.seq are optional fields. The flags F_STRUCT_CONN_PTNR(1,2)_LABEL_SEQ_ID can be used to determine if their value has been set. Ptnr(1,2)_label.seq is an index into the EntityPolySeq list such that the id field (ptnr(1,2)_label.seq.id) is equal to EntityPolySeq.num.

StructConn.ptnr(1,2)_label.asym are optional fields. The flags F_STRUCT_CONN_PTNR(1,2)_LABEL_ASYM_ID can be used to determine if their value has been set. Ptnr(1,2)_label.asym is an index into the StructAsym list such that the id field (ptnr(1,2)_label.asym.id) is equal to StructAsym.id.

StructConn.ptnr(1,2)_label.alt is an optional field. The flags F_STRUCT_CONN_PTNR(1,2)_LABEL_ALT_ID can be used to determine if their value has been set. Ptnr(1,2)_label.alt is an index into the AtomSite list such that the id field (ptnr(1,2)_label.alt.id) is equal to AtomSite.label.alt.id.

AtomIndex ptnr1_label;
AtomIndex ptnr2_label;

StructConn.ptnr(1,2)_auth

Identifiers provided by the author for the two partners of the structure connection.

StructConn.ptnr(1,2)_auth.atom are optional fields. The flags F_STRUCT_CONN_PTNR(1,2)_AUTH_ATOM_ID can be used to determine if their value has been set. Ptnr(1,2)_auth.atom is an index into the AtomSiteExt list such that the id field (ptnr(1,2)_auth.atom.id) is equal to AtomSiteExt.auth_atom_id.

StructConn.ptnr(1,2)_auth.comp are optional fields. The flags F_STRUCT_CONN_PTNR(1,2)_AUTH_COMP_ID can be used to determine if their value has been set. Ptnr(1,2)_auth.comp is an index into the AtomSiteExt list such that the id field (ptnr(1,2)_auth.comp.id) is equal to AtomSiteExt.auth_comp_id.

StructConn.ptnr(1,2)_auth.seq are optional fields. The flags F_STRUCT_CONN_PTNR(1,2)_AUTH_SEQ_ID can be used to determine if their value has been set. Ptnr(1,2)_auth.seq is an index into the AtomSiteExt list such that the id field (ptnr(1,2)_auth.seq.id) is equal to AtomSiteExt.auth_seq_id.

StructConn.ptnr(1,2)_auth.asym are optional fields. The flags F_STRUCT_CONN_PTNR(1,2)_AUTH_ASYM_ID can be used to determine if their value has been set. Ptnr(1,2)_auth.asym is an index into the AtomSiteExt list such that the id field (ptnr(1,2)_auth.asym.id) is equal to AtomSiteExt.auth_asym_id

AtomIndex ptnr1_auth;
AtomIndex ptnr2_auth;

StructConn.ptnr(1,2)_role

The chemical or structural role of the two partners in the structure connection.

StructConn.ptnr(1,2)_role is an optional field. The flag F_STRUCT_CONN_PTNR1_ROLE can be used to determine if its value has been set.

string ptnr1_role;
string ptnr2_role;

StructConn.ptnr(1,2)_symmetry

Describes the symmetry operation that should be applied to the atom set specified by StructConn.ptnr(1,2).label to generate the first partner in the structure connection.

StructConn.ptnr(1,2)_symmetry is an optional field. The flag F_STRUCT_CONN_PTNR1_SYMMETRY can be used to determine if its value has been set.

```
string ptnr1_symmetry;
string ptnr2_symmetry;
```

StructConnType

Data fields in the StructConnType valuetype record details about the criteria used to identify interactions between portions of structure.

The existence of the StructConnType valuetype in an Entry is optional. Its presence can be determined using the S_STRUCT_CONN_TYPE flag.

```
valuetype StructConnType
{
  ...
};

typedef sequence<StructConnType> StructConnTypeList;
```

StructConnType.criteria

The criteria used to define the interaction.

StructConnType.criteria is an optional field. The flag F_STRUCT_CONN_TYPE_CRITERIA can be used to determine if its value has been set.

```
string criteria;
```

StructConnType.id

The chemical or structural type of the interaction.

StructConnType.id is a mandatory field and will always be set to a valid value.

```
string id;
```

StructConnType.reference

A reference that specifies the criteria used to define the interaction.

StructConnType.reference is an optional field. The flag F_STRUCT_CONN_TYPE_REFERENCE can be used to determine if its value has been set.

```
string reference;
```

StructKeywords

Data fields in the StructKeywords valuetype specify keywords that describe the chemical structure in this entry.

The existence of the StructKeywords valuetype in an Entry is optional. Its presence can be determined using the S_STRUCT_KEYWORDS flag.

```
valuetype StructKeywords
{
  ...
};

typedef sequence<StructKeywords> StructKeywordsList;
```

StructKeywords.entry_id

Entry_id is the entry identifier.

StructKeywords.entry_id is a mandatory field and will always be set to a valid value.

```
EntryId entry_id;
```

StructKeywords.text

Keywords describing this struct.

StructKeywords.text is a mandatory field and will always be set to a valid value.

```
string text;
```

StructMonDetails

Data fields in the StructMonDetails valuetype record details about specifics of calculations summaries in data fields in the StructMonProt and StructMonNucl valuetypes. These can include the coefficients used in various maps calculations, the radii used for including points in a calculation, etc.

The existence of the StructMonDetails valuetype in an Entry is optional. Its presence can be determined using the S_STRUCT_MON_DETAILS flag.

```
valuetype StructMonDetails
{
  ...
};

typedef sequence<StructMonDetails> StructMonDetailsList;
```

StructMonDetails.entry_id

Entry_id is the entry identifier.

StructMonDetails.entry_id is a mandatory field and will always be set to a valid value.

EntryId entry_id;

StructMonDetails.prot_cis

An ideal cis peptide bound would have an omega torsion angle of zero. prot_cis gives the value in degrees by which the observed torsion angle can differ from 0.0 and still be considered cis.

StructMonDetails.prot_cis is an optional field. The flag F_STRUCT_MON_DETAILS_PROT_CIS can be used to determine if its value has been set.

float prot_cis;

StructMonDetails.rscc

Rscc describes the specifics of the calculations that generated the values given in given in StructMonProt.rscc_all, StructMonProt.rscc_main and StructMonProt.rscc_side. The coefficients used to calculate the p(o) and p(c) maps should be given as well as the criterion for inclusion of map grid points in the calculation.

StructMonDetails.rscc is an optional field. The flag F_STRUCT_MON_DETAILS_RSCC can be used to determine if its value has been set.

string rscc;

StructMonDetails.rsr

Rsr describes the specifics of the calculations that generated the values given in given in StructMonProt.rsr_all, StructMonProt.rsr_main and StructMonProt.rsr_side. The coefficients used to calculate the p(o) and p(c) maps should be given as well as the criterion for inclusion of map grid points in the calculation.

StructMonDetails.rsr is an optional field. The flag F_STRUCT_MON_DETAILS_RSR can be used to determine if its value has been set.

string rsr;

StructMonNucl

Data fields in the StructMonNucl valuetype record details about structural properties of a nucleic acid when analyzed at the monomer level. Analogous data fields for proteins are given in the StructMonProt valuetype. For fields where the value of the property depends on the method employed to calculate it, the details of the method of calculation are described in data fields in the StructMonDetails valuetype.

The existence of the StructMonNucl valuetype in an Entry is optional. Its presence can be determined using the S__STRUCT_MON_NUCL flag.

```
valuetype StructMonNucl
{
  ...
};

typedef sequence<StructMonNucl> StructMonNuclList;
```

StructMonNucl.alpha

The value in degrees of the backbone torsion angle alpha **o3'_p_o5'_c5'.**

StructMonNucl.alpha is an optional field. The flag F_STRUCT_MON_NUCL_ALPHA can be used to determine if its value has been set.

```
float alpha;
```

StructMonNucl.beta

The value in degrees of the backbone torsion angle beta **p_o5'_c5'_c4'.**

StructMonNucl.beta is an optional field. The flag F_STRUCT_MON_NUCL_BETA can be used to determine if its value has been set.

```
float beta;
```

StructMonNucl.chi1

The value in degrees of the sugar-base torsion angle chi **o4'_c1'_n1_c2.**

StructMonNucl.chi1 is an optional field. The flag F_STRUCT_MON_NUCL_CHI1 can be used to determine if its value has been set.

```
float chi1;
```

StructMonNucl.chi2

The value in degrees of the sugar-base torsion angle chi **o4'_c1'_n9_c4**.

StructMonNucl.chi2 is an optional field. The flag F_STRUCT_MON_NUCL_CHI2 can be used to determine if its value has been set.

```
float chi2;
```

StructMonNucl.delta

The value in degrees of the backbone torsion angle delta **c5'_c4'_c3'_o3'**.

StructMonNucl.delta is an optional field. The flag F_STRUCT_MON_NUCL_DELTA can be used to determine if its value has been set.

```
float delta;
```

StructMonNucl.details

A description of special aspects of the residue, its conformation, behavior in refinement, or any other aspect that requires annotation.

StructMonNucl.details is an optional field. The flag F_STRUCT_MON_NUCL_DETAILS can be used to determine if its value has been set.

```
float details;
```

StructMonNucl.epsilon

The value in degrees of the backbone torsion angle epsilon **c4'_c3'_o3'_p.**'

StructMonNucl.epsilon is an optional field. The flag F_STRUCT_MON_NUCL_EPSILON can be used to determine if its value has been set.

```
float epsilon;
```

StructMonNucl.gamma

The value in degrees of the backbone torsion angle gamma **o5'_c5'_c4'_c3'**.

StructMonNucl.gamma is an optional field. The flag F_STRUCT_MON_NUCL_GAMMA can be used to determine if its value has been set.

```
float gamma;
```

StructMonNucl.label

The identifier for participants in the site.

StructMonNucl.label.comp is a mandatory field and will always be set to a valid value. label.comp is an index into the ChemComp list such that the id field (label.comp.id) is equal to ChemComp.id.

StructMonNucl.label.seq is a mandatory field and will always be set to a valid value. label.seq is an index into the EntityPolySeq list such that the id field (label.seq.id) is equal to EntityPolySeq.num.

StructMonNucl.label.asym is a mandatory field and will always be set to a valid value. label.asym is an index into the StructAsym list such that the id field (label.asym.id) is equal to StructAsym.id.

```
SeqIndex label;
```

StructMonNucl.auth

An identifier provided by the author for participants in the site.

StructMonNucl.auth.comp is an optional field. The flag F_STRUCT_MON_NUCL_AUTH_COMP_ID can be used to determine if its value has been set. Auth.comp is an index into the AtomSiteExt list such that the id field (auth.comp.id) is equal to AtomSiteExt.auth_comp_id.

StructMonNucl.auth.seq is an optional field. The flag F_STRUCT_MON_NUCL_AUTH_SEQ_ID can be used to determine if its value has been set. Auth.seq is an index into the AtomSiteExt list such that the id field (auth.seq.id) is equal to AtomSiteExt.auth_seq_id.

StructMonNucl.auth.asym is an optional field. The flag F_STRUCT_MON_NUCL_AUTH_ASYM_ID can be used to determine if its value has been set. Auth.asym is an index into the AtomSiteExt list such that the id field (auth.asym.id) is equal to AtomSiteExt.auth_asym_id.

SeqIndex auth;

StructMonNucl.mean_b_all

The mean value of the isotropic temperature factor for all atoms in the monomer.

StructMonNucl.mean_b_all is an optional field. The flag F_STRUCT_MON_NUCL_MEAN_B_ALL can be used to determine if its value has been set.

float mean_b_all;

StructMonNucl.mean_b_base

The mean value of the isotropic temperature factor for atoms in the base moiety of the nucleic acid monomer.

StructMonNucl.mean_b_base is an optional field. The flag F_STRUCT_MON_NUCL_MEAN_B_BASE can be used to determine if its value has been set.

float mean_b_base;

StructMonNucl.mean_b_phos

The mean value of the isotropic temperature factor for atoms in the phosphate moiety of the nucleic acid monomer.

StructMonNucl.mean_b_phos is an optional field. The flag F_STRUCT_MON_NUCL_MEAN_B_PHOS can be used to determine if its value has been set.

float mean_b_phos;

StructMonNucl.mean_b_sugar

The mean value of the isotropic temperature factor for atoms in the sugar moiety of the nucleic acid monomer.

StructMonNucl.mean_b_sugar is an optional field. The flag F_STRUCT_MON_NUCL_MEAN_B_SUGAR can be used to determine if its value has been set.

float mean_b_sugar;

StructMonNucl.nu0

The value in degrees of the sugar torsion angle nu0 **c4'_o4'_c1'_c2'**.

StructMonNucl.nu0 is an optional field. The flag F_STRUCT_MON_NUCL_NU0 can be used to determine if its value has been set.

float nu0;

StructMonNucl.nu1

The value in degrees of the sugar torsion angle nu1 **o4'_c1'_c2'_c3'.**

StructMonNucl.nu1 is an optional field. The flag F_STRUCT_MON_NUCL_NU1 can be used to determine if its value has been set.

float nu1;

StructMonNucl.nu2

The value in degrees of the sugar torsion angle nu2 **c1'_c2'_c3'_c4'**.

StructMonNucl.nu2 is an optional field. The flag F_STRUCT_MON_NUCL_NU2 can be used to determine if its value has been set.

float nu2;

StructMonNucl.nu3

The value in degrees of the sugar torsion angle nu3 **c2'_c3'_c4'_o4'**.

StructMonNucl.nu3 is an optional field. The flag F_STRUCT_MON_NUCL_NU3 can be used to determine if its value has been set.

float nu3;

StructMonNucl.nu4

The value in degrees of the sugar torsion angle nu4 **c3'_c4'_o4'_c1'**.

StructMonNucl.nu4 is an optional field. The flag F_STRUCT_MON_NUCL_NU4 can be used to determine if its value has been set.

float nu4;

StructMonNucl.p

P is the phase angle of pseudorotation for five membered rings. This formulation is used for ribo and deoxyribo sugars in nucleic acids.

$$P = \operatorname{atan}\frac{(\tau_4 + \tau_1) - (\tau_3 + \tau_0)}{2\tau_2(\sin 36° + \sin 72°)}$$

$$\text{If } \tau_2 < 0 \text{ then } P = p + 180°$$

This formulation is by Altona and Sundaralingam (1972), J.a.c.s., 94, 8205-8212.

StructMonNucl.p is an optional field. The flag F_STRUCT_MON_NUCL_P can be used to determine if its value has been set.

```
float p;
```

StructMonNucl.rscc_all

The real-space (linear) correlation coefficient Rscc, as described by Jones et al., evaluated over all atoms in the nucleic acid monomer.

$$Rscc = \frac{\sum|\rho_{obs} - \langle\rho_{obs}\rangle| \cdot \sum|\rho_{calc} - \langle\rho_{calc}\rangle|}{\sqrt{\sum|\rho_{obs} - \langle\rho_{obs}\rangle|^2 \cdot \sum|\rho_{calc} - \langle\rho_{calc}\rangle|^2}}$$

ρ_{obs} = the density in an "experimental" map

ρ_{calc} = the density in a "calculated" map

The sum is taken over the specified grid points

The details of how these maps were calculated should be described in StructMonDetails.rscc. < > indicates an average and the sums are taken over all map grid points near the relevant atoms. The radius for including grid points in the calculation should also be given in StructMonDetails.rscc.

Ref: Jones, T. A., Zou, J. Y., Cowan, S. W. & Kjeldgaard, M. (1991). Acta Cryst. A47, 110-119.

StructMonNucl.rscc_all is an optional field. The flag F_STRUCT_MON_NUCL_RSCC_ALL can be used to determine if its value has been set.

```
float rscc_all;
```

StructMonNucl.rscc_base

The real-space (linear) correlation coefficient Rscc (defined above), as described by Jones et al., evaluated over all atoms in the base moiety of the nucleic acid monomer.

StructMonNucl.rscc_base is an optional field. The flag F_STRUCT_MON_NUCL_RSCC_BASE can be used to determine if its value has been set.

```
float rscc_base;
```

StructMonNucl.rscc_phos

The real-space (linear) correlation coefficient Rscc (defined above), as described by Jones et al., evaluated over all atoms in the phosphate moiety of the nucleic acid monomer.

StructMonNucl.rscc_phos is an optional field. The flag F_STRUCT_MON_NUCL_RSCC_PHOS can be used to determine if its value has been set.

```
float rscc_phos;
```

The real-space (linear) correlation coefficient Rscc (defined above), as described by Jones et al., evaluated over all atoms in the sugar moiety of the nucleic acid monomer.

StructMonNucl.rscc_sugar is an optional field. The flag F_STRUCT_MON_NUCL_RSCC_SUGAR can be used to determine if its value has been set.

```
float rscc_sugar;
```

StructMonNucl.rsr_all

The real-space residual Rsr, as described by Branden and Jones, evaluated over all atoms in the nucleic acid monomer.

$$Rsr = \frac{\sum|\rho_{obs} - \rho_{calc}|}{\sum|\rho_{obs} + \rho_{calc}|}$$

ρ_{obs} = the density in an "experimental" map

ρ_{calc} = the density in a "calculated" map

The sum is taken over the specified grid points

The details of how these maps were calculated should be described in StructMonDetails.rsr. The sums are taken over all map grid points near the relevant atoms. The radius for including grid points in the calculation should also be given in StructMonDetails.rsr.

Ref: Branden, C.-i. & Jones, T. A. (1990). Nature, 343, 687-689.

StructMonNucl.rsr_all is an optional field. The flag F_STRUCT_MON_NUCL_RSR_ALL can be used to determine if its value has been set.

float rsr_all;

StructMonNucl.rsr_base

The real-space residual Rsr (defined above), as described by Branden and Jones, evaluated over all atoms in the base moiety of the nucleic acid monomer.

StructMonNucl.rsr_base is an optional field. The flag F_STRUCT_MON_NUCL_RSR_BASE can be used to determine if its value has been set.

float rsr_base;

StructMonNucl.rsr_phos

The real-space residual Rsr, as described by Branden and Jones, evaluated over all atoms in the phosphate moiety of the nucleic acid monomer.

StructMonNucl.rsr_phos is an optional field. The flag F_STRUCT_MON_NUCL_RSR_PHOS can be used to determine if its value has been set.

float rsr_phos;

StructMonNucl.rsr_sugar

The real-space residual Rsr, as described by Branden and Jones, evaluated over all atoms in the sugar moiety of the nucleic acid monomer.

StructMonNucl.rsr_sugar is an optional field. The flag F_STRUCT_MON_NUCL_RSR_SUGAR can be used to determine if its value has been set.

float rsr_sugar;

StructMonNucl.tau0

The value in degrees of the sugar torsion angle tau0 **C4’O4’C1’C2’**.

StructMonNucl.tau0 is an optional field. The flag F_STRUCT_MON_NUCL_TAU0 can be used to determine if its value has been set.

float tau0;

StructMonNucl.tau1

The value in degrees of the sugar torsion angle tau1 **O4’C1’C2’C3’**.

StructMonNucl.tau1 is an optional field. The flag F_STRUCT_MON_NUCL_TAU1 can be used to determine if its value has been set.

float tau1;

StructMonNucl.tau2

The value in degrees of the sugar torsion angle tau2 **C1'C2'C3'C4'**.

StructMonNucl.tau2 is an optional field. The flag F_STRUCT_MON_NUCL_TAU2 can be used to determine if its value has been set.

float tau2;

StructMonNucl.tau3

The value in degrees of the sugar torsion angle tau2 **C2'C3'C4'O4'**.

StructMonNucl.tau3 is an optional field. The flag F_STRUCT_MON_NUCL_TAU3 can be used to determine if its value has been set.

float tau3;

StructMonNucl.tau4

The value in degrees of the sugar torsion angle tau4 **C3'C4'O4'C1'**.

StructMonNucl.tau4 is an optional field. The flag F_STRUCT_MON_NUCL_TAU4 can be used to determine if its value has been set.

float tau4;

StructMonNucl.taum

The maximum amplitide of puckering. It is derived from the pseudorotation value, P, and the torsion angles in the ribose ring.

$$\tau_2 = \tau_{aum}\cos(P)$$

$$\tau_3 = \tau_{aum}\cos(P + 144°)$$

$$\tau_4 = \tau_{aum}\cos(P + 288°)$$

$$\tau_0 = \tau_{aum}\cos(P + 72°)$$

$$\tau_1 = \tau_{aum}\cos(P + 216°)$$

StructMonNucl.taum is an optional field. The flag F_STRUCT_MON_NUCL_TAUM can be used to determine if its value has been set.

float taum;

StructMonNucl.zeta

The value in degrees of the backbone torsion angle zeta **c3'_o3'_p_o5'.'**

StructMonNucl.zeta is an optional field. The flag F_STRUCT_MON_NUCL_ZETA can be used to determine if its value has been set.

```
float zeta;
```

StructMonProt

Data fields in the StructMonProt valuetype record details about structural properties of a protein when analyzed at the monomer level. Analogous data fields for nucleic acids are given in the StructMonNucl valuetype. For fields where the value of the property depends on the method employed to calculate it, the details of the method of calculation are described in data fields in the StructMonDetails valuetype.

The existence of the StructMonProt valuetype in an Entry is optional. Its presence can be determined using the S_STRUCT_MON_PROT flag.

```
valuetype StructMonProt
{
 ...
};

typedef sequence<StructMonProt> StructMonProtList;
```

StructMonProt.chi1

The value in degrees of the side chain torsion angle chi1, for those residues containing such an angle.

StructMonProt.chi1 is an optional field. The flag F_STRUCT_MON_PROT_CHI1 can be used to determine if its value has been set.

```
float chi1;
```

StructMonProt.chi2

The value in degrees of the side chain torsion angle chi2, for those residues containing such an angle.

StructMonProt.chi2 is an optional field. The flag F_STRUCT_MON_PROT_CHI2 can be used to determine if its value has been set.

```
float chi2;
```

StructMonProt.chi3

The value in degrees of the side chain torsion angle chi3, for those residues containing such an angle.

StructMonProt.chi3 is an optional field. The flag F_STRUCT_MON_PROT_CHI3 can be used to determine if its value has been set.

float chi3;

StructMonProt.chi4

The value in degrees of the side chain torsion angle chi4, for those residues containing such an angle.

StructMonProt.chi4 is an optional field. The flag F_STRUCT_MON_PROT_CHI4 can be used to determine if its value has been set.

float chi4;

StructMonProt.chi5

The value in degrees of the side chain torsion angle chi5, for those residues containing such an angle.

StructMonProt.chi5 is an optional field. The flag F_STRUCT_MON_PROT_CHI5 can be used to determine if its value has been set.

float chi5;

StructMonProt.details

A description of special aspects of the residue, its conformation, behavior in refinement, or any other aspect that requires annotation.

StructMonProt.details is an optional field. The flag F_STRUCT_MON_PROT_DETAILS can be used to determine if its value has been set.

float details;

StructMonProt.label

The identifier for the monomer.

StructMonProt.label.comp is a mandatory field and will always be set to a valid value. label.comp is an index into the ChemComp list such that the id field (label.comp.id) is equal to ChemComp.id.

StructMonProt.label.seq is a mandatory field and will always be set to a valid value. label.seq is an index into the EntityPolySeq list such that the id field (label.seq.id) is equal to EntityPolySeq.num.

StructMonProt.label.asym is a mandatory field and will always be set to a valid value. label.asym is an index into the StructAsym list such that the id field (label.asym.id) is equal to StructAsym.id.

StructMonProt.label.alt is mandatory field and will always be set to a valid value. Label.alt is an index into the AtomSite list such that the id field (label.alt.id) is equal to AtomSite.label.alt.id.

SeqIndex label;

StructMonProtl.auth

An identifier provided by the author for the monomer.

StructMonProt.auth.comp is an optional field. The flag F_STRUCT_MON_PROT_AUTH_COMP_ID can be used to determine if its value has been set. Auth.comp is an index into the AtomSiteExt list such that the id field (auth.comp.id) is equal to AtomSiteExt.auth_comp_id.

StructMonProt.auth.seq is an optional field. The flag F_STRUCT_MON_PROT_AUTH_SEQ_ID can be used to determine if its value has been set. Auth.seq is an index into the AtomSiteExt list such that the id field (auth.seq.id) is equal to AtomSiteExt.auth_seq_id.

StructMonProt.auth.asym is an optional field. The flag F_STRUCT_MON_PROT_AUTH_ASYM_ID can be used to determine if its value has been set. Auth.asym is an index into the AtomSiteExt list such that the id field (auth.asym.id) is equal to AtomSiteExt.auth_asym_id.

SeqIndex auth;

StructMonProt.rscc_all

The real-space (linear) correlation coefficient Rscc, as described by Jones et al., evaluated over all atoms in the monomer.

$$Rscc = \frac{\sum \left|\rho_{obs} - \langle \rho_{obs} \rangle\right| \cdot \sum \left|\rho_{calc} - \langle \rho_{calc} \rangle\right|}{\sqrt{\sum \left|\rho_{obs} - \langle \rho_{obs} \rangle\right|^2 \cdot \sum \left|\rho_{calc} - \langle \rho_{calc} \rangle\right|^2}}$$

ρ_{obs} = the density in an "experimental" map

ρ_{calc} = the density in a "calculated" map

The sum is taken over the specified grid points

The details of how these maps were calculated should be described in StructMonDetails.rscc. < > indicates an average and the sums are taken over all map grid points near the relevant atoms. The radius for including grid points in the calculation should also be given in StructMonDetails.rscc.

Ref: Jones, T. A., Zou, J. Y., Cowan, S. W. & Kjeldgaard, M. (1991). Acta Cryst. A47, 110-119.

StructMonProt.rscc_all is an optional field. The flag F_STRUCT_MON_PROT_RSCC_ALL can be used to determine if its value has been set.

float rscc_all;

StructMonProt.rscc_main

The real-space (linear) correlation coefficient Rscc (defined above), as described by Jones et al., evaluated over all atoms in the main chain of the monomer.

StructMonProt.rscc_main is an optional field. The flag F_STRUCT_MON_PROT_RSCC_MAIN can be used to determine if its value has been set.

float rscc_main;

StructMonProt.rscc_side

The real-space (linear) correlation coefficient Rscc, as described by Jones et al., evaluated over all atoms in the side chain of the monomer.

StructMonProt.rscc_side is an optional field. The flag F_STRUCT_MON_PROT_RSCC_SIDE can be used to determine if its value has been set.

float rscc_side;

StructMonProt.rsr_all

The real-space residual Rsr, as described by Branden and Jones, evaluated over all atoms in the monomer.

$$Rsr = \frac{\sum |\rho_{obs} - \rho_{calc}|}{\sum |\rho_{obs} + \rho_{calc}|}$$

ρ_{obs} = the density in an "experimental" map

ρ_{calc} = the density in a "calculated" map

The sum is taken over the specified grid points

The details of how these maps were calculated should be described in StructMonDetails.rsr. The sums are taken over all map grid points near the relevant atoms. The radius for including grid points in the calculation should also be given in StructMonDetails.rsr.

Ref: Branden, C.-i. & Jones, T. A. (1990). Nature, 343, 687-689.

StructMonProt.rsr_all is an optional field. The flag F_STRUCT_MON_PROT_RSR_ALL can be used to determine if its value has been set.

float rsr_all;

StructMonProt.rsr_main

The real-space residual Rsr (defined above), as described by Branden and Jones, (1990) evaluated over all atoms in the main chain of the monomer.

StructMonProt.rsr_main is an optional field. The flag F_STRUCT_MON_PROT_RSR_MAIN can be used to determine if its value has been set.

float rsr_main;

The real-space residual Rsr (defined above), as described by Branden and Jones, (1990) evaluated over all atoms in the side chain of the monomer.

StructMonProt.rsr_side is an optional field. The flag F_STRUCT_MON_PROT_RSR_SIDE can be used to determine if its value has been set.

float rsr_side;

StructMonProt.mean_b_all

The mean value of the isotropic temperature factor for all atoms in the monomer.

StructMonProt.mean_b_all is an optional field. The flag F_STRUCT_MON_PROT_MEAN_B_ALL can be used to determine if its value has been set.

float mean_b_all;

StructMonProt.mean_b_main

The mean value of the isotropic temperature factor for atoms in the main chain of the monomer.

StructMonProt.mean_b_main is an optional field. The flag F_STRUCT_MON_PROT_MEAN_B_MAIN can be used to determine if its value has been set.

float mean_b_main;

StructMonProt.mean_b_side

The mean value of the isotropic temperature factor for atoms in the side chain of the monomer.

StructMonProt.mean_b_side is an optional field. The flag F_STRUCT_MON_PROT_MEAN_B_SIDE can be used to determine if its value has been set.

```
float mean_b_side;
```

StructMonProt.omega

The value in degrees of the main chain torsion angle omega.

StructMonProt.omega is an optional field. The flag F_STRUCT_MON_PROT_OMEGA can be used to determine if its value has been set.

```
float omega;
```

StructMonProt.phi

The value in degrees of the main chain torsion angle phi.

StructMonProt.phi is an optional field. The flag F_STRUCT_MON_PROT_PHI can be used to determine if its value has been set.

```
float phi;
```

StructMonProt.psi

The value in degrees of the main chain torsion angle psi.

StructMonProt.psi is an optional field. The flag F_STRUCT_MON_PROT_PSI can be used to determine if its value has been set.

```
float psi;
```

StructMonProtCis

Data fields in the StructMonProtCis valuetype identify monomers that have been found to have the peptide bond in the cis conformation. The criterion used to select residues to be designated as containing cis peptide bonds is given in StructMonDetails.prot_cis.

The existence of the StructMonProtCis valuetype in an Entry is optional. Its presence can be determined using the S_STRUCT_MON_PROT_CIS flag.

```
valuetype StructMonProtCis
{
 ...
};

typedef sequence<StructMonProtCis> StructMonProtCisList;
```

StructMonProtCis.label

The identifier for the monomer.

StructMonProtCis.label.comp is a mandatory field and will always be set to a valid value. label.comp is an index into the ChemComp list such that the id field (label.comp.id) is equal to ChemComp.id.

StructMonProtCis.label.seq is a mandatory field and will always be set to a valid value. label.seq is an index into the EntityPolySeq list such that the id field (label.seq.id) is equal to EntityPolySeq.num.

StructMonProtCis.label.asym is a mandatory field and will always be set to a valid value. label.asym is an index into the StructAsym list such that the id field (label.asym.id) is equal to StructAsym.id.

StructMonProtCis.label.alt is mandatory field and will always be set to a valid value. Label.alt is an index into the AtomSite list such that the id field (label.alt.id) is equal to AtomSite.label.alt.id.

```
SeqIndex label;
```

StructMonProtCisl.auth

An identifier provided by the author for the monomer.

StructMonProtCis.auth.comp is an optional field. The flag F_STRUCT_MON_PROT_CIS_AUTH_COMP_ID can be used to determine if its value has been set. Auth.comp is an index into the AtomSiteExt list such that the id field (auth.comp.id) is equal to AtomSiteExt.auth_comp_id.

StructMonProtCis.auth.seq is an optional field. The flag F_STRUCT_MON_PROT_CIS_AUTH_SEQ_ID can be used to determine if its value has been set. Auth.seq is an index into the AtomSiteExt list such that the id field (auth.seq.id) is equal to AtomSiteExt.auth_seq_id.

StructMonProtCis.auth.asym is an optional field. The flag F_STRUCT_MON_PROT_CIS_AUTH_ASYM_ID can be used to determine if its value has been set. Auth.asym is an index into the AtomSiteExt list such that the id field (auth.asym.id) is equal to AtomSiteExt.auth_asym_id.

StructNcsDom

Data fields in the StructNcsDom valuetype record information about the domains in an ensemble of domains related by one or more non-crystallographic symmetry operators.

A domain need not correspond to a complete polypeptide chain; it can be composed of one more more segments in a single chain, or by segments from more than one chain.

The existence of the StructNcsDom valuetype in an Entry is optional. Its presence can be determined using the S_STRUCT_NCS_DOM flag.

```
valuetype StructNcsDom
{
  ...
};

typedef sequence<StructNcsDom> StructNcsDomList;
```

StructNcsDom.details

A description of special aspects of the structural elements that comprise a domain in an ensemble of domains related by non- crystallographic symmetry.

StructNcsDom.details is an optional field. The flag F_STRUCT_NCS_DOM_DETAILS can be used to determine if its value has been set.

```
string details;
```

StructNcsDom.id

The value of StructNcsDom.id must uniquely identify a record in the StructNcsDom list. Note that this field need not be a number; it can be any unique identifier.

StructNcsDom.id is a mandatory field and will always be set to a valid value.

```
string id;
```

StructNcsDomLim

Data fields in the StructNcsDomLim valuetype identify the beginning and ending points of polypeptide chain segments that form all or part of a domain in an emsemble of domains related by non-crystallographic symmetry.

The existence of the StructNcsDomLim valuetype in an Entry is optional. Its presence can be determined using the S_STRUCT_NCS_DOM_LIM flag.

```
valuetype StructNcsDomLim
{
  ...
};

typedef sequence<StructNcsDomLim> StructNcsDomLimList;
```

StructNcsDomLim.(beg,end)_label

The identifiers for the monomers at which this segment of the domain begins and ends.

StructNcsDomLim.(beg,end)_label.comp are mandatory fields and will always be set to a valid value. (Beg,end)_label.comp is an index into the ChemComp list such that the id field ((beg,end)_label.comp.id) is equal to ChemComp.id.

StructNcsDomLim.(beg,end)_label.seq are mandatory fields and will always be set to a valid value. (Beg,end)_label.seq is an index into the EntityPolySeq list such that the id field ((beg,end)_label.seq.id) is equal to EntityPolySeq.num.

StructNcsDomLim.(beg,end)_label.asym are mandatory fields and will always be set to a valid value. (Beg,end)_label.asym is an index into the StructAsym list such that the id field ((beg,end)_label.asym.id) is equal to StructAsym.id.

StructNcsDomLim.(beg,end)_label.alt are mandatory fields and will always be set to a valid value. (Beg,end)_label.alt is an index into the StructAsym list such that the id field ((beg,end)_label.alt.id) is equal to AtomSite.label.alt.id.

```
SeqIndex beg_label;
SeqIndex end_label;
```

StructNcsDomLim.(beg,end)_auth

Identifiers provided by the author for the monomers at which this segment of the domain begins and ends.

StructNcsDomLim.(beg,end)_auth.comp is an optional field. The flag F_STRUCT_NCS_DOM_LIM_(BEG,END)_AUTH_COMP_ID can be used to determine if its value has been set. (Beg,end)_auth_(1,2).comp is an index into the AtomSiteExt list such that the id field ((beg,end)_auth_(1,2).comp.id) is equal to AtomSiteExt.auth_comp_id.

StructNcsDomLim.(beg,end)_auth.seq is an optional field. The flag F_STRUCT_NCS_DOM_LIM_(BEG,END)_AUTH_SEQ_ID can be used to determine if its value has been set. (Beg,end)_auth_(1,2).seq is an index into the AtomSiteExt list such that the id field ((beg,end)_auth_(1,2).seq.id) is equal to AtomSiteExt.auth_seq_id.

StructNcsDomLim.(beg,end)_auth.asym is an optional field. The flag F_STRUCT_NCS_DOM_LIM_(BEG,END)_AUTH_ASYM_ID can be used to determine if its value has been set. (Beg,end)_auth_(1,2).asym is an index into the AtomSiteExt list such that the id field ((beg,end)_auth_(1,2).asym.id) is equal to AtomSiteExt.auth_asym_id

```
SeqIndex beg_auth;
SeqIndex end_auth;
```

StructNcsDomLim.dom

Dom is a pointer to StructNcsDom.id in the StructNcsDom valuetype.

StructNcsDomLim.dom is a mandatory field and will always be set to a valid value. Dom is an index into the StructNcsDom list such that the id field (dom.id) is equal to StructNcsDom.id.

```
IndexId dom;
```

StructNcsEns

Data fields in the StructNcsEns valuetype record information about ensembles of domains related by non-crystallographic symmetry. The point group of the ensemble when taken as a whole may be specific, as well as any special aspect of the ensemble that require description.

The existence of the StructNcsEns valuetype in an Entry is optional. Its presence can be determined using the S_STRUCT_NCS_ENS flag.

```
valuetype StructNcsEns
{
  ...
};

typedef sequence<StructNcsEns> StructNcsEnsList;
```

StructNcsEns.details

A description of special aspects of the connect field.

StructNcsEns.details is an optional field. The flag F_STRUCT_NCS_ENS_DETAILS can be used to determine if its value has been set.

```
string details;
```

StructNcsEns.id

The value of StructNcsEns.id must uniquely identify a record in the StructNcsEns list. Note that this field need not be a number; it can be any unique identifier.

StructNcsEns.id is a mandatory field and will always be set to a valid value.

```
string id;
```

StructNcsEns.point_group

The point group of the ensemble of structural elements related by one or more non-crystallographic symmetry operations. The relationships need not be precise; This data fields is intended to give a rough description of the non-crystallographic symmetry relationships.

StructNcsEns.point_group is an optional field. The flag F_STRUCT_NCS_ENS_POINT_GROUP can be used to determine if its value has been set.

```
string point_group;
```

StructNcsEnsGen

Data fields in the StructNcsEnsGen valuetype list domains related by a non-crystallographic symmetry operation and identify the operator.

The existence of the StructNcsEnsGen valuetype in an Entry is optional. Its presence can be determined using the S_STRUCT_NCS_ENS_GEN flag.

```
valuetype StructNcsEnsGen
{
  ...
};

typedef sequence<StructNcsEnsGen> StructNcsEnsGenList;
```

StructNcsEnsGen.dom_id_1

The identifier for the domain that will remain unchanged by the transformation operator.

StructNcsEnsGen.dom_id_1 is a mandatory field and will always be set to a valid value. Dom_id_1 is an index into the StructNcsDom list such that the id field (dom_id_1) is equal to StructNcsDom.id.

```
IndexId dom_id_1;
```

StructNcsEnsGen.dom_id_2

The identifier for the domain that will be transformed by application of the transformation operator.

StructNcsEnsGen.dom_id_2 is a mandatory field and will always be set to a valid value. Dom_id_2 is an index into the StructNcsDom list such that the id field (dom_id_2) is equal to StructNcsDom.id.

```
IndexId dom_id_2;
```

StructNcsEnsGen.ens

Ens is a pointer to StructNcsEns.id in the StructNcsEns valuetype.

StructNcsEnsGen.ens is a mandatory field and will always be set to a valid value. Ens is an index into the StructNcsEns list such that the id field (ens.id) is equal to StructNcsEns.id.

```
IndexId ens;
```

StructNcsEnsGen.oper

Oper is a pointer to StructNcsOper.id in the StructNcsOper valuetype.

StructNcsEnsGen.oper is a mandatory field and will always be set to a valid value. Oper is an index into the StructNcsOper list such that the id field (oper.id) is equal to StructNcsOper.id.

```
IndexId oper;
```

StructNcsOper

Data fields in the StructNcsOper valuetype describe the non-crystallographic symmetry operations.

Each operator is specified as a matrix and a subsequent translation vector. Operators need not represent proper rotations.

The existence of the StructNcsOper valuetype in an Entry is optional. Its presence can be determined using the S_STRUCT_NCS_OPER flag.

```
valuetype StructNcsOper
{
 ...
};

typedef sequence<StructNcsOper> StructNcsOperList;
```

StructNcsOper.code

A code to indicate whether this operator describes a relationship between coordinates all of which are given in the entry (in which case the value of code is ‘given’), or whether the operator is used to generate new coordinates from those that are given in the entry (in which case the value of code is ‘generate’).

StructNcsOper.code is an optional field. The flag F_STRUCT_NCS_OPER_CODE can be used to determine if its value has been set.

```
string code;
```

StructNcsOper.details

A description of special aspects of the non-crystallographic symmetry operator.

StructNcsOper.details is an optional field. The flag F_STRUCT_NCS_OPER_DETAILS can be used to determine if its value has been set.

```
string details;
```

StructNcsOper.id

The value of StructNcsOper.id must uniquely identify a record in the StructNcsOper list. Note that this field need not be a number; it can be any unique identifier.

StructNcsOper.id is a mandatory field and will always be set to a valid value.

```
string id;
```

StructNcsOper.matrix

The elements of the 3x3 matrix component of a non- cyrstallographic symmetry operation.

StructNcsOper.matrix is an optional field. The flag F_STRUCT_NCS_OPER_MATRIX can be used to determine if its value has been set.

```
Matrix3 matrix;
```

StructNcsOper.vector

The elements of the 3 element vector component of a non- crystallographic symmetry operation.

StructNcsOper.vector is an optional field. The flag F_STRUCT_NCS_OPER_VECTOR can be used to determine if its value has been set.

```
Vector3 vector;
```

StructRef

Data fields in the StructRef valuetype allow the author of a entry to relate the biological units described in that entry to information archived in external databases.

For references to the sequence of a polymer, the value of the data field StructRef.seq_align is used to indicate whether the correspondence between the sequence of the entity or biological unit in the given entry and the sequence in the referenced database entry is 'complete' or 'partial'. If this value is 'partial', the region (or regions) of the alignment may be delimited using data fields in the StructRefSeq valuetype.

Also for references to the sequence of a polymer, the value of StructRef.seq_dif is used to indicate whether or not the two sequences contain point differences. If the value is yes, the differences may be identified and annotated using data data fields in the StructRefSeqDif valuetype.

The existence of the StructRef valuetype in an Entry is optional. Its presence can be determined using the S_STRUCT_REF flag.

```
valuetype StructRef
{
  ...
};

typedef sequence<StructRef> StructRefList;
```

StructRef.biol

Biol is a pointer to StructBiol.id in the StructBiol valuetype.

StructRef.biol is a mandatory field and will always be set to a valid value. Biol is an index into the StructBiol list such that the id field (biol.id) is equal to StructBiol.id.

```
IndexId biol;
```

StructRef.db_code

The code for this entity or biological unit or for a closely related entity or biological unit in the named database.

StructRef.db_code is a mandatory field and will always be set to a valid value.

string db_code;

StructRef.db_name

The name of the database containing reference information about this entity or biological unit.

StructRef.db_name is a mandatory field and will always be set to a valid value.

string db_name;

StructRef.details

A description of special aspects of the relationship between the entity or biological unit described in the entry and the referenced database entry.

StructRef.details is an optional field. The flag F_STRUCT_REF_DETAILS can be used to determine if its value has been set.

string details;

StructRef.entity

Entity is a pointer to Entity.id in the Entity valuetype.

StructRef.entity_id is a mandatory field and will always be set to a valid value. Entity is an index into the Entity list such that the id field (entity.id) is equal to Entity.id.

IndexId entity;

StructRef.id

The value of StructRef.id must uniquely identify a record in the StructRef list. Note that this field need not be a number; it can be any unique identifier.

StructRef.id is a mandatory field and will always be set to a valid value.

string id;

StructRef.seq_align

A flag to indicate the scope of the alignement between the sequence of the entity or biological unit described in this entry and the referenced database entry. ‘entire’ indicates that alignment spans the entire length of both sequences (although point differences may occur, and can be annotated using the data fields in the StructRefSeqDif valuetype.) ‘partial’ indicates a partial alignment, and the region (or regions) of the alignment may be delimited using data fields in the StructRefSeq valuetype. seq_align may also take the value '.', indicating that the reference is not to a sequence.

StructRef.seq_align is an optional field. The flag F_STRUCT_REF_SEQ_ALIGN can be used to determine if its value has been set.

```
string seq_align;
```

StructRef.seq_dif

A flag to indicate the presence ('yes') or absence ('no') of point differences between the sequence of the entity or biological unit described in the this entry and the referenced database entry. seq_dif may also take the value '.', indicating that the reference is not to a sequence.

StructRef.seq_dif is an optional field. The flag F_STRUCT_REF_SEQ_DIF can be used to determine if its value has been set.

```
string seq_dif;
```

StructRefSeq

Data fields in the StructRefSeq valuetype provide a mechanism for indicating and annotating a region (or regions) of alignment between the sequence of an entity or biological unit described in the this entry and the sequence in the referenced database entry.

The existence of the StructRefSeq valuetype in an Entry is optional. Its presence can be determined using the S_STRUCT_REF_SEQ flag.

```
valuetype StructRefSeq
{
  ...
};

typedef sequence<StructRefSeq> StructRefSeqList;
```

StructRefSeq.align_id

The value of StructRefSeq.align_id must uniquely identify a record in the StructRefSeq list. Note that this field need not be a number; it can be any unique identifier.

StructRefSeq.align_id is a mandatory field and will always be set to a valid value.

```
string align_id;
```

StructRefSeq.db_align_beg

The sequence position at which the alignment begins in the referenced database entry.

StructRefSeq.db_align_beg is a mandatory field and will always be set to a valid value.

long db_align_beg;

StructRefSeq.db_align_end

The sequence position at which the alignment ends in the referenced database entry.

StructRefSeq.db_align_end is a mandatory field and will always be set to a valid value.

long db_align_end;

StructRefSeq.details

A description of special aspects of the sequence alignment.

StructRefSeq.details is an optional field. The flag F_STRUCT_REF_SEQ_DETAILS can be used to determine if its value has been set.

string details;

StructRefSeq.ref

Ref is a pointer to StructRef.id in the StructRef valuetype.

StructRefSeq.ref is a mandatory field and will always be set to a valid value. Ref is an index into the StructRef list such that the id field (ref.id) is equal to StructRef.id.

IndexId ref;

StructRefSeq.seq_align_beg

The sequence position at which the alignment begins in the entity or biological unit described.

StructRefSeq.seq_align_beg is a mandatory field and will always be set to a valid value. Seq_align_beg is an index into the EntityPolySeq list such that the id field (seq_align_beg) is equal to EntityPolySeq.num.

IndexId seq_align_beg;

StructRefSeq.seq_align_end

The sequence position at which the alignment begins in the entity or biological unit described.

StructRefSeq.seq_align_end is a mandatory field and will always be set to a valid value. Seq_align_end is an index into the EntityPolySeq list such that the id field (seq_align_end) is equal to EntityPolySeq.num.

IndexId seq_align_end;

StructRefSeqDif

Data fields in the StructRefSeqDif valuetype provide a mechanism for indicating and annotating point differences between the sequence of the entity or biological unit described in this entry and the sequence of the referenced database entry.

The existence of the StructRefSeqDif valuetype in an Entry is optional. Its presence can be determined using the S_STRUCT_REF_SEQ_DIF flag.

```
valuetype StructRefSeqDif
{
  ...
};

typedef sequence<StructRefSeqDif> StructRefSeqDifList;
```

StructRefSeqDif.align

Align is a pointer to StructRefSeq.align_id in the StructRefSeq valuetype.

StructRefSeqDif.align is a mandatory field and will always be set to a valid value. Align is an index into the StructRefSeq list such that the id field (align.id) is equal to StructRefSeq.align.id.

```
IndexId align;
```

StructRefSeqDif.db_mon

The monomer type found at this position in the referenced database entry.

StructRefSeqDif.db_mon is a mandatory field and will always be set to a valid value. Db_mon is an index into the ChemComp list such that the id field (db_mon.id) is equal to ChemComp.id.

```
IndexId db_mon;
```

StructRefSeqDif.details

A description of special aspects of the point differences between the sequence of the entity of biological unit described in this entry and the referenced database entry.

StructRefSeqDif.details is an optional field. The flag F_STRUCT_REF_SEQ_DIF_DETAILS can be used to determine if its value has been set.

```
string details;
```

StructRefSeqDif.mon

The monomer type found at this position in the sequence of the entity or biological unit described in this entry.

StructRefSeqDif.mon is a mandatory field and will always be set to a valid value. Mon is an index into the ChemComp list such that the id field (mon.id) is equal to ChemComp.id.

```
IndexId mon;
```

StructRefSeqDif.seq_num

Seq_num is a pointer to EntityPolySeq.num in the EntityPolySeq valuetype.

StructRefSeqDif.seq_num is a mandatory field and will always be set to a valid value. Seq_num is an index into the EntityPolySeq list such that the id field (seq_num) is equal to EntityPolySeq.num.

```
IndexId seq_num;
```

StructSheet

Data fields in the StructSheet valuetype record details about the beta sheets.

The existence of the StructSheet valuetype in an Entry is optional. Its presence can be determined using the S_STRUCT_SHEET flag.

```
valuetype StructSheet
{
 ...
};

typedef sequence<StructSheet> StructSheetList;
```

StructSheet.details

A description of special aspects of the beta-sheet.

StructSheet.details is an optional field. The flag F_STRUCT_SHEET_DETAILS can be used to determine if its value has been set.

```
string details;
```

StructSheet.id

The value of StructSheet.id must uniquely identify a record in the StructSheet list. Note that this field need not be a number; it can be any unique identifier.

StructSheet.id is a mandatory field and will always be set to a valid value.

```
string id;
```

StructSheet.number_strands

The number of strands in the sheet. If a given range of residues is bulged out from the stands, it is still counted as one strand. If a strand is composed of two different regions of polypeptide, it is still counted as one strand, so long as the proper hydrogen bonding connections are made to adjacent strands.

StructSheet.number_strands is an optional field. The flag F_STRUCT_SHEET_NUMBER_STRANDS can be used to determine if its value has been set.

```
long number_strands;
```

StructSheet.type

A simple descriptor for the type of the sheet.

StructSheet.type is an optional field. The flag F_STRUCT_SHEET_TYPE can be used to determine if its value has been set.

```
string type;
```

StructSheetHbond

Data fields in the StructSheetHbond valuetype record details about the hydrogen bonding between residue ranges in a beta sheet. It is necessary to treat hydrogen bonding independently of the designation of ranges, because the hydrogen bonding may begin in different places for the interactions of a given strand with the one preceding it and the one following it in the sheet.

The existence of the StructSheetHbond valuetype in an Entry is optional. Its presence can be determined using the S_STRUCT_SHEET_HBOND flag.

```
valuetype StructSheetHbond
{
 ...
};

typedef sequence<StructSheetHbond> StructSheetHbondList;
```

StructSheetHbond.range_(1,2)_(beg,end)_label_atom

The identifiers for the residue atoms in the two partners of the first and last hydrogen bonds between the two residue ranges in a sheet.

StructSheetHbond.range_(1,2)_(beg,end)_label_atom is a mandatory field and will always be set to a valid value. Range_(1,2)_(beg,end)_label_atom is an index into the ChemCompAtom list such that the id field (range_(1,2)_(beg,end)_label_atom.id) is equal to ChemCompAtom.atom_id.

IndexId range_1_beg_label_atom;
IndexId range_1_end_label_atom;
IndexId range_2_beg_label_atom;
IndexId range_2_end_label_atom;

StructSheetHbond.range_(1,2)_(beg,end)_label_seq;

The identifiers for the residues in the two partners of the first and last hydrogen bonds between the two residue ranges in a sheet.

StructSheetHbond.range_(1,2)_(beg,end)_label_seq is a mandatory field and will always be set to a valid value. Range_(1,2)_(beg,end)_label_seq is an index into the EntityPolySeq list such that the id field (range_(1,2)_(beg,end)_label_seq.id) is equal to EntityPolySeq.num.

IndexId range_1_beg_label_seq;
IndexId range_1_end_label_seq;
IndexId range_2_beg_label_seq;
IndexId range_2_end_label_seq;

StructSheetHbond.range_(1,2)_(beg,end)_auth_atom

The identifiers provided by the author for the residue atoms in the two partners of the first and last hydrogen bonds between the two residue ranges in a sheet.

StructSheetHbond.range_(1,2)_(beg,end)_auth_atom are optional fields. The flags F_STRUCT_SHEET_HBOND_RANGE_(1,2)_(BEG,END)_AUTH_ATOM_ID can be used to determine if their value has been set. Range_(1,2)_(beg,end)_auth_atom is an index into the ChemCompAtom list such that the id field (range_(1,2)_(beg,end)_auth_atom.id) is equal to ChemCompAtom.atom_id.

IndexId range_1_beg_auth_atom;
IndexId range_1_end_auth_atom;
IndexId range_2_beg_auth_atom;
IndexId range_2_end_auth_atom;

StructSheetHbond.range_(1,2)_(beg,end)_auth_seq;

The identifiers provided by the author for the residues in the two partners of the first and last hydrogen bonds between the two residue ranges in a sheet.

StructSheetHbond.range_(1,2)_(beg,end)_auth_seq are optional fields. The flags F_STRUCT_SHEET_HBOND_RANGE_(1,2)_(BEG,END)_AUTH_SEQ_ID can be used to determine if their value has been set. Range_(1,2)_(beg,end)_auth_seq is an index into the EntityPolySeq list such that the id field (range_(1,2)_(beg,end)_auth_seq.id) is equal to EntityPolySeq.num.

```
IndexId range_1_beg_auth_seq;
IndexId range_1_end_auth_seq;
IndexId range_2_beg_auth_seq;
IndexId range_2_end_auth_seq;
```

StructSheetHbond.range_id_(1,2)

Range_id_(1,2) are pointers to StructSheetRange.id in the StructSheetRange valuetype.

StructSheetHbond.range_id_(1,2) are mandatory fields and will always be set to a valid value. Range_id_(1,2) are indices into the StructSheetRange list such that the id field (range_id_(1,2)) is equal to StructSheetRange.id.

```
IndexId range_id_1;
IndexId range_id_2;
```

StructSheetHbond.sheet

Sheet is a pointer to StructSheet.id in the StructSheet valuetype.

StructSheetHbond.sheet is a mandatory field and will always be set to a valid value. Sheet is an index into the StructSheet list such that the id field (sheet.id) is equal to StructSheet.id.

```
IndexId sheet;
```

StructSheetOrder

Data fields in the StructSheetOrder valuetype record details about the order of the residue ranges that form a beta sheet. All order linkages are pairwise, and the specified pairs are assumed to be adjacent to one another in the sheet. These data fields are an alternative to the StructSheetTopology data fields, and they allow for the formal description of all manner of sheets.

The existence of the StructSheetOrder valuetype in an Entry is optional. Its presence can be determined using the S__STRUCT_SHEET_ORDER flag.

```
valuetype StructSheetOrder
{
  ...
};

typedef sequence<StructSheetOrder> StructSheetOrderList;
```

StructSheetOrder.offset

Designated the relative position in the sheet, plus or minus, of the second residue range to the first.

StructSheetOrder.offset is an optional field. The flag F_STRUCT_SHEET_ORDER_OFFSET can be used to determine if its value has been set.

long offset;

StructSheetOrder.range_id_(1,2)

Range_id_(1,2) are pointers to StructSheetRange.id in the StructSheetRange valuetype.

StructSheetOrder.range_id_(1,2) are mandatory fields and will always be set to a valid value. Range_id_(1,2) are indices into the StructSheetRange list such that the id field (range_id_(1,2).id) is equal to StructSheetRange.id.

IndexId range_id_1;
IndexId range_id_2;

StructSheetOrder.sense

A flag to indicate whether the two designated residue ranges are parallel or antiparallel to one another.

StructSheetOrder.sense is an optional field. The flag F_STRUCT_SHEET_ORDER_SENSE can be used to determine if its value has been set.

string sense;

StructSheetOrder.sheet

Sheet is a pointer to StructSheet.id in the StructSheet valuetype.

StructSheetOrder.sheet is a mandatory field and will always be set to a valid value. Sheet is an index into the StructSheet list such that the id field (sheet.id) is equal to StructSheet.id.

IndexId sheet;

StructSheetRange

Data fields in the StructSheetRange valuetype record details about the residue ranges that form a beta sheet. Residues are included in a range if they made beta-sheet type hydrogen bonding interactions with at least one adjacent strand and if there are at least two residues in the range.

The existence of the StructSheetRange valuetype in an Entry is optional. Its presence can be determined using the S_STRUCT_SHEET_RANGE flag.

```
valuetype StructSheetRange
{
  ...
};

typedef sequence<StructSheetRange> StructSheetRangeList;
```

StructSheetRange.(beg,end)_label

Identifiers for the residues at which the beta sheet range begins and ends.

StructSheetRange.(beg,end)_label.comp are mandatory fields and will always be set to a valid value. (Beg,end)_label.comp is an index into the ChemComp list such that the id field ((beg,end)_label.comp.id) is equal to ChemComp.id.

StructSheetRange.(beg,end)_label.seq are mandatory fields and will always be set to a valid value. (Beg,end)_label.seq is an index into the EntityPolySeq list such that the id field ((beg,end)_label.seq.id) is equal to EntityPolySeq.num.

StructSheetRange.(beg,end)_label.asym are mandatory fields and will always be set to a valid value. (Beg,end)_label.asym is an index into the StructAsym list such that the id field ((beg,end)_label.asym.id) is equal to StructAsym.id.

```
  SeqIndex beg_label;
  SeqIndex end_label;
```

StructSheetRange.(beg,end)_auth

Identifiers provided by the author for the residues at which the beta sheet range begins and ends.

StructSheetRange.(beg,end)_auth.comp is an optional field. The flag F_STRUCT_SHEET_RANGE_(BEG,END)_AUTH_COMP_ID can be used to determine if its value has been set. (Beg,end)_auth_(1,2).comp is an index into the AtomSiteExt list such that the id field ((beg,end)_auth_(1,2).comp.id) is equal to AtomSiteExt.auth_comp_id.

StructSheetRange.(beg,end)_auth.seq is an optional field. The flag F_STRUCT_SHEET_RANGE_(BEG,END)_AUTH_SEQ_ID can be used to determine if its value has been set. (Beg,end)_auth_(1,2).seq is an index into the AtomSiteExt list such that the id field ((beg,end)_auth_(1,2).seq.id) is equal to AtomSiteExt.auth_seq_id.

StructSheetRange.(beg,end)_auth.asym is an optional field. The flag F_STRUCT_SHEET_RANGE_(BEG,END)_AUTH_ASYM_ID can be used to determine if its value has been set. (Beg,end)_auth_(1,2).asym is an index into the AtomSiteExt list such that the id field ((beg,end)_auth_(1,2).asym.id) is equal to AtomSiteExt.auth_asym_id

SeqIndex beg_auth;
SeqIndex end_auth;

StructSheetRange.id

The value of StructSheetRange.id must uniquely identify a range in a given sheet in the StructSheetRange list. Note that this field need not be a number; it can be any unique identifier.

StructSheetRange.id is a mandatory field and will always be set to a valid value.

string id;

StructSheetRange.sheet

Sheet is a pointer to StructSheet.id in the StructSheet valuetype.

StructSheetRange.sheet is a mandatory field and will always be set to a valid value. Sheet is an index into the StructSheet list such that the id field (sheet.id) is equal to StructSheet.id.

IndexId sheet;

StructSheetRange.symmetry

Describes the symmetry operation that should be applied to the residues delimited by the beginning and ending designators in order to generate the appropriate strand in this sheet.

StructSheetRange.symmetry is an optional field. The flag F_STRUCT_SHEET_RANGE_SYMMETRY can be used to determine if its value has been set.

string symmetry;

StructSheetTopology

Data fields in the StructSheetTopology valuetype record details about the topology of the residue ranges that form a beta sheet. All topology linkages are pairwise, and the specified pairs are assumed to be successive in the amino acid sequence. These data fields are useful in describing various simple and complex folds, but they become inadequate when the strands in the sheet come from more than one chain. One can alternatively used the StructSheetOrder data fields to describe both single and multiple chain-containing sheets.

The existence of the StructSheetTopology valuetype in an Entry is optional. Its presence can be determined using the S_STRUCT_SHEET_TOPOLOGY flag.

```
valuetype StructSheetTopology
{
  ...
};

typedef sequence<StructSheetTopology>
                StructSheetTopologyList;
```

StructSheetTopology.offset

Designated the relative position in the sheet, plus or minus, of the second residue range to the first.

StructSheetTopology.offset is an optional field. The flag F_STRUCT_SHEET_TOPOLOGY_OFFSET can be used to determine if its value has been set.

```
long offset;
```

StructSheetTopology.range_id_(1,2)

Range_id_(1,2) are pointers to StructSheetRange.id in the StructSheetRange valuetype.

StructSheetTopology.range_id_(1,2) are mandatory fields and will always be set to a valid value. Range_id_(1,2) are indices into the StructSheetRange list such that the id field (range_id_(1,2)) is equal to StructSheetRange.id.

```
IndexId range_id_1;
IndexId range_id_2;
```

StructSheetTopology.sense

A flag to indicate whether the two designated residue ranges are parallel or antiparallel to one another.

StructSheetTopology.sense is an optional field. The flag F_STRUCT_SHEET_TOPOLOGY_SENSE can be used to determine if its value has been set.

```
string sense;
```

StructSheetTopology.sheet

Sheet is a pointer to StructSheet.id in the StructSheet valuetype.

StructSheetTopology.sheet is a mandatory field and will always be set to a valid value. Sheet is an index into the StructSheet list such that the id field (sheet.id) is equal to StructSheet.id.

```
IndexId sheet;
```

StructSite

Data fields in the StructSite valuetype record details about portions of structure that contribute to certain structurally relevant sites (i.e., active sites, substrate-binding subsites, metal-coordination sites).

The existence of the StructSite valuetype in an Entry is optional. Its presence can be determined using the S_STRUCT_SITE flag.

```
valuetype StructSite
{
  ...
};

typedef sequence<StructSite> StructSiteList;
```

StructSite.details

A description of special aspects of the structural site.

StructSite.details is an optional field. The flag F_STRUCT_SITE_DETAILS can be used to determine if its value has been set.

```
string details;
```

StructSite.id

The value of StructSite.id must uniquely identify a record in the StructSite list. Note that this field need not be a number; it can be any unique identifier.

StructSite.id is a mandatory field and will always be set to a valid value.

```
string id;
```

StructSiteGen

Data fields in the StructSiteGen valuetype record details about the generation of portions of structure that contribute to structurally relevant sites.

The existence of the StructSiteGen valuetype in an Entry is optional. Its presence can be determined using the S_STRUCT_SITE_GEN flag.

```
valuetype StructSiteGen
{
  ...
};

typedef sequence<StructSiteGen> StructSiteGenList;
```

StructSiteGen.details

A description of special aspects of the symmetry generation of this portion of the structural site.

StructSiteGen.details is an optional field. The flag F_STRUCT_SITE_GEN_DETAILS can be used to determine if its value has been set.

string details;

StructSiteGen.id

The value of StructSiteGen.id must uniquely identify a record in the StructSiteGen list. Note that this field need not be a number; it can be any unique identifier.

StructSiteGen.id is a mandatory field and will always be set to a valid value.

string id;

StructSiteGen.label

The identifier for participants in the site.

StructSiteGen.label.atom is a mandatory field and will always be set to a valid value. Label.atom is an index into the ChemCompAtom list such that the id field (label.atom.id) is equal to ChemCompAtom.atom_id.

StructSiteGen.label.comp is a mandatory field and will always be set to a valid value. label.comp is an index into the ChemComp list such that the id field (label.comp.id) is equal to ChemComp.id.

StructSiteGen.label.seq is a mandatory field and will always be set to a valid value. label.seq is an index into the EntityPolySeq list such that the id field (label.seq.id) is equal to EntityPolySeq.num.

StructSiteGen.label.asym is a mandatory field and will always be set to a valid value. label.asym is an index into the StructAsym list such that the id field (label.asym.id) is equal to StructAsym.id.

StructSiteGen.label.alt is mandatory field and will always be set to a valid value. Label.alt is an index into the AtomSite list such that the id field (label.alt.id) is equal to AtomSite.label.alt.id.

AtomIndex label;

StructSiteGenl.auth

The identifier provided by the author for participants in the site.

StructSiteGen.auth.atom is an optional field. The flag F_STRUCT_SITE_GEN_AUTH_ATOM_ID can be used to determine if its value has been set. Auth.atom is an index into the AtomSiteExt list such that the id field (auth.atom.id) is equal to AtomSiteExt.auth_atom_id.

StructSiteGen.auth.comp is an optional field. The flag F_STRUCT_SITE_GEN_AUTH_COMP_ID can be used to determine if its value has been set. Auth.comp is an index into the AtomSiteExt list such that the id field (auth.comp.id) is equal to AtomSiteExt.auth_comp_id.

StructSiteGen.auth.seq is an optional field. The flag F_STRUCT_SITE_GEN_AUTH_SEQ_ID can be used to determine if its value has been set. Auth.seq is an index into the AtomSiteExt list such that the id field (auth.seq.id) is equal to AtomSiteExt.auth_seq_id.

StructSiteGen.auth.asym is an optional field. The flag F_STRUCT_SITE_GEN_AUTH_ASYM_ID can be used to determine if its value has been set. Auth.asym is an index into the AtomSiteExt list such that the id field (auth.asym.id) is equal to AtomSiteExt.auth_asym_id.

```
AtomIndex auth;
```

StructSiteGen.site

Site is a pointer to StructSite.id in the StructSite valuetype.

StructSiteGen.site is a mandatory field and will always be set to a valid value. Site is an index into the StructSite list such that the id field (site.id) is equal to StructSite.id.

```
IndexId site;
```

StructSiteGen.symmetry

Describes the symmetry operation that should be applied to the atom set specified by StructSiteGen.label to generate a portion of the structure site.

StructSiteGen.symmetry is an optional field. The flag F_STRUCT_SITE_GEN_SYMMETRY can be used to determine if its value has been set.

```
string symmetry;
```

StructSiteKeywords

Data fields in the StructSiteKeywords valuetype...

The existence of the StructSiteKeywords valuetype in an Entry is optional. Its presence can be determined using the S_STRUCT_SITE_KEYWORDS flag.

```
valuetype StructSiteKeywords
{
 ...
};

typedef sequence<StructSiteKeywords> StructSiteKeywordsList;
```

StructSiteKeywords.site

Site is a pointer to StructSite.id in the StructSite valuetype.

StructSiteKeywords.site is a mandatory field and will always be set to a valid value. Site is an index into the StructSite list such that the id field (site.id) is equal to StructSite.id.

```
IndexId site;
```

StructSiteKeywords.text

Keywords describing this structural site.

StructSiteKeywords.text is a mandatory field and will always be set to a valid value.

```
string text;
```

StructSiteView

Data fields in the StructSiteView valuetype record details about how to draw and annotate a useful didactic view of the structural site.

The existence of the StructSiteView valuetype in an Entry is optional. Its presence can be determined using the S_STRUCT_SITE_VIEW flag.

```
valuetype StructSiteView
{
  ...
};

typedef sequence<StructSiteView> StructSiteViewList;
```

StructSiteView.details

A description of special aspects of this view of the structural site. details can be used as a figure legend, if desired.

StructSiteView.details is an optional field. The flag F_STRUCT_SITE_VIEW_DETAILS can be used to determine if its value has been set.

```
string details;
```

StructSiteView.id

The value of StructSiteView.id must uniquely identify a record in the StructSiteView list. Note that this field need not be a number; it can be any unique identifier.

StructSiteView.id is a mandatory field and will always be set to a valid value.

```
string id;
```

StructSiteView.rot_matrix

The elements of the matrix used to rotate the subset of the Cartesian coordinates in the AtomSite valuetype identified in the StructSiteViewGen valuetype to a view useful for describing the structural site. The conventions used in the rotation are described in StructSiteView.details.

$$\begin{bmatrix} x' \\ y' \\ z' \end{bmatrix}_{\text{reoriented Cartesian}} = \begin{bmatrix} r_{11} & r_{12} & r_{13} \\ r_{21} & r_{22} & r_{23} \\ r_{31} & r_{32} & r_{33} \end{bmatrix} \cdot \begin{bmatrix} x \\ y \\ z \end{bmatrix}_{\text{Cartesian}}$$

StructSiteView.rot_matrix is an optional field. The flag F_STRUCT_SITE_VIEW_ROT_MATRIX can be used to determine if its value has been set.

```
Matrix3 rot_matrix;
```

StructSiteView.site

Site is a pointer to StructSite.id in the StructSite valuetype.

StructSiteView.site is a mandatory field and will always be set to a valid value. Site is an index into the StructSite list such that the id field (site.id) is equal to StructSite.id.

```
IndexId site;
```

5.4 The DsLSRMmsReference Module

5.4.1 The MmsReferenceEntry Interface

Relevant data about items such as literature references, citations, database identifiers and sturcture audits are retrieved using methods defined in the MmsReferenceEntry interface.

5.4.2 DsLSRMmsReference Summary

The following valuetypes make up the DsLsrMmsReference module.

CITATION

Citation

Literature cited in reference to the entry

CitationAuthor

Author(s) of the citations

CitationEditor

Editor(s) of citations where applicable

COMPUTING

Computing
Computer programs used in the structure analysis

Software
Description of the software used e.g. in the structure analysis

DATABASE

Database
Codes assigned to dictionary by maintainers of recognized databases

DatabasePdbCaveat
CAVEAT records originally found in the PDB version of the data file

DatabasePdbMatrix
MATRIX records originally found in the PDB version of the data file.

DatabasePdbRemark
REMARK records originally found in the PDB version of the data file

DatabasePdbRev
Taken from the PDB REVDAT records

DatabasePdbRevRecord
Taken from the PDB REVDAT records

DatabasePdbTvect
TVECT records originally found in the PDB version of the mmCIF data file

5.4.3 DsLSRMmsReference Valuetypes and Structs

Citation

Data fields in the Citation valuetype record details about the literature cited relevant to the contents of the entry.

The existence of the Citation valuetype in an Entry is optional. Its presence can be determined using the S_CITATION flag.

```
valuetype Citation
{
  ...
};

typedef sequence<Citation> CitationList;
```

Citation.abstract_tex

Abstract for the citation. This is used most when the citation is extracted from a bibliographic database that contains full text or abstract information.

Citation.abstract_text is an optional field. The flag F_CITATION_ABSTRACT_TEXT can be used to determine if its value has been set.

string abstract_text;

Citation.abstract_id_CAS

The Chemical Abstracts Service (cas) abstract identifier; relevant for journal articles.

Citation.abstract_id_CAS is an optional field. The flag F_CITATION_ABSTRACT_ID_CAS can be used to determine if its value has been set.

string abstract_id_CAS;

Citation.book_id_isbn

The International Standard Book Number (isbn) code assigned to the book cited; relevant for book chapters.

Citation.book_id_isbn is an optional field. The flag F_CITATION_BOOK_ID_ISBN can be used to determine if its value has been set.

string book_id_isbn;

Citation.book_publisher

The name of the publisher of the citation; relevant for book chapters.

Citation.book_publisher is an optional field. The flag F_CITATION_BOOK_PUBLISHER can be used to determine if its value has been set.

string book_publisher;

Citation.book_publisher_city

The location of the publisher of the citation; relevant for book chapters.

Citation.book_publisher_city is an optional field. The flag F_CITATION_BOOK_PUBLISHER_CITY can be used to determine if its value has been set.

string book_publisher_city;

Citation.book_title

The title of the book in which the citation appeared; relevant for book chapters.

Citation.book_title is an optional field. The flag F_CITATION_BOOK_TITLE can be used to determine if its value has been set.

string book_title;

Citation.coordinate_linkage

Citation.coordinate_linkage states whether or not this citation is concerned with precisely the set of coordinates given in the entry. If, for instance, the publication described the same structure, but the coordinates had undergone further refinement prior to creation of the entry, the value of this data field would be 'no'.

Citation.coordinate_linkage is an optional field. The flag F_CITATION_COORDINATE_LINKAGE can be used to determine if its value has been set.

```
string coordinate_linkage;
```

Citation.country

The country of publication; relevant for both journal articles and book chapters.

Citation.country is an optional field. The flag F_CITATION_COUNTRY can be used to determine if its value has been set.

```
string country;
```

Citation.database_id_medline

Ascession number used by Medline to categorize a specific bibliographic entry.

Citation.database_id_medline is an optional field. The flag F_CITATION_DATABASE_ID_MEDLINE can be used to determine if its value has been set.

```
long database_id_medline;
```

Citation.details

A description of special aspects that describe the relationship of the contents of the entry to the literature field cited.

Citation.details is an optional field. The flag F_CITATION_DETAILS can be used to determine if its value has been set.

```
string details;
```

Citation.id

The value of Citation.id must uniquely identify a record in the Citation list.

The Citation.id 'primary' should be used to indicate the citation that the author(s) consider to be the most pertinent to the contents of the entry. Note that this field need not be a number; it can be any unique identifier.

Citation.id is a mandatory field and will always be set to a valid value.

```
string id;
```

Citation.journal_abbrev

Abbreviated name of the journal cited as given in the Chemical Abstracts Service Source Index.

Citation.journal_abbrev is an optional field. The flag F_CITATION_JOURNAL_ABBREV can be used to determine if its value has been set.

```
string journal_abbrev;
```

Citation.journal_id_astm

The American Society for the Testing of Materials (astm) code assigned to the journal cited (also referred to as the Coden designator of the Chemical Abstracts Service); relevant for journal articles.

Citation.journal_id_astm is an optional field. The flag F_CITATION_JOURNAL_ID_ASTM can be used to determine if its value has been set.

```
string journal_id_astm;
```

Citation.journal_id_csd

The Cambridge Structural Database (csd) code assigned to the journal cited; relevant for journal articles.

Citation.journal_id_csd is an optional field. The flag F_CITATION_JOURNAL_ID_CSD can be used to determine if its value has been set.

```
string journal_id_csd;
```

Citation.journal_id_issn

The International Standard Serial Number (issn) code assigned to the journal cited; relevant for journal articles.

Citation.journal_id_issn is an optional field. The flag F_CITATION_JOURNAL_ID_ISSN can be used to determine if its value has been set.

```
string journal_id_issn;
```

Citation.journal_full

Full name of the journal cited; relevant for journal articles.

Citation.journal_full is an optional field. The flag F_CITATION_JOURNAL_FULL can be used to determine if its value has been set.

```
string journal_full;
```

Citation.journal_issue

Issue number of the journal cited; relevant for journal articles.

Citation.journal_issue is an optional field. The flag F_CITATION_JOURNAL_ISSUE can be used to determine if its value has been set.

string journal_issue;

Citation.journal_volume

Volume number of the journal cited; relevant for journal articles.

Citation.journal_volume is an optional field. The flag F_CITATION_JOURNAL_VOLUME can be used to determine if its value has been set.

string journal_volume;

Citation.language

Language in which the citation appears.

Citation.language is an optional field. The flag F_CITATION_LANGUAGE can be used to determine if its value has been set.

string language;

Citation.page_first

The first page of the citation; relevant for journal articles and book chapters.

Citation.page_first is an optional field. The flag F_CITATION_PAGE_FIRST can be used to determine if its value has been set.

string page_first;

Citation.page_last

The last page of the citation; relevant for journal articles and book chapters.

Citation.page_last is an optional field. The flag F_CITATION_PAGE_LAST can be used to determine if its value has been set.

string page_last;

Citation.title

The title of the citation; relevant for both journal articles and book chapters.

Citation.title is an optional field. The flag F_CITATION_TITLE can be used to determine if its value has been set.

string title;

Citation.year

The year of the citation; relevant for both journal articles and book chapters.

Citation.year is an optional field. The flag F_CITATION_YEAR can be used to determine if its value has been set.

```
long year;
```

CitationAuthor

Data fields in the CitationAuthor valuetype record details about the authors associated with the citations in the Citation list.

The existence of the CitationAuthor valuetype in an Entry is optional. Its presence can be determined using the S_CITATION_AUTHOR flag.

```
valuetype CitationAuthor
{
  ...
};

typedef sequence<CitationAuthor> CitationAuthorList;
```

CitationAuthor.citation

Citation is a pointer to Citation.id in the Citation valuetype.

CitationAuthor.citation is a mandatory field and will always be set to a valid value. Citation is an index into the Citation list such that the id field (citation.id) is equal to Citation.id.

```
DsLSRMacromolecularStructure::IndexId citation;
```

CitationAuthor.name

Name of an author of the citation; relevant for both journal articles and book chapters.

The family name(s), followed by a comma and including any dynastic components, precedes the first name(s) or initial(s).

CitationAuthor.name is a mandatory field and will always be set to a valid value.

```
string name;
```

CitationAuthor.ordinal

Ordinal defines the order of the author's name in the' list of authors of a citation.

CitationAuthor.ordinal is an optional field. The flag F_CITATION_AUTHOR_ORDINAL can be used to determine if its value has been set.

```
long ordinal;
```

CitationEditor

Data fields in the CitationEditor valuetype record details about the editor associated with book chapter citations in the Citation list.

The existence of the CitationEditor valuetype in an Entry is optional. Its presence can be determined using the S_CITATION_EDITOR flag.

```
valuetype CitationEditor
{
  ...
};

typedef sequence<CitationEditor> CitationEditorList;
```

CitationEditor.citation

Citation is a pointer to Citation.id in the Citation valuetype.

CitationEditor.citation is a mandatory field and will always be set to a valid value. Citation is an index into the Citation list such that the id field (citation.id) is equal to Citation.id.

```
DsLSRMacromolecularStructure::IndexId citation;
```

CitationEditor.name

Names of an editor of the citation; relevant for book chapters.

The family name(s), followed by a comma and including any dynastic components, precedes the first name(s) or initial(s).

CitationEditor.name is an optional field. The flag F_CITATION_EDITOR_NAME can be used to determine if its value has been set.

```
string name;
```

CitationEditor.ordinal

Ordinal defines the order of the editor's name in the' list of editors of a citation.

CitationEditor.ordinal is an optional field. The flag F_CITATION_EDITOR_ORDINAL can be used to determine if its value has been set.

```
long ordinal;
```

Database

Data fields in the Database valuetype record details about the database identifiers of the entry. These data fields are assigned by database managers and will only appear in an entry if they originate from that source.

The existence of the Database valuetype in an Entry is optional. Its presence can be determined using the S_DATABASE flag.

```
valuetype Database
{
  ...
};

typedef sequence<Database> DatabaseList;
```

Database.database_id

An abbreviation that identifies the database.

Database.database_id is a mandatory field and will always be set to a valid value.

```
    string database_id;
```

Database.database_code

The code assigned by the database identified in Database2.database_id.

Database.database_code is a mandatory field and will always be set to a valid value.

```
    string database_code;
```

DatabasePdbCaveat

Data fields in the DatabasePdbCaveat valuetype record details about features of the entry flagged as ‘caveats’ by the Brookhaven Protein Data Bank.

These data fields are included only for consistency with Pdb format files. They should appear in a entry only if that entry was created by reformatting a Pdb format file.

The existence of the DatabasePdbCaveat valuetype in an Entry is optional. Its presence can be determined using the S_DATABASE_PDB_CAVEAT flag.

```
valuetype DatabasePdbCaveat
{
  ...
};

typedef sequence<DatabasePdbCaveat>
                        DatabasePdbCaveatList;
```

DatabasePdbCaveat.id

A unique identifier for the Pdb caveat record.

DatabasePdbCaveat.id is a mandatory field and will always be set to a valid value.

```
long id;
```

DatabasePdbCaveat.text

The full text of the Pdb caveat record.

DatabasePdbCaveat.text is an optional field. The flag F_DATABASE_PDB_CAVEAT_TEXT can be used to determine if its value has been set.

```
string text;
```

DatabasePdbMatrix

The DatabasePdbMatrix valuetype provides placeholders for transformation matrices and vectors used by the Brookhaven Protein Data Bank.

These data fields are included only for consistency with older Pdb format files. They should appear in a entry only if that entry was created by reformatting a Pdb format file.

The existence of the DatabasePdbMatrix valuetype in an Entry is optional. Its presence can be determined using the S_DATABASE_PDB_MATRIX flag.

```
valuetype DatabasePdbMatrix
{
  ...
};

typedef sequence<DatabasePdbMatrix> DatabasePdbMatrixList;
```

DatabasePdbMatrix.entry_id

Entry_id is an entry identifier.

DatabasePdbMatrix.entry_id is a mandatory field and will always be set to a valid value.

```
EntryId entry_id;
```

DatabasePdbMatrix.origx

The elements of the Pdb Origx matrix.

DatabasePdbMatrix.origx is an optional field. The flag F_DATABASE_PDB_MATRIX_ORIGX can be used to determine if its value has been set.

```
DsLSRMacromolecularStructure::Matrix3 origx;
```

DatabasePdbMatrix.origx_vector

The elements of the Pdb Origx vector.

DatabasePdbMatrix.origx_vector is an optional field. The flag F_DATABASE_PDB_MATRIX_ORIGX_VECTOR can be used to determine if its value has been set.

```
DsLSRMacromolecularStructure::Vector3 origx_vector;
```

DatabasePdbMatrix.scale

The elements of the Pdb Scale matrix.

DatabasePdbMatrix.scale is an optional field. The flag F_DATABASE_PDB_MATRIX_SCALE can be used to determine if its value has been set.

```
DsLSRMacromolecularStructure::Matrix3 scale;
```

DatabasePdbMatrix.scale_vector

The elements of the Pdb Scale vector.

DatabasePdbMatrix.scale_vector is an optional field. The flag F_DATABASE_PDB_MATRIX_SCALE_VECTOR can be used to determine if its value has been set.

```
DsLSRMacromolecularStructure::Vector3 scale_vector;
```

DatabasePdbRemark

Data fields in the DatabasePdbRemark valuetype record details about the entry as archived by the Brookhaven Protein Data Bank.

Some data appearing in Pdb Remark records can be algorithmically extracted into the appropriate data fields in the entry.

These data fields are included only for consistency with older Pdb format files. They should appear in a entry only if that entry was created by reformatting a Pdb format file.

The existence of the DatabasePdbRemark valuetype in an Entry is optional. Its presence can be determined using the S_DATABASE_PDB_REMARK flag.

```
valuetype DatabasePdbRemark
{
  ...
};

typedef sequence<DatabasePdbRemark> DatabasePdbRemarkList;
```

DatabasePdbRemark.id

A unique identifier for the Pdb remark record.

DatabasePdbRemark.id is a mandatory field and will always be set to a valid value.

```
long id;
```

DatabasePdbRemark.text

The full text of the Pdb remark record.

DatabasePdbRemark.text is an optional field. The flag F_DATABASE_PDB_REMARK_TEXT can be used to determine if its value has been set.

```
string text;
```

DatabasePdbRev

Data fields in the DatabasePdbRev valuetype record details about the history of the entry as archived by the Brookhaven Protein Data Bank.

These data fields are assigned by the Pdb database managers and should only appear in a entry if they originate from that source.

The existence of the DatabasePdbRev valuetype in an Entry is optional. Its presence can be determined using the S_DATABASE_PDB_REV flag.

```
valuetype DatabasePdbRev
{
  ...
};

typedef sequence<DatabasePdbRev> DatabasePdbRevList;
```

DatabasePdbRev.author_name

The name of the person responsible for submitting this revision to the Pdb.

The family name(s) followed by a comma, precedes the first name(s) or initial(s).

DatabasePdbRev.author_name is an optional field. The flag F_DATABASE_PDB_REV_AUTHOR_NAME can be used to determine if its value has been set.

```
string author_name;
```

DatabasePdbRev.date

Date the Pdb revision took place. Taken from the Revdat record.

DatabasePdbRev.date is an optional field. The flag F_DATABASE_PDB_REV_DATE can be used to determine if its value has been set.

string date;

DatabasePdbRev.date_original

Date the entry first entered the Pdb database in the form: yyyy-mm-dd. Taken from the Pdb Header record.

DatabasePdbRev.date_original is an optional field. The flag F_DATABASE_PDB_REV_DATE_ORIGINAL can be used to determine if its value has been set.

string date_original;

DatabasePdbRev.mod_type

Taken from the Revdat record. Refer to the Protein Data Bank format description for details.

DatabasePdbRev.mod_type is an optional field. The flag F_DATABASE_PDB_REV_MOD_TYPE can be used to determine if its value has been set.

long mod_type;

DatabasePdbRev.num

The value of DatabasePdbRev.num must uniquely and sequentially identify a record in the DatabasePdbRevList.

Note that this field must be a number, and that modification numbers are assigned in increasing numerical order.

DatabasePdbRev.num is a mandatory field and will always be set to a valid value.

long num;

DatabasePdbRev.replaced_by

The Pdb code for a subsequent Pdb entry that replaced the Pdb file corresponding to this entry.

DatabasePdbRev.replaced_by is an optional field. The flag F_DATABASE_PDB_REV_REPLACED_BY can be used to determine if its value has been set.

string replaced_by;

DatabasePdbRev.replaces

The Pdb code for a previous Pdb entry that was replaced by the Pdb file corresponding to this entry.

DatabasePdbRev.replaces is an optional field. The flag F_DATABASE_PDB_REV_REPLACES can be used to determine if its value has been set.

```
string replaces;
```

DatabasePdbRev.status

This definition is preliminary - need to consult with Pdb about what they need here.

DatabasePdbRev.status is an optional field. The flag F_DATABASE_PDB_REV_STATUS can be used to determine if its value has been set.

```
string status;
```

DatabasePdbRevRecord

Data fields in the DatabasePdbRevRecord valuetype record details about specific record types that were changed in a given revision of a Pdb entry.

These data fields are assigned by the Pdb database managers and should only appear in a entry if they originate from that source.

The existence of the DatabasePdbRevRecord valuetype in an Entry is optional. Its presence can be determined using the S_DATABASE_PDB_REV_RECORD flag.

```
valuetype DatabasePdbRevRecord
{
  ...
};
typedef sequence<DatabasePdbRevRecord>
                 DatabasePdbRevRecordList;
```

DatabasePdbRevRecord.details

A description of special aspects of the revision of records in this Pdb entry.

DatabasePdbRevRecord.details is an optional field. The flag F_DATABASE_PDB_REV_RECORD_DETAILS can be used to determine if its value has been set.

```
string details;
```

DatabasePdbRevRecord.rev_num

Rev_num is a pointer to DatabasePdbRev.num in the DatabasePdbRev valuetype.

DatabasePdbRevRecord.rev_num is a mandatory field and will always be set to a valid value. Rev_num is an index into the DatabasePdbRev list such that the id field (rev_num) is equal to DatabasePdbRev.num.

```
DsLSRMacromolecularStructure::IndexId rev_num;
```

DatabasePdbRevRecord.type

The types of records that were changed in this revision to a Pdb entry.

DatabasePdbRevRecord.type is a mandatory field and will always be set to a valid value.

```
string type;
```

DatabasePdbTvect

The DatabasePdbTvect valuetype provides placeholders for the Tvect matrices and vectors.

These data fields are included only for consistency with older Pdb format files. They should appear in a entry only if that entry was created by reformatting a Pdb format file.

The existence of the DatabasePdbTvect valuetype in an Entry is optional. Its presence can be determined using the S_DATABASE_PDB_TVECT flag.

```
valuetype DatabasePdbTvect
{
  ...
};

typedef sequence<DatabasePdbTvect> DatabasePdbTvectList;
```

DatabasePdbTvect.details

A description of special aspects of this Tvect.

DatabasePdbTvect.details is an optional field. The flag F_DATABASE_PDB_TVECT_DETAILS can be used to determine if its value has been set.

```
string details;
```

DatabasePdbTvect.id

The value of DatabasePdbTvect.id must uniquely identify a record in the DatabasePdbTvect list. Note that this field need not be a number; it can be any unique identifier.

DatabasePdbTvect.id is a mandatory field and will always be set to a valid value.

```
string id;
```

DatabasePdbTvect.vector

The elements of the Pdb Tvect vector.

DatabasePdbTvect.vector is an optional field. The flag F_DATABASE_PDB_TVECT_VECTOR can be used to determine if its value has been set.

```
Vector3 vector;
```

PublManuscriptInc

Data fields in the PublManuscriptIncl valuetype allow the authors of a manuscript submitted for publication to list data names that should be added to the standard request list employed by journal printing software.

The existence of the PublManuscriptIncl valuetype in an Entry is optional. Its presence can be determined using the S__PUBL_MANUSCRIPT_INCL flag.

```
valuetype PublManuscriptIncl
{
  ...
};

typedef sequence<PublManuscriptIncl> PublManuscriptInclList;
```

PublManuscriptIncl.entry_id

Entry_id is an entry identifier.

PublManuscriptIncl.entry_id is a mandatory field and will always be set to a valid value.

```
EntryId entry_id;
```

PublManuscriptIncl.extra_defn

Flags whether the corresponding data field marked for inclusion in a journal request list is a standard definition or not (flags are ‘yes’ or ‘no’).

PublManuscriptIncl.extra_defn is an optional field. The flag F_PUBL_MANUSCRIPT_INCL_EXTRA_DEFN can be used to determine if its value has been set.

```
string extra_defn;
```

PublManuscriptIncl.extra_info

A short note indicating the reason why the author wishes the corresponding data field marked for inclusion in the journal request list to be published.

PublManuscriptIncl.extra_info is an optional field. The flag F_PUBL_MANUSCRIPT_INCL_EXTRA_INFO can be used to determine if its value has been set.

```
string extra_info;
```

PublManuscriptIncl.extra_item

Specifies the inclusion of specific data into a manuscript which is not normally requested by the journal. The values of this field are the extra data names (which Must be enclosed in single quotes) that will be added to the journal request list.

PublManuscriptIncl.extra_item is an optional field. The flag F_PUBL_MANUSCRIPT_INCL_EXTRA_ITEM can be used to determine if its value has been set.

```
string extra_item;
```

Computing

Data fields in the Computing valuetype record details about the computer programs used in the crystal structure analysis.

The existence of the Computing valuetype in an Entry is optional. Its presence can be determined using the S_COMPUTING flag.

```
valuetype Computing
{
  ...
};

typedef sequence<Computing> ComputingList;
```

Computing.entry_id

Entry_id is an entry identifier.

Computing.entry_id is a mandatory field and will always be set to a valid value.

```
EntryId entry_id;
```

Computing.cell_refinement

Software used in refining the cell, program or package name and a brief reference.

Computing.cell_refinement is an optional field. The flag F_COMPUTING_CELL_REFINEMENT can be used to determine if its value has been set.

```
string cell_refinement;
```

Computing.data_collection

Software used for data collection, the program or package name and a brief reference.

Computing.data_collection is an optional field. The flag F_COMPUTING_DATA_COLLECTION can be used to determine if its value has been set.

```
string data_collection;
```

Computing.data_reduction

Software used for data reduction, the program or package name and a brief reference.

Computing.data_reduction is an optional field. The flag F_COMPUTING_DATA_REDUCTION can be used to determine if its value has been set.

```
string data_reduction;
```

Computing.molecular_graphics

Software used for molecular graphics, the program or package name and a brief reference.

Computing.molecular_graphics is an optional field. The flag F_COMPUTING_MOLECULAR_GRAPHICS can be used to determine if its value has been set.

```
string molecular_graphics;
```

Computing.publication_material

Software used for generating material for publication, the program or package name and a brief reference.

Computing.publication_material is an optional field. The flag F_COMPUTING_PUBLICATION_MATERIAL can be used to determine if its value has been set.

```
string publication_material;
```

Computing.structure_refinement

Software used for refinement of the structure, the program or package name and a brief reference.

Computing.structure_refinement is an optional field. The flag F_COMPUTING_STRUCTURE_REFINEMENT can be used to determine if its value has been set.

```
string structure_refinement;
```

Computing.structure_solution

Software used for solution of the structure,the program or package name and a brief reference.

Computing.structure_solution is an optional field. The flag F_COMPUTING_STRUCTURE_SOLUTION can be used to determine if its value has been set.

```
string structure_solution;
```

Software

Data fields in the Software valuetype record details about the software used used in the structure analysis, which implies any software used in the generation of any data fields associated with the structure determination and structure representation. These data fields provide an alternative, and more thorough, method for referencing computer programs than do data fields in the Computing valuetype.

The existence of the Software valuetype in an Entry is optional. Its presence can be determined using the S_SOFTWARE flag.

```
valuetype Software
{
  ...
};

typedef sequence<Software> SoftwareList;
```

Software.citation

Citation is a pointer to Citation.id in the Citation valuetype.

Software.citation is a mandatory field and will always be set to a valid value. Citation is an index into the Citation list such that the id field (citation.id) is equal to Citation.id.

```
DsLSRMacromolecularStructure::IndexId citation;
```

Software.classification

The classification of the program according to its major function.

Software.classification is an optional field. The flag F_SOFTWARE_CLASSIFICATION can be used to determine if its value has been set.

```
string classification;
```

Software.compiler_name

The compiler used to compile the software.

Software.compiler_name is an optional field. The flag F_SOFTWARE_COMPILER_NAME can be used to determine if its value has been set.

string compiler_name;

Software.compiler_version

The version of the compiler used to compile the software.

Software.compiler_version is an optional field. The flag F_SOFTWARE_COMPILER_VERSION can be used to determine if its value has been set.

string compiler_version;

Software.contact_author

The recognized contact author of the software. This could be the original author, modifier of the code, or maintainer, but should be the individual most commonly associated with the code.

Software.contact_author is an optional field. The flag F_SOFTWARE_CONTACT_AUTHOR can be used to determine if its value has been set.

string contact_author;

Software.contact_author_email

The email address of the Software.contact_author.

Software.contact_author_email is an optional field. The flag F_SOFTWARE_CONTACT_AUTHOR_EMAIL can be used to determine if its value has been set.

string contact_author_email;

Software.date

The date the software was released.

Software.date is an optional field. The flag F_SOFTWARE_DATE can be used to determine if its value has been set.

string date;

Software.description

Description of the software.

Software.description is an optional field. The flag F_SOFTWARE_DESCRIPTION can be used to determine if its value has been set.

string description;

Software.dependencies

Any prerequistite software required to run Software.name.

Software.dependencies is an optional field. The flag F_SOFTWARE_DEPENDENCIES can be used to determine if its value has been set.

string dependencies;

Software.hardware

The hardware upon which the software was run.

Software.hardware is an optional field. The flag F_SOFTWARE_HARDWARE can be used to determine if its value has been set.

string hardware;

Software.language

The major computing language in which the software is coded.

Software.language is an optional field. The flag F_SOFTWARE_LANGUAGE can be used to determine if its value has been set.

string language;

Software.location

An Internet address in the form of a URL describing where details of the software can be found.

Software.location is an optional field. The flag F_SOFTWARE_LOCATION can be used to determine if its value has been set.

string location;

Software.mods

Any noteworthy modifications to the base software, if applicable.

Software.mods is an optional field. The flag F_SOFTWARE_MODS can be used to determine if its value has been set.

string mods;

Software.name

The name of the software.

Software.name is a mandatory field and will always be set to a valid value.

```
string name;
```

Software.os

The name of the operating system under which the software run.

Software.os is an optional field. The flag F_SOFTWARE_OS can be used to determine if its value has been set.

```
string os;
```

Software.os_version

The version of the operating system under which the software runs.

Software.os_version is an optional field. The flag F_SOFTWARE_OS_VERSION can be used to determine if its value has been set.

```
string os_version;
```

Software.type

The classification of the software according to the most common types.

Software.type is an optional field. The flag F_SOFTWARE_TYPE can be used to determine if its value has been set.

```
string type;
```

Software.version

The version of the software.

Software.version is a mandatory field and will always be set to a valid value.

```
string version;
```

6 Conformance Issues

6.1 Introduction

This section defines the optional compliance points of this specification and identifies the areas where there is significant overlap with other OMG specifications currently under development.

6.2 Interoperability

The overriding goal for LSR Conformance must be that all LSR specifications can be used independently or used together in any combination. Of particular importance are those areas that will affect the interoperability and useful transfer of data between various implementations of LSR specifications.

6.3 Compliance

Modules

Implementation of the DsLSRMmsReference module is optional.

BaseIDL

The implementation of the `get_extension_modules` method to return a `BaseIDL::ModuleDefSet` object specifying extension module descriptions is optional.

EntryGroups

The use of EntryGroups within the `EntryFactory` is optional. If used, the choice of the EntryGroup names is left to the implementation, but must conform to the stated use and restrictions of the `Identifier` type.

Presence Flags

The implementation of presence flags is optional. If however that are valid for any single Entry they must be valid for all Entries. I.e. they many not be selectively implemented for some entries but not others.

6.4 Bibliographic References

Several parts of the DsLSRMmsReference module contain elements that need to be aligned with the Bibliographic Query Services (BQS) specification currently under development. References from this specification however cannot be made until BQS is formally approved. Consequently the alignment work must be done during the specification finalization process. It is expected that, at that time, many of the value types in the DsLSRMmsReference module will be modified to contain or subclass BQS structures.

Appendices

A. DsLSRMacromolecularStructure IDL specification

```
// File: DsLSRMacromolecularStructure.idl

#ifndef _DS_LSR_MACROMOLECULAR_STRUCTURE_IDL_
#define _DS_LSR_MACROMOLECULAR_STRUCTURE_IDL_

#include <CosPropertyService.idl>
#include <TimeBase.idl>
#include <BaseIDL.idl>

#pragma prefix "omg.org"

module DsLSRMacromolecularStructure
{
   exception DataAccessException
   {
      string method_name;
      string description;
   };

   typedef string Identifier;

   typedef float Vector3[3];
   typedef Vector3 Matrix3[3];

   typedef string FormatType;
   typedef sequence<FormatType> FormatTypeList;
   typedef sequence<octet> EntryRepresentation;

   struct IndexId
   {
      string id;
      long index;
   };

   struct VectorXYZ
   {
      float x;
      float y;
      float z;
   };

   struct SeqIndex
```

```
{
   IndexId seq;
   IndexId comp;
   IndexId asym;
   IndexId alt;
};

struct AtomIndex
{
   IndexId atom;
   IndexId seq;
   IndexId comp;
   IndexId asym;
   IndexId alt;
};

struct AtomSite
{
   string id;
   IndexId type_symbol;
   AtomIndex label;
   IndexId label_entity;
   VectorXYZ cartn;
   float occupancy;
   float b_iso_or_equiv;
};

struct AtomSiteExt
{
   Matrix3 aniso_b;
   Matrix3 aniso_b_esd;
   float aniso_ratio;
   Matrix3 aniso_u;
   Matrix3 aniso_u_esd;
   long attached_hydrogens;
   string auth_asym_id;
   string auth_atom_id;
   string auth_comp_id;
   string auth_seq_id;
   float b_equiv_geom_mean;
   float b_equiv_geom_mean_esd;
   float b_iso_or_equiv_esd;
   string calc_attached_atom;
   string calc_flag;
   VectorXYZ cartn_esd;
   string constraints;
   string details;
   string disorder_group;
   IndexId footnote;
   VectorXYZ fract;
   VectorXYZ fract_esd;
   float occupancy_esd;
   string refinement_flags;
   string restraints;
```

```
    long symmetry_multiplicity;
    string thermal_displace_type;
    float u_equiv_geom_mean;
    float u_equiv_geom_mean_esd;
    float u_iso_or_equiv;
    float u_iso_or_equiv_esd;
    string wyckoff_symbol;
};
typedef sequence<AtomSite> AtomSiteList;
typedef sequence<AtomSiteExt> AtomSiteExtList;

valuetype AtomSiteAnisotrop
{
    factory createAtomSiteAnisotrop();

    public Matrix3 b;
    public Matrix3 b_esd;
    public float ratio;
    public IndexId id;
    public IndexId type_symbol;
    public Matrix3 u;
    public Matrix3 u_esd;
};
typedef sequence<AtomSiteAnisotrop> AtomSiteAnisotropList;

valuetype AtomType
{
    factory createAtomType();

    public float analytical_mass_percent;
    public string description;
    public long number_in_cell;
    public long oxidation_number;
    public float radius_bond;
    public float radius_contact;
    public float scat_cromer_mann_a1;
    public float scat_cromer_mann_a2;
    public float scat_cromer_mann_a3;
    public float scat_cromer_mann_a4;
    public float scat_cromer_mann_b1;
    public float scat_cromer_mann_b2;
    public float scat_cromer_mann_b3;
    public float scat_cromer_mann_b4;
    public float scat_cromer_mann_c;
    public float scat_dispersion_imag;
    public float scat_dispersion_real;
    public string scat_length_neutron;
    public string scat_source;
    public string scat_versus_stol_list;
    public string symbol;
};
typedef sequence<AtomType> AtomTypeList;

valuetype ChemComp
```

```
{
   factory createChemComp();

   public string formula;
   public float formula_weight;
   public string id;
   public string model_details;
   public string model_ext_reference_file;
   public string model_source;
   public string mon_nstd_class;
   public string mon_nstd_details;
   public string mon_nstd_flag;
   public string mon_nstd_parent;
   public IndexId mon_nstd_parent_comp;
   public string name;
   public long number_atoms_all;
   public long number_atoms_nh;
   public string one_letter_code;
   public string three_letter_code;
   public string type;
};
typedef sequence<ChemComp> ChemCompList;

valuetype ChemCompAngle
{
   factory createChemCompAngle();

   public IndexId atom_id_1;
   public IndexId atom_id_2;
   public IndexId atom_id_3;
   public IndexId comp;
   public float value_angle;
   public float value_angle_esd;
   public float value_dist;
   public float value_dist_esd;
};
typedef sequence<ChemCompAngle> ChemCompAngleList;

valuetype ChemCompAtom
{
   factory createChemCompAtom();

   public string alt_atom_id;
   public string atom_id;
   public long charge;
   public VectorXYZ model_cartn;
   public VectorXYZ model_cartn_esd;
   public IndexId comp;
   public float partial_charge;
   public string substruct_code;
   public IndexId type_symbol;
};
typedef sequence<ChemCompAtom> ChemCompAtomList;
```

```
valuetype ChemCompBond
{
   factory createChemCompBond();

   public IndexId atom_id_1;
   public IndexId atom_id_2;
   public IndexId comp;
   public string value_order;
   public float value_dist;
   public float value_dist_esd;
};
typedef sequence<ChemCompBond> ChemCompBondList;

valuetype ChemCompChir
{
   factory createChemCompChir();

   public IndexId atom;
   public string atom_config;
   public string id;
   public IndexId comp;
   public long number_atoms_all;
   public long number_atoms_nh;
   public string volume_flag;
   public float volume_three;
   public float volume_three_esd;
};
typedef sequence<ChemCompChir> ChemCompChirList;

valuetype ChemCompChirAtom
{
   factory createChemCompChirAtom();

   public IndexId atom;
   public IndexId chir;
   public IndexId comp;
   public float dev;
};
typedef sequence<ChemCompChirAtom> ChemCompChirAtomList;

valuetype ChemCompLink
{
   factory createChemCompLink();

   public IndexId link;
   public string details;
   public IndexId type_comp_1;
   public IndexId type_comp_2;
};
typedef sequence<ChemCompLink> ChemCompLinkList;

valuetype ChemCompPlane
{
   factory createChemCompPlane();
```

```
    public string id;
    public IndexId comp;
    public long number_atoms_all;
    public long number_atoms_nh;
};
typedef sequence<ChemCompPlane> ChemCompPlaneList;

valuetype ChemCompPlaneAtom
{
    factory createChemCompPlaneAtom();

    public IndexId atom;
    public IndexId comp;
    public IndexId plane;
    public float dist_esd;
};
typedef sequence<ChemCompPlaneAtom> ChemCompPlaneAtomList;

valuetype ChemCompTor
{
    factory createChemCompTor();

    public IndexId atom_id_1;
    public IndexId atom_id_2;
    public IndexId atom_id_3;
    public IndexId atom_id_4;
    public string id;
    public IndexId comp;
};
typedef sequence<ChemCompTor> ChemCompTorList;

valuetype ChemCompTorValue
{
    factory createChemCompTorValue();

    public IndexId comp;
    public IndexId tor;
    public float angle;
    public float angle_esd;
    public float dist;
    public float dist_esd;
};
typedef sequence<ChemCompTorValue> ChemCompTorValueList;

valuetype ChemLink
{
    factory createChemLink();

    public string id;
    public string details;
};
typedef sequence<ChemLink> ChemLinkList;
```

```
valuetype ChemLinkAngle
{
   factory createChemLinkAngle();

   public string atom_1_comp_id;
   public string atom_2_comp_id;
   public string atom_3_comp_id;
   public string atom_id_1;
   public string atom_id_2;
   public string atom_id_3;
   public IndexId link;
   public float value_angle;
   public float value_angle_esd;
   public float value_dist;
   public float value_dist_esd;
};
typedef sequence<ChemLinkAngle> ChemLinkAngleList;

valuetype ChemLinkBond
{
   factory createChemLinkBond();

   public string atom_1_comp_id;
   public string atom_2_comp_id;
   public string atom_id_1;
   public string atom_id_2;
   public IndexId link;
   public float value_dist;
   public float value_dist_esd;
   public string value_order;
};
typedef sequence<ChemLinkBond> ChemLinkBondList;

valuetype ChemLinkChir
{
   factory createChemLinkChir();

   public string atom_comp_id;
   public string atom_id;
   public string atom_config;
   public string id;
   public IndexId link;
   public long number_atoms_all;
   public long number_atoms_nh;
   public string volume_flag;
   public float volume_three;
   public float volume_three_esd;
};
typedef sequence<ChemLinkChir> ChemLinkChirList;

valuetype ChemLinkChirAtom
{
   factory createChemLinkChirAtom();
```

```
    public string atom_comp_id;
    public string atom_id;
    public IndexId chir;
    public float dev;
};
typedef sequence<ChemLinkChirAtom> ChemLinkChirAtomList;

valuetype ChemLinkPlane
{
    factory createChemLinkPlane();

    public string id;
    public IndexId link;
    public long number_atoms_all;
    public long number_atoms_nh;
};
typedef sequence<ChemLinkPlane> ChemLinkPlaneList;

valuetype ChemLinkPlaneAtom
{
    factory createChemLinkPlaneAtom();

    public string atom_comp_id;
    public string atom_id;
    public IndexId plane;
};
typedef sequence<ChemLinkPlaneAtom> ChemLinkPlaneAtomList;

valuetype ChemLinkTor
{
    factory createChemLinkTor();

    public string atom_1_comp_id;
    public string atom_2_comp_id;
    public string atom_3_comp_id;
    public string atom_4_comp_id;
    public string atom_id_1;
    public string atom_id_2;
    public string atom_id_3;
    public string atom_id_4;
    public string id;
    public IndexId link;
};
typedef sequence<ChemLinkTor> ChemLinkTorList;

valuetype ChemLinkTorValue
{
    factory createChemLinkTorValue();

    public IndexId tor;
    public float angle;
    public float angle_esd;
    public float dist;
    public float dist_esd;
```

```
};
typedef sequence<ChemLinkTorValue> ChemLinkTorValueList;

valuetype Entity
{
   factory createEntity();

   public string details;
   public float formula_weight;
   public string id;
   public string src_method;
   public string type;
};
typedef sequence<Entity> EntityList;

valuetype EntityKeywords
{
   factory createEntityKeywords();

   public IndexId entity;
   public string text;
};
typedef sequence<EntityKeywords> EntityKeywordsList;

valuetype EntityLink
{
   factory createEntityLink();

   public IndexId link;
   public string details;
   public IndexId entity_id_1;
   public IndexId entity_id_2;
   public IndexId entity_seq_num_1;
   public IndexId entity_seq_num_2;
};
typedef sequence<EntityLink> EntityLinkList;

valuetype EntityNameCom
{
   factory createEntityNameCom();

   public IndexId entity;
   public string name;
};
typedef sequence<EntityNameCom> EntityNameComList;

valuetype EntityNameSys
{
   factory createEntityNameSys();

   public IndexId entity;
   public string name;
   public string system;
};
```

```
typedef sequence<EntityNameSys> EntityNameSysList;

valuetype EntityPoly
{
   factory createEntityPoly();

   public IndexId entity;
   public string nstd_chirality;
   public string nstd_linkage;
   public string nstd_monomer;
   public long number_of_monomers;
   public string type;
   public string type_details;
};
typedef sequence<EntityPoly> EntityPolyList;

struct EntityPolySeq
{
   IndexId entity;
   string hetero;
   IndexId mon;
   long num;
};
typedef sequence<EntityPolySeq> EntityPolySeqList;

valuetype EntitySrcGen
{
   factory createEntitySrcGen();

   public IndexId entity;
   public string gene_src_common_name;
   public string gene_src_details;
   public string gene_src_genus;
   public string gene_src_species;
   public string gene_src_strain;
   public string gene_src_tissue;
   public string gene_src_tissue_fraction;
   public string host_org_common_name;
   public string host_org_details;
   public string host_org_genus;
   public string host_org_species;
   public string host_org_strain;
   public string plasmid_details;
   public string plasmid_name;
};
typedef sequence<EntitySrcGen> EntitySrcGenList;

valuetype EntitySrcNat
{
   factory createEntitySrcNat();

   public string common_name;
   public string details;
   public IndexId entity;
```

```
    public string genus;
    public string species;
    public string strain;
    public string tissue;
    public string tissue_fraction;
};
typedef sequence<EntitySrcNat> EntitySrcNatList;

valuetype EntryLink
{
    factory createEntryLink();

    public EntryId entry_id;
    public string id;
    public string details;
};
typedef sequence<EntryLink> EntryLinkList;

valuetype Geom
{
    factory createGeom();

    public EntryId entry_id;
    public string details;
};
typedef sequence<Geom> GeomList;

valuetype GeomAngle
{
    factory createGeomAngle();

    public IndexId atom_site_id_1;
    public AtomIndex atom_site_label_1;
    public IndexId atom_site_id_2;
    public AtomIndex atom_site_label_2;
    public IndexId atom_site_id_3;
    public AtomIndex atom_site_label_3;
    public AtomIndex atom_site_auth_1;
    public AtomIndex atom_site_auth_2;
    public AtomIndex atom_site_auth_3;
    public string publ_flag;
    public string site_symmetry_1;
    public string site_symmetry_2;
    public string site_symmetry_3;
    public float value;
    public float value_esd;
};
typedef sequence<GeomAngle> GeomAngleList;

valuetype GeomBond
{
    factory createGeomBond();

    public IndexId atom_site_id_1;
```

```
   public AtomIndex atom_site_label_1;
   public IndexId atom_site_id_2;
   public AtomIndex atom_site_label_2;
   public AtomIndex atom_site_auth_1;
   public AtomIndex atom_site_auth_2;
   public float dist;
   public float dist_esd;
   public string publ_flag;
   public string site_symmetry_1;
   public string site_symmetry_2;
};
typedef sequence<GeomBond> GeomBondList;

valuetype GeomContact
{
   factory createGeomContact();

   public IndexId atom_site_id_1;
   public AtomIndex atom_site_label_1;
   public IndexId atom_site_id_2;
   public AtomIndex atom_site_label_2;
   public AtomIndex atom_site_auth_1;
   public AtomIndex atom_site_auth_2;
   public float dist;
   public float dist_esd;
   public string publ_flag;
   public string site_symmetry_1;
   public string site_symmetry_2;
};
typedef sequence<GeomContact> GeomContactList;

valuetype GeomHbond
{
   factory createGeomHbond();

   public float angle_dha;
   public float angle_dha_esd;
   public string atom_site_id_a;
   public AtomIndex atom_site_label_a;
   public string atom_site_id_d;
   public AtomIndex atom_site_label_d;
   public string atom_site_id_h;
   public AtomIndex atom_site_label_h;
   public AtomIndex atom_site_auth_a;
   public AtomIndex atom_site_auth_d;
   public AtomIndex atom_site_auth_h;
   public float dist_da;
   public float dist_da_esd;
   public float dist_dh;
   public float dist_dh_esd;
   public float dist_ha;
   public float dist_ha_esd;
   public string publ_flag;
   public string site_symmetry_a;
```

```
   public string site_symmetry_d;
   public string site_symmetry_h;
};
typedef sequence<GeomHbond> GeomHbondList;

valuetype GeomTorsion
{
   factory createGeomTorsion();

   public IndexId atom_site_id_1;
   public AtomIndex atom_site_label_1;
   public IndexId atom_site_id_2;
   public AtomIndex atom_site_label_2;
   public IndexId atom_site_id_3;
   public AtomIndex atom_site_label_3;
   public IndexId atom_site_id_4;
   public AtomIndex atom_site_label_4;
   public AtomIndex atom_site_auth_1;
   public AtomIndex atom_site_auth_2;
   public AtomIndex atom_site_auth_3;
   public AtomIndex atom_site_auth_4;
   public string publ_flag;
   public string site_symmetry_1;
   public string site_symmetry_2;
   public string site_symmetry_3;
   public string site_symmetry_4;
   public float value;
   public float value_esd;
};
typedef sequence<GeomTorsion> GeomTorsionList;

valuetype Structure
{
   factory createStructure();

   public EntryId entry_id;
   public string title;
};
typedef sequence<Structure> StructureList;

valuetype StructAsym
{
   factory createStructAsym();

   public string details;
   public IndexId entity;
   public string id;
};
typedef sequence<StructAsym> StructAsymList;

valuetype StructBiol
{
   factory createStructBiol();
```

```
    public string details;
    public string id;
};
typedef sequence<StructBiol> StructBiolList;

valuetype StructBiolGen
{
    factory createStructBiolGen();

    public IndexId asym;
    public IndexId biol;
    public string details;
    public string symmetry;
};
typedef sequence<StructBiolGen> StructBiolGenList;

valuetype StructBiolKeywords
{
    factory createStructBiolKeywords();

    public IndexId biol;
    public string text;
};
typedef sequence<StructBiolKeywords> StructBiolKeywordsList;

valuetype StructBiolView
{
    factory createStructBiolView();

    public IndexId biol;
    public string details;
    public string id;
    public Matrix3 rot_matrix;
};
typedef sequence<StructBiolView> StructBiolViewList;

valuetype StructConf
{
    factory createStructConf();

    public SeqIndex beg_label;
    public SeqIndex beg_auth;
    public IndexId conf_type;
    public string details;
    public SeqIndex end_label;
    public SeqIndex end_auth;
    public string id;
};
typedef sequence<StructConf> StructConfList;

valuetype StructConfType
{
    factory createStructConfType();
```

```
    public string criteria;
    public string id;
    public string reference;
};
typedef sequence<StructConfType> StructConfTypeList;

struct StructConn
{
    IndexId conn_type;
    string details;
    string id;
    AtomIndex ptnr1_label;
    AtomIndex ptnr1_auth;
    string ptnr1_role;
    string ptnr1_symmetry;
    AtomIndex ptnr2_label;
    AtomIndex ptnr2_auth;
    string ptnr2_role;
    string ptnr2_symmetry;
};
typedef sequence<StructConn> StructConnList;

valuetype StructConnType
{
    factory createStructConnType();

    public string criteria;
    public string id;
    public string reference;
};
typedef sequence<StructConnType> StructConnTypeList;

valuetype StructKeywords
{
    factory createStructKeywords();

    public EntryId entry_id;
    public string text;
};
typedef sequence<StructKeywords> StructKeywordsList;

valuetype StructMonDetails
{
    factory createStructMonDetails();

    public EntryId entry_id;
    public float prot_cis;
    public string rscc;
    public string rsr;
};
typedef sequence<StructMonDetails> StructMonDetailsList;

valuetype StructMonNucl
{
```

```
    factory createStructMonNucl();

    public float alpha;
    public float beta;
    public float chi1;
    public float chi2;
    public float delta;
    public float details;
    public float epsilon;
    public float gamma;
    public SeqIndex label;
    public SeqIndex auth;
    public float mean_b_all;
    public float mean_b_base;
    public float mean_b_phos;
    public float mean_b_sugar;
    public float nu0;
    public float nu1;
    public float nu2;
    public float nu3;
    public float nu4;
    public float p;
    public float rscc_all;
    public float rscc_base;
    public float rscc_phos;
    public float rscc_sugar;
    public float rsr_all;
    public float rsr_base;
    public float rsr_phos;
    public float rsr_sugar;
    public float tau0;
    public float tau1;
    public float tau2;
    public float tau3;
    public float tau4;
    public float taum;
    public float zeta;
};
typedef sequence<StructMonNucl> StructMonNuclList;

valuetype StructMonProt
{
    factory createStructMonProt();

    public float chi1;
    public float chi2;
    public float chi3;
    public float chi4;
    public float chi5;
    public float details;
    public SeqIndex label;
    public SeqIndex auth;
    public float rscc_all;
    public float rscc_main;
```

```
    public float rscc_side;
    public float rsr_all;
    public float rsr_main;
    public float rsr_side;
    public float mean_b_all;
    public float mean_b_main;
    public float mean_b_side;
    public float omega;
    public float phi;
    public float psi;
};
typedef sequence<StructMonProt> StructMonProtList;

valuetype StructMonProtCis
{
    factory createStructMonProtCis();

    public SeqIndex label;
    public SeqIndex auth;
};
typedef sequence<StructMonProtCis> StructMonProtCisList;

valuetype StructNcsDom
{
    factory createStructNcsDom();

    public string details;
    public string id;
};
typedef sequence<StructNcsDom> StructNcsDomList;

valuetype StructNcsDomLim
{
    factory createStructNcsDomLim();

    public SeqIndex beg_label;
    public SeqIndex beg_auth;
    public IndexId dom;
    public SeqIndex end_label;
    public SeqIndex end_auth;
};
typedef sequence<StructNcsDomLim> StructNcsDomLimList;

valuetype StructNcsEns
{
    factory createStructNcsEns();

    public string details;
    public string id;
    public string point_group;
};
typedef sequence<StructNcsEns> StructNcsEnsList;

valuetype StructNcsEnsGen
```

```
{
   factory createStructNcsEnsGen();

   public IndexId dom_id_1;
   public IndexId dom_id_2;
   public IndexId ens;
   public IndexId oper;
};
typedef sequence<StructNcsEnsGen> StructNcsEnsGenList;

valuetype StructNcsOper
{
   factory createStructNcsOper();

   public string code;
   public string details;
   public string id;
   public Matrix3 matrix;
   public Vector3 vector;
};
typedef sequence<StructNcsOper> StructNcsOperList;

valuetype StructRef
{
   factory createStructRef();

   public IndexId biol;
   public string db_code;
   public string db_name;
   public string details;
   public IndexId entity;
   public string id;
   public string seq_align;
   public string seq_dif;
};
typedef sequence<StructRef> StructRefList;

valuetype StructRefSeq
{
   factory createStructRefSeq();

   public string align_id;
   public long db_align_beg;
   public long db_align_end;
   public string details;
   public IndexId ref;
   public IndexId seq_align_beg;
   public IndexId seq_align_end;
};
typedef sequence<StructRefSeq> StructRefSeqList;

valuetype StructRefSeqDif
{
   factory createStructRefSeqDif();
```

```
    public IndexId align;
    public IndexId db_mon;
    public string details;
    public IndexId mon;
    public IndexId seq_num;
};
typedef sequence<StructRefSeqDif> StructRefSeqDifList;

valuetype StructSheet
{
    factory createStructSheet();

    public string details;
    public string id;
    public long number_strands;
    public string type;
};
typedef sequence<StructSheet> StructSheetList;

valuetype StructSheetHbond
{
    factory createStructSheetHbond();

    public IndexId range_1_beg_label_atom;
    public IndexId range_1_beg_label_seq;
    public IndexId range_1_end_label_atom;
    public IndexId range_1_end_label_seq;
    public IndexId range_2_beg_label_atom;
    public IndexId range_2_beg_label_seq;
    public IndexId range_2_end_label_atom;
    public IndexId range_2_end_label_seq;
    public IndexId range_1_beg_auth_atom;
    public IndexId range_1_beg_auth_seq;
    public IndexId range_1_end_auth_atom;
    public IndexId range_1_end_auth_seq;
    public IndexId range_2_beg_auth_atom;
    public IndexId range_2_beg_auth_seq;
    public IndexId range_2_end_auth_atom;
    public IndexId range_2_end_auth_seq;
    public IndexId range_id_1;
    public IndexId range_id_2;
    public IndexId sheet;
};
typedef sequence<StructSheetHbond> StructSheetHbondList;

valuetype StructSheetOrder
{
    factory createStructSheetOrder();

    public long offset;
    public IndexId range_id_1;
    public IndexId range_id_2;
    public string sense;
```

```
   public IndexId sheet;
};
typedef sequence<StructSheetOrder> StructSheetOrderList;

valuetype StructSheetRange
{
   factory createStructSheetRange();

   public SeqIndex beg_label;
   public SeqIndex beg_auth;
   public SeqIndex end_label;
   public SeqIndex end_auth;
   public string id;
   public IndexId sheet;
   public string symmetry;
};
typedef sequence<StructSheetRange> StructSheetRangeList;

valuetype StructSheetTopology
{
   factory createStructSheetTopology();

   public long offset;
   public IndexId range_id_1;
   public IndexId range_id_2;
   public string sense;
   public IndexId sheet;
};
typedef sequence<StructSheetTopology> StructSheetTopologyList;

valuetype StructSite
{
   factory createStructSite();

   public string details;
   public string id;
};
typedef sequence<StructSite> StructSiteList;

valuetype StructSiteGen
{
   factory createStructSiteGen();

   public string details;
   public string id;
   public AtomIndex label;
   public AtomIndex auth;
   public IndexId site;
   public string symmetry;
};
typedef sequence<StructSiteGen> StructSiteGenList;

valuetype StructSiteKeywords
{
```

```
    factory createStructSiteKeywords();

    public IndexId site;
    public string text;
};
typedef sequence<StructSiteKeywords> StructSiteKeywordsList;

valuetype StructSiteView
{
    factory createStructSiteView();

    public string details;
    public string id;
    public Matrix3 rot_matrix;
    public IndexId site;
};
typedef sequence<StructSiteView> StructSiteViewList;

typedef sequence<octet> Flags;

interface Entry
{
    Flags get_presence_flags()
          raises (DataAccessException);
    CosPropertyService::Properties get_subentry_list()
          raises (DataAccessException);

    const short S_ATOM_SITE = 1;
    const short F_ATOM_SITE_LABEL_ATOM_ID = 2;
    const short F_ATOM_SITE_LABEL_SEQ_ID = 3;
    const short F_ATOM_SITE_LABEL_COMP_ID = 4;
    const short F_ATOM_SITE_LABEL_ASYM_ID = 5;
    const short F_ATOM_SITE_LABEL_ALT_ID = 6;
    const short F_ATOM_SITE_LABEL_ENTITY_ID = 7;
    const short F_ATOM_SITE_CARTN_X = 8;
    const short F_ATOM_SITE_CARTN_Y = 9;
    const short F_ATOM_SITE_CARTN_Z = 10;
    const short F_ATOM_SITE_OCCUPANCY = 11;
    const short F_ATOM_SITE_B_ISO_OR_EQUIV = 12;

    const short S_ATOM_SITE_EXT = 13;
    const short F_ATOM_SITE_EXT_ANISO_B = 14;
    const short F_ATOM_SITE_EXT_ANISO_B_ESD = 15;
    const short F_ATOM_SITE_EXT_ANISO_RATIO = 16;
    const short F_ATOM_SITE_EXT_ANISO_U = 17;
    const short F_ATOM_SITE_EXT_ANISO_U_ESD = 18;
    const short F_ATOM_SITE_EXT_ATTACHED_HYDROGENS = 19;
    const short F_ATOM_SITE_EXT_AUTH_ASYM_ID = 20;
    const short F_ATOM_SITE_EXT_AUTH_ATOM_ID = 21;
    const short F_ATOM_SITE_EXT_AUTH_COMP_ID = 22;
    const short F_ATOM_SITE_EXT_AUTH_SEQ_ID = 23;
    const short F_ATOM_SITE_EXT_B_EQUIV_GEOM_MEAN = 24;
    const short F_ATOM_SITE_EXT_B_EQUIV_GEOM_MEAN_ESD = 25;
    const short F_ATOM_SITE_EXT_B_ISO_OR_EQUIV_ESD = 26;
```

```
const short F_ATOM_SITE_EXT_CALC_ATTACHED_ATOM = 27;
const short F_ATOM_SITE_EXT_CALC_FLAG = 28;
const short F_ATOM_SITE_EXT_CARTN_ESD_X = 29;
const short F_ATOM_SITE_EXT_CARTN_ESD_Y = 30;
const short F_ATOM_SITE_EXT_CARTN_ESD_Z = 31;
const short F_ATOM_SITE_EXT_CONSTRAINTS = 32;
const short F_ATOM_SITE_EXT_DETAILS = 33;
const short F_ATOM_SITE_EXT_DISORDER_GROUP = 34;
const short F_ATOM_SITE_EXT_FOOTNOTE_ID = 35;
const short F_ATOM_SITE_EXT_FRACT_X = 36;
const short F_ATOM_SITE_EXT_FRACT_Y = 37;
const short F_ATOM_SITE_EXT_FRACT_Z = 38;
const short F_ATOM_SITE_EXT_FRACT_ESD_X = 39;
const short F_ATOM_SITE_EXT_FRACT_ESD_Y = 40;
const short F_ATOM_SITE_EXT_FRACT_ESD_Z = 41;
const short F_ATOM_SITE_EXT_OCCUPANCY_ESD = 42;
const short F_ATOM_SITE_EXT_REFINEMENT_FLAGS = 43;
const short F_ATOM_SITE_EXT_RESTRAINTS = 44;
const short F_ATOM_SITE_EXT_SYMMETRY_MULTIPLICITY = 45;
const short F_ATOM_SITE_EXT_THERMAL_DISPLACE_TYPE = 46;
const short F_ATOM_SITE_EXT_U_EQUIV_GEOM_MEAN = 47;
const short F_ATOM_SITE_EXT_U_EQUIV_GEOM_MEAN_ESD = 48;
const short F_ATOM_SITE_EXT_U_ISO_OR_EQUIV = 49;
const short F_ATOM_SITE_EXT_U_ISO_OR_EQUIV_ESD = 50;
const short F_ATOM_SITE_EXT_WYCKOFF_SYMBOL = 51;

const short S_ATOM_SITE_ANISOTROP = 52;
const short F_ATOM_SITE_ANISOTROP_B = 53;
const short F_ATOM_SITE_ANISOTROP_B_ESD = 54;
const short F_ATOM_SITE_ANISOTROP_RATIO = 55;
const short F_ATOM_SITE_ANISOTROP_U = 56;
const short F_ATOM_SITE_ANISOTROP_U_ESD = 57;

const short S_ATOM_TYPE = 58;
const short F_ATOM_TYPE_ANALYTICAL_MASS_PERCENT = 59;
const short F_ATOM_TYPE_DESCRIPTION = 60;
const short F_ATOM_TYPE_NUMBER_IN_CELL = 61;
const short F_ATOM_TYPE_OXIDATION_NUMBER = 62;
const short F_ATOM_TYPE_RADIUS_BOND = 63;
const short F_ATOM_TYPE_RADIUS_CONTACT = 64;
const short F_ATOM_TYPE_SCAT_CROMER_MANN_A1 = 65;
const short F_ATOM_TYPE_SCAT_CROMER_MANN_A2 = 66;
const short F_ATOM_TYPE_SCAT_CROMER_MANN_A3 = 67;
const short F_ATOM_TYPE_SCAT_CROMER_MANN_A4 = 68;
const short F_ATOM_TYPE_SCAT_CROMER_MANN_B1 = 69;
const short F_ATOM_TYPE_SCAT_CROMER_MANN_B2 = 70;
const short F_ATOM_TYPE_SCAT_CROMER_MANN_B3 = 71;
const short F_ATOM_TYPE_SCAT_CROMER_MANN_B4 = 72;
const short F_ATOM_TYPE_SCAT_CROMER_MANN_C = 73;
const short F_ATOM_TYPE_SCAT_DISPERSION_IMAG = 74;
const short F_ATOM_TYPE_SCAT_DISPERSION_REAL = 75;
const short F_ATOM_TYPE_SCAT_LENGTH_NEUTRON = 76;
const short F_ATOM_TYPE_SCAT_SOURCE = 77;
const short F_ATOM_TYPE_SCAT_VERSUS_STOL_LIST = 78;
```

```
const short S_CHEM_COMP = 79;
const short F_CHEM_COMP_FORMULA = 80;
const short F_CHEM_COMP_FORMULA_WEIGHT = 81;
const short F_CHEM_COMP_MODEL_DETAILS = 82;
const short F_CHEM_COMP_MODEL_EXT_REFERENCE_FILE = 83;
const short F_CHEM_COMP_MODEL_SOURCE = 84;
const short F_CHEM_COMP_MON_NSTD_CLASS = 85;
const short F_CHEM_COMP_MON_NSTD_DETAILS = 86;
const short F_CHEM_COMP_MON_NSTD_FLAG = 87;
const short F_CHEM_COMP_MON_NSTD_PARENT = 88;
const short F_CHEM_COMP_MON_NSTD_PARENT_COMP_ID = 89;
const short F_CHEM_COMP_NAME = 90;
const short F_CHEM_COMP_NUMBER_ATOMS_ALL = 91;
const short F_CHEM_COMP_NUMBER_ATOMS_NH = 92;
const short F_CHEM_COMP_ONE_LETTER_CODE = 93;
const short F_CHEM_COMP_THREE_LETTER_CODE = 94;

const short S_CHEM_COMP_ANGLE = 95;
const short F_CHEM_COMP_ANGLE_VALUE_ANGLE = 96;
const short F_CHEM_COMP_ANGLE_VALUE_ANGLE_ESD = 97;
const short F_CHEM_COMP_ANGLE_VALUE_DIST = 98;
const short F_CHEM_COMP_ANGLE_VALUE_DIST_ESD = 99;

const short S_CHEM_COMP_ATOM = 100;
const short F_CHEM_COMP_ATOM_ALT_ATOM_ID = 101;
const short F_CHEM_COMP_ATOM_CHARGE = 102;
const short F_CHEM_COMP_ATOM_MODEL_CARTN_X = 103;
const short F_CHEM_COMP_ATOM_MODEL_CARTN_Y = 104;
const short F_CHEM_COMP_ATOM_MODEL_CARTN_Z = 105;
const short F_CHEM_COMP_ATOM_MODEL_CARTN_ESD_X = 106;
const short F_CHEM_COMP_ATOM_MODEL_CARTN_ESD_Y = 107;
const short F_CHEM_COMP_ATOM_MODEL_CARTN_ESD_Z = 108;
const short F_CHEM_COMP_ATOM_PARTIAL_CHARGE = 109;
const short F_CHEM_COMP_ATOM_SUBSTRUCT_CODE = 110;

const short S_CHEM_COMP_BOND = 111;
const short F_CHEM_COMP_BOND_VALUE_ORDER = 112;
const short F_CHEM_COMP_BOND_VALUE_DIST = 113;
const short F_CHEM_COMP_BOND_VALUE_DIST_ESD = 114;

const short S_CHEM_COMP_CHIR = 115;
const short F_CHEM_COMP_CHIR_ATOM_CONFIG = 116;
const short F_CHEM_COMP_CHIR_NUMBER_ATOMS_ALL = 117;
const short F_CHEM_COMP_CHIR_NUMBER_ATOMS_NH = 118;
const short F_CHEM_COMP_CHIR_VOLUME_FLAG = 119;
const short F_CHEM_COMP_CHIR_VOLUME_THREE = 120;
const short F_CHEM_COMP_CHIR_VOLUME_THREE_ESD = 121;

const short S_CHEM_COMP_CHIR_ATOM = 122;
const short F_CHEM_COMP_CHIR_ATOM_DEV = 123;

const short S_CHEM_COMP_LINK = 124;
const short F_CHEM_COMP_LINK_DETAILS = 125;
```

```
const short S_CHEM_COMP_PLANE = 126;
const short F_CHEM_COMP_PLANE_NUMBER_ATOMS_ALL = 127;
const short F_CHEM_COMP_PLANE_NUMBER_ATOMS_NH = 128;

const short S_CHEM_COMP_PLANE_ATOM = 129;
const short F_CHEM_COMP_PLANE_ATOM_DIST_ESD = 130;

const short S_CHEM_COMP_TOR = 131;

const short S_CHEM_COMP_TOR_VALUE = 132;
const short F_CHEM_COMP_TOR_VALUE_DIST = 133;
const short F_CHEM_COMP_TOR_VALUE_DIST_ESD = 134;

const short S_CHEM_LINK = 135;
const short F_CHEM_LINK_DETAILS = 136;

const short S_CHEM_LINK_ANGLE = 137;
const short F_CHEM_LINK_ANGLE_ATOM_1_COMP_ID = 138;
const short F_CHEM_LINK_ANGLE_ATOM_2_COMP_ID = 139;
const short F_CHEM_LINK_ANGLE_ATOM_3_COMP_ID = 140;
const short F_CHEM_LINK_ANGLE_VALUE_ANGLE = 141;
const short F_CHEM_LINK_ANGLE_VALUE_ANGLE_ESD = 142;
const short F_CHEM_LINK_ANGLE_VALUE_DIST = 143;
const short F_CHEM_LINK_ANGLE_VALUE_DIST_ESD = 144;

const short S_CHEM_LINK_BOND = 145;
const short F_CHEM_LINK_BOND_ATOM_1_COMP_ID = 146;
const short F_CHEM_LINK_BOND_ATOM_2_COMP_ID = 147;
const short F_CHEM_LINK_BOND_VALUE_DIST = 148;
const short F_CHEM_LINK_BOND_VALUE_DIST_ESD = 149;
const short F_CHEM_LINK_BOND_VALUE_ORDER = 150;

const short S_CHEM_LINK_CHIR = 151;
const short F_CHEM_LINK_CHIR_ATOM_COMP_ID = 152;
const short F_CHEM_LINK_CHIR_ATOM_CONFIG = 153;
const short F_CHEM_LINK_CHIR_NUMBER_ATOMS_ALL = 154;
const short F_CHEM_LINK_CHIR_NUMBER_ATOMS_NH = 155;
const short F_CHEM_LINK_CHIR_VOLUME_FLAG = 156;
const short F_CHEM_LINK_CHIR_VOLUME_THREE = 157;
const short F_CHEM_LINK_CHIR_VOLUME_THREE_ESD = 158;

const short S_CHEM_LINK_CHIR_ATOM = 159;
const short F_CHEM_LINK_CHIR_ATOM_ATOM_COMP_ID = 160;
const short F_CHEM_LINK_CHIR_ATOM_DEV = 161;

const short S_CHEM_LINK_PLANE = 162;
const short F_CHEM_LINK_PLANE_NUMBER_ATOMS_ALL = 163;
const short F_CHEM_LINK_PLANE_NUMBER_ATOMS_NH = 164;

const short S_CHEM_LINK_PLANE_ATOM = 165;
const short F_CHEM_LINK_PLANE_ATOM_ATOM_COMP_ID = 166;

const short S_CHEM_LINK_TOR = 167;
```

```
const short F_CHEM_LINK_TOR_ATOM_1_COMP_ID = 168;
const short F_CHEM_LINK_TOR_ATOM_2_COMP_ID = 169;
const short F_CHEM_LINK_TOR_ATOM_3_COMP_ID = 170;
const short F_CHEM_LINK_TOR_ATOM_4_COMP_ID = 171;

const short S_CHEM_LINK_TOR_VALUE = 172;
const short F_CHEM_LINK_TOR_VALUE_DIST = 173;
const short F_CHEM_LINK_TOR_VALUE_DIST_ESD = 174;

const short S_ENTITY = 175;
const short F_ENTITY_DETAILS = 176;
const short F_ENTITY_FORMULA_WEIGHT = 177;
const short F_ENTITY_SRC_METHOD = 178;
const short F_ENTITY_TYPE = 179;

const short S_ENTITY_KEYWORDS = 180;

const short S_ENTITY_LINK = 181;
const short F_ENTITY_LINK_DETAILS = 182;
const short F_ENTITY_LINK_ENTITY_SEQ_NUM_1_ID = 183;
const short F_ENTITY_LINK_ENTITY_SEQ_NUM_2_ID = 184;

const short S_ENTITY_NAME_COM = 185;

const short S_ENTITY_NAME_SYS = 186;
const short F_ENTITY_NAME_SYS_SYSTEM = 187;

const short S_ENTITY_POLY = 188;
const short F_ENTITY_POLY_NSTD_CHIRALITY = 189;
const short F_ENTITY_POLY_NSTD_LINKAGE = 190;
const short F_ENTITY_POLY_NSTD_MONOMER = 191;
const short F_ENTITY_POLY_NUMBER_OF_MONOMERS = 192;
const short F_ENTITY_POLY_TYPE = 193;
const short F_ENTITY_POLY_TYPE_DETAILS = 194;

const short S_ENTITY_POLY_SEQ = 195;
const short F_ENTITY_POLY_SEQ_HETERO = 196;

const short S_ENTITY_SRC_GEN = 197;
const short F_ENTITY_SRC_GEN_GENE_SRC_COMMON_NAME = 198;
const short F_ENTITY_SRC_GEN_GENE_SRC_DETAILS = 199;
const short F_ENTITY_SRC_GEN_GENE_SRC_GENUS = 200;
const short F_ENTITY_SRC_GEN_GENE_SRC_SPECIES = 201;
const short F_ENTITY_SRC_GEN_GENE_SRC_STRAIN = 202;
const short F_ENTITY_SRC_GEN_GENE_SRC_TISSUE = 203;
const short F_ENTITY_SRC_GEN_GENE_SRC_TISSUE_FRACTION = 204;
const short F_ENTITY_SRC_GEN_HOST_ORG_COMMON_NAME = 205;
const short F_ENTITY_SRC_GEN_HOST_ORG_DETAILS = 206;
const short F_ENTITY_SRC_GEN_HOST_ORG_GENUS = 207;
const short F_ENTITY_SRC_GEN_HOST_ORG_SPECIES = 208;
const short F_ENTITY_SRC_GEN_HOST_ORG_STRAIN = 209;
const short F_ENTITY_SRC_GEN_PLASMID_DETAILS = 210;
const short F_ENTITY_SRC_GEN_PLASMID_NAME = 211;
```

```
const short S_ENTITY_SRC_NAT = 212;
const short F_ENTITY_SRC_NAT_DETAILS = 213;

const short S_ENTRY_LINK = 214;
const short F_ENTRY_LINK_DETAILS = 215;

const short S_GEOM = 216;
const short F_GEOM_DETAILS = 217;

const short S_GEOM_ANGLE = 218;
const short F_GEOM_ANGLE_ATOM_SITE_LABEL_1_ATOM_ID = 219;
const short F_GEOM_ANGLE_ATOM_SITE_LABEL_1_SEQ_ID = 220;
const short F_GEOM_ANGLE_ATOM_SITE_LABEL_1_COMP_ID = 221;
const short F_GEOM_ANGLE_ATOM_SITE_LABEL_1_ASYM_ID = 222;
const short F_GEOM_ANGLE_ATOM_SITE_LABEL_1_ALT_ID = 223;
const short F_GEOM_ANGLE_ATOM_SITE_LABEL_2_ATOM_ID = 224;
const short F_GEOM_ANGLE_ATOM_SITE_LABEL_2_SEQ_ID = 225;
const short F_GEOM_ANGLE_ATOM_SITE_LABEL_2_COMP_ID = 226;
const short F_GEOM_ANGLE_ATOM_SITE_LABEL_2_ASYM_ID = 227;
const short F_GEOM_ANGLE_ATOM_SITE_LABEL_2_ALT_ID = 228;
const short F_GEOM_ANGLE_ATOM_SITE_LABEL_3_ATOM_ID = 229;
const short F_GEOM_ANGLE_ATOM_SITE_LABEL_3_SEQ_ID = 230;
const short F_GEOM_ANGLE_ATOM_SITE_LABEL_3_COMP_ID = 231;
const short F_GEOM_ANGLE_ATOM_SITE_LABEL_3_ASYM_ID = 232;
const short F_GEOM_ANGLE_ATOM_SITE_LABEL_3_ALT_ID = 233;
const short F_GEOM_ANGLE_ATOM_SITE_AUTH_1_ATOM_ID = 234;
const short F_GEOM_ANGLE_ATOM_SITE_AUTH_1_SEQ_ID = 235;
const short F_GEOM_ANGLE_ATOM_SITE_AUTH_1_COMP_ID = 236;
const short F_GEOM_ANGLE_ATOM_SITE_AUTH_1_ASYM_ID = 237;
const short F_GEOM_ANGLE_ATOM_SITE_AUTH_2_ATOM_ID = 238;
const short F_GEOM_ANGLE_ATOM_SITE_AUTH_2_SEQ_ID = 239;
const short F_GEOM_ANGLE_ATOM_SITE_AUTH_2_COMP_ID = 240;
const short F_GEOM_ANGLE_ATOM_SITE_AUTH_2_ASYM_ID = 241;
const short F_GEOM_ANGLE_ATOM_SITE_AUTH_3_ATOM_ID = 242;
const short F_GEOM_ANGLE_ATOM_SITE_AUTH_3_SEQ_ID = 243;
const short F_GEOM_ANGLE_ATOM_SITE_AUTH_3_COMP_ID = 244;
const short F_GEOM_ANGLE_ATOM_SITE_AUTH_3_ASYM_ID = 245;
const short F_GEOM_ANGLE_PUBL_FLAG = 246;
const short F_GEOM_ANGLE_VALUE = 247;
const short F_GEOM_ANGLE_VALUE_ESD = 248;

const short S_GEOM_BOND = 249;
const short F_GEOM_BOND_ATOM_SITE_LABEL_1_ATOM_ID = 250;
const short F_GEOM_BOND_ATOM_SITE_LABEL_1_SEQ_ID = 251;
const short F_GEOM_BOND_ATOM_SITE_LABEL_1_COMP_ID = 252;
const short F_GEOM_BOND_ATOM_SITE_LABEL_1_ASYM_ID = 253;
const short F_GEOM_BOND_ATOM_SITE_LABEL_1_ALT_ID = 254;
const short F_GEOM_BOND_ATOM_SITE_LABEL_2_ATOM_ID = 255;
const short F_GEOM_BOND_ATOM_SITE_LABEL_2_SEQ_ID = 256;
const short F_GEOM_BOND_ATOM_SITE_LABEL_2_COMP_ID = 257;
const short F_GEOM_BOND_ATOM_SITE_LABEL_2_ASYM_ID = 258;
const short F_GEOM_BOND_ATOM_SITE_LABEL_2_ALT_ID = 259;
const short F_GEOM_BOND_ATOM_SITE_AUTH_1_ATOM_ID = 260;
const short F_GEOM_BOND_ATOM_SITE_AUTH_1_SEQ_ID = 261;
```

```
const short F_GEOM_BOND_ATOM_SITE_AUTH_1_COMP_ID = 262;
const short F_GEOM_BOND_ATOM_SITE_AUTH_1_ASYM_ID = 263;
const short F_GEOM_BOND_ATOM_SITE_AUTH_2_ATOM_ID = 264;
const short F_GEOM_BOND_ATOM_SITE_AUTH_2_SEQ_ID = 265;
const short F_GEOM_BOND_ATOM_SITE_AUTH_2_COMP_ID = 266;
const short F_GEOM_BOND_ATOM_SITE_AUTH_2_ASYM_ID = 267;
const short F_GEOM_BOND_DIST = 268;
const short F_GEOM_BOND_DIST_ESD = 269;
const short F_GEOM_BOND_PUBL_FLAG = 270;

const short S_GEOM_CONTACT = 271;
const short F_GEOM_CONTACT_ATOM_SITE_LABEL_1_ATOM_ID = 272;
const short F_GEOM_CONTACT_ATOM_SITE_LABEL_1_SEQ_ID = 273;
const short F_GEOM_CONTACT_ATOM_SITE_LABEL_1_COMP_ID = 274;
const short F_GEOM_CONTACT_ATOM_SITE_LABEL_1_ASYM_ID = 275;
const short F_GEOM_CONTACT_ATOM_SITE_LABEL_1_ALT_ID = 276;
const short F_GEOM_CONTACT_ATOM_SITE_LABEL_2_ATOM_ID = 277;
const short F_GEOM_CONTACT_ATOM_SITE_LABEL_2_SEQ_ID = 278;
const short F_GEOM_CONTACT_ATOM_SITE_LABEL_2_COMP_ID = 279;
const short F_GEOM_CONTACT_ATOM_SITE_LABEL_2_ASYM_ID = 280;
const short F_GEOM_CONTACT_ATOM_SITE_LABEL_2_ALT_ID = 281;
const short F_GEOM_CONTACT_ATOM_SITE_AUTH_1_ATOM_ID = 282;
const short F_GEOM_CONTACT_ATOM_SITE_AUTH_1_SEQ_ID = 283;
const short F_GEOM_CONTACT_ATOM_SITE_AUTH_1_COMP_ID = 284;
const short F_GEOM_CONTACT_ATOM_SITE_AUTH_1_ASYM_ID = 285;
const short F_GEOM_CONTACT_ATOM_SITE_AUTH_2_ATOM_ID = 286;
const short F_GEOM_CONTACT_ATOM_SITE_AUTH_2_SEQ_ID = 287;
const short F_GEOM_CONTACT_ATOM_SITE_AUTH_2_COMP_ID = 288;
const short F_GEOM_CONTACT_ATOM_SITE_AUTH_2_ASYM_ID = 289;
const short F_GEOM_CONTACT_DIST = 290;
const short F_GEOM_CONTACT_DIST_ESD = 291;
const short F_GEOM_CONTACT_PUBL_FLAG = 292;

const short S_GEOM_HBOND = 293;
const short F_GEOM_HBOND_ANGLE_DHA = 294;
const short F_GEOM_HBOND_ANGLE_DHA_ESD = 295;
const short F_GEOM_HBOND_ATOM_SITE_LABEL_A_ATOM_ID = 296;
const short F_GEOM_HBOND_ATOM_SITE_LABEL_A_SEQ_ID = 297;
const short F_GEOM_HBOND_ATOM_SITE_LABEL_A_COMP_ID = 298;
const short F_GEOM_HBOND_ATOM_SITE_LABEL_A_ASYM_ID = 299;
const short F_GEOM_HBOND_ATOM_SITE_LABEL_A_ALT_ID = 300;
const short F_GEOM_HBOND_ATOM_SITE_LABEL_D_ATOM_ID = 301;
const short F_GEOM_HBOND_ATOM_SITE_LABEL_D_SEQ_ID = 302;
const short F_GEOM_HBOND_ATOM_SITE_LABEL_D_COMP_ID = 303;
const short F_GEOM_HBOND_ATOM_SITE_LABEL_D_ASYM_ID = 304;
const short F_GEOM_HBOND_ATOM_SITE_LABEL_D_ALT_ID = 305;
const short F_GEOM_HBOND_ATOM_SITE_LABEL_H_ATOM_ID = 306;
const short F_GEOM_HBOND_ATOM_SITE_LABEL_H_SEQ_ID = 307;
const short F_GEOM_HBOND_ATOM_SITE_LABEL_H_COMP_ID = 308;
const short F_GEOM_HBOND_ATOM_SITE_LABEL_H_ASYM_ID = 309;
const short F_GEOM_HBOND_ATOM_SITE_LABEL_H_ALT_ID = 310;
const short F_GEOM_HBOND_ATOM_SITE_AUTH_A_ATOM_ID = 311;
const short F_GEOM_HBOND_ATOM_SITE_AUTH_A_SEQ_ID = 312;
const short F_GEOM_HBOND_ATOM_SITE_AUTH_A_COMP_ID = 313;
```

```
const short F_GEOM_HBOND_ATOM_SITE_AUTH_A_ASYM_ID = 314;
const short F_GEOM_HBOND_ATOM_SITE_AUTH_D_ATOM_ID = 315;
const short F_GEOM_HBOND_ATOM_SITE_AUTH_D_SEQ_ID = 316;
const short F_GEOM_HBOND_ATOM_SITE_AUTH_D_COMP_ID = 317;
const short F_GEOM_HBOND_ATOM_SITE_AUTH_D_ASYM_ID = 318;
const short F_GEOM_HBOND_ATOM_SITE_AUTH_H_ATOM_ID = 319;
const short F_GEOM_HBOND_ATOM_SITE_AUTH_H_SEQ_ID = 320;
const short F_GEOM_HBOND_ATOM_SITE_AUTH_H_COMP_ID = 321;
const short F_GEOM_HBOND_ATOM_SITE_AUTH_H_ASYM_ID = 322;
const short F_GEOM_HBOND_DIST_DA = 323;
const short F_GEOM_HBOND_DIST_DA_ESD = 324;
const short F_GEOM_HBOND_DIST_DH = 325;
const short F_GEOM_HBOND_DIST_DH_ESD = 326;
const short F_GEOM_HBOND_DIST_HA = 327;
const short F_GEOM_HBOND_DIST_HA_ESD = 328;
const short F_GEOM_HBOND_PUBL_FLAG = 329;

const short S_GEOM_TORSION = 330;
const short F_GEOM_TORSION_ATOM_SITE_LABEL_1_ATOM_ID = 331;
const short F_GEOM_TORSION_ATOM_SITE_LABEL_1_SEQ_ID = 332;
const short F_GEOM_TORSION_ATOM_SITE_LABEL_1_COMP_ID = 333;
const short F_GEOM_TORSION_ATOM_SITE_LABEL_1_ASYM_ID = 334;
const short F_GEOM_TORSION_ATOM_SITE_LABEL_1_ALT_ID = 335;
const short F_GEOM_TORSION_ATOM_SITE_LABEL_2_ATOM_ID = 336;
const short F_GEOM_TORSION_ATOM_SITE_LABEL_2_SEQ_ID = 337;
const short F_GEOM_TORSION_ATOM_SITE_LABEL_2_COMP_ID = 338;
const short F_GEOM_TORSION_ATOM_SITE_LABEL_2_ASYM_ID = 339;
const short F_GEOM_TORSION_ATOM_SITE_LABEL_2_ALT_ID = 340;
const short F_GEOM_TORSION_ATOM_SITE_LABEL_3_ATOM_ID = 341;
const short F_GEOM_TORSION_ATOM_SITE_LABEL_3_SEQ_ID = 342;
const short F_GEOM_TORSION_ATOM_SITE_LABEL_3_COMP_ID = 343;
const short F_GEOM_TORSION_ATOM_SITE_LABEL_3_ASYM_ID = 344;
const short F_GEOM_TORSION_ATOM_SITE_LABEL_3_ALT_ID = 345;
const short F_GEOM_TORSION_ATOM_SITE_LABEL_4_ATOM_ID = 346;
const short F_GEOM_TORSION_ATOM_SITE_LABEL_4_SEQ_ID = 347;
const short F_GEOM_TORSION_ATOM_SITE_LABEL_4_COMP_ID = 348;
const short F_GEOM_TORSION_ATOM_SITE_LABEL_4_ASYM_ID = 349;
const short F_GEOM_TORSION_ATOM_SITE_LABEL_4_ALT_ID = 350;
const short F_GEOM_TORSION_ATOM_SITE_AUTH_1_ATOM_ID = 351;
const short F_GEOM_TORSION_ATOM_SITE_AUTH_1_SEQ_ID = 352;
const short F_GEOM_TORSION_ATOM_SITE_AUTH_1_COMP_ID = 353;
const short F_GEOM_TORSION_ATOM_SITE_AUTH_1_ASYM_ID = 354;
const short F_GEOM_TORSION_ATOM_SITE_AUTH_2_ATOM_ID = 355;
const short F_GEOM_TORSION_ATOM_SITE_AUTH_2_SEQ_ID = 356;
const short F_GEOM_TORSION_ATOM_SITE_AUTH_2_COMP_ID = 357;
const short F_GEOM_TORSION_ATOM_SITE_AUTH_2_ASYM_ID = 358;
const short F_GEOM_TORSION_ATOM_SITE_AUTH_3_ATOM_ID = 359;
const short F_GEOM_TORSION_ATOM_SITE_AUTH_3_SEQ_ID = 360;
const short F_GEOM_TORSION_ATOM_SITE_AUTH_3_COMP_ID = 361;
const short F_GEOM_TORSION_ATOM_SITE_AUTH_3_ASYM_ID = 362;
const short F_GEOM_TORSION_ATOM_SITE_AUTH_4_ATOM_ID = 363;
const short F_GEOM_TORSION_ATOM_SITE_AUTH_4_SEQ_ID = 364;
const short F_GEOM_TORSION_ATOM_SITE_AUTH_4_COMP_ID = 365;
const short F_GEOM_TORSION_ATOM_SITE_AUTH_4_ASYM_ID = 366;
```

```
const short F_GEOM_TORSION_PUBL_FLAG = 367;
const short F_GEOM_TORSION_VALUE = 368;
const short F_GEOM_TORSION_VALUE_ESD = 369;

const short S_STRUCTURE = 370;
const short F_STRUCTURE_TITLE = 371;

const short S_STRUCT_ASYM = 372;
const short F_STRUCT_ASYM_DETAILS = 373;

const short S_STRUCT_BIOL = 374;
const short F_STRUCT_BIOL_DETAILS = 375;

const short S_STRUCT_BIOL_GEN = 376;
const short F_STRUCT_BIOL_GEN_DETAILS = 377;

const short S_STRUCT_BIOL_KEYWORDS = 378;

const short S_STRUCT_BIOL_VIEW = 379;
const short F_STRUCT_BIOL_VIEW_DETAILS = 380;
const short F_STRUCT_BIOL_VIEW_ROT_MATRIX = 381;

const short S_STRUCT_CONF = 382;
const short F_STRUCT_CONF_BEG_AUTH_SEQ_ID = 383;
const short F_STRUCT_CONF_BEG_AUTH_COMP_ID = 384;
const short F_STRUCT_CONF_BEG_AUTH_ASYM_ID = 385;
const short F_STRUCT_CONF_DETAILS = 386;
const short F_STRUCT_CONF_END_AUTH_SEQ_ID = 387;
const short F_STRUCT_CONF_END_AUTH_COMP_ID = 388;
const short F_STRUCT_CONF_END_AUTH_ASYM_ID = 389;

const short S_STRUCT_CONF_TYPE = 390;
const short F_STRUCT_CONF_TYPE_CRITERIA = 391;
const short F_STRUCT_CONF_TYPE_REFERENCE = 392;

const short S_STRUCT_CONN = 393;
const short F_STRUCT_CONN_DETAILS = 394;
const short F_STRUCT_CONN_PTNR1_LABEL_ALT_ID = 395;
const short F_STRUCT_CONN_PTNR1_AUTH_ATOM_ID = 396;
const short F_STRUCT_CONN_PTNR1_AUTH_SEQ_ID = 397;
const short F_STRUCT_CONN_PTNR1_AUTH_COMP_ID = 398;
const short F_STRUCT_CONN_PTNR1_AUTH_ASYM_ID = 399;
const short F_STRUCT_CONN_PTNR1_ROLE = 400;
const short F_STRUCT_CONN_PTNR1_SYMMETRY = 401;
const short F_STRUCT_CONN_PTNR2_LABEL_ALT_ID = 402;
const short F_STRUCT_CONN_PTNR2_AUTH_ATOM_ID = 403;
const short F_STRUCT_CONN_PTNR2_AUTH_SEQ_ID = 404;
const short F_STRUCT_CONN_PTNR2_AUTH_COMP_ID = 405;
const short F_STRUCT_CONN_PTNR2_AUTH_ASYM_ID = 406;
const short F_STRUCT_CONN_PTNR2_ROLE = 407;
const short F_STRUCT_CONN_PTNR2_SYMMETRY = 408;

const short S_STRUCT_CONN_TYPE = 409;
const short F_STRUCT_CONN_TYPE_CRITERIA = 410;
```

```
const short F_STRUCT_CONN_TYPE_REFERENCE = 411;

const short S_STRUCT_KEYWORDS = 412;

const short S_STRUCT_MON_DETAILS = 413;
const short F_STRUCT_MON_DETAILS_PROT_CIS = 414;
const short F_STRUCT_MON_DETAILS_RSCC = 415;
const short F_STRUCT_MON_DETAILS_RSR = 416;

const short S_STRUCT_MON_NUCL = 417;
const short F_STRUCT_MON_NUCL_ALPHA = 418;
const short F_STRUCT_MON_NUCL_BETA = 419;
const short F_STRUCT_MON_NUCL_CHI1 = 420;
const short F_STRUCT_MON_NUCL_CHI2 = 421;
const short F_STRUCT_MON_NUCL_DELTA = 422;
const short F_STRUCT_MON_NUCL_DETAILS = 423;
const short F_STRUCT_MON_NUCL_EPSILON = 424;
const short F_STRUCT_MON_NUCL_GAMMA = 425;
const short F_STRUCT_MON_NUCL_AUTH_SEQ_ID = 426;
const short F_STRUCT_MON_NUCL_AUTH_COMP_ID = 427;
const short F_STRUCT_MON_NUCL_AUTH_ASYM_ID = 428;
const short F_STRUCT_MON_NUCL_MEAN_B_ALL = 429;
const short F_STRUCT_MON_NUCL_MEAN_B_BASE = 430;
const short F_STRUCT_MON_NUCL_MEAN_B_PHOS = 431;
const short F_STRUCT_MON_NUCL_MEAN_B_SUGAR = 432;
const short F_STRUCT_MON_NUCL_NU0 = 433;
const short F_STRUCT_MON_NUCL_NU1 = 434;
const short F_STRUCT_MON_NUCL_NU2 = 435;
const short F_STRUCT_MON_NUCL_NU3 = 436;
const short F_STRUCT_MON_NUCL_NU4 = 437;
const short F_STRUCT_MON_NUCL_P = 438;
const short F_STRUCT_MON_NUCL_RSCC_ALL = 439;
const short F_STRUCT_MON_NUCL_RSCC_BASE = 440;
const short F_STRUCT_MON_NUCL_RSCC_PHOS = 441;
const short F_STRUCT_MON_NUCL_RSCC_SUGAR = 442;
const short F_STRUCT_MON_NUCL_RSR_ALL = 443;
const short F_STRUCT_MON_NUCL_RSR_BASE = 444;
const short F_STRUCT_MON_NUCL_RSR_PHOS = 445;
const short F_STRUCT_MON_NUCL_RSR_SUGAR = 446;
const short F_STRUCT_MON_NUCL_TAU0 = 447;
const short F_STRUCT_MON_NUCL_TAU1 = 448;
const short F_STRUCT_MON_NUCL_TAU2 = 449;
const short F_STRUCT_MON_NUCL_TAU3 = 450;
const short F_STRUCT_MON_NUCL_TAU4 = 451;
const short F_STRUCT_MON_NUCL_TAUM = 452;
const short F_STRUCT_MON_NUCL_ZETA = 453;

const short S_STRUCT_MON_PROT = 454;
const short F_STRUCT_MON_PROT_CHI1 = 455;
const short F_STRUCT_MON_PROT_CHI2 = 456;
const short F_STRUCT_MON_PROT_CHI3 = 457;
const short F_STRUCT_MON_PROT_CHI4 = 458;
const short F_STRUCT_MON_PROT_CHI5 = 459;
const short F_STRUCT_MON_PROT_DETAILS = 460;
```

```
const short F_STRUCT_MON_PROT_AUTH_SEQ_ID = 461;
const short F_STRUCT_MON_PROT_AUTH_COMP_ID = 462;
const short F_STRUCT_MON_PROT_AUTH_ASYM_ID = 463;
const short F_STRUCT_MON_PROT_RSCC_ALL = 464;
const short F_STRUCT_MON_PROT_RSCC_MAIN = 465;
const short F_STRUCT_MON_PROT_RSCC_SIDE = 466;
const short F_STRUCT_MON_PROT_RSR_ALL = 467;
const short F_STRUCT_MON_PROT_RSR_MAIN = 468;
const short F_STRUCT_MON_PROT_RSR_SIDE = 469;
const short F_STRUCT_MON_PROT_MEAN_B_ALL = 470;
const short F_STRUCT_MON_PROT_MEAN_B_MAIN = 471;
const short F_STRUCT_MON_PROT_MEAN_B_SIDE = 472;
const short F_STRUCT_MON_PROT_OMEGA = 473;
const short F_STRUCT_MON_PROT_PHI = 474;
const short F_STRUCT_MON_PROT_PSI = 475;

const short S_STRUCT_MON_PROT_CIS = 476;
const short F_STRUCT_MON_PROT_CIS_AUTH_SEQ_ID = 477;
const short F_STRUCT_MON_PROT_CIS_AUTH_COMP_ID = 478;
const short F_STRUCT_MON_PROT_CIS_AUTH_ASYM_ID = 479;

const short S_STRUCT_NCS_DOM = 480;
const short F_STRUCT_NCS_DOM_DETAILS = 481;

const short S_STRUCT_NCS_DOM_LIM = 482;
const short F_STRUCT_NCS_DOM_LIM_BEG_AUTH_SEQ_ID = 483;
const short F_STRUCT_NCS_DOM_LIM_BEG_AUTH_COMP_ID = 484;
const short F_STRUCT_NCS_DOM_LIM_BEG_AUTH_ASYM_ID = 485;
const short F_STRUCT_NCS_DOM_LIM_END_AUTH_SEQ_ID = 486;
const short F_STRUCT_NCS_DOM_LIM_END_AUTH_COMP_ID = 487;
const short F_STRUCT_NCS_DOM_LIM_END_AUTH_ASYM_ID = 488;

const short S_STRUCT_NCS_ENS = 489;
const short F_STRUCT_NCS_ENS_DETAILS = 490;
const short F_STRUCT_NCS_ENS_POINT_GROUP = 491;

const short S_STRUCT_NCS_ENS_GEN = 492;

const short S_STRUCT_NCS_OPER = 493;
const short F_STRUCT_NCS_OPER_CODE = 494;
const short F_STRUCT_NCS_OPER_DETAILS = 495;
const short F_STRUCT_NCS_OPER_MATRIX = 496;
const short F_STRUCT_NCS_OPER_VECTOR = 497;

const short S_STRUCT_REF = 498;
const short F_STRUCT_REF_DETAILS = 499;
const short F_STRUCT_REF_SEQ_ALIGN = 500;
const short F_STRUCT_REF_SEQ_DIF = 501;

const short S_STRUCT_REF_SEQ = 502;
const short F_STRUCT_REF_SEQ_DETAILS = 503;

const short S_STRUCT_REF_SEQ_DIF = 504;
const short F_STRUCT_REF_SEQ_DIF_DETAILS = 505;
```

```
const short S_STRUCT_SHEET = 506;
const short F_STRUCT_SHEET_DETAILS = 507;
const short F_STRUCT_SHEET_NUMBER_STRANDS = 508;
const short F_STRUCT_SHEET_TYPE = 509;

const short S_STRUCT_SHEET_HBOND = 510;
const short F_STRUCT_SHEET_HBOND_RANGE_1_BEG_AUTH_ATOM_ID = 511;
const short F_STRUCT_SHEET_HBOND_RANGE_1_BEG_AUTH_SEQ_ID = 512;
const short F_STRUCT_SHEET_HBOND_RANGE_1_END_AUTH_ATOM_ID = 513;
const short F_STRUCT_SHEET_HBOND_RANGE_1_END_AUTH_SEQ_ID = 514;
const short F_STRUCT_SHEET_HBOND_RANGE_2_BEG_AUTH_ATOM_ID = 515;
const short F_STRUCT_SHEET_HBOND_RANGE_2_BEG_AUTH_SEQ_ID = 516;
const short F_STRUCT_SHEET_HBOND_RANGE_2_END_AUTH_ATOM_ID = 517;
const short F_STRUCT_SHEET_HBOND_RANGE_2_END_AUTH_SEQ_ID = 518;

const short S_STRUCT_SHEET_ORDER = 519;
const short F_STRUCT_SHEET_ORDER_OFFSET = 520;
const short F_STRUCT_SHEET_ORDER_SENSE = 521;

const short S_STRUCT_SHEET_RANGE = 522;
const short F_STRUCT_SHEET_RANGE_BEG_AUTH_SEQ_ID = 523;
const short F_STRUCT_SHEET_RANGE_BEG_AUTH_COMP_ID = 524;
const short F_STRUCT_SHEET_RANGE_BEG_AUTH_ASYM_ID = 525;
const short F_STRUCT_SHEET_RANGE_END_AUTH_SEQ_ID = 526;
const short F_STRUCT_SHEET_RANGE_END_AUTH_COMP_ID = 527;
const short F_STRUCT_SHEET_RANGE_END_AUTH_ASYM_ID = 528;
const short F_STRUCT_SHEET_RANGE_SYMMETRY = 529;

const short S_STRUCT_SHEET_TOPOLOGY = 530;
const short F_STRUCT_SHEET_TOPOLOGY_OFFSET = 531;
const short F_STRUCT_SHEET_TOPOLOGY_SENSE = 532;

const short S_STRUCT_SITE = 533;
const short F_STRUCT_SITE_DETAILS = 534;

const short S_STRUCT_SITE_GEN = 535;
const short F_STRUCT_SITE_GEN_DETAILS = 536;
const short F_STRUCT_SITE_GEN_AUTH_ATOM_ID = 537;
const short F_STRUCT_SITE_GEN_AUTH_SEQ_ID = 538;
const short F_STRUCT_SITE_GEN_AUTH_COMP_ID = 539;
const short F_STRUCT_SITE_GEN_AUTH_ASYM_ID = 540;
const short F_STRUCT_SITE_GEN_SYMMETRY = 541;

const short S_STRUCT_SITE_KEYWORDS = 542;

const short S_STRUCT_SITE_VIEW = 543;
const short F_STRUCT_SITE_VIEW_DETAILS = 544;
const short F_STRUCT_SITE_VIEW_ROT_MATRIX = 545;

const short MAX_FLAG = 545;

long atom_site_list_size()
    raises (DataAccessException);
```

```
AtomSiteList get_atom_site_list()
      raises (DataAccessException);
AtomSiteList get_atom_site_block_n(
      in long from,
      in long to)
      raises (DataAccessException);
long atom_site_ext_list_size()
      raises (DataAccessException);
AtomSiteExtList get_atom_site_ext_list()
      raises (DataAccessException);
AtomSiteExtList get_atom_site_ext_block_n(
      in long from,
      in long to)
      raises (DataAccessException);
long atom_site_anisotrop_list_size()
      raises (DataAccessException);
AtomSiteAnisotropList get_atom_site_anisotrop_list()
      raises (DataAccessException);
long atom_type_list_size()
      raises (DataAccessException);
AtomTypeList get_atom_type_list()
      raises (DataAccessException);
long chem_comp_list_size()
      raises (DataAccessException);
ChemCompList get_chem_comp_list()
      raises (DataAccessException);
long chem_comp_angle_list_size()
      raises (DataAccessException);
ChemCompAngleList get_chem_comp_angle_list()
      raises (DataAccessException);
long chem_comp_atom_list_size()
      raises (DataAccessException);
ChemCompAtomList get_chem_comp_atom_list()
      raises (DataAccessException);
long chem_comp_bond_list_size()
      raises (DataAccessException);
ChemCompBondList get_chem_comp_bond_list()
      raises (DataAccessException);
long chem_comp_chir_list_size()
      raises (DataAccessException);
ChemCompChirList get_chem_comp_chir_list()
      raises (DataAccessException);
long chem_comp_chir_atom_list_size()
      raises (DataAccessException);
ChemCompChirAtomList get_chem_comp_chir_atom_list()
      raises (DataAccessException);
long chem_comp_link_list_size()
      raises (DataAccessException);
ChemCompLinkList get_chem_comp_link_list()
      raises (DataAccessException);
long chem_comp_plane_list_size()
      raises (DataAccessException);
ChemCompPlaneList get_chem_comp_plane_list()
      raises (DataAccessException);
```

```
long chem_comp_plane_atom_list_size()
      raises (DataAccessException);
ChemCompPlaneAtomList get_chem_comp_plane_atom_list()
      raises (DataAccessException);
long chem_comp_tor_list_size()
      raises (DataAccessException);
ChemCompTorList get_chem_comp_tor_list()
      raises (DataAccessException);
long chem_comp_tor_value_list_size()
      raises (DataAccessException);
ChemCompTorValueList get_chem_comp_tor_value_list()
      raises (DataAccessException);
long chem_link_list_size()
      raises (DataAccessException);
ChemLinkList get_chem_link_list()
      raises (DataAccessException);
long chem_link_angle_list_size()
      raises (DataAccessException);
ChemLinkAngleList get_chem_link_angle_list()
      raises (DataAccessException);
long chem_link_bond_list_size()
      raises (DataAccessException);
ChemLinkBondList get_chem_link_bond_list()
      raises (DataAccessException);
long chem_link_chir_list_size()
      raises (DataAccessException);
ChemLinkChirList get_chem_link_chir_list()
      raises (DataAccessException);
long chem_link_chir_atom_list_size()
      raises (DataAccessException);
ChemLinkChirAtomList get_chem_link_chir_atom_list()
      raises (DataAccessException);
long chem_link_plane_list_size()
      raises (DataAccessException);
ChemLinkPlaneList get_chem_link_plane_list()
      raises (DataAccessException);
long chem_link_plane_atom_list_size()
      raises (DataAccessException);
ChemLinkPlaneAtomList get_chem_link_plane_atom_list()
      raises (DataAccessException);
long chem_link_tor_list_size()
      raises (DataAccessException);
ChemLinkTorList get_chem_link_tor_list()
      raises (DataAccessException);
long chem_link_tor_value_list_size()
      raises (DataAccessException);
ChemLinkTorValueList get_chem_link_tor_value_list()
      raises (DataAccessException);
long entity_list_size()
      raises (DataAccessException);
EntityList get_entity_list()
      raises (DataAccessException);
long entity_keywords_list_size()
      raises (DataAccessException);
```

```
EntityKeywordsList get_entity_keywords_list()
      raises (DataAccessException);
long entity_link_list_size()
      raises (DataAccessException);
EntityLinkList get_entity_link_list()
      raises (DataAccessException);
long entity_name_com_list_size()
      raises (DataAccessException);
EntityNameComList get_entity_name_com_list()
      raises (DataAccessException);
long entity_name_sys_list_size()
      raises (DataAccessException);
EntityNameSysList get_entity_name_sys_list()
      raises (DataAccessException);
long entity_poly_list_size()
      raises (DataAccessException);
EntityPolyList get_entity_poly_list()
      raises (DataAccessException);
long entity_poly_seq_list_size()
      raises (DataAccessException);
EntityPolySeqList get_entity_poly_seq_list()
      raises (DataAccessException);
EntityPolySeqList get_entity_poly_seq_block_n(
      in long from,
      in long to)
      raises (DataAccessException);
long entity_src_gen_list_size()
      raises (DataAccessException);
EntitySrcGenList get_entity_src_gen_list()
      raises (DataAccessException);
long entity_src_nat_list_size()
      raises (DataAccessException);
EntitySrcNatList get_entity_src_nat_list()
      raises (DataAccessException);
long entry_link_list_size()
      raises (DataAccessException);
EntryLinkList get_entry_link_list()
      raises (DataAccessException);
long geom_list_size()
      raises (DataAccessException);
GeomList get_geom_list()
      raises (DataAccessException);
long geom_angle_list_size()
      raises (DataAccessException);
GeomAngleList get_geom_angle_list()
      raises (DataAccessException);
long geom_bond_list_size()
      raises (DataAccessException);
GeomBondList get_geom_bond_list()
      raises (DataAccessException);
long geom_contact_list_size()
      raises (DataAccessException);
GeomContactList get_geom_contact_list()
      raises (DataAccessException);
```

```
long geom_hbond_list_size()
      raises (DataAccessException);
GeomHbondList get_geom_hbond_list()
      raises (DataAccessException);
long geom_torsion_list_size()
      raises (DataAccessException);
GeomTorsionList get_geom_torsion_list()
      raises (DataAccessException);
long structure_list_size()
      raises (DataAccessException);
StructureList get_structure_list()
      raises (DataAccessException);
long struct_asym_list_size()
      raises (DataAccessException);
StructAsymList get_struct_asym_list()
      raises (DataAccessException);
long struct_biol_list_size()
      raises (DataAccessException);
StructBiolList get_struct_biol_list()
      raises (DataAccessException);
long struct_biol_gen_list_size()
      raises (DataAccessException);
StructBiolGenList get_struct_biol_gen_list()
      raises (DataAccessException);
long struct_biol_keywords_list_size()
      raises (DataAccessException);
StructBiolKeywordsList get_struct_biol_keywords_list()
      raises (DataAccessException);
long struct_biol_view_list_size()
      raises (DataAccessException);
StructBiolViewList get_struct_biol_view_list()
      raises (DataAccessException);
long struct_conf_list_size()
      raises (DataAccessException);
StructConfList get_struct_conf_list()
      raises (DataAccessException);
long struct_conf_type_list_size()
      raises (DataAccessException);
StructConfTypeList get_struct_conf_type_list()
      raises (DataAccessException);
long struct_conn_list_size()
      raises (DataAccessException);
StructConnList get_struct_conn_list()
      raises (DataAccessException);
long struct_conn_type_list_size()
      raises (DataAccessException);
StructConnTypeList get_struct_conn_type_list()
      raises (DataAccessException);
long struct_keywords_list_size()
      raises (DataAccessException);
StructKeywordsList get_struct_keywords_list()
      raises (DataAccessException);
long struct_mon_details_list_size()
      raises (DataAccessException);
```

```
StructMonDetailsList get_struct_mon_details_list()
      raises (DataAccessException);
long struct_mon_nucl_list_size()
      raises (DataAccessException);
StructMonNuclList get_struct_mon_nucl_list()
      raises (DataAccessException);
long struct_mon_prot_list_size()
      raises (DataAccessException);
StructMonProtList get_struct_mon_prot_list()
      raises (DataAccessException);
long struct_mon_prot_cis_list_size()
      raises (DataAccessException);
StructMonProtCisList get_struct_mon_prot_cis_list()
      raises (DataAccessException);
long struct_ncs_dom_list_size()
      raises (DataAccessException);
StructNcsDomList get_struct_ncs_dom_list()
      raises (DataAccessException);
long struct_ncs_dom_lim_list_size()
      raises (DataAccessException);
StructNcsDomLimList get_struct_ncs_dom_lim_list()
      raises (DataAccessException);
long struct_ncs_ens_list_size()
      raises (DataAccessException);
StructNcsEnsList get_struct_ncs_ens_list()
      raises (DataAccessException);
long struct_ncs_ens_gen_list_size()
      raises (DataAccessException);
StructNcsEnsGenList get_struct_ncs_ens_gen_list()
      raises (DataAccessException);
long struct_ncs_oper_list_size()
      raises (DataAccessException);
StructNcsOperList get_struct_ncs_oper_list()
      raises (DataAccessException);
long struct_ref_list_size()
      raises (DataAccessException);
StructRefList get_struct_ref_list()
      raises (DataAccessException);
long struct_ref_seq_list_size()
      raises (DataAccessException);
StructRefSeqList get_struct_ref_seq_list()
      raises (DataAccessException);
long struct_ref_seq_dif_list_size()
      raises (DataAccessException);
StructRefSeqDifList get_struct_ref_seq_dif_list()
      raises (DataAccessException);
long struct_sheet_list_size()
      raises (DataAccessException);
StructSheetList get_struct_sheet_list()
      raises (DataAccessException);
long struct_sheet_hbond_list_size()
      raises (DataAccessException);
StructSheetHbondList get_struct_sheet_hbond_list()
      raises (DataAccessException);
```

```
   long struct_sheet_order_list_size()
         raises (DataAccessException);
   StructSheetOrderList get_struct_sheet_order_list()
         raises (DataAccessException);
   long struct_sheet_range_list_size()
         raises (DataAccessException);
   StructSheetRangeList get_struct_sheet_range_list()
         raises (DataAccessException);
   long struct_sheet_topology_list_size()
         raises (DataAccessException);
   StructSheetTopologyList get_struct_sheet_topology_list()
         raises (DataAccessException);
   long struct_site_list_size()
         raises (DataAccessException);
   StructSiteList get_struct_site_list()
         raises (DataAccessException);
   long struct_site_gen_list_size()
         raises (DataAccessException);
   StructSiteGenList get_struct_site_gen_list()
         raises (DataAccessException);
   long struct_site_keywords_list_size()
         raises (DataAccessException);
   StructSiteKeywordsList get_struct_site_keywords_list()
         raises (DataAccessException);
   long struct_site_view_list_size()
         raises (DataAccessException);
   StructSiteViewList get_struct_site_view_list()
         raises (DataAccessException);
};

typedef Identifier EntryId;
typedef sequence<EntryId> EntryIdList;

typedef Identifier EntryGroupId;
typedef sequence<EntryGroupId> EntryGroupIdList;

struct ModificationDate
{
   EntryId entry_id;
   TimeBase::TimeT date;
};
typedef sequence<ModificationDate> ModificationDateList;

interface EntryFactory
{
   string get_version();
   BaseIDL::ModuleDefSet get_extension_modules();
   EntryIdList get_entry_id_list()
         raises (DataAccessException);
   long get_entry_id_list_size()
         raises (DataAccessException);
   EntryIdList get_entry_id_list_block_n(
         in long from,
         in long to)
```

```
            raises (DataAccessException);
        ModificationDateList get_entry_modification_dates()
            raises (DataAccessException);
        ModificationDateList get_entry_modification_dates_block_n(
            in long from,
            in long to)
            raises (DataAccessException);
        EntryGroupIdList get_entry_group_list()
            raises (DataAccessException);
        EntryIdList get_entries_in_group(in EntryGroupId group)
            raises (DataAccessException);
        Entry get_entry_from_id(in EntryId entry_id)
            raises (DataAccessException);
        FormatTypeList native_formats_supported()
            raises (DataAccessException);
        EntryRepresentation get_native_entry_representation(
            in FormatType format,
            in EntryId entry_id)
            raises (DataAccessException);
    };
};

#endif // _DS_LSR_MACROMOLECULAR_STRUCTURE_IDL_
```

B. DsLSRMmsReference IDL specification

```
// File: DsLSRMmsReference.idl

#ifndef _DS_LSR_MMS_REFERENCE_IDL_
#define _DS_LSR_MMS_REFERENCE_IDL_

#include “DsLSRMacromolecularStructure.idl”

#pragma prefix “omg.org”

module DsLSRMmsReference
{

   valuetype Citation
   {
      factory createCitation();

      public string abstract_text;
      public string abstract_id_CAS;
      public string book_id_isbn;
      public string book_publisher;
      public string book_publisher_city;
      public string book_title;
      public string coordinate_linkage;
      public string country;
      public long database_id_medline;
      public string details;
      public string id;
      public string journal_abbrev;
      public string journal_id_astm;
      public string journal_id_csd;
      public string journal_id_issn;
      public string journal_full;
      public string journal_issue;
      public string journal_volume;
      public string language;
      public string page_first;
      public string page_last;
      public string title;
      public long year;
   };
   typedef sequence<Citation> CitationList;

   valuetype CitationAuthor
   {
      factory createCitationAuthor();
```

```
    public DsLSRMacromolecularStructure::IndexId citation;
    public string name;
    public long ordinal;
};
typedef sequence<CitationAuthor> CitationAuthorList;

valuetype CitationEditor
{
    factory createCitationEditor();

    public DsLSRMacromolecularStructure::IndexId citation;
    public string name;
    public long ordinal;
};
typedef sequence<CitationEditor> CitationEditorList;

valuetype Database
{
    factory createDatabase();

    public string database_id;
    public string database_code;
};
typedef sequence<Database> DatabaseList;

valuetype DatabasePdbCaveat
{
    factory createDatabasePdbCaveat();

    public long id;
    public string text;
};
typedef sequence<DatabasePdbCaveat> DatabasePdbCaveatList;

valuetype DatabasePdbMatrix
{
    factory createDatabasePdbMatrix();

    public EntryId entry_id;
    public DsLSRMacromolecularStructure::Matrix3 origx;
    public DsLSRMacromolecularStructure::Vector3 origx_vector;
    public DsLSRMacromolecularStructure::Matrix3 scale;
    public DsLSRMacromolecularStructure::Vector3 scale_vector;
};
typedef sequence<DatabasePdbMatrix> DatabasePdbMatrixList;

valuetype DatabasePdbRemark
{
    factory createDatabasePdbRemark();

    public long id;
    public string text;
};
```

```
typedef sequence<DatabasePdbRemark> DatabasePdbRemarkList;

valuetype DatabasePdbRev
{
   factory createDatabasePdbRev();

   public string author_name;
   public string date;
   public string date_original;
   public long mod_type;
   public long num;
   public string replaced_by;
   public string replaces;
   public string status;
};
typedef sequence<DatabasePdbRev> DatabasePdbRevList;

valuetype DatabasePdbRevRecord
{
   factory createDatabasePdbRevRecord();

   public string details;
   public DsLSRMacromolecularStructure::IndexId rev_num;
   public string type;
};
typedef sequence<DatabasePdbRevRecord> DatabasePdbRevRecordList;

valuetype DatabasePdbTvect
{
   factory createDatabasePdbTvect();

   public string details;
   public string id;
   public DsLSRMacromolecularStructure::Vector3 vector;
};
typedef sequence<DatabasePdbTvect> DatabasePdbTvectList;

valuetype Computing
{
   factory createComputing();

   public EntryId entry_id;
   public string cell_refinement;
   public string data_collection;
   public string data_reduction;
   public string molecular_graphics;
   public string publication_material;
   public string structure_refinement;
   public string structure_solution;
};
typedef sequence<Computing> ComputingList;

valuetype Software
{
```

```
   factory createSoftware();

   public DsLSRMacromolecularStructure::IndexId citation;
   public string classification;
   public string compiler_name;
   public string compiler_version;
   public string contact_author;
   public string contact_author_email;
   public string date;
   public string description;
   public string dependencies;
   public string hardware;
   public string language;
   public string location;
   public string mods;
   public string name;
   public string os;
   public string os_version;
   public string type;
   public string version;
};
typedef sequence<Software> SoftwareList;

interface MmsReferenceEntry
{
   DsLSRMacromolecularStructure::Flags get_presence_flags()
         raises (DsLSRMacromolecularStructure::DataAccessException);

   const short S_CITATION = 1;
   const short F_CITATION_ABSTRACT_TEXT = 2;
   const short F_CITATION_ABSTRACT_ID_CAS = 3;
   const short F_CITATION_BOOK_ID_ISBN = 4;
   const short F_CITATION_BOOK_PUBLISHER = 5;
   const short F_CITATION_BOOK_PUBLISHER_CITY = 6;
   const short F_CITATION_BOOK_TITLE = 7;
   const short F_CITATION_COORDINATE_LINKAGE = 8;
   const short F_CITATION_COUNTRY = 9;
   const short F_CITATION_DATABASE_ID_MEDLINE = 10;
   const short F_CITATION_DETAILS = 11;
   const short F_CITATION_JOURNAL_ABBREV = 12;
   const short F_CITATION_JOURNAL_ID_ASTM = 13;
   const short F_CITATION_JOURNAL_ID_CSD = 14;
   const short F_CITATION_JOURNAL_ID_ISSN = 15;
   const short F_CITATION_JOURNAL_FULL = 16;
   const short F_CITATION_JOURNAL_ISSUE = 17;
   const short F_CITATION_JOURNAL_VOLUME = 18;
   const short F_CITATION_LANGUAGE = 19;
   const short F_CITATION_PAGE_FIRST = 20;
   const short F_CITATION_PAGE_LAST = 21;
   const short F_CITATION_TITLE = 22;
   const short F_CITATION_YEAR = 23;

   const short S_CITATION_AUTHOR = 24;
   const short F_CITATION_AUTHOR_ORDINAL = 25;
```

```
const short S_CITATION_EDITOR = 26;
const short F_CITATION_EDITOR_NAME = 27;
const short F_CITATION_EDITOR_ORDINAL = 28;

const short S_DATABASE = 29;

const short S_DATABASE_PDB_CAVEAT = 30;
const short F_DATABASE_PDB_CAVEAT_TEXT = 31;

const short S_DATABASE_PDB_MATRIX = 32;
const short F_DATABASE_PDB_MATRIX_ORIGX = 33;
const short F_DATABASE_PDB_MATRIX_ORIGX_VECTOR = 34;
const short F_DATABASE_PDB_MATRIX_SCALE = 35;
const short F_DATABASE_PDB_MATRIX_SCALE_VECTOR = 36;

const short S_DATABASE_PDB_REMARK = 37;
const short F_DATABASE_PDB_REMARK_TEXT = 38;

const short S_DATABASE_PDB_REV = 39;
const short F_DATABASE_PDB_REV_AUTHOR_NAME = 40;
const short F_DATABASE_PDB_REV_DATE = 41;
const short F_DATABASE_PDB_REV_DATE_ORIGINAL = 42;
const short F_DATABASE_PDB_REV_MOD_TYPE = 43;
const short F_DATABASE_PDB_REV_REPLACED_BY = 44;
const short F_DATABASE_PDB_REV_REPLACES = 45;
const short F_DATABASE_PDB_REV_STATUS = 46;

const short S_DATABASE_PDB_REV_RECORD = 47;
const short F_DATABASE_PDB_REV_RECORD_DETAILS = 48;

const short S_DATABASE_PDB_TVECT = 49;
const short F_DATABASE_PDB_TVECT_DETAILS = 50;
const short F_DATABASE_PDB_TVECT_VECTOR = 51;

const short S_COMPUTING = 52;
const short F_COMPUTING_CELL_REFINEMENT = 53;
const short F_COMPUTING_DATA_COLLECTION = 54;
const short F_COMPUTING_DATA_REDUCTION = 55;
const short F_COMPUTING_MOLECULAR_GRAPHICS = 56;
const short F_COMPUTING_PUBLICATION_MATERIAL = 57;
const short F_COMPUTING_STRUCTURE_REFINEMENT = 58;
const short F_COMPUTING_STRUCTURE_SOLUTION = 59;

const short S_SOFTWARE = 60;
const short F_SOFTWARE_CLASSIFICATION = 61;
const short F_SOFTWARE_COMPILER_NAME = 62;
const short F_SOFTWARE_COMPILER_VERSION = 63;
const short F_SOFTWARE_CONTACT_AUTHOR = 64;
const short F_SOFTWARE_CONTACT_AUTHOR_EMAIL = 65;
const short F_SOFTWARE_DATE = 66;
const short F_SOFTWARE_DESCRIPTION = 67;
const short F_SOFTWARE_DEPENDENCIES = 68;
const short F_SOFTWARE_HARDWARE = 69;
```

```
const short F_SOFTWARE_LANGUAGE = 70;
const short F_SOFTWARE_LOCATION = 71;
const short F_SOFTWARE_MODS = 72;
const short F_SOFTWARE_OS = 73;
const short F_SOFTWARE_OS_VERSION = 74;
const short F_SOFTWARE_TYPE = 75;

const short MAX_FLAG = 75;

long citation_list_size()
    raises (DsLSRMacromolecularStructure::DataAccessException);
CitationList get_citation_list()
    raises (DsLSRMacromolecularStructure::DataAccessException);
long citation_author_list_size()
    raises (DsLSRMacromolecularStructure::DataAccessException);
CitationAuthorList get_citation_author_list()
    raises (DsLSRMacromolecularStructure::DataAccessException);
long citation_editor_list_size()
    raises (DsLSRMacromolecularStructure::DataAccessException);
CitationEditorList get_citation_editor_list()
    raises (DsLSRMacromolecularStructure::DataAccessException);
long database_list_size()
    raises (DsLSRMacromolecularStructure::DataAccessException);
DatabaseList get_database_list()
    raises (DsLSRMacromolecularStructure::DataAccessException);
long database_pdb_caveat_list_size()
    raises (DsLSRMacromolecularStructure::DataAccessException);
DatabasePdbCaveatList get_database_pdb_caveat_list()
    raises (DsLSRMacromolecularStructure::DataAccessException);
long database_pdb_matrix_list_size()
    raises (DsLSRMacromolecularStructure::DataAccessException);
DatabasePdbMatrixList get_database_pdb_matrix_list()
    raises (DsLSRMacromolecularStructure::DataAccessException);
long database_pdb_remark_list_size()
    raises (DsLSRMacromolecularStructure::DataAccessException);
DatabasePdbRemarkList get_database_pdb_remark_list()
    raises (DsLSRMacromolecularStructure::DataAccessException);
long database_pdb_rev_list_size()
    raises (DsLSRMacromolecularStructure::DataAccessException);
DatabasePdbRevList get_database_pdb_rev_list()
    raises (DsLSRMacromolecularStructure::DataAccessException);
long database_pdb_rev_record_list_size()
    raises (DsLSRMacromolecularStructure::DataAccessException);
DatabasePdbRevRecordList get_database_pdb_rev_record_list()
    raises (DsLSRMacromolecularStructure::DataAccessException);
long database_pdb_tvect_list_size()
    raises (DsLSRMacromolecularStructure::DataAccessException);
DatabasePdbTvectList get_database_pdb_tvect_list()
    raises (DsLSRMacromolecularStructure::DataAccessException);
long computing_list_size()
    raises (DsLSRMacromolecularStructure::DataAccessException);
ComputingList get_computing_list()
    raises (DsLSRMacromolecularStructure::DataAccessException);
long software_list_size()
```

```
            raises (DsLSRMacromolecularStructure::DataAccessException);
        SoftwareList get_software_list()
            raises (DsLSRMacromolecularStructure::DataAccessException);
    };
};

#endif // _DS_LSR_MMS_REFERENCE_IDL_
```

Glossary

anisotropic Having unequal physical properties along different directions.

anomalous scattering A phase change that occurs upon the scattering of X rays by a crystal containing one or more atoms that strongly absorb the X rays.

asymmetric unit The smallest part of a crystal structure from which the complete structure can be obtained from the space group symmetry operations.

atomic coordinates A set of numbers that specifies the position of an atom in a crystal structure with respect to the axial directions of the unit cell of the crystal.

conformation The shape of a molecule, produced by the specific spatial arrangement of the units that compose it.

diffractometry The branch of science that determines the structure of a crystal by observing the changes in amplitude or phase of an X-ray beam or other energy waves penetrating its structure.

factory An object whose primary function is to produce other objects.

Miller indices The plane with Miller indices h, k, and l makes intercepts a/h, b/k, and c/l with the unit-cell axes a, b, and c. The positions of structure factors in reciprocal space are represented by the Miller indices.

phase calculations The measured intensities of diffracted beams produce only the squares of the amplitudes. These calculations determine the phase angle associated with each structure factor, so that an electron-density map may be calculated from a Fourier series that requires both amplitude and phase coefficients.

R Factor A discrepancy index or residual based on differences in structure factor amplitudes. The R factor may be used to measure the agreement between different measurements of the structure factor data or the agreement between the data and the model.

reciprocal space A mathematical dual-space used to calculate the positions in the crystal diffraction pattern.

space group A space group may be considered the group of transformations that converts one molecule or asymmetric unit into an infinitely extending three-dimensional pattern. There are 230 theoretically possible space groups. In a crystal structure determination the space group symmetry is identified from systematic absences in the diffraction pattern.

structure factor A factor that determines the intensity of a reflected beam in crystal diffraction analysis. The magnitude of the structure factor |F| is the ratio of the amplitude of X-rays scattered in a particular direction to that scattered by a point electron at the origin of the unit cell under the same conditions.

temperature factor An expression by which the scattering of an atom is reduced as a consequence of vibration or a simulated vibration resulting from static disorder.

unit cell The basic building block of a crystal. It is the smallest unit of the lattice for a given crystal that displays the symmetry of the lattice.

valuetype IDL keyword defined in the OMG Objects-by-Value specification. Designates an entity which contains state similar to an IDL struct but also inheritance functionality similar to an IDL interface.

References

Abola88 Abola, E.E., Bernstein, F.C., and Koetzle, T.F. (1988) in Computational Methods for Sequence Analysis (Lesk, A.M. ed.) pp 69-81, Oxford University Press.

Ab98 Object Management Group. 1998. OMG IDL Style Guide. OMG Document ab/98-06-03.

Bernstein77 Bernstein F.C., Koetzle T.F., Williams G.J., Meyer E.E. Jr, Brice M.D., Rodgers J.R., Kennard O., Shimanouchi T., Tasumi The Protein Data Bank: a computer-based archival file for macromolecular structures. J. Mol. Biol. 112 pp. 535-542 (1977)

Bernstein98 Bernstein, H. J., Bernstein F.C., and Bourne P.E., pdb2cif translating PDB entries into mmCIF format. Journal of Applied Crystallography (1998) 31, 282-295

Bourne97 Bourne, P.E., Berman, H.M., McMahon, B., Watenpaugh, K., Westbrook, J., and Fitzgerald, P.M.D. The Macromolecular CIF Dictionary (mmCIF). Methods in Enzymology. (1997) 277, 571-590

BSA99 Object Management Group. 1998. Biomolecular Sequence Analysis RFP Response. OMG Document lifesci/99-10-01.

Comp99 Object Management Group. 1999. Component Specification - Volume II OMG Document orbos/99-07-02.

Formal98a Object Management Group. 1998. CORBAservices: Common Object Services Specification. OMG Document formal/98-12-09.

Formal98b Object Management Group. 1998. CORBAservices: Common Object Services IDL. OMG Document formal/98-10-53.

Gamma95 Gamma, Erich, Richard Helm, Ralph Johnson, and John Vlissides. 1995. Design Patterns: Elements of Reusable Object-Oriented Software. Addison-Wesley. ISBN: 0-201-63361-2.

GM99 Object Management Group. 1998. Genomic Maps RFP Response. OMG Document lifesci/99-10-12.

IUCr The International Union of Crystallography (IUCr) CIF web page http://www.iucr.ac.uk/iucr-top/cif/index.html

IUPAC85 IUPAC-IUB symbols for nucleotide nomenclature. Cornish-Bowden (1985) Nucl. Acids Res. 13: 3021-3030.

IUPAC84 IUPAC-IUB symbols for amino acid nomenclature Biochem J. 1984 Apr 15; 219(2): 345-373.

MOF99 Object Management Group. 1999. Meta-Object Facility 1.3 OMG Document ad/99-09-05.

Orbos98 Object Management Group. 1998. Interoperable Naming Service. OMG Document orbos/98-10-11.

OBV98 Object Management Group. 1998. Joint Revised Objects by Value Submission – with Errata. OMG TC Document orbos/98-01-18.

PDB The web page for PDB file formats and dictionaries http://www.rcsb.org/pdb/info.html

Ptc98 Object Management Group. 1998. CORBA v2.3a - Core final revision. OMG PC Document ptc/98-12-04.