Journal of
Applied Crystallography

ISSN 0021-8898

Received 6 April 2010
Accepted 16 May 2010

ASTAR: a .NET class library for STAR/CIF manipulation

Yun Lin

Instrumental Analysis and Measurement Center, Fuzhou University, Fuzhou, Fujian, 350002, People's Republic of China. Correspondence e-mail: transi@fzu.edu.cn

A .NET class library for STAR/CIF manipulation, ASTAR, has been developed and is available at http://xstar.sourceforge.net/astar/. The library provides facilities to read and write STAR/CIF files and an object model to manipulate data in STAR/CIF files. While the library is written in the C# programming language, it can be utilized by programs written in various programming languages targeting the .NET platform.

1. Introduction

The STAR (Self-defining Text Archive and Retrieval) syntax (Hall, 1991) provides a way for simple, easy-to-comprehend, flexible and extensible data exchange. The syntax permits most types of data items, data structures and data cells.

The CIF (Crystallographic Information File) syntax (Hall *et al.*, 1991), derived from the STAR syntax, is widely used in crystallography for data archiving and exchange. While CIF was primarily designed to describe small-molecule structures and crystallographic experiments, a specialized CIF format, mmCIF (Bourne *et al.*, 1997), was developed to describe protein structures and is an alternative to the PDB (Protein Data Bank; Bernstein *et al.*, 1977; Bermann *et al.*, 2000) format. Besides the advantages mentioned above, another important feature of CIF is that data in CIF files can be validated against standard dictionaries which are written in the Dictionary Definition Language (DDL) (Hall & Cook, 2005; Westbrook *et al.*, 2005). A DDL dictionary is itself a CIF file containing definitions of data items and categories used in CIF data files.

Another application of STAR is NMR-STAR, which was developed for archiving and exchanging data from NMR spectroscopic studies on biomolecules, at the Biological Magnetic Resonance Bank (BMRB, http://www.bmrb.wisc.edu/).

Programming libraries for manipulation of STAR/CIF files are available in various programming languages, including Fortran (Hall & Bernstein, 1996), C (Westbrook *et al.*, 1997), Objective C (Chang & Bourne, 1998), Perl (Bluhm, 2000) and Python (Hester, 2006). These libraries can significantly ease the task of adding CIF or STAR functionality to software projects. ASTAR, described in this article, is a class library developed for working with CIF and STAR files on the .NET platform.

The .NET Framework is a new platform for building, deploying and running applications. It was first developed by the Microsoft Corporation (http://www.microsoft.com/net/). Part of the Microsoft .NET Framework was standardized by the ECMA (European Computer Manufacturers Association, 2001, 2006; http://www.ecma-international.org/publications/standards/Ecma-335.htm) and the ISO (International Organization for Standardization, 2003, 2006; http://www.iso.org/iso/catalogue_detail.htm?csnumber=42927) as an open specification under the name of Common Language Infrastructure (CLI), which was published as ECMA-335 and ISO/IEC 23271. Mono (http://www.mono-project.com/) is an open-source and cross-platform implementation of the Common Language Runtime (CLR), which is binary compatible with Microsoft .NET and supports Windows, Linux, Mac OS X, Free BSD and other UNIX-like operating systems. The .NET Framework offers a number of advantages to software developers, including support for many modern programming features (such as memory management, exception handling, thread management and concurrent computing), a consistent programming model and a foundational class library across different languages. Programs and modules on .NET are compiled into Common Intermediate Language (CIL) assemblies which are portable across software and hardware platforms. The .NET platform does just-in-time (JIT) compilation at runtime to achieve performance close to native machine code. While .NET is widely used in information technologies, much work has been done in scientific computing with .NET, *e.g.* Math.NET (http://www.mathdotnet.com/), ILNumerics.NET (http://ilnumerics.net/) and SCINET (http://www.obacs.com/).

ASTAR, exploiting the advantages of .NET, can be utilized for STAR/CIF manipulation on various hardware and software platforms supported by .NET (either the Microsoft .NET or Mono implementation) and can interoperate with different programming languages targeting the .NET platform.

2. Implementation

2.1. Programming

ASTAR is written in the C# programming language. *GPLEX* (Gough, 2008) and *GPPG* (Kelly, 2008) are used to generate the lexical scanners and parsers from the STAR/CIF grammar files. By default, the library is compiled into a .NET assembly file (AStar.dll) and the dependencies of the lexical scanners and parsers are included. Programs written in different .NET programming languages can utilize ASTAR by adding references to this assembly file at compile-time or runtime.

2.2. Namespaces

The classes in the ASTAR library are defined in different namespaces.

(1) `AStar.Common`: the common classes for STAR and CIF are defined in this namespace, including the `NamedItemCollection<T>` class, which is the base class of most collection classes in ASTAR, and the `Measurand` structure, which represents a measurand value and the associated standard uncertainty.

(2) AStar.Cif: the classes related to CIF are defined in this namespace.

(3) AStar.Star: the classes related to STAR are defined in this namespace.

2.3. Differences between STAR and CIF

CIF is similar to STAR except for some restrictions on the CIF syntax: (i) the exclusion of the ASCII characters 0x0B and 0x0C; (ii) the limit on the maximum number of characters in a text line or a data name; (iii) the limit on the use of save-frames and save-frame references; (iv) the exclusion of global_ blocks; (v) the limit on looping levels (a recursive loop is not allowed).

ASTAR supports both specifications. Because most of these differences must be handled at the lexical scanning and parsing stages, the lexical scanner, parser and other classes for each specification are implemented independently and defined in different namespaces. The implementation of CIF has been tested against the IUCr 'trip' test suite (http://www.iucr.org/iucr-top/cif/developers/trip). In the rest of this article, we will only focus on the CIF part of ASTAR.

2.4. Object model to represent a CIF file

Fig. 1 shows the representation of a CIF file using the ASTAR objects. The important object classes in the AStar.Cif and AStar.Common namespaces are briefly described below.

(1) A CifPack object represents a CIF file and contains CifBlock objects. CifPack is the top-level collection class of the ASTAR CIF object model and is derived from the base class NamedItemCollection<CifBlock>.

(2) A CifBlock object represents a data block in the CIF file.

(3) A CifFrame object represents a save-frame in the data block.

(4) CifBlock and CifFrame are both derived from the same abstract base class CifFieldCollection.

(5) A CifFieldCollection object contains CifField objects which represent data items in the data block or save-frame. CifFieldCollection is derived from the base class NamedItemCollection<CifField>.

(6) A CifPrimitiveField object represents a tag–value pair in the data block or save-frame and holds a single CifValue object.

(7) A CifLoop object represents a loop structure in the data block or save-frame and contains CifLoopField objects.

(8) A CifLoopField object represents a tag and the associated value list in the loop structure and holds a CifValueList object.

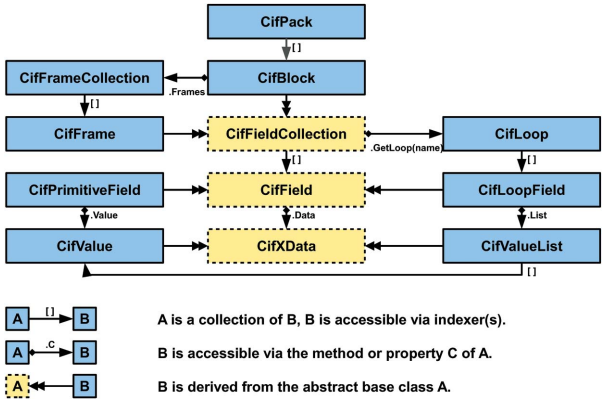

Figure 1
Representation of a CIF file with the ASTAR CIF object model.

(9) CifPrimitiveField and CifLoopField are both derived from the same abstract base class CifField.

(10) The class CifLoopRow is provided for the convenience of row-oriented data access on CifLoop.

(11) A CifValue object represents a single value. Various methods are provided for conversion between CifValue and other .NET data types, including String, Int32, Int64, Double and Measurand.

(12) A CifValueList object contains CifValue objects. CifValueList implements the IList<CifValue> interface.

(13) CifValue and CifValueList are both derived from the same abstract base class CifXData.

(14) The structure Measurand is provided for the convenience of grouping a measurand value and the associated standard uncertainty.

(15) The generic class NamedItemCollection<T> is a collection of T objects, where the class T implements the INamed interface. NamedItemCollection<T> implements the ICollection<T> interface and provides the indexing functionality based on either the order or the name (case insensitive) of the item in the collection.

3. Using ASTAR

3.1. Input, output and data manipulation

Figs. 2 and 3 show two examples written in C# and Python (the IronPython implementation, http://ironpython.codeplex.com/). More examples written in other programming languages, including VB .NET, F# (http://research.microsoft.com/en-us/um/cambridge/projects/fsharp/) and Ruby (the IronRuby implementation, http://ironruby.codeplex.com/), are available in the 'src/LangInterop' directory of the source package of ASTAR. These examples demonstrate basic CIF input, output and data manipulation with the ASTAR class library. The details are described below.

(1) File input. As shown in Part A of each example, a CifPack object is created by calling the constructor with the path to the CIF

```
using System;
using AStar.Cif;
using AStar.Common;

namespace Sample.CSharp {
    public class Program {
      public static void Main() {
        string fn = "../data/fa3203.cif";
        Console.WriteLine("Read the CIF file: {0}", fn);
        CifPack pack = new CifPack(fn);
        foreach(CifBlock block in pack) {
          Console.WriteLine("[Block: {0}]", block.Name);
          if(block.Contains("_chemical_formula_sum")) {
            Console.WriteLine("  Formula: {0}",
              block["_chemical_formula_sum"].Value);
            Console.WriteLine("  atom  U(iso/eq)  s.u.");
            foreach(CifLoopRow row in
              block.GetLoop("_atom_site_U_iso_or_equiv").Rows) {
              CifValue val = row["_atom_site_U_iso_or_equiv"];
              Console.WriteLine("  {0,-4}  {1,-9:F6}  ({2:F6})",
                row["_atom_site_label"],
                val.ToNumber(), val.ToMeasurand().Su);
            }
          }
        }

        CifBlock test = new CifBlock("TEST");
        pack.Add(test);
        test.CreateField("_astar_test_description", "CSharp");
        test.CreateField("_astar_test_measurand",
          new Measurand(2.3533, 0.0022));
        test.Add(new CifLoop("_astar_test_la", "_astar_test_lb"));
        test["_astar_test_la"].List = new CifValueList("AAA", "AAB");
        test["_astar_test_lb"].List = new CifValueList("BBB", "BBA");
        fn = "output/fa3203_test.cif";
        Console.WriteLine("Write the CIF file: {0}", fn);
        pack.Save(fn);
      }
    }
}
```

Figure 2
A code snippet which demonstrates the use of ASTAR, written in C#.

file as the argument. In more practical use, an empty `CifPack` object can be created by calling the constructor with no arguments; then various `Load` or `Parse` methods can be used to load data from different CIF sources. In this way, the behaviour of the parser is finely controlled and a list of syntax errors is also available.

(2) Data manipulation. As shown in Part B of each example, data blocks (`CifBlock` objects) within the CIF file (`CifPack` object) are iterated using the `foreach` or `for` statement, the fields (`CifField` objects) in the data block are accessible *via* the square-bracket notations (indexers). The `GetLoop` method of `CifBlock` is used to obtain the loop structure (`CifLoop` object) that contains the field with the specified name. Data in the loop structure are retrieved *via* the iteration over the `Rows` property of `CifLoop`, which implements the `IEnumerable<CifLoopRow>` interface. As shown in Part C of each example, an empty `CifBlock` object is created and added to the `CifPack` object, two primitive data items (`CifPrimitiveField` objects) are created within the newly created `CifBlock` object, one accepts a `String` as its value and the other accepts a `Measurand` as its value. A `CifLoop` object containing two fields is created, and the value list of each loop field (`CifLoopField` object) is set *via* the `List` property of `CifLoopField`.

(3) File output. As shown in Part D of each example, the `CifPack` object with the content modified in the previous procedure is saved to a new CIF file by calling the `Save` method with the specified file path. Alternatively, a `String` representation of the `CifPack` object can be created by calling the `ToText` method.

3.2. Performance test

The performance of file reading (lexical scanning, parsing) and writing (checking on names, tags, values *etc.*) with ASTAR has been tested on various CLR/OS combinations. Table 1 gives some representative times for reading and writing typical CIF files, including one common CIF data file (from the IUCr electronic archives) for a small-molecule structure (Lipstman & Goldberg, 2009), two mmCIF files (from RCSB PDB, http://www.pdb.org/) for protein structures (Varghese & Colman, 1991; Gamblin *et al.*, 2004), and two dictionary files (from ftp://ftp.iucr.org/pub/) written in DDL1 and DDL2.

The overall reading speed on the Mono/Linux combination is slightly faster than on the Microsoft .NET/Windows combination. The overall writing speed on the Mono/Linux combination is about two to four times slower than on the Microsoft .NET/Windows combination, which is due to the reduced efficiency of the implementation of Regex on Mono. Despite these differences caused by different runtime implementations, the performance can be considered high enough for common desktop or server applications on modern computer platforms, though it may not compete with some STAR/CIF libraries which are implemented in lower-level languages such as C and Fortran.

4. Applications

Two programs that utilize the ASTAR class library to manipulate CIF files are included in the ASTAR project.

4.1. *AXC*: conversion between CIF and XML

The *AXC* program does conversion between CIF and XML. XML is widely supported by computer software for data exchange. *AXC* represents CIF data with the XML Document Object Model (DOM) in a straightforward way, which is detailed in the online documentation. With the help of XSLT, the XML files can be transformed into various XML-compliant formats, *e.g.* XHTML. The following gives some sample uses of *AXC*:

(1) `axc ciftoxml a.cif`

Converts the CIF file 'a.cif' to an XML file 'a.xml'.

(2) `axc xml2cif b.xml`

Converts the XML file 'b.xml' to a CIF file 'b.cif'.

(3) `axc xslt -l ctod.xsl -o d.xml c.xml`

Transforms the XML file 'c.xml' to another XML file 'd.xml' using the XSL style-sheet 'ctod.xsl'.

Table 1
Execution times on a Pentium 4, 3.0 GHz PC running (*a*) Microsoft .NET 2.0/Windows XP and (*b*) Mono 2.4.2.3/Linux 2.6.24.

		Read time (ms)		Write time (ms)	
File name	File size (kB)	(*a*)	(*b*)	(*a*)	(*b*)
fa3203.cif	36	13	15	8	20
1nn2.cif	523	392	335	252	530
1ruz..cif	1344	1287	933	570	2093
cif_core.dic	479	77	93	60	171
cif_mm.dic	1757	343	366	230	572

4.2. *ATrans*: template-based CIF-to-text transformation

The *ATrans* program is used to generate files in arbitrary text formats from CIF files based on template files. The following gives a sample use of *ATrans*:

```
atrans -t report.tex.cif -o a.tex a.cif
```

The command generates a LaTeX file 'a.tex' from the CIF data file 'a.cif', using the template file 'report.tex.cif'.

A template file is a CIF file containing the template entries defined with a particular set of tags. Fig. 4 shows a sample template entry, which is defined within a data block. The value of `_tdl_context_type` indicates the type of item that the template should be applied to. In this example, the value 'BLOCK' indicates that the item must be a data block. The looping of `_tdl_item_type`, `_tdl_item_select` and `_tdl_item_apply` is used to select child items in the context and apply templates to them. The values of `_tdl_item_type` and `_tdl_item_select` specify the type and name of the child item to be selected. The values of `_tdl_item_apply` specify the template text or the names of the

```
import clr
clr.AddReference('AStar.dll')
from AStar.Cif import *
from AStar.Common import Measurand

A
fn = '../data/fa3203.cif'
print 'Read the CIF file: %s' % fn
pack = CifPack(fn)

B
for block in pack:
  print '[Block: %s]' % block.Name
  if block.Contains('_chemical_formula_sum'):
    print '  Formula: %s' % block['_chemical_formula_sum'].Value
    print '  atom  U(iso/eq)  s.u.'
    for row in block.GetLoop('_atom_site_U_iso_or_equiv').Rows:
      val = row['_atom_site_U_iso_or_equiv']
      print '  %-4s  %-9f  (%f)' % (row['_atom_site_label'],
        val.ToNumber(), val.ToMeasurand().Su)

C
test = CifBlock('TEST')
pack.Add(test)
test.CreateField('_astar_test_description', 'PYthon')
test.CreateField('_astar_test_measurand',
  Measurand(2.3533, 0.0022))
test.Add(CifLoop('_astar_test_la', '_astar_test_lb'))
test['_astar_test_la'].List = CifValueList('AAA', 'AAB')
test['_astar_test_lb'].List = CifValueList('BBB', 'BBA')

D
fn = 'output/fa3203_test.cif'
print 'Write the CIF file: %s' % fn
pack.Save(fn)
```

Figure 3
A code snippet which demonstrates the use of ASTAR, written in Python.

```
data_block
    _tdl_context_type BLOCK
    loop_
        _tdl_item_type
        _tdl_item_select
        _tdl_item_apply
        PRIMITIVE '_chemical_name_systematic'
;
\section{${(cif_markup_to_latex value)}}
;
        LOOP '_atom_site_label' tbl_atom
```

Figure 4
A sample template entry.

template entries (data blocks). In this example, the primitive field '_chemical_name_systematic' in the context (a data block) is selected and the template text is applied to it (with the variables and expressions evaluated and expanded), then the loop structure which contains the field '_atom_site_label' is selected and the template 'tbl_atom' is applied to it.

Sample template files to transform CIF into various text formats, including HTML, LaTeX, RTF and the plain text format, are provided in the 'data/tdl' directory of the source package of ASTAR. The syntax of template files is detailed in the online documentation.

5. Availability

ASTAR is released under the LGPL or BSD licence. The source and binary packages are freely available at the project web site, http://xstar.sourceforge.net/astar/. The binaries are compiled with Microsoft .NET SDK 2.0 on Windows XP and can be used on Microsoft .NET (2.0 and above) and Mono (1.2 and above). Full documentation and examples are also available online.

The author is grateful to the project hosting service provider, SourceForge.

References

Berman, H. M., Westbrook, J., Feng, Z., Gilliland, G., Bhat, T. N., Weissig, H., Shindyalov, I. N. & Bourne, P. E. (2000). *Nucleic Acids Res.* **28**, 235–242.
Bernstein, F. C., Koetzle, T. F., Williams, G. J. B., Meyer, E. F. Jr, Brice, M. D., Rodgers, J. R., Kennard, O., Shimanouchi, T. & Tasumi, M. (1977). *J. Mol. Biol.* **112**, 535–542.
Bluhm, W. (2000). *STAR (CIF) Parser*, http://pdb.sdsc.edu/STAR/index.html.
Bourne, P., Berman, H. M., Watenpaugh, K., Westbrook, J. & Fitzgerald, P. M. D. (1997). *Methods Enzymol.* **277**, 571–590.
Chang, W. & Bourne, P. E. (1998). *J. Appl. Cryst.* **31**, 505–509.
European Computer Manufacturers Association (2001). *Standard ECMA-335 Common Language Infrastructure (CLI)*, 1st ed. ECMA, Geneva, Switzerland.
European Computer Manufacturers Association (2006). *Standard ECMA-335 Common Language Infrastructure (CLI)*, 4th ed. ECMA, Geneva, Switzerland.
Gamblin, S. J., Haire, L. F., Russell, R. J., Stevens, D. J., Xiao, B., Ha, Y., Vasisht, N., Steinhauer, D. A., Daniels, R. S., Elliot, A., Wiley, D. C. & Skehel, J. J. (2004). *Science*, **303**, 1838–1842.
Gough, J. (2008). *GPLEX*. Version 1.0.1, http://plas.fit.qut.edu.au/gplex/.
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. (1996). *J. Appl. Cryst.* **29**, 598–603.
Hall, S. R. & Cook, A. P. F. (2005). *International Tables for Crystallography*, Vol. G, edited by S. R. Hall & B. McMahon, ch. 2.5, pp. 53–60. Heidelberg: IUCr/Springer.
Hester, J. R. (2006). *J. Appl. Cryst.* **39**, 621–625.
International Organization for Standardization (2003). *ISO/IEC 23271:2003 Common Language Infrastructure (CLI) Partitions I to VI,* 1st ed. ISO, Geneva, Switzerland.
International Organization for Standardization (2006). *ISO/IEC 23271:2006 Common Language Infrastructure (CLI) Partitions I to VI,* 2nd ed. ISO, Geneva, Switzerland.
Kelly, W. (2008). *GPPG*. Version 1.3.1. http://plas.fit.qut.edu.au/gppg/.
Lipstman, S. & Goldberg, I. (2009). *Acta Cryst.* C**65**, m371–m373.
Varghese, J. N. & Colman, P. M. (1991). *J. Mol. Biol.* **221**, 473–486.
Westbrook, J. D., Berman, H. M. & Hall, S. R. (2005). *International Tables for Crystallography*, Vol. G, edited by S. R. Hall & B. McMahon, ch. 2.6, pp. 61–70. Heidelberg: IUCr/Springer.
Westbrook, J. D., Hsieh, S.-H. & Fitzgerald, P. M. D. (1997). *J. Appl. Cryst.* **30**, 79–83.