Reading ReDIF files: the *ReDIF-Perl* package

Ivan Kurmanov with Thomas Krichel

July 2000

1 Introduction

This document describes *ReDIF-perl*, a set of Perl modules to read ReDIF) data. It is is available on the web at `http://openlib.org/acmes/root/docu/redif-perl.html`.
The main component of *ReDIF-perl* is *rr.pm* is a Perl module to read ReDIF data. It reads ReDIF data, validates the structure of the contents against a ReDIF specification contained in a separate specification file, and puts the valid templates into a hash structure where they can be easily accessed for further processing.

2 Installing ReDIF-perl

2.1 Downloading the package

ReDIF-perl is available at ftp://openlib.org/acmes/root/soft/ReDIF-perl/. Unpack the software, then read the file `README` for the latest changes that may not be documented in this page yet.
The package contains

- the *rr.pm* perl module as an application interface to the ReDIF parser
- the *ReDIF::init.pm* perl module as a shared tool for ReDIF developers
- `rech`, the ReDIF checking script
- `rere`, the ReDIF reading (filtering) script
- `Makefile.PL`, the installation tool (makefile generator)
- `Configure`, the configuration utility
- related documentation about *rr.pm*, `rech` and `redif.spec`

2.2 Choosing a ReDIF home

If you have a normal directory structure as set out by the Guildford protocol, then should set the ReDIF home to be your archive directory. If you do not have such a directory structure, proceed as follows.
Choose and create a directory that will be the ReDIF home and copy the `spec/`, `etc/` and `doc/` directories of the package. If you didn't install the package with "make install" command into a generally accessibly Perl library tree, then you also need to copy the lib/ dir and rech and rere scripts. Create a `data/` directory as a subdirectory of the ReDIF home and store your ReDIF data files in it. This method is known as a "stand-alone" installation of ReDIF-perl. Don't forget to look at the REDIFHOME/etc/rech.conf.eg file.
Generally, you will want to use this package if you have or are going to have your own archive that contains ReDIF metadata as suggested by the Guildford protocol. Alternatively, you may find it useful if you produce or process, or otherwise work with the ReDIF data or if you developing some software that uses ReDIF data.
Note that the package also contains a ReDIF specification file in the `spec/` directory. However, please check with the community that uses ReDIF if that specification is current. You will find in the `doc/` directory of the package. In the `etc/` you will find default configuration file for rech. The Perl modules themselves live in the `lib/` directory.

2.3 Installing the package files

To install this package onto your machine use the following standard commands (I assume you have already unpacked the package onto your harddisk):

```
perl Makefile.PL
make
make test
make install
```

The `perl Makefile.PL` command will also execute the Configure utility which will help you to set the only one, but valuable parameter: your ReDIF home directory. The is your archive's directory if you have one. It may be any other directory, where you will create a so-called stand-alone ReDIF home (see next section). Configure will ask you about the directory name and will suggest to you to create one if it does not already exist. However it will not create more than one level of new directories.
The last step, `make install` installs *rr.pm* and other related libraries into a directory where perl will find them. For this step you will most likely wish to use superuser rights, if you have them. If you need to make a private installation of ReDIF-Perl use the `PREFIX=`*DIR* option of `perl Makefile.PL` command or execute *Makefile.PL* with your private copy of perl. After this procedure the *rech* and *rere* scripts shall be installed to a directory which is already included into your `$PATH` (e.g. `/usr/bin`). They can then be executed by from everywhere on your system.
You do not need to determine the ReDIF home directory at the installation stage. In that case you use the environment variable, REDIFDIR or you run the software from that directory, or you give a command-line parameter `-rdir` (or `-redif.dir`). The choice is yours.

3 The functions provided by *rr.pm*

To call a function from *rr.pm*, you need to make Perl find it by a `use` or `require` Perl statement.
`use rr`
The *rr.pm* module provides a ReDIF reading interface through the functions `&rr::OpenRDF`, `&rr::OpenDir`, `&rr::OpenDirTree` and `&rr:NextTemplate`. The first three of these are for opening the data source of different types. They initiate a data stream through reading. The `&rr::NextTemplate` function iterates through the stream of data, opened by an `&rr::Open...` function, allowing you to access the actual data, template by template.

3.1 `&rr::OpenRDF`

Use:
`&rr::OpenRDF` *`filename`*
`&rr::OpenRDF` *`filename`*[`,` *`filepos`*]
) `&rr::OpenRDF` *`filename`*[`,` *`filepos`*`,`][*`showfilename`*]

This is the basic function, which initiates reading of a ReDIF file *`filename`*. It returns 1 if successful, 0 if not.
The optional *`filepos`* parameter (integer number) shall be used to read the file from a specific position within the file. This may be useful if you need to get a specific template and as quickly as possible. By default the file is being read from the beginning, of course.
The optional *`showfilename`* shows how the file should be referred to, i.e. the logical file name. This will be the name in the `$::HashT{'FILENAME'}`. Imagine that you read a file called `/home/tim/RePEc/bob/bobseri.rdf`. You may want it to be referred to as just `bob/bobseri.rdf`, because `/home/tim/RePEc/` is not important and will be the same for all ReDIF files being read. In that case, use
`&rr::tt(rr::OpenRDF)( '/home/tim/RePEc/bob/bobseri.rdf', 0, 'bob/bobseri.rdf');`
This feature may appear superfluous, but is quite useful in a range of applications. By default, the logical name is the same as physical *`filename`*.

3.2 `&rr::OpenDir`

Use:
`&rr::OpenDir` *`directory`*

`&rr::OpenDir` *`directory`*[, *`showprefixlength`*]
)`&rr::OpenDir` *`directory`*[, *`showprefixlength`*][, *`filtersub`*]

Whereas `&rr::OpenRDF` function starts one-file data stream, this function opens a whole directory of ReDIF files for consequent reading template by template. Thus it is a higher level function. The exact filename the template has been read from is stored in the `$::HashT{'FILENAME'}` variable at each specific template. Following the conventions of the Guildford protocol, only files with the extension `.rdf` (case insensitive) are taken. The function returns the number of ReDIF files found if successful, 0 if not. You give the pathname of the directory to be read in *`directory`*.
The optional *`showprefixlength`* parameter is for the same reasons as *`showfilename`* parameter at `rr::OpenRDF`. But this value just sets how many of the starting characters of the *`directory`* to strip as meaningless to get a nice logical name. So if you, say, read the directory `/home/bob/RePEc/bob/dylan0`, and you want logical names of the files to be `bob/dylan0/...` then give length of `/home/bob/RePEc/` as a *`showprefixlength`*, e.g.

```
rr::tt(rr::OpenDir)('/home/tim/RePEc/bob/dylan0',length('/home/tim/RePEc/'));
```

By default, the logical name is the same as the physical full filename.
If the optional *`filtersub`* parameter is specified, it applies a user-defined filter function to choose templates that satisfy a user-defined set of criteria. This feature is described Section 5.

3.3 `&rr::OpenDirTree`

Use:
`&rr::OpenDirTree` *`directory`*
`&rr::OpenDirTree` *`directory`*, [*`showprefixlength`*]
)`&rr::OpenDirTree` *`directory`*, [*`showprefixlength`*], [*`filtersub`*]

This function is very much like the previous one, but it does a sub-directory tree search for all ReDIF files. And similarly it returns number of ReDIF files found if successful, 0 if not. The *`showprefix`* and *`filtersub`* parameters have the same meaning as for the `rr::OpenDir` function.

3.4 `&rr::NextTemplate`

This function iterates the current data stream (previously opened) to the next template. It returns 1 if successful, 0 if not. In case of a one-file stream `&rr::OpenRDF`, if 0 is returned from the `&rr::NextTemplate` it means that you have reached the end of the file. In case of multi-file streams `&rr::OpenDir` and `&rr::OpenDirTree` it means that you have reached the last correct template of the last ReDIF file that was found.

4 How to access the data

There are two ways of accessing the data in your scripts. You may choose to use one of them or both (or none) by setting appropriate options in the hash `%rr::Options`. The selected modes will influence the way reading goes internally and each way of accessing the data takes some run-time resources to prepare.
The two major forms of the data presentation are

- full-text in one multi-line string variable. This is the output of the `rere` script.
- structured (to reflect template logic) hash variable

Both forms allow processing of one template by at a time only. We discuss the structured hash form first.

4.1 The structure of `%::HashT`

When the structured presentation is enabled, then after a successful `rr::NextTemplate` function, a variable `%::HashT` will be filled up with a ReDIF template data according to the following rules.

4.1.1 Rule (a): non-cluster attributes

Simple (non-cluster) attributes of the read template become keys of the `%::HashT` and these keys have the values associated with them. The names of keys are converted to lowercase. For example, *Handle* is a simple, non-cluster attribute. After reading a template, `$::HashT{'handle'}` (mind lower case!) will give you the value of the handle. For other simple attributes like "Title", "Abstract", "Creation-Date" (for Paper or Article templates), "Name" (Archive, Series), "Template-Type" and many others, it is exactly the same. Example:

```
print "Paper: $::HashT{'title'} ($::HashT{'handle'})\n";
```

If `$T` is a reference to the hash, then you may use it to access the data from `%::HashT` by writing a bit less code. For example, in the following piece of code you can observe example of using a reference to `%::HashT` instead of using `%::HashT` itself. After reading a template the script processes it depending on its type:

```
 # Assigning to the $T variable a value of the reference to %::HashT
  $T = \%::HashT;
  #
  # some rr::Open... function call is supposed here
  #
  return if !&rr::NextTemplate;  # loading a new template
  #
  # checking the template type:
  #
  if ($T->{'template-type'} eq 'redif-paper 1.0') {

      print "Keywords: $T->{'keywords'}\n";
  } elsif ($T->{'template-type'} eq 'redif-archive 1.0') {

      print "Archive description: $T->{'description'}\n";
  }
```

4.1.2 Rule (b): clusters

Clusters of templates are repeatable. They may be repeated several times in one template and the same attributes in different repeated clusters should not be mixed up, as they are supposed to only have sense within a cluster. Each cluster of a template read by *rr.pm* will be represented as an array element. For example, a document's author names are set referred to by "Author-Name" attribute and more generally described by "Author-(*PERSON)" cluster. In the hash `%::HashT` they will be represented as elements of the `@$::HashT{'author'}` array.
This means that the "author" key of `%::HashT` will point to (reference to) an array, which has as many entries as there are authors. Each value will point to an independent hash, containing the author's "(*PERSON)" cluster data.
For example,

```
  # assume a paper template has been successfully loaded
  print "The paper has " , $#$::HashT{'author'}+1, " author(s):\n";
  #
  # now iterate through each author
  foreach $au ( @$::HashT{'author'} ) {
        # now $au contains a reference to an author's data
        print "The author is $au -> {'name'}\n" ;
  }
```

or

```
  # initializations
  $T = \%::HashT;
  $authors = $T->{'author'};
```

```
#
# an effort to be correct in English: checking the number of authors
#
if ($#$authors > 0) { $suffix = 's'; }
else { $suffix = ''; }
#
print "Author$suffix:\n";
```

4.1.3 Rule (c): attributes in clusters

Each cluster data will in turn be coded as a hash with cluster attributes as keys, similar to the rule (a). Clusters attributes will have the cluster prefix (e.g. `author-`) stripped off. For instance, `$::HashT{'author'}->[0]->{'name'}` will give you the first author's name, i.e. the value contained in the "Author-Name" tag). Note that the `->` are here optional. If there are more than just one author, then `$::HashT{'author'}[1]{'postal'}` will give second author's address, i.e. the "author-postal" attribute in the second author cluster. This rule is valid for all clusters at all levels. If we have one cluster nested within another (like "workplace-(*ORGANIZATION)" cluster in a "(*PERSON)" cluster), then the latter cluster's hash will give us access to the second-level cluster hash. A long expression like `$::HashT{'editor'}[1]{'workplace'}[0]{'postal'}` would specify the postal address of the first workplace of the second editor of a series.

4.1.4 Other hash `%::HashT` elements

The hash will also contain some additional information that is local to your site. Uppercase letters are used for this purpose. At the moment the following information will be provided:
`%::HashT{'FILENAME'}` is the name of a file where the template has been read from
`%::HashT{'STARTFPOS'}` is an integer number, signifying starting position of a template in the file
`$::HashT{'BUFFER'}` will be assigned a value if only the `Buffer` option is turned on. It will contain a multi-line string with the full-text of the preprocessed ReDIF template. This is actually the way how the second form of the data can be accessed.
At the sub-hash (clusters) level there is one more technical uppercase variable: `PREFIX`, e.g. `$T->{'file'}[0]{'PREFIX'}` or `$T->{'author'}[0]{'PREFIX'}`. This key stores the cluster attributes' prefix: `'file-'` and `'author-'` respectively for the examples. The prefix and cluster's hash keys may be used to get the original attributes of the template by uniting them in one string.
Other uppercase keys of `%::HashT` may be used for internal or other reasons in the future as software development goes on. User scripts can use the keys of `%::HashT` as listed above, but should ignore any other of them.

4.2 Buffer output

With this method you get the whole template in one string variable. Each line contains one attribute, each attribute is separated from each value by a ': ' combination, each line is separated from each other by a newline character. Extra whitespace, tabulation or new-line characters are removed. All multi-line values are converted to single-line. All attribute:value pairs come checked and pre-processed.
By default this data supply method is disabled. To enable this method you need to turn on the *'Buffer'* option of the `%rr::Options` (before opening a file), for example

```
$rr::Options {'Buffer'} = 1;
&rr::OpenRDF ( $file ) || die;
```

You can switch this option between reading different files if you need, but I can't guarantee you positive results if you switch it while reading one file.
After a successful `&rr::NextTemplate`, you get a template in a string as described above in a `$::HashT{'BUFFER'}`. For example

```
print "\$::HashT\{'BUFFER'} = '$::HashT{'BUFFER'}' ;\n";
```

will produce the following output (for instance)

```
$::HashT{'BUFFER'} = 'template-type: ReDIF-Series 1.0
name: CEP Discussion Papers
description: Discussion papers on Macroeconomics and Labour Economics
keywords: Macroeconomics, Labour
editor-name: Richard Layard
publisher-name: Centre for Economic Performance and ESRC
publisher-homepage: http://cep.lse.ac.uk/
maintainer-name: Anita Bardhan-Roy
maintainer-email: a.bardhan-roy@lse.ac.uk
handle: RePEc:cep:cepdps' ;
```

4.3 *rr.pm* Options

The user of *rr.pm* module can influence some aspects of the way it works. We have already mentioned several of the options. Here comes a full description.
Option: `'HashT'`
Default: enabled (1)
Meaning: This options sets whether to build the `%::HashT` variable from the template attributes and values. If this is enabled, the full template data: attribute:value pairs will be put into the `%::HashT`. If disabled, only the `FILENAME`, `STARTFPOS` and `BUFFER` keys will have a value in `%::HashT`.
option: `'Buffer'`
Default: disabled (0)
Meaning: This option sets whether you want to get a full-text of a template in `%::HashT{'BUFFER'}`. By default, it is disabled.
You may enable both of this options, but we recommend you to choose only what is necessary, to avoid performance losses.
Option: `'BufferEmpty'`
Default: disabled (0)
Meaning: If 'Buffer' is enabled this option sets how to treat empty attributes (with null value). If enabled, empty-value attributes will be included into the `%::HashT{'BUFFER'}` and as a `%::HashT{...}` lower case attribute, otherwise it would be ignored as meaningless.
Option: `'ReadX-Attr'`
Default: disabled (0)
Meaning: This option determines whether to process and show to the so-called X-attributes. X-attributes are the attributes that start with the 'X-' sequence. If disabled, X-attributes will be ignored. If enabled they will be included into `%::HashT{'BUFFER'}` and as a `%::HashT{...}` lower case attribute.
Other options exist that are used with scripts *rech* and *rere* scripts. They should not be interesting to the *rr.pm* users.

5 Search filters

When you use the `rr::OpenDir` and `rr::OpenDirTree` functions for accessing a bunch of ReDIF files, you may set a filter for the templates. Such a filter may guarantee that while going through an opened data stream with `rr::NextTemplate`, you will only get the templates which meet a certain criteria. For example, you may want to choose templates by type: archive, series, paper, article, software and so on.
To execute such a search with filter, you can prepare a function, that checks the criteria you need and returns the result. If the template is fine, it returns true (e.g. 1), if not it return zero or the undefined value.
When you call the `rr::NextTemplate` subroutine, it will find a next piece of data for you and then run the filter you have set. If the filter returns true, then `rr::NextTemplate` will allow your programme to process it. If filter returns a false value, `rr::NextTemplate` will look for a next one template to offer. Clearly, setting the filter may cause the module to read many templates in vain, thus wasting precious run-time resources. But some people may find it efficient from programming efficiency point of view: convenient, easy, etc.
Here is a small example of using the filter. It should make things clearer.

```
# $RepecRemo = mirrored RePEc archives data directory
#
# this is a filter function that checks a criteria
sub articlefilter {
    return 1 if $T->{'template-type'} eq 'redif-article 1.0';
    return 0;
}
sub checkOpenDirTree {
    my $f, $c, $l = length ("$RepecRemo") + 1;
    $T = \%::HashT;

    # executing a search with a filter here:
    $f = &rr::OpenDirTree ( "$RepecRemo/cre/", $l, \&articlefilter )  ;

    print "\nOpenDirTree $RepecRemo/cre/ : found ", $f ,
         " .RDF file entries\n" ;

    # processing ... (only article templates will get here)
    while ($c = &rr::NextTemplate ) {
       print $T->{'FILENAME'} , ' : ' , $T->{'handle'}, "\n";
       $count ++;
    }
}
```

This `checkOpenDirtree` subroutine will seek for ReDIF files in the *cre* archive directory and in its subdirectories and will report filenames and handles of each "article" template found.

6 The *ReDIF::init.pm* module

If you are writing an application that just needs to work in ReDIF environment and wants to get the same installation and configuration info as included software does, you may use perl module *ReDIF::init.pm*. Function `initialize()` from that package will look for a configuration information saved in a Perl-reachable module, will analyze @ARGV array (command-line arguments), environment variables and the current working directory, if necessary. It can help your program to identify: the ReDIF home directory, it's type (Guildford protocol complient or stand-alone), will find the configuration file for your program in the apropriate directory (if it needs one).
It will report the main configuration values worked out to the user (unless you make it silent, which is easy) and will save them for you in the `%ReDIF::CONFIG` hash variable. It will not import any symbols to your package's namespace (unless you ask it to).
For a detailed discussion of this module please see it's manpage (`manReDIF::init`) or pod data in `lib/ReDIF/init.pm`.