

CASL libraries on the web

Till Mossakowski

Version 0.3

November 13, 2003

1 What the CASL summary says

A library may be located at a particular *site* on the Internet. The library is referenced from other sites by a name which determines the location and perhaps identifies a particular version of the library. To allow libraries to be relocated without this invalidating existing references to them, library names may be interpreted relative to a *global directory* that maps names to URIs. Libraries may also be referenced directly by their (relative or absolute) URIs, independently of their registration in the global directory.

```
LIB-NAME      ::= LIB-ID | LIB-VERSION
LIB-VERSION   ::= lib-version LIB-ID VERSION-NUMBER
VERSION-NUMBER ::= version-number NUMBER+
```

```
LIB-ID        ::= DIRECT-LINK | INDIRECT-LINK
DIRECT-LINK   ::= direct-link URI
INDIRECT-LINK ::= indirect-link PATH
```

A direct link to a library is simply written as the URI of the library. The location of a library is always a directory, giving access not only to the individual specifications defined by the current version of the library but also to previously-defined versions, various indexes, and perhaps other documentation.

An indirect link is written:

$$FI_1/\dots/FI_n$$

where each file identifier FI_i is a valid file name, as for use in a path in a URI. An indirect link is interpreted as a URI by the current global library directory.

URI and PATH are recognized as lexical symbols only directly following the key words ‘library’ and ‘from’ in specification libraries.

CASL/Summary
9, ...:
URL changed
to URI.

The following grammar provides a minimal syntax for URI: further forms may be recognized and supported.

```

PATH-CHAR ::= A | ... | Z | a | ... | z | 0 | ... | 9
            | $ | - | _ | @ | . | & | + | ! | *
            | ' ' | " " | ( | ) | , | : | ~
            | % HEX-CHAR HEX-CHAR
HEX-CHAR  ::= A | ... | F | a | ... | f | 0 | ... | 9

PATH-WORD ::= PATH-CHAR ... PATH-CHAR
PATH      ::= PATH-WORD /.../ PATH-WORD
URI       ::= http:// PATH
            | ftp:// PATH
            | file:/// PATH

```

The PATH-CHAR characters include all the 'safe' and 'extra' characters, plus the 'reserved' character ':' and the 'national' character '~'.

2 Proposal for implementation in the Heterogeneous Tool Set (HETS)

Briefly some words about what HETS is: HETS (<http://www.tzi.de/cofi/hets>) is a tool combining various tools for different specification languages, thus providing a tool for a heterogeneous specification language. The structuring constructs of this language are those of CASL, plus some new heterogeneous constructs like

```
logic <language-name>
```

for indicating the language of the subsequent specification text. Languages currently supported by HETS are CASL, HASCASL, CSP-CASL and Haskell.

Now the proposal how to implement the CASL library mechanism in HETS:

1. The global directory for the indirect links is accessed by a list of mirror sites that can be obtained from

```
www.cofi.info/Libraries.
```

The mirror sites are copied from a central server (initially `cvcs-agbkb.informatik.uni-bremen.de/repository/CASL-lib` later based on the UniForm Workbench). The central server can be accessed through an http interface.

2. The http interface provides access to versioned libraries, following the versioning scheme below. The http interface is used by HETS (and both the http interface and HETS should do some caching). For directly browsing through the libraries `viewcvs` should suffice. Setting up an Apache server in order to be able to use `cvcsweb` is too much for the moment.

3. Direct absolute URIs are taken as such. Direct relative URIs are written `./path` or `../path`. They are interpreted relative to the URI of the referencing library.

Local files can be accessed via `file://path`. `path` must begin with a `/` (otherwise the first name in `path` it is interpreted as a hostname).

Often, it will be more convenient to access local files via `./path` or `../path`. Should `./path` then also be allowed as defining library name in a local library (and only there)?

4. Versioning: should the location of a library be a directory containing all the versions (which means that library `name` in path `path` version `nnn` is accessed via `path/name/v_nnn`, and the current version via `path/name/name`), or should the version be appended to the name of the library (which means that library `name` in path `path` version `nnn` is accessed via `path/name_v_nnn`, and the current version via `path/name`) ?

A decision in favour of the former approach is taken in the summary, the main argument being all the versions and development graphs, proof object, proof scripts etc. that shall be stored together with the library. On the other hand, editing and browsing will be easier with the latter approach, because fewer subdirectories have to be entered. Moreover, many other specification and programming language take the latter approach, and their integration into HETS should be eased. Hence, the following reconciliation of the approaches is proposed:

A CASL library is stored in a file ending “`.casl`”. A directory having the same name (but without the ending) contains previously-defined versions of the library, various indexes, tool-generated information and perhaps other documentation.

For files with a language-specific module mechanism, a language-specific ending is chosen (e.g. “`.hs`” for Haskell). For heterogeneous libraries, the ending is “`.het`”¹, indicating that the needed logic(s) have to specified within the library itself.

5. The versioning is realized in cvs with tags², one for each version (note that one cannot use cvs version numbers, because they are numbered consecutively, while CASL version numbers are chosen by the user). The versioning scheme shall be supported by a shell script that in connection with cvs automatically generates all available versions, named by the above versioning scheme. This solution uses more disk space than a dynamic checkout of individual version via cgi scripts. However, the advantage is that the versioning scheme is simple, such that it also can be simulated by people who want to provide access to their libraries via direct links but who do not want to install cgi scripts or use cvs: they can just manually

¹“`.spec`” is already reserved for RPM specification files, and Emacs comes with a standard mode for these.

²Probably, `viewcvs` orders tags alphabetically. This coincides with the intended lexicographic version ordering only if main version numbers are not greater than 9.

create the needed directories and files. (By contrast, using cgi scripts would require a different versioning scheme, with the file name and version number added as parameters to the URL, using, say, ?name and ?version.) A further advantage is that it is easy to store development graphs, proof object, proof scripts together with the particular library version they belong to.

6. What happens if different versions of the same specification are imported (via different import paths)? At least, HETS should issue a warning. But still the user could get confused by the fact that (s)he has to work with, say, two different copies of `Nat` and cannot interchange theorems and proofs between the two copies. Hence, an alternative might be to require global version numbers for each package of libraries. Here, a package is a collection of libraries needed for a specific project, or a standard package like the collection of libraries of basic datatypes. HETS should provide some means of specifying which libraries belong to a package.

Packages ease the download of library versions that together form a consistent global configuration. It would be bad if the user who does not want to use latest versions but rather particular fixed versions would have to care about the fact that say `StructuredDatatypes` version 0.9 uses `Numbers` version 0.7, and therefore (s)he also should use `Numbers` version 0.7 when importing `StructuredDatatypes` version 0.9 in order to avoid having two different copies of the `Numbers`. It is much easier if the `Basic Datatypes` have global version numbers which are the same for each library, such that `StructuredDatatypes` version 0.9 uses `Numbers` version 0.9. Actually, we are following this policy already for the `Basic Datatypes`, but it would be easier if also HETS would support some kind of configuration management along these lines. `cvs` uses tags for this, and since these shall correspond to CASL version numbers, perhaps a package is just a `cvs` project or folder in such a project, and we just should provide a mechanism for globally tagging such a folder, which would mean that not only the `cvs` tag is set, but also the version numbers used in the CASL libraries are updated accordingly.

Note that this is a tools issue that should not affect the summary.

7. URNs are not needed: the syntax of usual URIs suffices, and moreover, these are directly understood by browsers and web servers.

Acknowledgements

Thanks to Christoph Lüth, Klaus Lüttich, Achim Mahnke and Peter D. Mosses for useful discussions and comments on earlier versions of this note.