refdb tutorial

A day with the refdb clients

Markus Hoenicka

mhoenicka@users.sourceforge.net

refdb tutorial: A day with the refdb clients

by Markus Hoenicka

Revision History

Revision 1.1 2004-02-08

Revision 1.0 2002-11-22

Table of Contents

List of Tables

Preface

refdb is a reference database and bibliography tool for markup languages. The software is available under the GNU Public License (GPL) and runs on a variety of modern operating systems. For further information please visit the refdb homepage (http://refdb.sourceforge.net).

This brief tutorial shows the typical use of refdb and accessory applications from the average user's point of view. The tutorial assumes that refdb is installed on your box, that the application server and the database server are humming in the background, and that you are eager to learn how to access references, how to load new ones, and how to create bibliographies.

This tutorial does not replace the refdb manual (http://refdb.sourceforge.net/doc.html). In order to install or administer refdb, you'll need the manual. To see all available command line options, configuration variables, and less common commands, you'll need the manual. To learn how to set up and operate the web interface, you'll need the manual. This tutorial tries to get new *users* up to speed, and for everything else... well, I think by now you've got it.

Chapter 1. Getting started

So now you sit in front of a box that has the refdb clients installed and wonder what you should do next. At first you should do some very simple customizations to make further use as smooth as possible.

What your refdb administrator needs to tell you

refdb uses an external database server to store all sorts of data. In order to read from or to write to a database, the database server wants to know your username and perhaps also a password. These are not necessarily identical with your login name and password (the ones you use to log into your computer), but they could very well be. Your administrator will tell you. For the purposes of this tutorial, we'll assume your database username is "markus" and your database password is "secret".

refdb uses a client/server architecture. You'll be using refdb clients for various tasks. The clients are basically pretty dumb, and they can't do much without asking the server for more information. The server interacts with the database server and has most of the application logic. In order to talk to the application server, the clients need to know where it runs. This can be on the same box, or somewhere else in your network. Your administrator will tell you. In this tutorial we'll assume the server runs on the same box, which is the IP address "127.0.0.1" in computer lingo.

refdb can basically use several reference databases. Your administrator has to give you access rights to at least one reference database. He should provide a list of databases that you can access. Let us assume for this tutorial that you will have access to a single database called "refs".

refdb uses a *pager* to display output which would otherwise scroll off the screen. Pagers allow you to do nifty things, but the most important feature is that you can scroll back and forth. Some pagers like **less** need to be terminated explicitly (by typing **q**), others like **more** terminate automatically as soon as you reach the end of the output. You can use the commands **which less** and **which more** to find out which pagers are installed on your system.

Creating your configuration files

You can set various parameters for the refdb clients on the command line each time you start them, but this is tedious and error-prone. We're all lazy by nature, so refdb allows you to set defaults in configuration files. You can still override these defaults by using command-line options on a per-session base, but usually you won't have to.

You'll have to deal with two refdb clients: refdbc is the reference management client, and refdbib is the bibliography client (you won't run this client directly and rather use a script, but you'll have to configure the client anyway). The configuration files are aptly named `.refdbcrc` and `.refdbibrc` for refdbc and refdbib, respectively, and they should reside in your home directory. These are plain-text files with a simple syntax that you can create and edit with your favourite text editor.

The following code example shows the configuration file `.refdbcrc`, using the values that we assumed above.

```
# This is the user configuration file for refdbc
serverip        127.0.0.1
```

```
username        markus
passwd          secret
defaultdb       refs
pager           less
toencoding      ISO-8859-1
fromencoding    ISO-8859-1
```

The line starting with the hash sign (#) is a *comment*. You can add more comment lines if you want to, or leave out the comment line in the example if you remember what that file is good for anyway.

If you (or your administrator) don’t feel comfortable with storing your password in a plain-text file, you can use an asterisk "*" instead. This will cause the refdb clients to ask for the password interactively at startup.

Now what does the configuration file `.refdbibrc` look like? It is the same with the exception of the last line. You can simply create a copy and name it `.refdbibrc`, remove the last line, and you’re done.

Chapter 2. Managing references and notes

Now everything is ready to do some real work. We'll go ahead and learn how to use the reference management client.

Starting the reference management client

refdbc is the command-line client used for all tasks related to manage your references. The following sections will tell you how to search for existing references and how to add new ones (and more). This section just tells you how to start the client.

If you did a good job creating your `~/.refdbcrc` configuration file, all you have to do is:

```
~$ refdbc

refdbc:
```

The client has started and displays a prompt, waiting for your commands. If you tweaked the configuration file to let refdbc ask you for your password interactively, it will do so before displaying its prompt. If you type in your password and don't see anything of what you type displayed on the screen, that's ok. This is done on purpose to keep others from reading your password on the screen.

Getting help

Although the refdb command-line programs seem intimidating to many people used to graphical user interfaces, they're not as arcane as you may fear. Help is available with a few keystrokes. Now that refdbc waits for your commands, how can you find out what commands are available? What do you shout when you fear to drown? You'll be yelling "help" (if your audience speaks English, that is), and this is what works well with refdbc too. The uncommunicative among us can use a question mark "?" instead, although I wouldn't recommend this when you're about to drown. With refdbc you'll see something like:

```
refdbc: help
This is a command overview. To get specific information about a command, run this com-
mand with the -h option.
help                    Display this text.
?                       Synonym for 'help'.
quit                    Quit refdbc.
addref                  Add references to the database.
deleteref               Delete references from the database.
getas                   Get a list of series editors.
getau                   Get a list of authors.
geted                   Get a list of editors.
getkw                   Get a list of keywords.
getjf                   Get a list of journal names (full).
getjo                   Get a list of journal names (abbrev).
getj1                   Get a list of journal names (custom abbrev1).
getj2                   Get a list of journal names (custom abbrev2).
getref                  Get a list of references.
```

```
listdb                  List databases.
liststyle               List bibliography styles.
pickref                 Add references to your personal interest list.
selectdb                Select a database.
set                     Set new value of config variable.
updateref               Update references in the database.
verbose                 Toggle verbose mode.
whichdb                 Show current database.
refdbc:
```

The left column shows the names of the available commands, while the right column shows a brief command description.

Most of the commands need some arguments in order to make sense. As the help output above said, running any of the available commands with the `-h` option will display a short message about the purpose and the syntax of the command. The command will not run, so it is safe to play around with this option. Just to get a feel for this help system, look up how you would list available databases. You set a default database in your configuration file, but maybe there are more databases available (not all of them are necessarily refdb databases; your administrator can tell you). As you can see above, you'd use the **listdb** command. To see how it works, just type

```
refdbc: listdb -h
Lists the names of the available databases. If an argument is given as a SQL reg-
ular expression, only the matching databases are shown.
Syntax: listdb [-h] [SQL-regexp]
Options: -h           prints this mini-help
         All other arguments are interpreted as an SQL regular expression.
refdbc:
```

This is a fairly simple command that takes just one argument: A SQL regular expression to specify a group of databases. If you never had the pleasure to get in touch with regular expressions: they are somewhat like the "wildcards" you may know from command shells, but more powerful. But this is not the point here. The square brackets around the argument tell you that the argument is *optional*, that is you can run the command without the argument if you wish. If you leave out the argument, you do not restrict the search to a certain pattern, so the command will return *all* database names.

This is how quite a lot of the refdbc commands work. Of course there are more complicated ones, and you'll be introduced to some of these below.

Command line basics

The command line of refdbc works like that of many other sophisticated interactive command-line tools. If you are used to graphical user interfaces, the following hints might help to get you started:

You can edit the commands

If you type something wrong and notice this only half a line later, you don't have to erase everything back to the error: instead, you can move the cursor back, fix the error, and continue editing.

There is a tab-completion feature

> If you type the first few characters of a command, like "sel", and then hit the **tab** key, refdbc will attempt to complete this command. In this case you'll get "selectdb" as this is the only command starting with "sel". If the specified characters are ambiguous, refdbc will complete as far as possible. If you then hit **tab** a second time, it will display all available possibilities, and you can enter more characters until the completion is unambiguous. If you type an argument after the command name and try the tab completion with that, refdb will attempt to complete a filename relative to the current working directory. You can use this feature to conveniently select input files for commands that read input from files. E.g. if you type "addref ~/refs/n" and then hit **tab**, refdbc might expand this to "addref ~/refs/new.ris" if such a file exists.

There is a command history

> If you want to run a previous command again or reuse it with a slight modification, you can either scroll backwards through your previous commands with the **up** key or you can start a search for a particular command with the **Ctrl-r** keystroke (hold down the **Ctrl** key and then press **r**). You'd then type in the first few characters of the command you're looking for and refdbc displays the first match. Press **Ctrl-r** again to jump to the next match or type additional characters as required. Once you see the command you're looking for, hit **enter** to run this command again or use the arrow keys to edit the command.
>
> This command history is refdbc's way to implement the incremental search that you might know from other bibliographic software. However the command history is way more versatile as you are not limited to adding another restriction to the end of the list or removing the last restriction. Instead you can modify the query as you see fit.

Adding references

If you're new to refdb and don't have a database yet, you'll want to start by adding a couple of references. This chapter first teaches you how to add references in the main input formats RIS and risx. The subsequent sections cover the import of data from various data sources like PubMed, BibTeX, or Z39.50 servers.

> **Tip:** If you're really new to refdb but have access to an existing database, e.g. the one your department built, you might want to get acquainted with refdb by retrieving existing references. Retrieving references does not alter the database, so this is safe to play around. Once you feel comfortable, return to this section.

How to create RIS datasets

The RIS format is a plain-text tagged file format used by most Windows- and Mac-based reference management tools. A variety of other data formats supported by refdb can be converted to RIS using the conversion tools described in the following sections.

What a RIS dataset looks like

The main advantage of RIS are:

- The tag set is well suited for managing references and offprints used by scientists. There is no overhead for library capabilities.
- As a plain-text, tagged format it is easy to create and edit with basically any text editor.

To get started, we'll assume you have some reference data in RIS format handy. This could be your existing Reference Manager or EndNote database exported to RIS. However, for your first experiments the example file shipped with refdb is just fine. This file is usually installed as `/usr/local/share/refdb/examples/testrefs.ris` (ask your administrator if there is no such file).

At first you should have a look at the data, just to know what they look like. Use either a pager or a text editor to display the example file:

```
~$  less /usr/local/share/refdb/examples/testrefs.ris
     ❶
     TY  - BOOK                                    ❷
     ID  - smith1975metalloporphyrins              ❸
     T1  - Porphyrins and metalloporphyrins        ❹
     A1  - Smith,K.M.                              ❺
     Y1  - 1975///                                 ❻
     KW  - Porphyrins                              ❼
     KW  - Metalloporphyrins
     KW  - Spectrophotometry [methods]
     KW  - spectroscopy
     RP  - NOT IN FILE                             ❽
     CY  - Amsterdam                               ❾
     PB  - Elsevier Scientific Publishing Company  (10)
     ER  -                                         (11)
```

Ok, I've cheated a little. This reference is not the first and only one you'll see if you run that command, but it is the shortest one, good enough to discuss the general anatomy of a RIS dataset (in order to see this reference in the real example file, scroll just about halfway down).

We can easily discover these main features of the RIS format:

- A RIS file can contain one or more datasets.
- Each dataset starts with an empty line (more precisely, with a linefeed character). This also means that each file containing RIS datasets starts with an empty line.
- The tags start at the beginning of a line and consist of two uppercase characters, followed by a double space, a dash, and another space.
- The TY tag is always the first one, and the ER tag is always the last one in each dataset
- The sequence of the other tags is arbitrary with one exception: The sequence of the author fields (A1 or AU) determines the sequence of the authors in bibliographies or citations. The sequence should follow the sequence in the original publication.

Now let us look at the RIS dataset shown above. It contains the following tags:

❶ The mandatory empty first line

❷ The type tag is always the first one. The type of the reference is specified using one of about two dozen predefined types. This reference describes a complete book. Other common types are JOUR for a journal article and CHAP for a chapter in a book. See below for a full listing.

❸ This field contains an optional citation key. This is an easy-to-remember name for a reference, often constructed from the first author, the publication year, and maybe a word or two from the title. If you do not provide a citation key, refdb will create one for you.

❹ This is the title of the publication, in this case the title of the book. Other title tags are available. E.g. if you cite a chapter in a book which is part of a series, T2 and T3 can be used to code the book title and the series title, respectively, whereas T1 would be the chapter title.

❺ The name of the author, written in the order Last,First Middle,Suffix, or Last,F.M.,Suffix. If a publication has several authors, list each of them on a separate line with the A1 tag.

❻ The publication date in the format YYYY/MM/DD/other info. You can leave out any information that is not available, but you must keep the slashes regardless.

❼ A keyword that allows to retrieve the reference by topic. You can specify as many keywords as you like by putting each one on a separate KW line.

❽ This denotes the reprint status. This field can hold only the values "NOT IN FILE", "ON REQUEST", or "IN FILE".

❾ The publication place.

(10) The name of the publishing company.

(11) This is the mandatory last tag of each reference. Please be aware that this tag consists of "ER", two spaces, a dash, and another space, just like all other tags.

The following list gives an overview over all available tags and their use.

TY

This tag specifies the type of the reference and must be the first tag of each RIS dataset, preceeded by a newline.

Format: This can be any of the following strings:

- ABST (abstract reference)
- ADVS (audiovisual material)
- ART (art work)
- BILL (bill/resolution)
- BOOK (whole book reference)
- CASE (case)
- CHAP (book chapter reference)
- COMP (computer program)
- CONF (conference proceeding)
- CTLG (catalog)

- DATA (data file)
- ELEC (electronic citation)
- GEN (generic)
- ICOMM (internet communication)
- INPR (in press reference)
- JFULL (journal - full)
- JOUR (journal reference)
- MAP (map)
- MGZN (magazine article)
- MPCT (motion picture)
- MUSIC (music score)
- NEWS (newspaper)
- PAMP (pamphlet)
- PAT (patent)
- PCOMM (personal communication)
- RPRT (report)
- SER (serial - book, monograph)
- SLIDE (slide)
- SOUND (sound recording)
- STAT (statute)
- THES (thesis/dissertation)
- UNBILL (unenacted bill/resolution)
- UNPB (unpublished work reference)
- VIDEO (video recording)

ER

This empty tag denotes the end of the reference. It must be the last tag of each RIS dataset.

ID

This tag is used to uniquely identify the reference in the database. The value is either the unique ID that refdb generates when a reference is imported into a database, or a unique citation key. The latter can be supplied by the user. If no citation key is specified when adding a reference, refdb will automatically generate a unique citation key, based on the name of the first author and the publication year. refdb will create an unique ID value for internal use regardless of whether a citation key is provided or not.

> **Note:** ID values are always numerical (e.g. "11"), whereas citation keys are alphanumerical (e.g. "Miller1999").
>
> While you are free to choose any reasonable citation key (as long as it is unique within the database), you should not attempt to create a ID value manually. It is ignored when adding the dataset, but it may overwrite an existing entry if you update a reference. Along the same line, you should leave the ID tag alone if you retrieve a dataset from the database and plan to update it. The citation key in the retrieved data set is essential to match the modified data with the copy in the database.

ID Format: Integer >0.

Citation key Format: A string with up to 255 characters

TI

This is the title of a publication. For BOOK and UNPB references this is the same as the BT tag.

Format: A string with unlimited length.

T2

This is the secondary title of a publication, e.g. the book title for a CHAP reference.

Format: A string with unlimited length.

T3

This is the tertiary title of a publication, e.g. the series title for a CHAP reference.

Format: A string with unlimited length.

AU

Synonym: A1. This is the name of one author of the reference. If a reference has multiple authors, each author is specified with an AU tag on a separate line. The number of authors per RIS dataset is not limited. The sequence of the authors in the authorlist will be determined from the sequence as they appear in the RIS dataset.

Format: A string with up to 255 characters in the form: Lastname[,(F.|First)[(M.|Middle)[,Suffix]]]. First and middle names can either be abbreviated or spelled out. Some examples for valid entries:

- King,B.B.
- Benberg,Steven C.
- Mellencamp,John Cougar,Jr.
- Van Zandt,Steven

A2

Synonym: ED. This is the name of an editor of the reference, e.g. an editor of the book in which a CHAP reference was published. The same formatting requirements as for AU apply.

A3

This is the name of a series editor of the reference, e.g. an editor of a series of books in one of which a CHAP reference was published. The same formatting requirements as for AU apply.

PY

Synonym: Y1. This is the primary publication date.

Format: A string with the format “YYYY/MM/DD/otherinfo”, where YYYY denotes the four-digit year, MM and DD denote the two-digit month and day, respectively, and otherinfo denotes any other information with up to 255 characters. If any of these parts is not available, it can be left out, but the slashes must be present. E.g. “1999///Christmas edition” is a valid string.

Y2

This is the secondary publication date.

Format: A string with the format “YYYY/MM/DD/otherinfo”, where YYYY denotes the four-digit year, MM and DD denote the two-digit month and day, respectively, and otherinfo denotes any other information with up to 255 characters. If any of these parts is not available, it can be left out, but the slashes must be present. E.g. “1999///Christmas edition” is a valid string.

N1

The notes. This can be any form of additional information, like pointers to corrections or editorials, or just personal notes about the contents of the reference.

Format: A string with unlimited length

N2

Synonym: AB. The abstract of a reference.

Format: A string with unlimited length

KW

A keyword. If a publication has multiple keywords, each goes on a separate line preceeded with this tag. Keywords are crucial to find references in larger databases.

Format: A string with up to 255 characters

RP

The reprint status of a reference. This can be any of the following strings:

- IN FILE
- NOT IN FILE
- ON REQUEST MM/DD/YY

AV

The availability information. This is a hint where you can find an offprint or a file containing the reference.

Format: A string with up to 255 characters. This can either be a plain-text description like "methods folder, second drawer from top in the green cabinet on the yellow hallway", or an URL pointing to a file. In the latter case, this field has to start with the string "PATH:" with no space between this and the path proper. Using this feature requires some thought and is therefore explained in a separate section.

SP

The start page of the reference

Format: A string with up to 255 characters

EP

The end page of the reference

Format: A string with up to 255 characters

JO

The abbreviated name of a journal.

Format: A string with up to 255 characters. The journal words should be separated by a single space without a period after abbreviated words. If you use periods, these should not be followed by spaces.

JF

The full name of a journal.

Format: A string with up to 255 characters

J1

The abbreviated name of a journal (user abbreviation 1).

Format: A string with up to 255 characters

J2

The abbreviated name of a journal (user abbreviation 2).

Format: A string with up to 255 characters

VL

The volume of the journal.

Format: A string with up to 255 characters

IS

The issue of the journal

Format: A string with up to 255 characters

CY

City of publication of a book.

Format: A string with up to 255 characters

PB

Name of the publisher or the publishing company.

Format: A string with up to 255 characters

SN

The ISBN or ISSN number.

Format: A string with up to 255 characters

AD

The contact address, usually the any combination of postal or email address and the phone or fax number of the corresponding author.

Format: A string of unlimited length

UR

The URL of an online version of the reference.

Format: A string with up to 255 characters

U1 through U5

The user-defined fields 1 through 5. These fields are not intended to be filled with random bits of information. Each database should have a set of rules what information is to be stored in these fields.

A possible use for these fields is some relevance indicator (e.g. “*” means low, “*****” means high relevance).

You may also use one of these fields to create the equivalents of “folders” that some other reference databases praise as the panacea to organize your references. Just assign the same value to one of these fields for all references that belong to the same folder. Retrieve them by specifying this value in addition to your other search criteria.

Format: A string with up to 255 characters

M1 through M3

The miscellaneous fields 1 through 3. The distinction between Ux and Mx fields is somewhat unclear, and maybe only the inventors of the RIS format have a vague idea why there are two different types of fields for user-defined information.

Format: A string with up to 255 characters

Creating a RIS dataset from scratch

As noted previously, you can use any text editor that creates Unix-style line endings (linefeed) to create and edit RIS files. refdb ships a ris-mode for Emacs which makes the task a little more pleasant. Ask your administrator whether this mode is available on your system.

Creating a dataset is basically monkey business. The only intelligence required is to use the correct tags for your chunks of bibliographic information. Use the example RIS file as guidelines for the most common reference types journal article, book chapter, and book. We’ll look at a few issues related to these references first.

The first issue is certainly the type of the reference. There are pretty clear cases if you look at the list, but there’s some bordercases too which are not covered by the predefined types. You should keep in mind that refdb does not restrict the fields available for a particular reference type. You can fill any available field for any reference type. The only restriction is that the bibliography styles for most reference types use only a subset of the available fields. E.g. a bibliography entry of a journal article will not show a series editor even if you filled in the A3 field, whereas the bibliography entry of a book chapter might show the series editor if the book the chapter is published in is a part of a series.

The general rule is to use the closest matching type, to be consistent in this decision (all similar bordercases should use the same type), and to use the GEN type if nothing else helps. Most bibliography styles display all available fields for the GEN type.

To people new to bibliographic software, the various levels of titles and authors is often confusing. RIS offers three levels of authors:

AU (synonym: A1)

: The author of a publication. This is the person (or the persons) responsible for the smallest unit of the publication you're looking at.

ED (synonym: A2)

: The editor of a collection of publications.

A3

: The editor of a series of collections of publications.

Lets consider a few examples. If your reference contains a journal article, published in some scientific journal, the AU fields contain the names of those who wrote the article. The same holds true for the authors of a chapter published in a book like "Methods in Enzymology". The chapter authors would be in the AU fields, the volume editors in the ED field. The editors of the whole "Methods in Enzymology" series of books would be entered in A3 fields. However, if your reference points to one particular volume of "Methods in Enzymology" as a whole, you'd rather put the volume editors in the AU fields. The same logic holds true for the title fields:

TI (synonym: T1)

: The title of the smallest unit of publication you're looking at.

A2

: The title of the collection of publications

A3

: The title of the series of collections of publications

Using our previous example, an article published in "Methods in Enzymology" might have a TI field "An apparatus to turn urine into gold". The T2 field would be the title of the volume, e.g. "Alchemy and related techniques", whereas the T3 field would contain "Methods in Enzymology". However, if your reference points to the "Alchemy and related techniques" volume as a whole, this title would go into the T1 field.

Retrieving datasets from PubMed

The primary source of reference data in the biomedical field is the PubMed database maintained by the National Center for Biotechnology Information (http://www.ncbi.nlm.nih.gov). This section shows the simplest and most common way to retrieve bibliographic information about interesting articles from this database using a web browser (other methods use web service clients or email subscription services, but this is beyond the scope of this tutorial).

After visiting the site with your favourite web browser, select "PubMed" from the drop-down box called "Search" and type a query in the provided field. Something like "Doe J 2002" to find articles published by J. Doe in 2002. After hitting "Enter" you'll receive a list of publications matching your query. Select the ones you're interested in by clicking the check box right next to the publication (convenience beating logic, you can also check *none* of the boxes in order to retrieve *all* publications). Select "XML" from the drop-down box next to the Display button and hit the latter. You'll receive the list of the publications in the Pubmed XML format. Now click the Save button on that page and save the information to a plain-text file, e.g. `pm001.xml`. You could then return to the search, run a few more queries, and save your results in additional files according to this pattern.

We'll use the Perl script **med2ris.pl** to turn our XML data into RIS data. This tool either reads Pubmed data from standard input or from files specified as arguments. The result will be sent to standard output, so you can either view it with a pager or write it to a file.

```
~$ med2ris.pl < pm001.xml | less
```

This command converts the data in the file `pm001.xml` and displays the result in a pager.

```
~$ med2ris.pl pm*.xml > pm.ris
```

This command converts all files that match the pattern `pm*.xml`, like `pm001.xml` or `pmnew.xml`, and writes the resulting RIS datasets to `pm.ris`.

Now you should add your personal information to each dataset, as outlined above. Then you could go ahead and add the references to your default database with refdbc:

```
refdbc: addref pm.ris
```

Importing BibTeX datasets

If you have a BibTeX database that you want to import into refdb, you'll have to convert these data to RIS first. This is again best done by using one of the converters shipped with refdb. The tool **bib2ris** will by default convert all standard BibTeX fields to the RIS equivalents. If you used non-standard fields in your BibTeX database, **bib2ris** can be configured to import these too, but this requires additional entries in the `~/.bib2risrc` configuration file. The manual has all the details, but for the purposes of this tutorial we'll assume that you only use an "ABSTRACT" field as the only additional non-standard field.

Just like refdbc, bib2ris reads configuration files at startup which you can modify to permanently set some defaults. The syntax of the configuration file is the same as outlined above, but the only line we have to enter at this time is the following:

```
nsf_abstract N2
```

This will tell bib2ris to import your non-standard abstract field (this is case-insensitive, so your BibTeX file might use ABSTRACT or Abstract as well) into the N2 RIS field. Use the following command to see the results:

```
~$ bib2ris < myrefs.bib | less
```

This command will convert the contents of `myrefs.bib` in the current directory and display the result in a pager.

```
~$ bib2ris *.bib > myrefs.ris
```

This command reads the data from all `*.bib` files in the current directory and redirects the output into the file `myrefs.ris`.

Now there is a little issue with the data generated by bib2ris: they still contain all TeX markup that you may have used in your input data. If you want to use refdb only to maintain references for LaTeX files, this is probably ok, but if you want to use the data for SGML and XML documents too, it is necessary to strip the TeX markup before adding the references to the database. To this end, run the RIS file through the Perl script **tex2mail** shipped with refdb.

```
~$ tex2mail -noindent -ragged -linelength 65535 -ris < myrefs.ris > myrefs-notex.ris
```

As described in the previous section, you should now add your personal information and then use refdbc to add the datasets to your database.

Retrieving datasets from a Z39.50 server

Many libraries allow remote access to their electronic catalogs via the Z39.50 protocol. With a suitable client you can search the catalogs and retrieve the bibliographic information of interesting references to your local computer. For this tutorial we'll use the free client provided in the YAZ toolkit (http://www.indexdata.dk/yaz/), although you could use any other client as well. One of the largest libraries accessible via the Z39.50 protocol is the Library of Congress (http://www.loc.gov). We'll try to find computer books written by some Mr. Knuth in this library.

The following command connects to the library using the host name "z3950.loc.gov", the port 7090, and the database name "Voyager". All this information is usually provided either online or in a printed pamphlet about electronic catalog access published by libraries offering Z39.50 services:

```
     ~$ yaz-client z3950.loc.gov:7090/Voyager
Sent initrequest.
Connection accepted by target.
ID     : 34
Name   : Voyager LMS - Z39.50 Server
Version: 1.13
Options: search present
Elapsed: 5.176489
Z>
```

Now you can go ahead and type a query. The full query syntax of Z39.50 is beyond the scope of this tutorial, but the following query retrieves entries with the authorname "knuth" and the topic "computer":

```
Z> f @and @attr 1=1003 knuth @attr 1=4 @attr 5=1 computer
Sent searchRequest.
Received SearchResponse.
Search was a success.
Number of hits: 28
records returned: 0
Elapsed: 5.187307
```

```
Z>
```

We've found 28 entries that match our search pattern. We could just go ahead and display some or all of them, but we'd like to write them to a file, so we let our client dump all retrieved references to the file `knuth.loc.usmarc` and make sure the data are retrieved as "usmarc" (again, the library should be able to inform you which formats are available). Other formats acceptable for refdb are "ukmarc" and "unimarc".

```
Z> set_marcdump knuth.loc.usmarc
Z> format usmarc
Z>
```

Now we retrieve all of the matching entries. The **show** command uses an argument like X+Y, where X is the record number where the retrieval should start and Y is the number of consecutive records to be retrieved. The data will be displayed on the screen and written to our file in the background.

```
Z> show 1+28
Sent presentRequest (1+28).
Records: 28
[VOYAGER]Record type: USmarc
[...real data left out for brevity...]
nextResultSetPosition = 29
Elapsed: 8.438264
Z>
```

Finally we can leave the client by typing:

```
Z> quit
~$
```

If you attempt to open the resulting MARC file with a text editor or display it with a pager, you'll notice a couple of strange characters. MARC is a binary data format which is not supposed to be readable as plain text. If you want to display the file in a human-readable form, use the tool **marcdump** (this is part of the MARC::Record perl module which is required for the **marc2ris.pl** converter shipped with refdb; if there is no marcdump on your system, ask your administrator):

```
~$ marcdump knuth.loc.usmarc | less
```

The structure of a MARC record is quite complex. It is divided into a leader, fields with three-digit names, indicators, and subfields. No need to understand it at this point, though.

refdb ships the Perl script **marc2ris.pl** which attempts to convert MARC datasets to the RIS format like this:

```
~$ marc2ris.pl knuth.loc.usmarc > knuth.loc.ris
```

This will convert the references we downloaded to corresponding references in RIS format which will be written to the file `knuth.loc.ris`. If you retrieved the data in a different format, use the `-t` command line option to specify the input file format: "marc21", which is equivalent to USMARC, "unimarc", or "ukmarc". Now you can proceed as described above and add the contents of this RIS file to your database.

How to create risx datasets

risx datasets basically carry the same information as RIS datasets, but they use an XML format instead of tagged lines.

What a risx dataset looks like

The main advantages of risx are:

- The XML format is a good target for transformation of other bibliographic data in SMGL or XML formats.
- XML can be edited using either a general-purpose editor or, even more conveniently, with any XML editor.
- XML datasets can be validated, i.e. checked for completeness and for an appropriate structure.

Now lets have a look what the same dataset we had in RIS format above would look like in risx:

```
<?xml version="1.0" encoding="UTF-8" ?>
<!DOCTYPE ris PUBLIC "-//Markus Hoenicka//DTD Ris V1.0.2//EN" "http://refdb.sourceforge.net
<ris>
  <entry type="BOOK" citekey="smith1975metalloporphyrins">
    <publication>
      <title>Porphyrins and metalloporphyrins</title>
      <author>
 <lastname>Smith</lastname>
 <firstname>K</firstname>
 <middlename>M</middlename>
      </author>
      <pubinfo>
 <pubdate>
   <date>
     <year>1975</year>
   </date>
 </pubdate>
 <city>Amsterdam</city>
 <publisher>Elsevier Scientific Publishing Company</publisher>
      </pubinfo>
    </publication>
    <libinfo user="markus">
      <reprint status="NOTINFILE" />
    </libinfo>
    <contents>
      <keyword>Porphyrins</keyword>
      <keyword>Metalloporphyrins</keyword>
      <keyword>Spectrophotometry [methods]</keyword>
      <keyword>spectroscopy</keyword>
    </contents>
  </entry>
</ris>
```

This is a complete file with just one reference (you could add more `entry` elements for additional references). As you can see, the entry is a lot more verbose compared to RIS due to the spelled-out start and end tags. However, modern XML editors compensate this verbosity with nifty feature like tab completion and automatic insertion of end tags.

You will notice three large blocks of data in this dataset:

- The `publication` element contains the bulk of the bibliographic data proper, like the title, the author(s), and the publication date. Simple entry types like the book we see here make do with one level of bibliographic information. Complex types need more than one level. A journal article needs the `part` element for the article proper and the `publication` element for the information related to the journal. A book published as a part of the series needs the `set` element for the series information in addition to the `publication` element.
- The `libinfo` element contains the information specific to one refdb user, like availability information and personal notes. As a matter of fact, a risx dataset can contain an unlimited number of `libinfo` elements, one per user of the system. See also the section about global and personal fields.
- The `contents` element holds the abstract (not shown here) and keywords.

For further information please visit the documentation of the risx DTD (http://refdb.sourceforge.net/risx/book1.html).

Writing risx datasets from scratch

Using your favourite XML editor, writing a risx dataset from scratch should not be exceedingly difficult. The editor should prompt you for required elements and attributes, and refuse to enter an invalid structure. See the example datasets shipped with refdb to get an idea what different entry types should look like.

Transforming SGML and XML bibliographic data to risx

This topic is somewhat beyond the scope of this introductory tutorial, but if you're familiar with SGML or XML transformations in general, this should not be too hard either. Each input bibliographic format will require a custom DSSSL (for SGML data) or XSLT (for XML data) stylesheet that transforms the data to risx.

Validating risx datasets

If you write risx datasets from scratch or develop your own stylesheets for SGML/XML transformations, it is strongly recommended to validate the results of your laborious efforts. refdbd uses a non-validating parser to map the risx data to the appropriate database columns. If your input data are invalid, the results might not be to your liking. Two tools come in handy to validate your input data:

onsgmls

This tool is part of the OpenJade package of SGML/XML tools. The following command can be used to validate a risx document:

```
~$ onsgmls -wxml -s /usr/local/share/refdb/declarations/xml.dcl risxrefs.xml
```

xmllint

This tool is part of the libxml2 package. Use it like this to validate a risx document:

```
~$ xmllint --valid  --noout risxrefs.xml
```

Global and personal fields

refdb differs from other reference management tools because a main goal of its design is to encourage people to share their references. However, you may have figured from the tag list above that some of these entries only make sense if they can be maintained by each user individually. This is precisely the approach used by refdb: The "hard" bibliographic data are global and identical for each user. The "soft" personal data, which are the only ones likely to change after the reference was added anyway, are maintained for each user individually. These personal fields are:

- The reprint status (RP)
- The availability field (AV)
- The notes field (N1)

Even if you use one of the import filters described below or if you use RIS files exported from other bibliographic software, you should take the time to fill these fields with useful values. If you don’t specify values, the AV and N1 field will be blank (this is ok), and the RP field will have the default value "NOT IN FILE".

Character encodings

One seemingly intimidating detail about reference data is the character encoding issue. At the lowest level, a computer doesn’t understand anything but two states of a bit: off and on, usually represented as 0 (zero) and 1 (one). Concatenating several bits still doesn’t make a text, but a series of binary numbers at best. This is why even text strings are represented as numbers in a computer’s memory. Simply put, a character encoding is a lookup table that tells a computer which character to print if it encounters a particular binary number in a byte sequence that represents a text. The well-known ASCII encoding is understood by most computers but covers only 127 characters. Other encodings like Latin-1 contain all ASCII characters plus many special characters used in European languages. Still other encodings are far more versatile in that they allow to encode all characters used by recent and extinct languages. These are the various forms of the Unicode character encodings.

Although many encodings are known by several names, each character encoding has a preferred name which is usually identical with the MIME encoding name (the one you sometimes see in the header of emails). The names are case-insensitive, but otherwise the spelling must match precisely. E.g. UTF-16 and utf-16 are both recognized, whereas utf16 is incorrect.

Now where do these character encodings come into play? First of all, your reference database uses one particular character encoding, set by your administrator. All data that you add to this database will be converted to that encoding, and all data that you retrieve from this database will have to be converted from that encoding if necessary. To find out what encoding your database uses, run this command in refdbc:

```
refdbc: whichdb
Current database: refs
Number of references: 34
Highest reference ID: 34
Number of notes: 0
Highest note ID: 0
Encoding: UTF-8
Database type: risx
Database server: pgsql
Created: 2004-02-07 20:39:02 UTC
Using refdb version: 0.9.4-pre6
Last modified: 2004-02-09 21:43:10 UTC
```

In this example, the database uses the UTF-8 encoding, one of the most versatile Unicode encodings. Now, how do you convert your input data to e.g. UTF-8? Fortunately you don't have to, at least in most cases, as refdb does this for you on the fly. It just needs to know what encoding your input data use. The two input data formats employ two different ways to specify the character encoding:

RIS

: RIS data do not have a built-in mechanism to record the character encoding. You will have to tell refdb explicitly. You do this by using the `-E encoding` option with the addref command. Allowed are all encodings that your operating system can deal with. The most common examples are UTF-8, ASCII, and ISO-8859-1 through -15 (the various character sets for European languages).

risx

: As shown in the risx example above, each file containing risx data should specify the encoding in the processing instructions (the very first line of an XML file). Allowed are only four encodings: UTF-8, UTF-16, ASCII, and ISO-8859-1.

After you've added your data to the database, you're not yet done with encodings. The same issue pops up when you retrieve datasets from the database. By default, refdb sends the data using the same encoding as the database uses. However, you can retrieve the data using any encoding that your platform supports.

How to add and update references

In most cases you have a new set of references and want to add them to your database. Assuming your RIS references are stored in a file `newrefs.ris` in the present working directory, all you need to do is:

```
refdbc: addref newrefs.ris
```

To simply add the references in the example RIS file, use this command:

```
refdbc: addref /usr/local/share/refdb/examples/testrefs.ris
```

We have not used the `-E` option to select an encoding here, as the example data uses the ISO-8859-1 encoding which we have set as the default in our config file. If the input data were encoded in UTF-8, we'd do the following instead:

```
refdbc: addref -E UTF-8 /usr/local/share/refdb/examples/testrefs.ris
```

If you have references in the risx format instead, you'll have to tell refdb so. Do this with the `-t risx` switch:

```
refdbc: addref -t risx newrefs.xml
```

To add the example risx data, use this command:

```
refdbc: addref -t risx /usr/local/share/refdb/examples/testrefs.xml
```

Remember that risx data carry their encoding information in the processing instructions, so there is no need for using the `-E` option.

refdbc will try to add the references stored in these files to your default database. The diagnostic messages will be displayed in your pager, so if you send a dozen or so references it might take a few seconds until the results are displayed. You'll see a message for each reference found in the input file: Which ID was assigned, which citation key was used (or generated if you didn't specify one), and whether the operation was successful. Adding references usually fails only for two reasons:

- The input data were not the type of data that refdb expected. Either the data were simply corrupt, or there was a mismatch between the data type and the value of the `-t` option.
- A citation key was specified in the reference but the same citation key already exists in the database. refdb will refuse to accept the reference because citation keys *must* be unique. You can fix this by changing or deleting the offending citation key in the input file.

Once a reference is added to the database, you might still feel the urge to change it. Be it that you would like to add further keywords or that your personal information, like the reprint status, have changed since. The most straightforward way to change a reference is to retrieve it in either RIS or risx format, save it to a file, edit it, and send the updated copy back to where it came from. The following sequence of commands shows this, assuming we want to use the RIS format:

> **Note:** In this example we'll assume that you already know the ID of the reference that you want to change. You'll learn later how to find a reference by all sorts of criteria like authors, keywords and the like. This section also has additional information on the **getref** command used here.

```
refdbc: getref -t ris -o editme.ris :ID:=7
2595 byte written to /usr/home/markus/refdb/editme.ris
refdbc:
```

Now you can use your favourite text editor to change the file `editme.ris` as you see fit. For this exercise we'll just change the reprint status (the RP field) from "NOT IN FILE" to "IN FILE". When you're done, save the file and go back to refdbc:

```
refdbc: updateref -P editme.ris
Updating input set 1 successful
1 dataset(s) updated, 0 added, 0 failed
1 dataset(s) sent.
```

Now what was that `-P` switch good for? This switch tells refdb that it should only update your personal information of this reference, i.e. the RP, AV, and N1 fields. This is a lot faster than updating the whole reference. It is also more secure as you might have changed the file somewhere else accidentally without noticing. On the other hand, if you e.g. correct a typo you noticed in the title (TI) field, you *must not* use the `-P` switch.

Finding references

The easiest way to get started with a refdb database is to see what's in there. Running queries does not alter the database, so it is safe to play around. We're assuming now that your default database already contains a few references. If this is not the case, you'll have to read about adding references first.

Once you've got used to using a reference management software, you'll find out that retrieving references is one of the most common tasks. refdbc offers the command **getref** command for this purpose.

Before we look at the syntax of this command, let's discuss what is important about the data you retrieve:

- Which references to retrieve. In most cases, you are interested in particular references, not in a complete listing. Therefore you'll specify search criteria that will narrow down the list of matching references.
- Which output format to use. References can be retrieved in a variety of formats, and which one you choose depends on what you'd like to do with them.
- Where to put them. You can either display the results on the screen, or send the references to a file for later processing. Advanced users can even pipe the results to a shell command for further processing.

> **Note:** This is why you won't find commands like "Export" or "Print". Instead you can write any output format to a file in order to export it or pipe any output format to **lpr** in order to print it, respectively.

Let's now look at the syntax of the command. This is admittedly the most complex refdbc command you'll have to deal with:

```
refdbc: getref -h
Displays the result of a database search.
Syntax: getref  [-c command] [-d database] [-E encoding] [-h] [-o outfile] [-O outfile][-
P] [-R pdfroot] [-s format] [-S tag] [-t output-format] {search-string|-f infile}
Search-string: {:XY:{<|=|~|!=|!~|>}{string|regexp}} [AND|OR|AND NOT] [...]
where XY specifies the field to search in
Options: -c command   pipe the output through command
         -d database  specify the database to work with
```

```
          -E encoding  set the output character encoding
          -h           prints this mini-help
          -o outfile   save the output in outfile (overwrite)
          -O outfile   append the output to outfile
          -P           limit search to personal interest list
          -R           use pdfroot as root for path of pdf files
          -s format    specify fields for screen or style for DocBook output
          -S tag       sort output by tag ID (default) or PY
          -t output-format display as format scrn, html, xhtml, db31, db31x, teix, ris, risx
          -f infile    use the saved search line in file infile
          All other arguments are interpreted as the search string.
refdbc:
```

Now let us dissect this syntax into manageable chunks of information. We'll look at the bits and pieces of this command along the lines of the list above.

Which references do you want to retrieve?

refdbc uses a query language that is fairly simple yet powerful enough for more advanced tasks. Let's start with a very simple example:

```
refdbc: getref :ID:=7
ID*:7 (2002)                ❶
Key: Bellamy2002            ❷
Bellamy,T.C., Garthwaite,J. ❸
Pharmacology of the nitric oxide receptor, soluble guanylyl cyclase,
in cerebellar cells         ❹
Br.J.Pharmacol. 136(1):95-103 ❺
```

The only argument we passed to the **getref** command is the string ":ID:=7" (please note that there are no spaces on either side of the comparison operator). That is, we asked for the reference that has the numeric identifier "7" (all references have an unique identifier like this; it is automatically created when the reference is added to the database). Every refdbc command that accepts command line options uses some defaults if they're not specified, so in this case refdb sent back the reference in the "scrn" format and displayed the results on the screen using your favourite pager (more about the available output formats will be presented shortly). By default, the "scrn" format shows the following information:

❶ This is the ID of the reference, followed by the publication year in parentheses. The asterisk "*" tells you that the reference is part of your personal reference list, an advanced feature discussed below.

❷ This is the unique citation key of the reference. The citation key can be supplied by the user when a reference is added, otherwise refdb creates one for you. The purpose of the citation key is to provide a "handle" for a reference with an easy-to-remember name. Citation keys are preferred to IDs when writing citations in documents.

❸ This line lists all authors of the reference.

❹ This is the title of the reference.

❺ This line contains the publication information. As this is a citation of a journal article, it shows the name of the journal as well as the volume, issue, and page information of the particular article.

You'll learn in the next section how to tell **getref** to display more or all available information about the retrieved references.

Available fields to query

Retrieving references by their ID value is not very common with one exception: If for some reason you want to retrieve *all* references, you can use the command:

```
refdbc: getref :ID:>0
```

As all references have an ID value greater than 0, this command catches them all. But beware, this could literally be thousands of references, so don't keep your server busy just for fun.

In most cases you have some idea about the reference you're looking for. Either you know the name of an author, or when the document was published. Maybe you know a word or a phrase from the title, or you want to use keywords to look up references about a particular topic. The **getref** query language allows you to use all data fields in any combination to retrieve references. Let's first have a look at what fields are available besides the ID field we saw above.

:TY:

Type of the reference.

:ID:

The unique identifier of a reference. This is the numeric identifier that refdb assigns to each new reference.

:CK:

The unique citation key of a reference. This is the alphanumeric string that was either supplied by the user or automatically generated by refdb.

:TI:, :T2:, :T3:

The title of the reference, of the secondary title, and of the series title, respectively.

:AU:, :A2:, :A3:

The name of an author, of a secondary author/editor, and of a series author, respectively.

:PY:, :Y2:

The publication date and the secondary date, respectively (numeric).

:N1:

The notes that user can add to the reference.

:KW:

A keyword.

:RP:

The reprint status of the reference.

:AV:

The location of an offprint (physical, URL, or path)

:SP:

The start page.

:EP:

The end page.

:JO:, :JF:, :J1:, :J2:

The abbreviated name, the full name, the user abbreviation 1, and the user abbreviation 2 of a journal name, respectively.

:VL:

The volume number.

:ED:

The name of an editor.

:IS:

The issue (article) or chapter (book part) number.

:CY:

City of publication of a book.

:PB:

Name of the publishing company.

:U1: through :U5:

The user-defined fields 1 through 5. The intended contents of these fields are site-specific. Ask your administrator.

:N2:

The abstract of the reference.

:SN:

The ISSN or ISBN number.

:M1: through :M3:

The Miscellaneous fields 1 through 3. The intended contents of these fields are site-specific. Ask your administrator.

:AD:

The address of the contact person.

:UR:

The URL of a web page related to the reference.

In addition to the above field specifiers, there are a few that allow to retrieve references based on extended notes attached to them (see below for an explanation of extended notes):

:NID:

The ID of an extended note.

:NCK:

The alphanumeric key of an extended note.

Doing comparisons

We have to distinguish between numeric and alphanumeric fields, as indicated in the list above. This determines which comparison operators are available for that particular field. Alphanumeric fields can use the operators "=" (literal equality) and "!=" (literal non-equality) as well as "~" (regular expression equality) and "!~" (regular expression non-equality). Numeric fields can use the operators "=" (equality), "!=" (non-equality), "<" (less than), and ">" (greater than). The comparison of numeric fields should be straightforward, but the comparison of alphanumeric field needs a little further elucidation. Alphanumeric comparisons use either literal strings or *regular expressions* on the right-hand side of the operator. In the first case, the query matches if the whole string in the field indicated on the left is identical ("=") or not identical ("!=") to the string on the right. In the second case, the query matches if the regular expression finds a match somewhere in the text stored in the requested field. That is, the comparison ":N1:~'warts'" would return a result if the character sequence "warts" is somewhere in the text stored in the notes (N1) field. On the other hand, the comparison ":N1:='warts' would match only if a dataset contains a note that consists of the single word "warts".

Regular expressions are a complex but powerful feature. For the purposes of this tutorial we'll keep it simple and work only with these features:

- All regular characters like 'a', 'b', '3', and so forth, simply match themselves. That is, the regular expression "ab" will match the string "ab", but also "cab", "cabby", but not "ba" or "lumbar".
- The regular expression "." (period) matches any single character, and ".*" (period asterisk) matches zero or more occurrences of any character (the asterisk works with any other character than the period just as well). That is, "b.*b" matches "bb", "bob", "beeeebop"
- So the period and the asterisk are special characters within a regular expression. But what if you want to match those literally? Then you'll have to *escape* the special character with a backslash. So "bad\." will match "bad." but not "bad;". What if you want to match a backslash? Then you obviously use a double backslash.
- The '^' and '$' special characters serve as *anchors* for the regular expression. If you precede a regular expression with '^', the expression will match only at the beginning of the text. For example, the regular expression "^mule" will match the string "mules", but not "two mules". The '$' special

character does the same at the end of the line. The bright among the readers may have figured out already how to specify a full match: "^mule$" will match only the string "mule", but not "two mules" or "mules".

With this knowledge, we can try a few more simple queries (the query results are not shown in these examples for the sake of brevity):

```
refdbc: getref :PY:>2000
```

This will list all references that were published in 2001 or later.

```
refdbc: getref :AU:~^Miller
```

This will list all references with at least one author whose name starts with "Miller".

```
refdbc: getref :KW:~'guanyl.* cyclase'
```

This will list all references with at least one keyword that may be "guanylyl cyclase" or "guanylate cyclase" or something related. Note that we had to enclose the regular expression in quotation marks because the expression contains a space.

```
refdbc: getref :RP:="IN FILE"
```

This will list all references with a reprint status "IN FILE".The reprint status field may only contain one of the values "NOT IN FILE", "ON REQUEST", or "IN FILE". As we used a literal match instead of a regular expression match, the query does not return the datasets with the value "NOT IN FILE" although "IN FILE" is a part of the former string.

Combining and grouping fields

What if using just one field doesn't narrow down your search sufficiently? Obviously there must be a way to combine fields in a search. refdb understands the boolean operators "AND", "OR", and "AND NOT" for this purpose. If you want to retrieve articles published by Dr. Miller in 1999, you'd use a query like:

```
refdbc: getref :AU:~"^Miller" AND :PY:=1999
```

You can join as many fields with boolean operators as you see fit. If you run queries involving several fields it might be necessary to group parts of your query using parentheses. Consider the following query:

```
refdbc: getref :AU:~"^Miller" AND :PY:=1999 OR :PY:=2000
```

You wanted to retrieve all papers by Dr. Miller published in either 1999 or 2000, but this query would return tons of papers published by other authors as well. How come? The reason is that the queries are evaluated from left to right *unless parentheses change this order*. In the example above, refdb would put all references with an author starting with "Miller" in a bag. Then it would take all of Dr. Miller's references out of the bag that are not published in 1999. Finally it would add all references published in 2000 by whichever author to the bag.

This is clearly different from what you had in mind. Now look at this query:

```
refdbc: getref :AU:~"^Miller" AND (:PY:=1999 OR :PY:=2000)
```

This would again put all of Dr. Miller's publications into a big bag, but this time refdb would go ahead and remove all of Dr. Miller's publications not dated 1999 or 2000 from the bag. Now you win.

Which format do you want to use?

refdb can output references in a variety of formats. To request a specific output format, use the `-t` option of the **getref** command. The following table gives a short overview over the available formats. The name is what you have to specify with the `-t` option. The specifics of these formats will be described below.

Table 2-1. refdb reference output formats

Name	File format	Purpose
scrn	plain text	display of search results in a terminal window
html	HTML 4.01	display of search results in a web browser
xhtml	XHTML 1.0	display of search results in an XML-aware web browser
ris	RIS	editing references, export of references to other reference management programs, backup of databases
risx	risx	editing references, export of references to XML-aware programs, backup of databases
db31	DocBook	manually creating bibliographies for DocBook SGML documents
db31x	DocBook	manually creating bibliographies for DocBook XML documents
teix	TEI P4	manually creating bibliographies for TEI XML documents
bibtex	BibTeX	export of references to a BibTeX file

scrn

The screen backend provides a basic data output for viewing in a terminal, preferably through a pager. By default, the reference ID, the publication year, the authors, the title, and the source information are displayed. You can use the `-s` option to additionally display the abstract (AB or N2), the notes (N1), the reprint info (RP), the address (AD), the publisher (PB), the city (CY), the URL (UR), and the user (U1 through U5) and misc (M1 through M3) fields. You can concatenate these identifiers, e.g. `-s N1N2` would additionally display the notes and the abstract. `-s ALL` will display all available fields.

html

The html backend works just like the scrn backend, but encodes this information in a HTML text. This comes in handy if you would like to view the results of your queries in a web browser rather than in a terminal window. You simply use the `-o` switch to write the results of your queries to a file (see below), reusing the same filename for each query. After each query you just have to hit the reload button of your browser to view the results of the most recent query.

ris

This is identical to the input RIS format. Use it to edit references, to export them to other reference management systems, or to make RIS backups of your databases.

risx

The same holds true for this format which is the XML equivalent of RIS. The advantage of this format over RIS in terms of backing up your database is that it holds the personal information of all users of the database, instead of the information of only the current user.

bibtex

This backend provides output formatted for use as a bibtex reference database. This can be used with the tex and bibtex applications to create bibliographies for documents written with Donald Knuth's famous TeX (http://www.tug.org) typesetting system. The `-s` option cannot be used with this backend and will be ignored.

db31

The DocBook SGML backend formats the query result as a `bibliography` element in a SGML document using the DocBook DTD. refdb outputs an appropriate doctype string at the beginning of the data. The string is commented out so the contents can be directly inserted into a larger document by some processing application. If you need the data as a standalone document, simply uncomment the first line. The `-s` option cannot be used with this backend and will be ignored. The name "db31" means that the output will work with DocBook SGML 3.1 or later.

db31x

The output is essentially the same as with the preceeding backend but you'll get a DocBook XML document instead, compatible with all released versions of the DocBook XML DTD (it is sort of a misnomer because there was no XML version of DocBook 3.1).

teix

The TEI XML backend formats the query results as a TEI `listBibl` element according to TEI P4. refdb outputs an appropriate processing instruction and doctype string at the beginning of the data. The string

is commented out so the contents can be directly inserted into a larger document by some processing application. If you need the data as a standalone document, simply uncomment the first line. The `-s` option cannot be used with this backend and will be ignored.

Where do you want to put the results?

By default, all **getref** output will be sent to your favourite pager unless you request a different target. The pager is fine as long as you try to find references, but when you want to edit a reference or reuse references in a markup document, you have to save the search results in a file. To this end, use one of the `-o` and `-O` options along with the filename where the data should end up. The lowercase `-o` option will either create the file or overwrite an existing file with the same name. The uppercase `-O` option will either create the file or append to an existing file with the same name. For example, the following command will write the query results to a file in HTML format. If the file already exists, e.g. from a previous query, it will be overwritten.

```
refdbc: getref -t html -o refs.html :AU:~^Miller
```

The `-c` option pipes the data into a shell command. This shell command can basically be anything that is able to read from the standard input. You could even specify a different pager with this option, although this is arguably not too exciting. However, there are more interesting commands as shown in the following examples:

```
refdbc: getref -c lpr :AU:~^Miller
```

This will output all references with authors whose names start with "Miller" on your printer. The plain text scrn format is used as we didn’t request any particular format.

To find out how many words and characters the abstract of a particular reference contains, try the following:

```
refdbc: getref -t ris -c "grep ’^N2  - ’ | wc" :ID:=7
       1     255    1696
```

The output of reference 7 in RIS format will be piped to the command specified with the `-c` option. This happens to be another pipe. The refdb output will first be piped into **grep**, which isolates the abstract field. This line is then piped into **wc** which in turn outputs the line, word, and character count.

> **Note:** This example assumes that all of the abstract field is in one line. If this is not the case, you’d need a more complex command. This is left to your ingenuity.

Along the same lines we could also make a listing of the frequency of all words in the abstract, regardless of whether this really provides usable hints for categorizing the reference:

```
refdbc: getref -t ris -c "grep ’^N2  - ’ | tr ’ ’ ’\n’ | sort | uniq -c | less" :ID:=7
```

As above, the **grep** command isolates the abstract field. This time the field is piped into **tr** which we ask to replace every space with a newline, so the resulting data will have one word per line. We then use **sort**

and **uniq** to sort the list alphabetically and to remove duplicates, respectively. The result will then be displayed in the pager **less**. Phew. This is admittedly rather an exercise in self-indulgence, but by now you should have gathered that you can use arbitrarily complex shell commands as the argument to the `-c` option.

I can’t resist to present a final example which is a variation of the old "eat your own dogfood" theme. refdbc can also run in a batch mode (without entering the interactive console mode) and receive data at standard input for certain commands. We can use this feature to update references on the fly. Look at the following command:

```
refdbc: getref -t ris -c "sed ’s/ON REQUEST.*/IN FILE/’ | refdbc -C updateref -P" :ID:=7
```

getref retrieves the reference with the ID 7 in RIS format and pipes the result into **sed**. We have **sed** replace the string "ON REQUEST" (including the trailing date info, if present) with "IN FILE", which is common when you receive a long-awaited reprint of a paper which is already in your database. Then we pipe the modified reference into another instance of refdbc. The `-C` command line option causes this instance of refdbc to run the **updateref** command using the data received at standard input and exit when done.

> **Note:** To avoid undesired results it is recommended to do a dry run first and see whether the converted data look like they should. To this end, simply pipe the results of the **sed** command into **less**. If you like the results, hit the **up** key and replace **less** with the **refdbc** call as in the example above.

Adding extended notes

While it is possible for each user to keep a personal note along with each dataset in a database, this mechanism is not very flexible. Extended notes go far beyond these simple notes:

- You can attach several notes to a single reference. Of course several users can attach notes to the same references.
- You can attach each note to several references. This way you can create a note about a topic and link all relevant references to it.
- You’re not even restricted to references: you can attach notes to author/editor names, to journal names, and to keywords as well.
- The extended notes are accessible with a similar query language like the one you use to retrieve references.

How to create xnote datasets

Just like the risx reference data, extended notes use a particular XML format. It is best explained with a short example:

```
<?xml version="1.0" encoding="utf-8"?>
<!DOCTYPE xnoteset PUBLIC "-//Markus Hoenicka//DTD Xnote V1.1//EN" "http://refdb.sourceforg
1.1/xnote.dtd" [
<!ENTITY lt "&#38;#60;">
<!ENTITY gt "&#62;">
<!ENTITY amp "&#38;#38;">
]>
<xnoteset>
  <xnote key="notekey">
    <title>mynote</title>

    <content type="application/xhtml+xml" xml:lang="en-us"><para id='2' style='bold'>the no
    <keyword>biochemistry</keyword>
    <keyword>enzymes</keyword>
    <link type="reference" target="Phadke1994"/>
    <link type="author" target="Walsh,N."/>
    <link type="journalabbrev" target="Biochem.Pharmacol."/>
  </xnote>
</xnoteset>
```

We can easily recognize the following features:

- Just like with any other XML file, the processing instructions (the very first line in the file) may carry an `encoding` attribute. The value can be one of utf-8 (the default if none is specified), utf-16, ascii, and iso-8859-1.
- The `xnoteset` is a wrapper for one or more extended notes, each of which consists of one `xnote` element.
- Each `xnote` requires a `key` attribute. This is a mnemonic handle, like a citation key of a reference. If you export extended notes, you'll also see a `date` attribute as well as an `id` attribute which is managed by refdb. You should just leave it alone.
- The note itself is wrapped into a `content` element. This can hold either plain text, or markup as shown in the example above. The optional `type` attribute can be used to tell a processing application about the nature of the contents. Use the optional `xml:lang` attribute to specify the language of the contents.
- Just like references, extended notes may contain `keyword` elements to make notes searchable.
- The `link` elements connect a note to an unlimited number of references, keywords, authors/editors, or journals. If the link target does not exist in the database, the extended note is still added, but you'll get a warning.

How to add and update extended notes

You'll notice that managing extended notes is very similar to managing references. There are analogous commands to add and update extended notes. However, there is also one command specific to extended

notes that helps you maintain the links to other database objects. Let's first go ahead and add a file containing extended notes:

```
refdbc: addnote /usr/local/share/refdb/examples/xnoteset.xml
try to add set as note 1
Note key: markus2003
Adding input set 1 successful
try to add set as note 2
Note key: secondnote
Adding input set 2 successful
2 note(s) added, 0 skipped, 0 failed
```

If you want to edit an existing note, you've got two options, depending on what you want to change. If you want to change the contents proper, or maybe add or change keywords, use the following commands:

> **Note:** In this example we'll assume that you already know the ID of the note that you want to change. You'll learn later how to find a note by all sorts of criteria like title, keywords and the like. This section also has additional information on the **getnote** command used here.

```
refdbc: getnote -t xnote :NID:=1 -o editme.xml
543 byte written to /usr/home/markus/refdb/editme.xml
1 note(s) retrieved
```

This command will retrieve the note with the ID 1 in the xnote XML format and write the data to the file `editme.xml`. Now you can edit this file to your liking, and then send it back to the database like this:

```
refdbc: updatenote editme.xml
try to replace note 1
Note key: firstnote
Updating input set 1 successful
1 note(s) updated, 0 added, 0 skipped, 0 failed
```

If you want to add or remove links within an existing extended note, you may also find the following command helpful:

```
refdbc: addlink :NID:=1 :CK:=Miller1999
:NID: 1 -> :CK: Miller1999
1 link(s) added, 0 skipped, 0 failed
```

The first argument to this command must be either the ID of an extended note (":NID:") or the key (":NCK:"). The following arguments are interpreted as a list of objects that the note should be linked to. In this case, the citation key of a reference is specified. If you use the `-r` option with this command, the specified links will be removed.

Finding extended notes

You will easily guess that the command to retrieve extended notes is called **getnote**. The syntax is very similar to the **getref** command described above:

```
refdbc: getnote -h
Displays the result of a database search for notes.
Syntax: getnote  [-c command] [-d database] [-E encoding] [-h] [-o outfile] [-
O outfile][-P] [-R pdfroot] [-s format] [-S tag] [-t output-format] {search-string|-
f infile}
Search-string: {:XY:{<|=|~|!=|!~|>}{string|regexp}} [AND|OR|AND NOT] [...]
where XY specifies the field to search in
Options: -c command   pipe the output through command
         -d database  specify the database to work with
         -E encoding  specify the input character encoding
         -h           prints this mini-help
         -o outfile   save the output in outfile (overwrite)
         -O outfile   append the output to outfile
         -P           limit search to personal interest list
         -R           use pdfroot as root for path of pdf files
         -s format    specify fields for screen or style for DocBook output
         -S tag       sort output by tag ID (default) or PY
         -t output-format display as format scrn, html, xhtml, or xnote
         -f infile    use the saved search line in file infile
        All other arguments are interpreted as the search string.
```

As you can see, most of what we said about finding references also applies to extended notes. We'll only look at the differences here. The most obvious difference is the available fields to query for. For extended notes, the following fields are available:

:NID:

The unique id of an extended note.

:NCK:

The unique citation key of an extended note.

:NPY:

The date of an extended note.

:NTI:

The title of an extended note.

:NKW:

A keyword. This is a keyword attached to a note in order to categorize the latter, similar to a keyword in a reference.

:KW:

A keyword. This is a keyword that the note is linked to, i.e. a keyword that the note was attached to in order to supply additional information.

:AU:

The name of an author or editor. Use this field specifier to locate notes that are linked to a particular author.

:JF:, :JO:, :J1:, :J2:

The full, abbreviated, or user-abbreviated name of a periodical. Use this field specifier to locate notes that are linked to a periodical.

:ID:

The id of a reference. Use this field specifier to locate notes linked to a particular reference.

:CK:

The citation key of a reference. Use this field specifier to locate notes linked to a particular reference.

The other major difference is the available output formats.

Table 2-2. refdb notes output formats

Name	File format	Purpose
scrn	plain text	display of search results in a terminal window
html	HTML 4.01	display of search results in a web browser
xhtml	XHTML 1.0	display of search results in an XML-aware web browser
xnote	xnote	editing references, backup of databases

Some basic notes queries using these fields are shown here:

```
refdbc: getnote :NCK:=biochemistry1999
```

This query retrieves the note with the key "biochemistry1999".

```
refdbc: getnote :CK:=Miller1999
```

This query retrieves the note which is attached to the reference carrying the citation key "Miller1999".

Some advanced issues

So far you've heard only the very basic stuff. Some more advanced issues are discussed in this chapter. You'll learn how to manage your personal reference list, how to use the pdfroot configuration variable, and how to customize the HTML output of the **getref** command.

Using personal reference lists

If you share a common reference database with other users, you should know about the concept of personal reference lists. If you don't share your reference database with someone else, you can safely skip this section.

Each refdb database keeps a personal reference list for each user who adds references to this database. The reason you didn't notice so far is that it all happens automatically. If you add a reference with the citation key "miller1999asteroids" to the database, this reference will be tagged with your database username. That is, the database knows that the user "markus" added this particular dataset. However, by default refdb ignores this information when you run queries. This is by design, as it allows you to benefit from the references (and offprints) that others added. But if you want to, refdb allows you to restrict all database queries to only those references that you personally added. There are a few cases where this might be useful:

- You're about to move on in your nomadic life as a scientist. You want to take your collection of offprints with you, along with your part of the database. The personal reference list allows you to retrieve selectively those references in the common database that you added.
- If your queries return too many hits, you can use the personal reference list to cut down the number of results to those that you added.

All you need to do is to use the `-P` switch with the **getref**. Without that option, refdb will search the entire database. With the option, refdb will search only those entries that you added.

Of course refdb offers commands to manage your personal reference list. You can manually include existing references that someone else added to the database into your personal reference list. If you feel a reference is no longer of interest for you, you can remove it from the list.

> **Note:** Removing a reference from your personal reference list does not delete the reference data from the database. The reference will still show up if you search the entire database. In general this is preferable to deleting a reference because someone else might be interested in that reference. It is so much preferable that this little tutorial doesn't even tell you how to delete references.

The command to manage your personal reference list is called **pickref**. If you use the `-r` command line option, the specified references will be removed from your personal reference list. If you do not use this switch, the specified references will be added to your personal reference list. Let us assume someone else added a paper written by Dr. Miller in 1999 to the database that both of you share. Let's see what **getref** tells us about this entry:

```
refdbc: getref :AU:~Miller AND :PY:=1999
ID:5 (1999)
Key: Miller1999
Miller,J.D.
Recent advances in sleeping disorder research
Int.J.Sleep Res. 14(2):121-7
```

If you run the same command with the `-P` option, this reference would not show up in your result list. To add this reference to your personal interest list, run the command:

```
refdbc: pickref 5
ID 5 successfully picked
1 datasets added to personal interest list, 0 ignored
```

If you now re-run the previous **getref** command, you should see the following:

```
refdbc: getref :AU:~Miller AND :PY:=1999
ID*:5 (1999)
Key: Miller1999
Miller,J.D.
Recent advances in sleeping disorder research
Int.J.Sleep Res. 14(2):121-7
```

The little asterisk (*) now tells you that this reference is part of your personal reference list.

Using pdfroot and other abuses of the AV field

Like most other bibliographic tools, refdb can manage collections of electronic offprints. You can store a path to an electronic file along with the bibliographic information in the AV field. refdb can use this path to construct a hyperlink in both the HTML output and in the web interface, so your offprint is accessible with a single mouseclick.

While this sounds like a piece of cake, you should think twice. If you store a full path to the file, what happens if you access your database from a different computer? If you (or the IT guys of your department) keep the offprints on a web server and have to move the server to a different machine? Will you still be able to access your offprints? Keep in mind that changing the paths of several thousand references does not sound like much fun.

refdb uses a simple feature to take most of the hassle out of this. The URLs used in the HTML pages generated by refdb are made up of two components: A (potentially variable) root path and a (supposedly static) relative path. What you store along with the reference is only the relative path. The root path can be provided to refdbc as a command-line option or by a setting in its configuration file.

Let's look at an example. You keep your PDF and Postscript offprints in `~/references/` and subdirectories thereof, so the full path of one example file might be `/home/markus/references/surgery/miller1999cirrhosis.pdf`. Of course you could keep all files in one subdirectory, but this example shows that you're not limited to that. The AV field for this reference should then read:

```
AV  - PATH:surgery/miller1999cirrhosis.pdf
```

The appropriate setting for the root path for local access is `/home/markus/references/`, as simple concatenation of these paths would return the full path of the file in question (please note the trailing slash in the root path). The following entry in your `~/.refdbcrc` configuration file would set this as your default pdfroot:

```
pdfroot     /home/markus/references/
```

Now you're doing some interesting lab work. You're way too lazy to walk over to your office but you still need that offprint real quick. Just go ahead and mount your drive remotely (you might need root permissions to do that):

```
~$ mount dogear:/home /mnt
```

assuming that your box is named "dogear". Your references directory is now accessible as `/mnt/markus/references`. All you need to do is to start refdbc with the -R option set to this new pdfroot:

```
~$ refdbc -R /mnt/markus/references
```

For the current session you will be able to access your offprints with a mouseclick in the HTML query output although you use a different computer.

Along the same lines, if your department convinces you to donate all offprints to a central repository which will be made accessible through a web server, all you need to do is to change your `~/.refdbcrc` accordingly:

```
pdfroot     http://serverbox.labnet/offprints/markus/
```

assuming that the web server is accessible with the host name "serverbox.labnet" in your local network and that you get your own subdirectory, thus keeping your subdirectory structure intact.

Using cssurl

You've learned earlier in this tutorial that you can display refdbc query results in a web browser. The HTML that refdb creates is suitable for use with CSS stylesheets. By default, an example stylesheet (installed as `/usr/local/share/refdb/css/refdb.css`) is used for this purpose. If you don't like how the HTML output looks, you can hack your personal stylesheet and use that instead.

It is probably the easiest way to get a copy of the default stylesheet and edit that:

```
~$ cp /usr/local/share/refdb/css/refdb.css ~/
```

Then use your favourite text editor to change the visual appearance of the HTML pages. In order to test your new stylesheet you need to tell refdbc about it with the `-G` option:.

```
~$ refdbc -G /home/markus/refdb.css
```

Now you can retrieve some references in HTML format and save them to a file:

```
refdbc: getref -o test.html -t html :ID:<10
```

Now open the file `test.html` in your web browser and see whether you like it. You can change the stylesheet and redisplay the file until you like the results.

To make your new stylesheet the permanent default, add the following line to your `~/.refdbcrc`:

```
cssurl      /home/markus/refdb.css
```

If you have access to a web server in your local network, you could leave your stylesheet there and enter its URL as the cssurl value. This way you could enjoy the benefits of your stylesheet from any computer in your local network.

Chapter 3. Bibliographies

Now you've had some success lately with your research and would like to publish your results (or rather, you have to publish *something* in order to get the next grant). Your data are ready, all papers that you've read are in your refdb database, and you have the instructions for authors of your favourite journal. refdb can create formatted citations and bibliographies for several document type definitions:

- DocBook SGML (3.1 or later)
- DocBook XML (all published versions)
- TEI XML (P4 or later)
- LaTeX

We'll leave out LaTeX here as refdb merely acts as a substitute for the flat file database of the BibTeX system in this case; see the manual for details. Instead we'll focus on the two DocBook variants and show the processing steps for both the SGML/DSSSL and the XML/XSL combos. This tutorial assumes that you are familiar with the basics of either SGML or XML authoring and transformation. This tutorial also assumes that suitable bibliography styles are available on your refdb installation.

refdb ships with a couple of example documents. They are usually installed in `/usr/local/share/refdb/examples/` (ask your administrator if you can't find the files). If you want to follow the instructions using these test documents, please copy these files to a private folder so you can mess around as much as you like.

Writing documents

In order to use an existing DocBook or TEI document with refdb, you need just a few small changes in the document itself. You do *not* need any changes to either the DTD or the stylesheets for the subsequent transformation. The basic idea is outlined in the following three steps:

1. Declare an external entity in the local DTD subset that points to a file to be generated by refdb.
2. Include this external entity at a location where you wish the bibliography to appear
3. Whenever you cite a publication in your text, mark this citation element for use with refdb.

For the purpose of this tutorial we'll start from scratch and create a new file rather than modifying an existing one. refdb ships with a script that does the first step for you: It creates an empty file with the appropriate document type declaration and declares the external entity. There's no magic involved, though, so if you want to do this manually, feel free to do so. Now we'll first have a look at that script and at the files it creates.

Create a new document

If you want to start a new document for use with refdb from scratch, create a new subdirectory and **cd** to it:

```
~$ mkdir publications/refdbtest
```

```
~$ cd publications/refdbtest
```

Now run the **refdbnd** (like in New Document) script. This is a simple interactive shell script that will ask you some questions. Based on your answers it will then create a new skeleton document and a custom-tailored Makefile which makes the subsequent processing steps a breeze.

Let's have a look at an example session. The displayed help messages should be sufficient to guide you through the process:

```
~$ refdbnd
I'll be happy to assist you in setting up a new document along with a
Makefile. First we'll collect a few answers, and only if you accept
your settings any files will be created. Press Ctrl-C anytime to exit.

Each question will present a default value which you can accept by
pressing ENTER

Please enter the basename of your document. This is the name without
any suffix. For example, if your printed output file is supposed to
be called refdbtest.pdf, the basename will be 'refdbtest'
[refdbtest]


Please enter the type of the document. Available types are 'db31',
'db40', and 'db41' for DocBook SGML versions 3.1, 4.0, and 4.1,
respectively, 'db41x' for DocBook XML version 4.1.2, and 'teix' for
TEI XML P4
[db41]
db41x

Please enter the element root which determines the type of the
publication. Common are 'set', 'book', and 'article' for DocBook
documents and 'TEI.2' for TEI documents
[book]


Please enter the name of the RefDB database where you take your
references from
[refdbtest]


Please enter the bibliography style that your document should use
[J.Biol.Chem.]
Eur.J.Pharmacol.


You've selected the following values:
Basename:           refdbtest
Document type:      db41x
Publication type:   book
Database:           refdbtest
Bibliography style: Eur.J.Pharmacol.
```

```
Is this ok?
[y]

Fine, so then ...

Makefile created.

Document refdbtest.short.xml created.

After editing this file you can use the following commands to create
formatted output:
make pdf to create a Portable Document Format (PDF) file (refdbtest.pdf)
make html to create HTML output (book1.html)
make rtf to create a Rich Text Format (RTF) file (refdbtest.rtf)
make all to create all available output formats
```

The defaults and the messages might be slightly different if you select a different document type. As the session log tells you, the script creates these two files: the Makefile and your new document's source file. Why is the source file called `refdbtest.short.sgml`? The Makefile assumes that you will be using the short notation for citations, as explained in the next section. This warrants an additional processing step to create the file `refdbtest.sgml` which uses the full notation for citations. This document in turn determines the name of the output files.

The source file contains the document type declaration and the declaration of your bibliography as an external entity in the internal subset. Feel free to add any additional entity declarations that your document requires. The documents look like the following examples. The first one is a DocBook SGML document:

```
<!DOCTYPE BOOK PUBLIC "-//OASIS//DTD DocBook V4.1//EN" [
<!ENTITY bibliography SYSTEM "refdbtest.bib.sgml">
]>
```

The corresponding chunk of a DocBook XML file looks like this:

```
<?xml version="1.0"?>
<!DOCTYPE book PUBLIC "-//OASIS//DTD DocBook XML V4.1.2//EN"
                   "http://www.oasis-open.org/docbook/xml/4.1.2/docbookx.dtd" [
<!ENTITY bibliography SYSTEM "refdbtest.bib.xml">
]>
```

> **Note:** Your local tools might not support using URLs as system identifiers (or you loathe the inconvenience to fire up your modem connection each time you validate or transform the document). Then you'll have to use a local copy of the DTD instead and replace the URL with the full path of your local copy.

Include the external entity

refdb bibliographies are wrapped in `bibliography` and `div` elements for DocBook and TEI, respectively. Include the external entity at a location where such an element is valid, like in the following chunk:

```
...
  </chapter>
&bibliography;
</book>
```

Write your citations

refdb offers two ways to enter citations in your documents. The short notation which will be described shortly is simpler but requires a mandatory preprocessing step before the bibliography can be created and before the document can be transformed (you won't notice, though, as all this is handled by the `Makefile`). Working with multipart documents is also a little trickier. If you have good reasons to use the full notation, please consult the manual.

refdb citations in SGML and XML documents use the native elements for citations, which are `citation` and `seg` elements in DocBook and TEI, respectively. However, you need to set an attribute in order to "tag" these elements for use with refdb. This way refdb can easily distinguish between citations which it is supposed to process and citations which are not meant as refdb citations. Citations in both DocBook SGML and DocBook XML documents look like this:

```
<citation role="REFDB">2;5;9</citation>
<citation role="REFDB">miller1999;jones2001</citation>
```

The first example shows a citation of three publications whose IDs are listed in a semicolon-separated list. The second example shows that you can use the citation keys as well.

The corresponding syntax for TEI XML documents is quite similar, except that we abuse the general-purpose `seg` element and tag it for use with refdb by setting the `type` attribute to `REFDBCITATION` in all caps:

```
<seg type="REFDBCITATION" part="N" TEIform="seg">2;5;9</seg>
```

All of the above examples will create citations which are separate from the flow of your text. This is shown in the following example output:

```
The pharmacological effect of YC-1 was discovered in a rat model (Doe, 1993).
```

Occasionally you might prefer to include parts of the citation in the flow of the text, like in the following text:

```
Miller et al. suggested in a recent review (2000) that ...
```

You can achieve this effect by preceeding the ID or citation key with either "A:" or "Y:" for an author-only and a year-only citation, respectively:

```
<para><citation role="REFDB">A:miller2000</citation> suggested in a recent re-
view <citation role="REFDB">Y:miller2000</citation> that ...</para>
```

Processing your documents

Once you have finished writing your document (or at least a first draft), you want to create a printable or online version with the formatted citations and bibliographies. This section explains how to do this conveniently with the Makefiles generated by **refdbnd**.

Makefile targets

As mentioned previously, **refdbnd** creates a custom-tailored `Makefile` along with every new document. This Makefile offers the following main targets to process your document:

pdf

: This target generates a PDF file from your source document. PDF is a widely accepted document format with free viewers for essentially all current operating systems.

html

: This runs all required commands to create HTML output, viewable with any web browser. Depending on your local setup, the output will be chunked into a collection of HTML files.

rtf

: This target generates a Rich Text Format (RTF) file. This plain text format is sort of a word processor interchange format understood by most current word processors, including MS Word, WordPerfect, and OpenOffice/StarOffice.

ps

: This target is only available for SGML documents. It will create a Postscript document from your source. Postscript is the universal document format on Unix systems and can be printed directly on Postscript printers. Viewers are available for all current operating systems.

The Makefile also offers a few more targets. For each of the above targets there is a corresponding '<target>dist' target which creates a `.tar.gz` archive of the output document. The target 'all', which is also the default if you don't specify a target to **make**, builds all available output formats. Accordingly, the target 'dist' creates all archives. And finally, the target 'clean' removes all intermediate files and returns your directory to the original state.

Using the Makefile is simple. For example, to turn your document into a PDF file, just run:

```
~$ make pdf
```

The Makefile tests whether any of the intermediate files is outdated and re-creates them if required. For example, if you run **make pdfdist** after the previous command, **make** will only run the command to

create the archive; it will not go through the whole processing procedure for the PDF document again. However, if you edit the document source and then run **make pdfdist** again, it will first bring the PDF output document up to date and then create the archive.

Customizing your Makefile

There are a few cases when you might want to modify the refdbnd-generated `Makefile`. As a Makefile is but plain text you can use any text editor to customize it. The first case is quite common in the life cycle of a scientific publication: You've nicely formatted your document for journal 'A', but it was rejected and you have to submit it to journal 'B' (with any number of possible reiterations). In order to re-create the formatted document using a different citation and bibliography style, use the following procedure:

1. Remove all intermediate files:

   ```
   ~$ make clean
   ```

2. Modify your `Makefile`. Edit the line that sets the variable `style` and specify the new bibliography style.

3. Transform your document again. E.g. if you want to get a PDF version using the new bibliography style, just run:

   ```
   ~$ make pdf
   ```

The second good reason to customize your `Makefile` is to get additional files into your `.tar.gz` archives. If your document includes images or should be accompanied by a README file, you can let **make** include them into the archive as well. All you need to do is to extend the file list of the appropriate **tar** commands.

Chapter 4. Where to go from here?

If you use refdb for your daily work you'll probably get to a point where this tutorial can't answer your questions any longer. To pick up a theme from the preface, you should first consult the refdb manual (http://refdb.sourceforge.net/doc.html). This has a reference section for all clients and scripts used by refdb, installation and maintenance instructions, instructions how to author bibliography styles, and a complete description of the full citation notation which we skipped in this tutorial to keep things simple.

Sometimes it might help to chat with others who use refdb. There is a mailing list for refdb users devoted to this purpose. Visit the refdb-users list page (http://lists.sourceforge.net/lists/listinfo/refdb-users) to subscribe or to view the archives. To send mail to the list, use the address refdb-users@lists.sourceforge.net (mailto:refdb-users@lists.sourceforge.net).