JsCifBrowser: A Primer

JsCifBrowser is a JavaScript codebase for viewing and applying dictionary Relational Expression Language (dREL) methods to Crystallographic Information Files (CIFs). To read and process local CIF and dictionary files within a web-browser it requires the HTML5 file reading extensions. The file access API is supported by several modern browsers, alegedly including Internet Explorer IE10. The Google Chrome browser requires the command line option "--allow-file-access-from-files" to switch local file access on.

Some of the algorithms in dREL methods scripts are numerically demanding, particularly if they involve looped data tables. Consequently a relatively efficient JavaScript engine is preferable, e.g. *SpiderMonkey* in *Firefox*, or even better, the V8 engine in *Chrome*. Although JsCifBrowser is handled by *Safari*, the processing speed is at present too slow for general use.

This version of JsCifBrowser is intended, and designed, to run purely locally, i.e. it doesn't load remote dictionaries, or CIF files.

Using the Browser

Point your HTML5 web browser at either the src/index.html file (for a *dictionary hierarchical view*), or src/interp.html (for a *flat-file* view) of CIF instance data. The two browser types are described below.

Basic steps to follow for the Index browser are:

A. In the *Control* window load the dictionary file(s). Start with dictionaries in the **dic** folder: **cif.dic.js** or **cif.dic.xrodic** or all of the files in the **dic/ddlm** folder.

B. In the *Control* window load the CIF instance data. Start with the test CIFs in the **data** folder.

C. In the *Data* window (once a CIF is loaded this opens automatically) try a few of the control links, such as **[++]** and **Eval**. These are described below.

D. In the *Control* window hit the **Validate** button. This will automatically compare all instance values with values evaluated from the dictionary methods scripts. The report on this process are shown in the *Checks* window.

E. In the Control window hit the **Eval all** button to inject new values for all instances set as **'?'**. This is achieved by evaluation from the methods scripts, otherwise using the defined default values.

F. If you wish to retain a copy of the modified CI, in the Control window hit the **Export** button, and save the file shown.

Some other things to try:

1. Load every dictionary file from **dic/ddlm** simultaneously (shift click shift click in *Firefox* file browser popup) or you can just choose the top-level dictionary **cif.dic**, and you will be prompted to load all of its dependencies as they are being resolved. The latter approach is a bit tedious, though necessary to work within the Web Browser security model.

1. 2. Because DDLm dictionaries support aliased tags, you should be able to load a CIF-1.0 file (like **xtest0.cif**) and evaluate some missing items. The resulting CIF will be a mix of DDL tags. Note that tags not recognised (i.e. not defined or aliased in the loaded dictionary) will be shown in **magenta**. Unrecognised tags will not affect the working of the browser, apart from being *inert* to any of the valid/evaluation operations.

2. 3. One could try to load a DDL2 dictionary and corresponding CIF, but in the absence of methods it may not be too rewarding.

3. 4. One could could also try to load a DDL1 dictionary. This has been tested and seems to be OK.

JsCifBrowser index.html: Hierarchy View

In this view, it is essential to load your dictionary(s) before you read in your CIF. Loading the CIF establishes all the cross linking and categorisation of the various data attributes, vital for building the hierarchy. When the CIF is loaded it is displayed simply as an abbreviated list of data_block_names. Selecting the "**[+]**" links permits individual blocks to be navigated as desired. The "**[++]**" link opens up all categories containing instance data.

For each data_block_name, there is a "**Gen**"(erate) control that causes certain categories to be "calculated" via their category Evaluation methods (e.g. model_site). Hover the mouse cursor over the Gen control for a second or so to activate the popup, then choose the desired category. Categories below the marker line in the popup list are missing

and have no Evaluation method so will just be added as empty categories.

Similarly, for sub categories and sub-sub categories there are "**Add**" controls to add absent sub-categories or missing data_items to selective categories.

If particular data items have an associated dictionary Evaluation method then you can click the "**Eval**" control (don't be concerned if various "Alert" windows pop up - this is a feature of different Java versions! Various "Evaluation" methods are being harvested from the loaded dictionary, transformed to javascript, compiled and evaluated as necessary.

JsCifBrowser interp.html: Flat-file View

In this mode a default, pre-parsed dictionary is pre-loaded with the page. You can optionally load an alternative dictionary if you wish, but you should do this before loading your CIF.

Because there are no hierarchical cues regarding possible missing categories or attributes, you will need to remember them and enter them as text strings into the command menu. These commands only apply to a single CIF data_block, so you will need to choose the relevant data_block from the selector first. Category commands like "geom_bond". etc. should have the desired effect.

Command Line Based Operation

To use this software from the command line in a Linux or Mac environment you need to have Node.js suite installed. Node.js is built on Google Chrome's V8 JavaScript engine and is prebuilt and packaged for various computing platforms (e.g. as "node" in Ubuntu, or "nodejs" in Debian)

1. When a CIF file is read it is tokenized and parsed using a STAR grammar built by Jison. The output is a Javascript hierarchical object rendered to text as a JSON object.

```
cd data

../src/util/star_reader.js  nick.cif
```

4. Similarly we can read a CIF dictionary and render it as JSON, albeit with a wrapper layer to assist with loading:

```
cd data

../src/util/dic2json.js  core_3.dic
```

7. For folks specifically interested in the parsing of dREL expressions, the expression parser can be invoked directly to see a JSON representation of the Abstract Syntax Tree as well as the transformed JavaScript as follows:

```
cd src/drel

./ast.js  ./ex1.drel
```

10. But more usefully, we can write a script like drel.js, provide it with a dictionary and a CIF data file and ask it to evaluate a few things programmatically:

```
cd data

../src/util/drel.js  ../dic/ddlm/cif.dic  nick.cif
```

Software acknowledgments:

This implementation uses

numeric-1.0.2.js
a linear algebra library
Downloadify
a flash utility for saving local files
Jison

a damn good parser generator, though parsers are built statically and Jison is not part of the source (Jison runs under node.js).
a python-2.7(?)
stock grammar syntax, cut down for dREL and Jison.