CIF APPLICATIONS

Authors of any software that reads, writes or validates CIF data are invited to contribute to this series. Authors should state clearly when submitting a manuscript to a Co-editor that the paper should be included as part of the CIF Applications series. An appropriate series number will be assigned by the Editorial Office.

J. Appl. Cryst. (1998). **31**, 505–509

CIF Applications. IX. A new approach for representing and manipulating STAR files

WEIDER CHANG[a] AND PHILIP E. BOURNE[b,c,d]* *at* [a]*Java Design Center, Sun Microsystems Inc., 55 Broad Street, New York NY 10004, USA,* [b]*San Diego Supercomputer Center, PO Box 85608, San Diego CA 92186, USA,* [c]*Department of Pharmacology, University of California, San Diego, 9500 Gilman Drive, La Jolla CA 92093, USA, and* [d]*The Burnham Institute, 10901 North Torrey Pines Road, La Jolla CA 92037, USA. E-mail: bourne@sdsc.edu*

(*Received 20 August 1997; accepted 17 November 1997*)

Abstract

The Self-defining Text Archival and Retrieval (STAR) format provides a low-level data description from which Crystallographic Information Files (CIFs) are derived. STAR is independent of the two Dictionary Definition Languages (DDLs) currently used in crystallography, is used to represent scientific data beyond crystallographic data and is thus quite general. This paper describes an object-oriented approach to representing STAR files (OOSTAR) which is then used in several applications of importance to crystallographers. Applications are described for validating the integrity of STAR-compliant files, extracting subsets of data from STAR files and converting STAR files to HyperText Markup Language (HTML) for use with a Web browser. SunOS 4.1 binaries, Objective-C source code and instructions for compiling and testing on other Unix-based hardware platforms are available *via* the World Wide Web from http://www.sdsc.edu/pb/cif/OOSTAR.html.

1. Introduction

The Self-defining Text Archive and Retrieval (STAR) file format is a simple, easy to comprehend, extensible data exchange format. The motivation for developing STAR and the resultant syntax specifications has been described previously (Hall, 1991; Hall & Spadaccini, 1994; Hall & Cook, 1995) and only a brief summary is given here. The basic STAR format consists of a set of tag-value pairs. These tag-value pairs are referred to as data items. Thus, a data item is identified by its value and the unique tag that the value has associated with it. Syntax and semantics are clearly separated since any semantics associated with the data item are defined in separate domain-specific dictionaries. Crystallographic dictionaries include the core dictionary (Hall *et al.*, 1991), the macromolecular dictionary (Bourne *et al.*, 1997) and the powder diffraction dictionary (Toby, 1997). Tag-value pairs are enclosed in a data block. A data block starts with a `data_-blockcode` tag, where 'blockcode' is a unique identifier for the data block, and is followed by the associated tag-value pairs. A data block ends when another data block starts or at an end-of-file. A STAR file consists of one or more data blocks and an optional leading global data block, which contains information that is applicable across multiple data blocks in a STAR file. A global data block is identified by the `global_blockcode` statement. A save frame is an optional referenced subcomponent nested inside a data block. A save frame starts with a `save_savecode` where 'savecode' is an identifier used to reference a save frame within a data block. A save frame ends with the reserved word `save_`. The application of a save frame in crystallography is currently restricted to the macromolecular crystallographic information file (mmCIF) dictionary (Fitzgerald *et al.*, 1997). Repetitive data items can be packaged into a loop structure contained within a single data block. A data loop structure consists of a `loop_` statement followed by a list of data names and then a repeated list of data values that can be decomposed and matched to a corresponding data name. To maintain the correct correspondence between tags and values, values cannot be missing from the loop. If a data value is not known it must be represented as either a period (.) to signify that is is missing, or a question mark (?) to signify that it is not relevant in the current context. A loop structure can be nested inside another data loop structure to construct arbitrarily complex data loop structures. Each level of loop must be terminated by a `stop_` statement, except the outermost loop, which is terminated by the occurrence of a new data item, a save frame, a data block, or an end-of-file. Nested loops are not currently used by any crystallographic implementations of the STAR format which leads to some unnecessarily awkward data representations.

OOSTAR, written in the Objective-C programming language (Pinson & Richard, 1991; Free Software Foundation, 1997), provides an object-oriented representation of these STAR encoding rules. The advantages of Objective-C over the more commonly used C++ language include: (i) looser data typing; (ii) run-time type checking; and (iii) better message-passing ability. Loose typing allows software designers to avoid certain immutable design decisions and results in a flexible software system capable of adapting to various applications. The disadvantage of this approach relates to efficiency, but this was not a problem in the applications described here. Loose-typed languages like Objective-C query type information at run-time. Objective-C provides a run-time library that enables a software system to query the *meta* information of objects at run-time while C++ provides relatively weak run-time typing support.

Message passing between objects facilitating dynamic binding, compared to functional calls (*i.e.* static binding) found

in structured programming languages, is one of the major benefits of the object-oriented approach. C++ supports message passing through the virtual method mechanism. In order for this mechanism to work, a proper class inheritance tree is a prerequisite. Objective-C dynamically interprets messages using the run-time system and does not require a supporting class inheritance tree. Flexible system design and software adaptability were two of the objectives of this project; therefore the Objective-C programming language was more appropriate than the C++ programming language at the price of being less familiar to potential application developers. It should be noted that the Java programming language, the most popular object-oriented language at the time of writing this paper, was not available when this project began. It would be straightforward, however, to apply the OOSTAR design when writing Java code.

To reduce development effort, OOSTAR and the accompanying applications (jointly referred to as the tool set in this paper) were built on top of two public-domain Objective-C class libraries. These libraries are Thorup's Objective-C object and run-time library (Thorup, 1996) and McCallum's collection library (McCallum, 1997). The tool set is organized as two groups of programs, a STAR file representation (*i.e.* OOSTAR) and three application programs described below.

2. OOSTAR

A bottom-up design approach was applied by modeling STAR syntax (Fig. 1). *ItemAssoc* is implemented as a class consisting of two objects, an item and the value or a list of values associated with the item. *ItemAssoc* can be used to represent any relationship between two objects. *StarAssoc* is a class that is derived from the basic *ItemAssoc* class. A `_name` data member and additional methods which facilitate the manipulation of STAR files were added to this class, for example *setItemName:*, *getItemName:* and *addvalue:*. *DataBlock* is a class that is derived from the *StarAssoc* class. The `_name` attribute in the *DataBlock* class is interpreted as the data-block identifier, blockcode, and the value is the first data item in the data block. Methods were added to this class to provide STAR data-block behavior, for example *dataBlockName:*, *associatedDictionaries:* and *getDataItem:*. *ItemAssoc* contains pointers to its previous and next elements and thus can be used to represent one or more associations as well as being iterative. Since *StarAssoc* and *DataBlock* are both derived from *ItemAssoc* they are also iterative, making it a simple matter, for example, to iterate over multiple data blocks. *StarFile* is a class that is derived from *DataBlock*. The `_name` attribute of *StarFile* is the STAR file name and the value is the first data block in the file.

As a result of Objective-C's loose compile-time typing and run-time *meta* information support, the design described above provides sufficient power to represent various combinations of STAR files while remaining a relatively straightforward design. A set of rules, in keeping with STAR syntax rules, has been developed to use the classes described. These rules are:

(i) A STAR file consists of one or more *DataBlocks*.

(ii) *DataBlock* consists of a data-block name and one or more *StarAssocs* as its values.

(iii) *StarAssoc* consists of a name, a pointer to the data block which provides the definition of the data item, and the value associated with the name.

(iv) If the *StarAssocs* name is not `loop_` then the value attribute is either a string or a list of strings (*LineBuffer*).

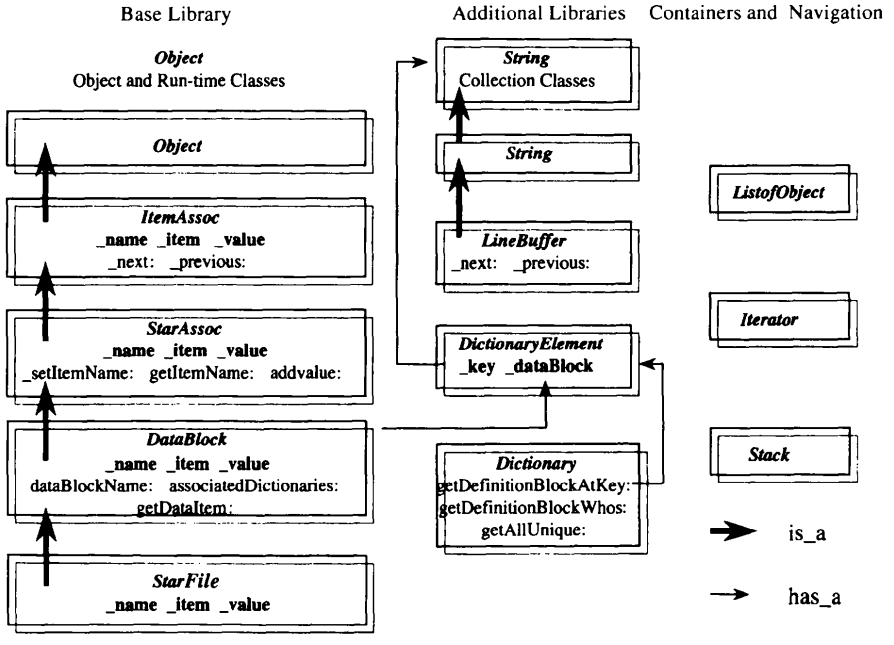

Fig. 1. OOSTAR classes. Classes are shown in rectangular boxes along with important attributes (bold) and methods (followed by colons). Classes are organized into three columns representing their source libraries. Thick arrows indicate an 'is_a' relationship, thin arrows a 'has_a' relationship.

Otherwise, the value attribute consists of a list of *StarAssocs* and the data block is null.

(v) Rule (iv) can be applied recursively to represent STAR nested loop structures.

The above rules have been translated into a STAR parser based on *Bison* (a GNU version of *yacc*; Free Software Foundation, 1997). The parser parses the token retrieved from a scanner, which was written in *Flex* (a GNU version of *lex*; Free Software Foundation, 1997), to construct the in-memory OOSTAR representation. The scanner was developed independently of the parser and can be used for any STAR application.

A total of 18 classes have been developed for OOSTAR. Beside the essential classes described above, other classes including *ListofObject*, *Iterator* and *LineBuffer* have been implemented as container classes and for navigation. Several classes from McCallum's collection library were supplemented to serve specific STAR requirements. These classes include *String*, *Stack* and *Dictionary* (Fig. 1).

Specialized query methods were added to the collection library's *Dictionary* class to assist the study of STAR and associated dictionaries using existing DDLs. These methods include *getDefinitionBlockAtKey*:, *getDefinitionBlocksWhos*:, *is*: and *getAllUnique*:. These methods are used in combination by the various applications. For example, a message of `getDefinitionBlocksWhos: '_category' is: 'atom_site'` to a crystallographic information file returns a collection of data blocks which contain the CIF category `atom_site`.

3. Applications

Three applications have been developed using OOSTAR to investigate the usefulness of this approach; they are *StarHtml*, *stargrep* and *startuple*.

3.1. *StarHtml*

StarHtml converts a set of STAR files into HyperText Markup Language (HTML) files for use with a World Wide Web browser, thus providing a useful viewer for exploring STAR/CIF, including complex dictionaries. *StarHtml* generates a set of interconnected HTML files based on the referential information embedded in the STAR files. Currently, the referential information is between a data item and the corresponding definition as found in a STAR (and hence CIF) dictionary. Knowledge associating both STAR syntax and application semantic information with the HTML syntax was programmed into the specialized *StarHtml* object. A set of STAR files and STAR dictionaries are used as input to the application and a comprehensive in-memory representation of these STAR files generated. The *StarHtml* object navigates the OOSTAR network and generates HTML files to represent the associations that the object has encountered in the network. *StarHtml* is executed with the following command:

`StarHtml dir dic1 [dic2 [dic3]. ..] [file1 [file2]...]`

where *dir* is the target directory to contain all HTML files, *dic1*. . *dicn* are the paths to the STAR dictionaries and *file1*. . *filen* are the paths to the STAR data files.

Two HTML files are generated for every STAR file or dictionary that was passed to the *StarHtml* application and the master HTML file SFBrowser.html is generated to serve as the

```
data_Ex3

loop_
  _display_id
  _display_object
  _display_symbol
  _display_colour
  _display_size
  _display_coord_x
  _display_coord_y
  loop_
    _display_conn_id
    _display_conn_symbol

 1 text BH   blue   5  35 70 2 1b  7 1b  8 1b 12 1b stop_
 2 text BH   blue   5  10 60 3 1b  7 1b  8 1b  8 1b stop_
 3 text BH   blue   5  10 30 4 1b  8 1b  9 1b 10 1b stop_
 4 text BH   blue   5  35 20 5 1b  9 1b 10 1b 11 1b stop_
 5 text BH   blue   5  60 30 6 1b 10 1b 11 1b 12 1b stop_
 6 text BH   blue   5  60 30  1 1b 11 1b 12 1b  7 1b stop_
 7 text BH   blue   5  35 57  9 1b stop_
15 text Me   black  5  80 50 stop_
16 text Me   black  5 100 50 stop_
```

(*a*)

```
% stargrep  ex.3  data_Ex3  _display_conn_id

2
7
8
12
.
.
.
.
.
.
11
12
7
9
```

(*b*)

```
% startuple ex.4 data_Ex3  _display_id _display_colour   _display_conn_id
_display_conn_symbol

1 blue 2 1b
1 blue 7 1b
1 blue 8 1b
1 blue 12 1b
2 blue 3 1b
2 blue 7 1b
2 blue 8 1b
2 blue 8 1b
.
.
5 blue 6 1b
5 blue 10 1b
5 blue 11 1b
5 blue 12 1b
6 blue 1 1b
.
6 blue 7 1b
7 blue 9 1b
15 black (null) (null)
16 black (null) (null)
```

(*c*)

Fig. 2. (*a*) A Molecular Information File (MIF). (*b*) Application of *stargrep*: extracting values associated with the tag `_display_conn_id` from the data block *Ex*3 contained in the MIF called ex.3. (*c*) Application of *startuple*.

entry point to all the HTML files. StarHtml has been used to process STAR files from Molecular Information File (MIF; Allen *et al.*, 1996) and CIF applications using DDL1.4. *StarHtml* was later enhanced to process the mmCIF dictionary and data files developed using DDL2.1.1.

DDL2.1.1 extends DDL1.4 by providing a richer set of definitions for relationships between data items and stronger data typing. In the core CIF dictionary, which is based upon DDL1.4, all data items are defined in a single data block without the use of save frames. Save frames have been used in DDL2.1.1 to delineate data items. Both of these situations are STAR compliant; hence, the underlying OOSTAR representation required only minor change to encompass DDL2.1.1 from a version starting with DDL1.4. The only modification made to the basic structure was during the building of *Dictionary* objects. Instead of creating a key table against `data_blockcode`, the *Dictionary* object builds the key table against `save_framecode`. Other minor modifications were made at the application level. For example, *StarHtml* objects were modified to reflect the alias mechanism developed in DDL2.1.1. Similarly, instead of locating the information for `_list_link_parent` in DDL1.4, `_item_linked.parent_name` was searched when using DDL2.1.1. HTML versions of several crystallographic dictionaries can be found at http://www.sdsc.edu/pb/cif/dictionaries.html.

3.2. *stargrep and startuple*

Two applications that query STAR files (see Fig. 2*a* for an example of a STAR-compliant file) have been developed. The *stargrep* application is a query utility that enables a user to query information associated with a specific data item within a specific data block in a specific STAR file. The target item can be either single- or multiple-valued. The result of the query is presented as one value per line. The command for running stargrep is:

`stargrep starfilename datablockname dataitemname`

The result of running *stargrep* on the STAR input file given in Fig. 2(*a*) is shown in Fig. 2(*b*).

The *startuple* application is another query utility which flattens the data loop structure in a STAR file and presents data in a tabular form. The command to run startuple is:

`startuple filename datablockname dataitemname1 [dataitemname2...]. ...`

The result of this query is presented as one row of the table per line. An example of the use of *startuple* is given in Fig. 2(*c*) using the STAR-compliant input file given in Fig. 2(*a*).

4. Discussion

Object-oriented languages are well suited for modeling STAR and CIF encoding rules, and have the advantages of code reuse and extensibility at the price of poorer performance and a more limited audience of application developers capable of using the code than for structured languages like Fortran and C. Performance is not an issue in the simple translators described here. Even the macromolecular CIF dictionary with over 3200 data items can be processed in a few seconds on a typical Unix workstation. Likewise, usability is less of an issue as more crystallographers become familiar with object-oriented programming. We think it unlikely that Objective-C will become a popular object-oriented language among crystallographers with the advent of Java (Java did not exist when this project began). Hence, we offer the code described here as more of a template for the object-oriented representation of STAR files than for use in further application development. Nevertheless, the applications themselves may prove useful.

Two other applications that convert CIFs to HTML beside *StarHtml* described here are known to exist. First, *CIFLIB* (Westbrook *et al.*, 1997), a C++ class library, has been used to develop a convertor which operates on DDL2.1.1-compliant files. Versions of the DDL and mmCIF dictionaries converted to HTML with this tool are available at http://ndbserver.rutgers.edu/mmcif. Second, Murray-Rust (1994) developed a prototype convertor for DDL1.4-compliant CIFs written in the interpretive language Tcl. Each offers a somewhat different view of a STAR/CIF, but provides the same subset of hyperlinks since these are specified by STAR syntax rules rather than the application developer.

The functionality of *stargrep* and *startuple* could be provided in Fortran using *CIFtbx* (Hall & Bernstein, 1997) for both STAR/CIF DDL1.4- and DDL2.1.1-compliant files. *stargrep* and *startuple* have been incorporated into the Python programming language to provide an interactive query environment. Python (Lutz, 1996) is an interpreted, interactive, object-oriented programming environment. It incorporates modules, exceptions, dynamic typing and classes. The Objective-C STAR representation was enclosed as a Python module. Through Python's interpreter, we could interrogate the information stored in STAR format interactively. This system has been used to study, for example, the integrity of a STAR file and derived parent/child relationships in STAR dictionaries.

This work was supported by the National Science Foundation grant BIR 9310154 and the US Department of Energy. This work owes much to the community of CIF and mmCIF developers and users.

References

Allen, F. H., Barnard, J. M., Cook, A. F. P. & Hall, S. R. (1996). *J. Chem. Inf. Comput. Sci.* **35**, 412–427.

Bourne, P. E., Berman, H. M., McMahon, B., Watenpaugh, K., Westbrook, J. D. & Fitzgerald, P. M. D. (1997). *Methods Enzymol.* **277**, 571–590.

Fitzgerald, P. M. D., Berman, H. M., Bourne, P. E., McMahon, B., Watenpaugh, K. & Westbrook, J. D. (1997). *mmCIF*, http://ndbserver.rutgers.edu/NDB/mmcif/dictionary/.

Free Software Foundation (1997). *GNU's Not Unix! – the GNU Project and the Free Software Foundation (FSF)*, http://www.gnu.ai.mit.edu/.

Hall, S. R. (1991). *J. Chem. Inf. Comput. Sci.* **31**, 326–333.

Hall, S. R., Allen, F. H. & Brown, I. D. (1991). *Acta Cryst.* A**47**, 655–685.

Hall, S. R. & Bernstein, H. J. (1997). *CiFtbx, Cyclops, cif2cif*, http://ndb.rutgers.edu/NDB/mmcif/software/ciftbx/.

Hall, S. R. & Cook, A. P. F. (1995). *J. Chem. Inf. Comput. Sci.* **35**, 819–825.

Hall, S. R. & Spadaccini, N. (1994). *J. Chem. Inf. Comput. Sci.* **34**, 505–508.

Lutz, M. (1996). *Programming Python*. Sebastopol, California, USA: O'Reilly.

McCallum, A. K. (1997). *GNU Objective C Class Library Home Page*, http://www.cs.rochester.edu/u/mccallum/libobjects/home.html.

Murray-Rust, P. (1994). *CIF Resources at CBMT*, http://www.seqnet.dl.ac.uk/CBMT/cif/HOME.html.

Pinson, L. J. & Richard, R. S. (1991). *Objective-C Object-oriented Programming Techniques*. Reading, Massachusetts, USA: Addison Wesley (http://devworld.apple.com/techinfo/techdocs/rhapsody/NextLibrary/Documentation/NextDev/TasksAndConcepts/ObjectiveC/Welcome.html).

Thorup, K. K. (1996). *User's Guide to the GNU Objective-C Class Library – Table of Contents*, http://www.math.utah.edu/docs/info/libobjects_toc.html.

Toby, B. (1997). *(IUCr) Powder CIF Dictionary*, http://www.iucr.org/cif/pd/index.html.

Westbrook, J. D., Hsieh, S.-H. & Fitzgerald, P. M. D. (1997). *J. Appl. Cryst.* **30**, 79–83.