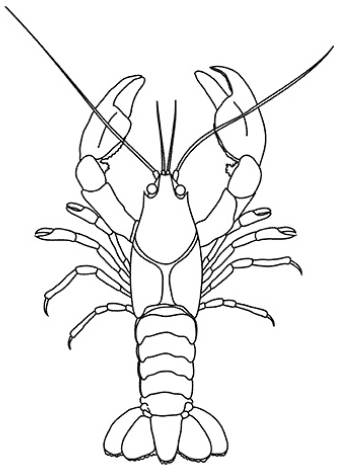

Yabby User Guide

YABBY version 0.1

Yabby is a lightweight bioinformatics workbench that provides a centralized framework for the management and reuse of Perl program scripts. The specific design goal of Yabby is to allow easy writing of extensions and new functionality.

Contents

Basics

1.1 Introduction

Many bioinformatics tasks are solved by writing of specialised programs, typically using scripting languages that allow rapid development and are supported by well developed libraries. When the level of generality is required, or when such programs manipulate large data files, testing and debugging may involve considerable effort and time. In spite of their inherent value, such programs typically end up scattered with project-specific data files, and are often lost after the completion of the project as a result of data, interest, or people migration.

Applications in bioinformatics involve a variety of tasks, for example sequence analysis, database searching, analysis of instrumentation data, and so on. General purpose scripting languages such as Perl, Python and others are particularly well suited for applications in bioinformatics. However, the advantage of scripting languages are often diminished by the *ad hoc* development process, where little thought given to long term retention and re-use of resulting program scripts.

Bioinformatics data structures are often represented as text. Because of its exceptional text processing capabilities, strong library support, and open source nature, the Perl programming language is very popular for applications in bioinformatics. Prominent examples of bioinformatics packages developed in Perl include BioPerl, a set of programming libraries with particular strengths in manipulation and analysis of sequence data, and MMTSB, a set of utilities and libraries for the analysis of structural, simulation, and prediction data. The availability of high quality libraries makes Perl even more attractive for applications in bioinformatics. However, inasmuch as the totality of Perl features aids in rapid development of custom program scripts, this also exacerbates the problem of proliferation of in-house program scripts, and therefore the problem of their management, long term retention, and re-use.

Yabby aims to support a long term retention of dissipate Perl program scripts developed for bioinformatics, and is also an environment for the execution of these scripts. Yabby consists of an interactive shell and a library of program scripts that provide a specific functionality. The shell provides the command line interface to the user, and a mechanism to execute library program scripts. Program scripts can have either stand-alone functionality, or they can be a part of a larger processing pipeline. In the latter case, they exchange the data through the common data model. For this purpose two different data formats are used, two-dimensional tables and XML. Yabby is modular and extensible, and allows users to modify existing functionality and to easily add new functionality.

Yabby is released as open source, under the GNU Public License version 2.

1.2 Installation

1.2.1 Checking Perl

Yabby has been developed with Perl version 5.X, a freely available general purpose scripting language. Perl stands for "Practical Extraction and Report Language" and is particularly well suited for the manipulation of data stored in plain text files.

Before attempting the installation, it is highly recommended to check if the Perl interpreter is present on your computer system. For example, on a Linux system,

```
$ perl -v

This is perl, v5.8.5 built for i386-linux-thread-multi

Copyright 1987-2004, Larry Wall

Perl may be copied only under the terms of either the Artistic License or the
GNU General Public License, which may be found in the Perl 5 source kit.

Complete documentation for Perl, including FAQ lists, should be found on
this system using 'man perl' or 'perldoc perl'.  If you have access to the
Internet, point your browser at http://www.perl.com/, the Perl Home Page.
```

Perl 5 and later is required for Yabby. The next step is to find where exactly is the Perl interpreter located, as this information will be required for Yabby installation:

```
$ which perl
/usr/local/bin/perl
```

1.2.2 Downloading Yabby

Yabby source code can be browsed from the Google Code servers, at the URL: http://code.google.com/p/yabby/. Under the section "Source" one can find the instructions for downloading the source code. The same page provides the link under "This project's Subversion repository can be viewed in your web browser" which allows one to browse the source code on the server without actually downloading it.

Google servers maintain the source code by the program called 'subversion' (an open-source version control system). To download the source code one needs to use the subversion client program called 'svn'. The

'svn' client exists for all mainstream operating systems[1], for more information see http://subversion.tigris.org/. The book about subversion is freely available on-line at http://svnbook.red-bean.com/. Subversion has extensive functionality however only the very basic functionality is needed to download Yabby. Assuming that the computer is connected to the internet, the following command will download the latest Yabby source code in the current directory:

```
$ svn checkout http://yabby.googlecode.com/svn/trunk/ yabby
A    yabby/yabby.pl
A    yabby/LICENSE
A    yabby/lib
A    yabby/lib/blast.pl
A    yabby/lib/hmm_score2seq.pl
A    yabby/lib/seq_strip.pl
A    yabby/lib/seq_comment.pl
A    yabby/lib/hmm_score_extract.pl
A    yabby/lib/blastg.pl
A    yabby/lib/motif_cmp.pl
A    yabby/lib/seq_unique.pl
A    yabby/lib/yabby_seq.pm
....further output deleted....
```

1.2.3 Linux installation

The installation of Yabby requires that the file 'yabby/lib/yabby.pl' is modified in order to set the path to the Perl language interpreter, and the path to Yabby libraries.

If Yabby code was downloaded in the directory /home/jake/ (and therefore the script 'yabby.pl' is located in /home/jake/yabby/), to set the path to Yabby libraries the following two lines need to be set:

```
use lib "/home/jake/yabby/lib";
$LIB_DIR = "/home/jake/yabby/lib";
```

The path to the Perl interpreter is set in the first line of the file 'yabby.pl':

```
#!/usr/bin/perl
```

The script yabby.pl needs to have executable permissions:

```
$ chmod +x yabby.pl
```

[1] For example, on Linux CentOS 4 the RPM package 'subversion-1.3.2-1.rhel4.i386.rpm' provides the subversion client 'svn'.

If this is all set, running the script 'yabby.pl' will start Yabby:

```
$ yabby.pl

 - YABBY version 0.1 -

 [ 35 command(s) ready ]

yabby>
```

It is often useful to create a symbolic link in a directory which is included the PATH variable, such as /usr/local/bin or /bin:

```
ln -s /home/jake/yabby/yabby.pl /home/jake/bin/yabby
```

This would allow Yabby to be run from any directory, simply by typing 'yabby'. When a particular Yabby command relies on an external program or library (such as NCBI BLAST), additional configuration may be needed to allow Yabby to find the appropriate files.

1.2.4 Windows Installation

Note that Yabby has been developed and tested on Linux, and should work equally well on any Unix system. There are two ways to install yabby on Windows:

1. Using Cygwin. Cygwin is a complete Unix environment on Windows, and Linux/Unix installation instructions apply. Yabby is expected to work with full functionality under Cygwin.

2. Using ActivePerl. Yabby can work in this environment, however the testing was limited and not all commands will work. A detailed installation instructions are given below.

Installing ActivePerl

The perl interpreter must be installed, we recommend to to install ActiveState's ActivePerl.

```
http://www.activestate.com/Products/activeperl/
```

At this site one can obtain a free copy of the ActivePerl installer, which will then need to executed to install the interpreter.

Installing a subversion client

Google servers maintain the Yabby source code by the program called 'subversion' (an open-source version control system). An easy-to-use interface to subversion for Windows is TortoiseSVN. It is an extension of the shell, so it can be (and must be) used straight from Windows Explorer. It is available for free download from here:

`http://tortoisesvn.net/downloads`

Run the .msi executable, and it will install. This will also require a reboot.

Downloading Yabby

The following instructions refer specifically to TortoiseSVN. It is recommended, but not necessary, that a new directory be created anywhere in the filesystem where the Yabby source code is to be located. For example:

C:\

1 create C:\. Then open this directory.

2 Right-click the empty space and select SVN Checkout.

3 The URL is http://yabby.googlecode.com/svn/trunk/. Click OK.

Yabby installation

Yabby installation requires that the file 'yabby.pl' is modified to set the path to Yabby libraries.

If Yabby code was downloaded in the root directory C:\, to set the path to Yabby libraries the following two lines need to be set (yabby.pl is stored in binary format, not text. So use Wordpad.exe, found in `Start Menu`⟶`All Programs`→`Accessories`):

```
use lib "C:\\Yabby\\lib";
$LIB_DIR = "C:\\Yabby\\lib";
```

NOTE: '\' is an escape character in Perl strings, which is also the character Windows uses to separate directories. A double escape character escapes the escape. Hence the need for two.

If you double-click yabby.pl now, it will start Yabby.

```
 - YABBY version 0.1 -
```

```
 [ 38 command(s) ready ]

yabby>
```

1.3 Running Yabby

1.3.1 An interactive session

Yabby can be run interactively or from the command script. To start an Yabby interactive session one needs to start the Yabby interface from the Unix shell. Here is the simplest Yabby session:

```
$ yabby

 - YABBY version 0.1 -

 [ 35 command(s) ready ]

yabby> quit

 bye-bye
```

1.3.2 Yabby command scripts

In order to run Yabby from the command script, the command file needs to be prepared first. Such a file lists Yabby commands one per line, with optional blank lines (lines which start with the % character are ignored). For example, the following input file, named test.yab,

```
% test.yab -- test input script

seq_load cad3.seq cad3
seq_info -l cad3
```

could be run with the Unix shell with input redirection:

```
$ yabby < test.yab

 - YABBY version 0.1 -

 [ 35 command(s) ready ]

yabby> % test.yab -- test input script
```

```
yabby> yabby>
 Reading the file 'cad3.seq' ..
 3 sequence(s) found.

yabby>
 'cad3' contains 3 sequence(s)
  1 -> Q53650_STAAU, 192 residues
  2 -> Q97PJ0_STRPN, 193 residues
  3 -> P95773_STALU, 192 residues

yabby> yabby>
 bye-bye
```

When run from the command script the actual commands are not echoed back, only the command's screen output as well as the comments are seen.

1.3.3 Executing Unix commands

Any command which is not recognized by Yabby is assumed to be an Unix command, and Yabby will attempt to execute it. Consider the following example:

```
yabby> ls
1BT0.pdb  cox1.seq     LmjFmockup.pep  needle.out
cad3.seq  dna.seq      m2.blocks       README
cad.seq   hmmpfam.out  meme.out        test.yab
yabby> l

 [ UNIX command 'l' failed ]
```

Because there is no Yabby command `ls`, it was assumed to be a system command and executed. The output was printed on the screen, listing the files and directories in the directory where Yabby was started.

Subsequently, the command `l` was given but failed because there is no such Yabby or Unix command. If `l` was an alias to something (say `ls -CF`) the command would fail regardless, because Yabby does not know about shell aliases.

There are no inherent limitations to which Unix commands can be executed within Yabby. It is possible to run a text editor, such as "vi" (and then simply resume the Yabby session after exiting the editor), or even start programs with GUI such as "gnuplot", or a Unix terminal window.

A subtle but important point is that Unix commands are not executed through the Unix shell. The consequence of this is that the (sometimes important) functions provided by the Unix shell, such as file globing, are not available. For example:

```
yabby> ls *
ls: *: No such file or directory

 [ UNIX command 'ls *' failed ]
```

A handy trick which allows one to go about Unix business is to temporarily suspend Yabby. This is actually a feature provided by some unix shells (in combination with the system's terminal driver), and has little to do with Yabby. In short, typing Ctrl-Z within Yabby will suspend the current Yabby session, and return user to the Unix shell. Issuing the command "fg" to the same shell will return the suspended Yabby session:

```
yabby> [Cntrl-Z]
[1]+  Stopped                 yabby
$ ls -CF
1BT0.pdb  cox1.seq     LmjFmockup.pep  needle.out
cad3.seq  dna.seq      m2.blocks       README
cad.seq   hmmpfam.out  meme.out        test.yab
$ fg

yabby>
```

Ctrl-Z was typed on the first line, which was not echoed back. Note that the second command prompt (starting with the $ character) is the Unix shell prompt, and typing "fg" had returned user to the suspended Yabby session.

1.4 Understanding how Yabby works

To understand how Yabby works it is important to understand the relationship between three directories: the working directory (where the Yabby session has been started), the Yabby library (this is the lib/ subdirectory in the Yabby installation directory), and the workspace directory.

The workspace directory is created automatically when the Yabby session is initialized. By default, the workspace directory is called .yabby, and it is created in the working directory. The workspace directory contains all the yabby objects created within the session. Upon a graceful exit from Yabby the workspace directory is destroyed together with its content.

1.4.1 The exchange of data between command scripts

Consider what happens when the sequences were read from the file 'tom20.fas' to create a sequence object in the workspace:

```
1  $ yabby
```

```
2
3   - YABBY version 0.1 -
4
5   [ 35 command(s) ready ]
6
7  yabby> seq_load tom20.fas tom20
8
9   Reading the file 'tom20.fas' ..
10  3 sequence(s) found.
11
12 yabby> what
13
14     object(s)      type
15   -------------------------------
16     tom20          seq
17
18 yabby> print tom20.seq
19
20 >A.thaliana2 [ A.thaliana2 ]
21 MEFSTADFERFIMFEHARKNSEAQYKNDPLDSENLLKWGGALLELSQFQPIPEAKLMLND
22 AISKLEEALTINPGKHQALWCIANAYTAHAFYVHDPEEAKEHFDKATEYFQRAENEDPGN
23 DTYRKSLDSSLKAPELHMQFMNQGMGQQILGGGGGGGGGGMASSNVSQSSKKKKRNTEFT
24 YDVCGWIILACGIVAWVGMAKSLGPPPPAR
25 >O.sativa [ O.sativa ]
26 MDMGAMSDPERMFFFDLACQNAKVTYEQNPHDADNLARWGGALLELSQMRNGPESLKCLE
27 DAESKLEEALKIDPMKADALWCLGNAQTSHGFFTSDTVKANEFFEKATQCFQKAVDVEPA
28 NDLYRKSLDLSSKAPELHMEIHRQMASQASQAASSTSNTRQSRKKKKDSDFWYDVFGWVV
29 LGVGMVVWVGLAKSNAPPQAPR
30 >L.esculentum [ L.esculentum ]
31 MDMQSDFDRLLFFEHARKTAETTYATDPLDAENLTRWAGALLELSQFQSVSESKKMISDA
32 ISKLEEALEVNPQKHDAIWCLGNAYTSHGFLNPDEDEAKIFFDKAAQCFQQAVDADPENE
33 LYQKSFEVSSKTSELHAQIHKQGPLQQAMGPGPSTTTSSTKGAKKKSSDLKYDVFGWVIL
34 AVGLVAWIGFAKSNMPXPAHPLPR
```

In the line 1 the Yabby was started from the Unix shell command line. During the initialization process the workspace directory .yabby was silently created. The command

```
seq_load tom20.fas tom20
```

will read the sequence from the file 'tom20.fas', and save the sequences under the name 'tom20'. The command 'what' shown on the line 12 allows one to inspect the content of the workspace. It shows that the workspace contains one object, named 'tom20', and this object is of the type 'seq' (it is a sequence object). The command 'print' on the line 18 has printed the object 'tom20' on the terminal screen.

The command 'seq_load' has executed the script 'seq_load.pl' located in the Yabby library. This command reads the FASTA sequence file and converts it into an Yabby object of the type 'seq'. The command 'print' has executed the script 'print.pl' from the Yabby library. The scripts 'seq_load.pl' and 'print.pl' are completely independent.

How was the script 'print.pl' aware of the data object 'tom20.seq', and how was the data transferred between the script 'seq_load.pl' to the script 'print.pl' for printing?

This was achieved by the use of the workspace directory. When Yabby was started (line 1) the workspace directory .yabby/ was silently created in the current working directory. The newly created object 'tom20' (the result of the command 'seq_load') was stored in the file 'tom20.seq' in the workspace directory. On the line 18, the command 'print' was requested to print the object 'tom20' of the type 'seq'. The command 'print' has attempted to find the object 'tom20.seq' in the workspace. This was successful, and the object was loaded from the workspace directory, and then printed on the terminal screen.

Consider printing a non-existent sequence object:

```
yabby> print tom99.seq

 _print_seq:: missing requirement: 'tom99.seq'

 print:: ERROR: system script '_print_seq' failed
 [ error occurred in the subroutine call() ]
 [ command 'print' failed ]
```

The message above shows that 'tom99.seq' ('tom99' object of the type 'seq') is not in the workspace. Furthermore, printing of the sequence objects is handled by the system script '_print_seq.pl', which is normally invoked by the command script 'print.pl'. Hence the error message comes from '_print_seq'.

Consider printing of an object of a non-exiting type:

```
yabby> print tom20.ttt

 print:: ERROR: printing the property 'ttt' not yet implemented
 [ command 'print' failed ]
```

In this case the error message originated from the command script 'print.pl'. This command script needs to decide which system script to call for printing. It first determines if the printing of the requested object type has been implemented, and bombs out if not.

The relationship between the working directory, workspace, and Yabby library is shown in Figure 1.1.

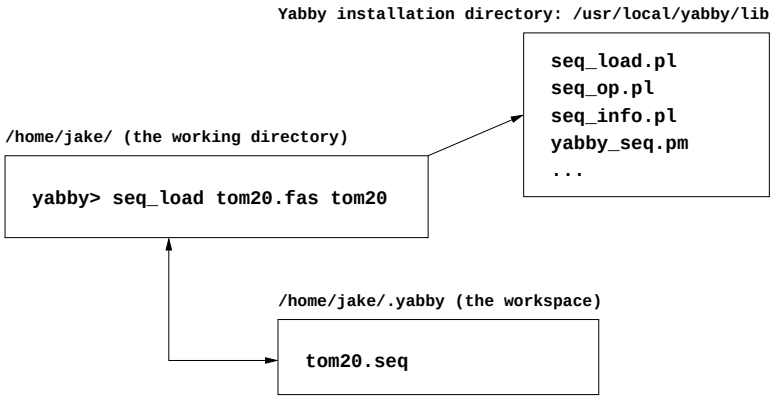

Figure 1.1: The relationship between the working directory, Yabby library, and the workspace directory. In this example the Yabby session was started in the user's home directory (/home/jake). During the initialization, Yabby has created the workspace directory (/home/jake/.yabby). The execution of the command seq_load from the Yabby shell has executed the script seq_load.pl from the library, and has created the tom20.seq object. This object is stored in the file tom20.seq, created in the workspace directory.

1.4.2 The workspace

The workspace is the area where persistent objects are held, normally an area of the computer memory. In Yabby, the workspace is an area of the local persistent storage (hard disk), and more specifically, the workspace directory (or folder). This directory is automatically created when Yabby is started, and is silently destroyed upon a graceful exit from Yabby. The consequence of this is that one can directly inspect what is in Yabby's workspace. If the Yabby session has been started in /home/jake, and the following command is executed:

```
yabby> seq_load tom20.fas tom20

 Reading the file 'tom20.fas' ..
 3 sequence(s) found.
```

This has created the object 'tom20.seq'. One can inspect this object directly by using system tools:

```
$ cd /home/jake/.yabby
$ ls
tom20.seq
$ cat tom20.seq
<?xml version="1.0"?><seqroot><seqentry><seqid>A.thaliana2</
seqid><comment>A.thaliana2</comment><sequence>MEFSTADFERFIMF
EHARKNSEAQYKNDPLDSENLLKWGGALLELSQFQPIPEAKLMLNDAISKLEEALTINPG
```

```
KHQALWCIANAYTAHAFYVHDPEEAKEHFDKATEYFQRAENEDPGNDTYRKSLDSSLKAP
ELHMQFMNQGMGQQILGGGGGGGGGGMASSNVSQSSKKKKRNTEFTYDVCGWIILACGIV
AWVGMAKSLGPPPPAR</sequence></seqentry><seqentry><seqid>O.sat
iva</seqid><comment>O.sativa</comment><sequence>MDMGAMSDPERM
FFFDLACQNAKVTYEQNPHDADNLARWGGALLELSQMRNGPESLKCLEDAESKLEEALKI
DPMKADALWCLGNAQTSHGFFTSDTVKANEFFEKATQCFQKAVDVEPANDLYRKSLDLSS
KAPELHMEIHRQMASQASQAASSTSNTRQSRKKKKDSDFWYDVFGWVVLGVGMVVWVGLA
KSNAPPQAPR</sequence></seqentry><seqentry><seqid>L.esculentu
m</seqid><comment>L.esculentum</comment><sequence>MDMQSDFDRL
LFFEHARKTAETTYATDPLDAENLTRWAGALLELSQFQSVSESKKMISDAISKLEEALEV
NPQKHDAIWCLGNAYTSHGFLNPDEDEAKIFFDKAAQCFQQAVDADPENELYQKSFEVSS
KTSELHAQIHKQGPLQQAMGPGPSTTTSSTKGAKKKSSDLKYDVFGWVILAVGLVAWIGF
AKSNMPXPAHPLPR</sequence></seqentry></seqroot>
```

The object 'tom20.seq' is a plain text XML document which captures the data about sequences.

The ability to access the workspace allows one to inspect the existing objects while they reside in Yabby's "memory", which often comes handy. For example, this may facilitate the understanding of an object's internal representation, and helps the implementation (and debugging) of data models that need to be developed for new data types.

1.4.3 The data formats

Currently Yabby implements two different data formats: two-dimensional list (table) and XML document. In many bioinformatics applications data can be effectively represented by two dimensional tables. Examples include the list of atoms in a molecule, the list of atom coordinates, the list of sequence IDs and their rankings, etc. For such data, two dimensional tables are the ideal representation. Occassionally, the objects which have more complex internal structure need to be handled. For example, a list of protein sequences where each sequence may need to be characterized with several attributes: sequence ID, organism name, the sequence string consisting of amino acid residues and so on.

As an example for a two-dimensional list data format, consider the property 'mol' (molecule).

```
yabby> mol_load 1BT0.pdb rub

 661 atoms found in the molecule 'rub'

yabby> what

    object(s)      type
  ------------------------------
    rub            mol
```

The command 'mol_load' has created the object 'rub.mol'. This object is stored in the workspace as the

file 'rub.mol':

```
$ cd .yabby
/home/jack/data/.yabby
$ ls
rub.mol
$ head rub.mol
1 MET N
1 MET CA
1 MET C
1 MET O
1 MET CB
1 MET CG
1 MET SD
1 MET CE
2 LEU N
2 LEU CA
```

This is an example where the two-dimensional list fits perfectly the data type: a molecule is a list of atoms, and each atom is characterized by residue number, residue name, and atom name. This information can be naturally arranged into a two-dimensional table, with one atom per table row.

On the other hand, the protein or DNA sequence is not naturally amenable to the two-dimensional list representation. A sequence may be characterized by several attributes (sequence ID, organism name, database specific IDs, and so on) in addition to the sequence itself, which may contain a few to many thousands of letters. The matter is further complicated if the data model needs to handle transparently one or more sequences. Yabby uses XML to represent such data which requires more capable representations.

Consider the content of the file 'tom20.seq' which contains the information about three tom20 sequences given previously. If arranged in a more readable format, this sequence object looks as given below:

```
<?xml version="1.0"?>
<seqroot>
  <seqentry>
    <seqid>A.thaliana2</seqid>
    <comment>A.thaliana2</comment>
    <sequence>MEFSTADFERFIMFEHARKNSEAQYKNDPLDSENLLKWGGALLELSQF
      QPIPEAKLMLNDAISKLEEALTINPGKHQALWCIANAYTAHAFYVHDPEEAKEHFD
      KATEYFQRAENEDPGNDTYRKSLDSSLKAPELHMQFMNQGMGQQILGGGGGGGGGG
      MASSNVSQSSKKKKRNTEFTYDVCGWIILACGIVAWVGMAKSLGPPPPAR
    </sequence>
  </seqentry>
  <seqentry>
    <seqid>O.sativa</seqid>
```

```
    <comment>O.sativa</comment>
    <sequence>MDMGAMSDPERMFFFDLACQNAKVTYEQNPHDADNLARWGGALLELSQM
      RNGPESLKCLEDAESKLEEALKIDPMKADALWCLGNAQTSHGFFTSDTVKANEFFEK
      ATQCFQKAVDVEPANDLYRKSLDLSSKAPELHMEIHRQMASQASQAASSTSNTRQSR
      KKKKDSDFWYDVFGWVVLGVGMVVWVGLAKSNAPPQAPR</sequence>
  </seqentry>
  <seqentry>
  <seqid>L.esculentum</seqid>
    <comment>L.esculentum</comment>
    <sequence>MDMQSDFDRLLFFEHARKTAETTYATDPLDAENLTRWAGALLELSQFQS
    VSESKKMISDAISKLEEALEVNPQKHDAIWCLGNAYTSHGFLNPDEDEAKIFFDKAAQC
    FQQAVDADPENELYQKSFEVSSKTSELHAQIHKQGPLQQAMGPGPSTTTSSTKGAKKKS
    SDLKYDVFGWVILAVGLVAWIGFAKSNMPXPAHPLPR</sequence>
  </seqentry>
</seqroot>
```

Each sequence is enclosed by the XML tags <seqentry> and </seqentry>. Furthermore, the individual attributes for each sequence are enclosed in tags such as <seqid> and </seqid>, <comment> and </comment> and so on. This allows considerable flexibility in the representation of data with complex internal structure. The price for this flexibility is complexity, as more effort is required to design and then implement an XML based format compared to a two-dimensional list format.

Tutorial

The data files used in this tutorial can be found in docs/data/ directory. Starting Yabby in this directory should allow one to execute all examples given in the tutorial.

2.1 Working with sequences

In Yabby sequences are represented as sequence objects. A sequence object may contain one or more sequences.

One way to create a sequence object is to load sequences from a file. Consider the file 'cad3.fas' which contains three sequences from the Pfam CAD family in the FASTA format:

```
>Q53650_STAAU
YVATGIDYLVILILLFSQVKKGQVKHIWIGQYIGTAIVIGASLLVAQGVVNLIPQQWVIG
LLGLLPLYLGVKIWIKGEEDEDESSILSLFSSGKFNQLFLTMIFIVLASSADDFSIYIPY
FTTLSMSEIFIVTIVFLIMVGVLCYVSYRLASFDFISETIEKYERWIVPIVFIGLGIYIL
FENGTSNALISF
>Q97PJ0_STRPN
YISTSIDYLIILIILFAQLSQNKQKWHIYAGQYLGTGLLVGASLVAAYVVNFVPEEWMVG
LLGLIPIYLGIRFAIVGEDAEEEEEEIIERLEQSKANQLFWTVTLLTIASGGDNLGIYIP
YFASLDWSQTLVALLVFVIGIIIFCEISRVLSSIPLIFETIEKYERIIVPLVFILLGLYI
MYENGTIETFLIV
>P95773_STALU
YIAQALDLLVILLMFFARAKTRKEYRDIYIGQYVGSVALIVISLFFAFVLNYVPEKWILG
LLGLIPIYLGIKVAIYGDSDGEERAKKELNEKGLSKLVGTIAIVTIASCGADNIGLFVPY
FVTLSVTNLLITLFVFLILIFFLVFAAQKLANIPEVGEIVEKFGRWIMAVIYIALGLFII
IENDTIQTILGF
```

To load this file in the workspace use the command 'seq_load':

```
yabby> seq_load cad3.fas cad3
```

```
 Reading the file 'cad3.fas' ..
 3 sequence(s) found.
```

Most Yabby commands which create a new object in the workspace conform to the same pattern:

```
COMMAND [ options ] ARGS OBJ_NAME
```

Where '[options]' is where flags and options are passed (if any), ARGS are the arguments to the command, and OBJ_NAME is the name of the object to be created. In the example given above, there were no options to the 'seq_load' command; there was one argument, the name of the file; and the object to be created was named 'cad3'.

It is possible to inspect objects currently available in the workspace:

```
yabby> what

    object(s)      type
  ------------------------------
    cad3           seq
```

This listing shows that one object is currently in the workspace, of the 'seq' type. It is possible to load the same sequences under a different name:

```
yabby> seq_load cad3.fas cad3_second

 Reading the file 'cad3.fas' ..
 3 sequence(s) found.

yabby> what

    object(s)      type
  ------------------------------
    cad3           seq
    cad3_second    seq
```

The two objects 'cad3.seq' and 'cad3_second.seq' are identical.

The command 'print' allows one to output the sequence object:

```
yabby> print cad3.seq

>Q53650_STAAU [ Q53650_STAAU ]
YVATGIDYLVILILLFSQVKKGQVKHIWIGQYIGTAIVIGASLLVAQGVVNLIPQQWVIG
```

```
LLGLLPLYLGVKIWIKGEEDEDESSILSLFSSGKFNQLFLTMIFIVLASSADDFSIYIPY
FTTLSMSEIFIVTIVFLIMVGVLCYVSYRLASFDFISETIEKYERWIVPIVFIGLGIYIL
FENGTSNALISF
>Q97PJ0_STRPN [ Q97PJ0_STRPN ]
YISTSIDYLIILIILFAQLSQNKQKWHIYAGQYLGTGLLVGASLVAAYVVNFVPEEWMVG
LLGLIPIYLGIRFAIVGEDAEEEEEEIIERLEQSKANQLFWTVTLLTIASGGDNLGIYIP
YFASLDWSQTLVALLVFVIGIIIFCEISRVLSSIPLIFETIEKYERIIVPLVFILLGLYI
MYENGTIETFLIV
>P95773_STALU [ P95773_STALU ]
YIAQALDLLVILLMFFARAKTRKEYRDIYIGQYVGSVALIVISLFFAFVLNYVPEKWILG
LLGLIPIYLGIKVAIYGDSDGEERAKKELNEKGLSKLVGTIAIVTIASCGADNIGLFVPY
FVTLSVTNLLITLFVFLILIFFLVFAAQKLANIPEVGEIVEKFGRWIMAVIYIALGLFII
IENDTIQTILGF
```

Upon reading the sequences Yabby has taken the first string in the FASTA comment to be the sequence ID, and the full comment is re-inserted within the square brackets. In this case the only comment was the ID string, and this is merely repeated within the square brackets.

The command 'print' can send sequences to a file, instead of printing them in the terminal window:

```
yabby> print -f tmp.fasta cad3.seq

 'cad3.seq' written to the file 'tmp.fasta'
```

Currently there are several useful options of the 'print' command. The option '-l' causes protein sequences to be printed in a three-letter format:

```
yabby> print -l cad3.seq

>Q53650_STAAU
TYR VAL ALA THR GLY ILE ASP TYR LEU VAL ILE LEU
ILE LEU LEU PHE SER GLN VAL LYS LYS GLY GLN VAL
....further output deleted....
```

The option '-t N' truncates all sequences at N residues:

```
yabby> print -t 10 cad3.seq

>Q53650_STAAU [ Q53650_STAAU ]
YVATGIDYLV
>Q97PJ0_STRPN [ Q97PJ0_STRPN ]
YISTSIDYLI
>P95773_STALU [ P95773_STALU ]
YIAQALDLLV
```

The command 'seq_info' prints additional information about sequence objects:

```
yabby> seq_info cad3

 'cad3' contains 3 sequence(s)
   min number of residues: 192 (sequence 'Q53650_STAAU')
   max number of residues: 193 (sequence 'Q97PJ0_STRPN')
```

The option '-l' causes the number of residues to be printed for each sequence:

```
yabby> seq_info -l cad3

 'cad3' contains 3 sequence(s)
  1 -> Q53650_STAAU, 192 residues
  2 -> Q97PJ0_STRPN, 193 residues
  3 -> P95773_STALU, 192 residues
```

It is often required to select one or more sequences from the sequence object. The command 'seq_pick' allows one to select sequences from their order number or sequence ID. For example:

```
yabby> seq_pick -n 2 cad3 s2

 Fetching the sequence 2 ('Q97PJ0_STRPN')
 Saving the extracted sequence as 's2'
```

The above command has picked the sequence number 2 from the 'cad3' object, and saved this sequence under the name 's2':

```
yabby> what

    object(s)      type
  ------------------------------
    cad3           seq
    s2             seq

yabby> print s2.seq

>Q97PJ0_STRPN [ Q97PJ0_STRPN ]
YISTSIDYLIILIILFAQLSQNKQKWHIYAGQYLGTGLLVGASLVAAYVVNFVPEEWMVG
LLGLIPIYLGIRFAIVGEDAEEEEEEIIERLEQSKANQLFWTVTLLTIASGGDNLGIYIP
YFASLDWSQTLVALLVFVIGIIIFCEISRVLSSIPLIFETIEKYERIIVPLVFILLGLYI
MYENGTIETFLIV
```

Another useful option is to select a sequence by its ID string:

```
yabby> seq_pick -q Q53650_STAAU cad3 s1

 Fetching the sequence 'Q53650_STAAU'
 Saving the extracted sequence as 's1'

yabby> what

    object(s)      type
  ------------------------------
    cad3           seq
    s1             seq
    s2             seq
```

Suppose that we wanted to take the sequence Q53650_STAAU, and extract residues 21-40. The above command would take care of the first part, while the command 'seq_strip' could be used to select a residue range:

```
yabby> seq_strip 21:40 s1 s1_portion

 's1' contains 1 sequence(s)
 stripping 'Q53650_STAAU'

yabby> print s1_portion.seq

>Q53650_STAAU_21:40 [ Q53650_STAAU ]
KGQVKHIWIGQYIGTAIVIG
```

The command 'seq_pattern' allows one to search for a pattern in a sequence. For example, to search for a pattern 'IDY' use:

```
yabby> seq_pattern IDY cad3

 'IDY' matches in 'Q53650_STAAU'
 'IDY' matches in 'Q97PJ0_STRPN'
 3 sequences examined, 2 match(es) found
```

The option '-s NAME' allows one to save the matching sequences under a the new name:

```
yabby> seq_pattern -s IDY_matches IDY cad3
```

```
 'IDY' matches in 'Q53650_STAAU'
 'IDY' matches in 'Q97PJ0_STRPN'
 3 sequences examined, 2 match(es) found
 Saving matches as 'IDY_matches'
 [ seq_pattern: 'IDY_matches.seq' exists, overwritten ]
```

The option '-c' allows one to search the sequence comment for a pattern, rather than the sequence residues:

```
yabby> seq_pattern -c STAA cad3

 'STAA' matches in 'Q53650_STAAU'
 3 sequences examined, 1 match(es) found
```

The command 'seq_op' allows one to combine two sequence objects based on their IDs. This command has the following form:

```
seq_op [ options ] SEQ1_OBJ SEQ2_OBJ RES_OBJ
```

where SEQ1_OBJ SEQ2_OBJ are two sequence objects to be combined and RES_OBJ is the name under which the result will be stored. This command can calculate the union, intersection, symmetric difference.

For example, consider the sets of sequences stored in files 'cad.fas' and 'cad3.fas', where the first file contains six CAD sequences, and cad3.fas contains three out of six sequences present in 'cad.fas'.

```
yabby> seq_load cad.fas cad

 Reading the file 'cad.fas' ..
 6 sequence(s) found.

yabby> seq_load cad3.fas cad3

 Reading the file 'cad3.fas' ..
 3 sequence(s) found.

yabby> seq_op -i cad cad3 cad_i

 Found 6 sequence(s) in 'cad'
 Found 3 sequence(s) in 'cad3'
  INTERSECTION contains 3 sequence(s)
  DIFFERENCE contains 3 sequence(s)
  UNION contains 6 sequence(s)
 [ Saving INTERSECTION as 'cad_i' ]
 [ seq_op: 'cad_i.seq' exists, overwritten ]
```

The last command has saved the intersection of objects 'cad.seq' and 'cad3.seq' as a new sequence object, named 'cad_i'. Since 'cad3.seq' is simply the subset of 'cad.seq' this intersection is identical to 'cad3.seq'. Options '-u' and '-d' save the union and difference. It should be noted that the command 'seq_op' compares only sequence IDs, *it does not compare the sequences themselves.* Furthermore, unpredictable results may occur if the sequence IDs are not unique within each set of sequences.

The command 'seq_comment' modifies the comment of each sequence to append the sequence number to its ID. To modify the IDs of the sequences in 'cad3':

```
yabby> seq_comment cad3

 Comments modified in 3 sequence(s).

yabby> print cad3.seq

>1-Q53650_STAAU [ Q53650_STAAU ]
YVATGIDYLVILILLFSQVKKGQVKHIWIGQYIGTAIVIGASLLVAQGVVNLIPQQWVIG
LLGLLPLYLGVKIWIKGEEDEDESSILSLFSSGKFNQLFLTMIFIVLASSADDFSIYIPY
FTTLSMSEIFIVTIVFLIMVGVLCYVSYRLASFDFISETIEKYERWIVPIVFIGLGIYIL
FENGTSNALISF
>2-Q97PJ0_STRPN [ Q97PJ0_STRPN ]
YISTSIDYLIILIILFAQLSQNKQKWHIYAGQYLGTGLLVGASLVAAYVVNFVPEEWMVG
LLGLIPIYLGIRFAIVGEDAEEEEEEIIERLEQSKANQLFWTVTLLTIASGGDNLGIYIP
YFASLDWSQTLVALLVFVIGIIIFCEISRVLSSIPLIFETIEKYERIIVPLVFILLGLYI
MYENGTIETFLIV
>3-P95773_STALU [ P95773_STALU ]
YIAQALDLLVILLMFFARAKTRKEYRDIYIGQYVGSVALIVISLFFAFVLNYVPEKWILG
LLGLIPIYLGIKVAIYGDSDGEERAKKELNEKGLSKLVGTIAIVTIASCGADNIGLFVPY
FVTLSVTNLLITLFVFLILIFFLVFAAQKLANIPEVGEIVEKFGRWIMAVIYIALGLFII
IENDTIQTILGF
```

Since this command will change the sequence IDs, the output of the command 'seq_op' would be different. Specifically, the intersection of the two sets of sequences would contain no sequences:

```
yabby> seq_op -i cad cad3 cad_u

 Found 6 sequence(s) in 'cad'
 Found 3 sequence(s) in 'cad3'
  INTERSECTION contains 0 sequence(s)
  DIFFERENCE contains 9 sequence(s)
  UNION contains 9 sequence(s)
 [ No sequences to save ]

yabby> seq_op -u cad cad3 cad_u
```

```
 Found 6 sequence(s) in 'cad'
 Found 3 sequence(s) in 'cad3'
  INTERSECTION contains 0 sequence(s)
  DIFFERENCE contains 9 sequence(s)
  UNION contains 9 sequence(s)
 [ Saving UNION as 'cad_u' ]
```

The command 'seq_unique' finds sequences present in one set not present in the other, by *comparing sequence strings*.

```
yabby> seq_unique cad cad3 cadu

 3 unique sequences found.
 Saving sequences as 'cadu'
```

Note that in this case the order of sequence objects is important. This command find the sequences present in the first sequence object and not present in the second, i.e.:

```
yabby> seq_unique cad3 cad tmp

 No unique sequences found.
```

The command 'seq_compl' calculates the reverse complement of a DNA sequence:

```
yabby> seq_load dna.fas dna

 Reading the file 'dna.fas' ..
 1 sequence(s) found.

yabby> print dna.seq

>chr01 [ chr01 ]
taaccctaaccctaaccctgaccctaaccctaaccctaaccctaaccctaaccctaaccagtacac
gcgtacacgtacaagcacccgtacccccagtatacttggacacccgtactcagttatcct
ttttattagtgtacccgcctcttgcacgcatgccacagttcttcagcagaagaacacgca
caatgctctttgataaacgtgcggacatgaaaaaaagggaaaaacgcagctacgtgtgct
gtcgttggtttcacagcgtcaagccgcgtcggtgtaccaaagaggaggtgacccatcgag

yabby> seq_compl dna dna_c

 'dna' contains 1 sequence(s)
 Working on 'chr01'
```

```
yabby> print dna_c.seq

>chr01 [ chr01 ]
attgggattgggattgggactgggattgggattgggattgggattgggattggtcatgtg
cgcatgtgcatgttcgtgggcatgggggtcatatgaacctgtgggcatgagtcaatagga
aaaataatcacatgggcggagaacgtgcgtacggtgtcaagaagtcgtcttcttgtgcgt
gttacgagaaactatttgcacgcctgtactttttttccctttttgcgtcgatgcacacga
cagcaaccaaagtgtcgcagttcggcgcagccacatggtttctcctccactgggtagctc
```

The option '-r' calculates the reverse complement:

```
yabby> seq_compl -r dna dna_cr

 'dna' contains 1 sequence(s)
 Working on 'chr01'

yabby> print dna_cr.seq

>chr01 [ chr01 ]
ctcgatgggtcacctcctctttggtacaccgacgcggcttgacgctgtgaaaccaacgac
agcacacgtagctgcgtttttccctttttttcatgtccgcacgtttatcaaagagcattg
tgcgtgttcttctgctgaagaactgtggcatgcgtgcaagaggcgggtacactaataaaa
aggataactgagtacgggtgtccaagtatactgggggtacgggtgcttgtacgtgtacgc
gtgtactggttagggttagggttagggttagggttagggtcagggttagggttagggtta
```

The command 'seq_genbank' fetches a sequence from GenBank by default using the sequence accession number:

```
yabby> seq_genbank -a J00522 mig

 Saving 'J00522' as 'mig'

yabby> what

    object(s)      type
  -------------------------------
    mig            seq
```

This command requires the module Bio::DB::GenBank and illustrates how BioPerl [1] can be used within Yabby.

The command 'seq_os' inserts additional information into sequence comments, where additional information is read from an external file. For example, the file 'sprot_test.dat.os' contains the list of sequences IDs and associated organism names, one line per sequence:

```
104K_THEAN Theileria annulata.
104K_THEPA Theileria parva.
108_SOLLC Solanum lycopersicum (Tomato) (Lycopersicon esculentum).
10KD_VIGUN Vigna unguiculata (Cowpea).
110KD_PLAKN Plasmodium knowlesi.
11S2_SESIN Sesamum indicum (Oriental sesame) (Gingelly).
11S3_HELAN Helianthus annuus (Common sunflower).
11SB_CUCMA Cucurbita maxima (Pumpkin) (Winter squash).
128UP_DROME Drosophila melanogaster (Fruit fly).
12AH_CLOS4 Clostridium sp. (strain C 48-50).
12KD_FRAAN Fragaria ananassa (Strawberry).
12KD_MYCSM Mycobacterium smegmatis.
12S1_ARATH Arabidopsis thaliana (Mouse-ear cress).
12S2_ARATH Arabidopsis thaliana (Mouse-ear cress).
12S_PROFR Propionibacterium freudenreichii subsp. shermanii.
13KDA_SCYCA Scyliorhinus canicula (Spotted dogfish) (Spotted catshark).
13KDA_TRISC Triakis scyllium (Leopard shark) (Triakis scyllia).
13S1_FAGES Fagopyrum esculentum (Common buckwheat).
```

The command 'seq_os' could be used to insert the organism name in the sequence object comments by using the file listed above. To illustrate this, we first load the test set of sequences:

```
yabby> seq_load sprot_test.fas test

 Reading the file 'sprot_test.fas' ..
 3 sequence(s) found.

yabby> print test.seq

>104K_THEPA [ 104K_THEPA 104 kDa microneme/rhoptry antigen precursor (p104) ]
MKFLILLFNILCLFPVLAADNHGVGPQGASGVDPITFDINSNQTGPAFLTAVEMAGVKYL
QVQHGSNVNIHRLVEGNVVIWENASTPLYTGAIVTNNDGPYMAYVEVLGDPNLQFFIKSG
DAWVTLSEHEYLAKLQEIRQAVHIESVFSLNMAFQLENNKYEVETHAKNGANMVTFIPRN
GHICKMVYHKNVRIYKATGNDTVTSVVGFFRGLRLLLINVFSIDDNGMMSNRYFQHVDDK
YVPISQKNYETGIVKLKDYKHAYHPVDLDIKDIDYTMFHLADATYHEPCFKIIPNTGFCI
TKLFDGDQVLYESFNPLIHCINEVHIYDRNNGSIICLHLNYSPPSYKAYLVLKDTGWEAT
THPLLEEKIEELQDQRACELDVNFISDKDLYVAALTNADLNYTMVTPRPHRDVIRVSDGS
EVLWYYEGLDNFLVCAWIYVSDGVASLVHLRIKDRIPANNDIYVLKGDLYWTRITKIQFT
QEIKRLVKKSKKKLAPITEEDSDKHDEPPEGPGASGLPPKAPGDKEGSEGHKGPSKGSDS
SKEGKKPGSGKKPGPAREHKPSKIPTLSKKPSGPKDPKHPRDPKEPRKSKSPRTASPTRR
PSPKLPQLSKLPKSTSPRSPPPPTRPSSPERPEGTKIIKTSKPPSPKPPFDPSFKEKFYD
DYSKAASRSKETKTTVVLDESFESILKETLPETPGTPFTTPRPVPPKRPRTPESPFEPPK
DPDSPSTSPSEFFTPPESKRTRFHETPADTPLPDVTAELFKEPDVTAETKSPDEAMKRPR
SPSEYEDTSPGDYPSLPMKRHRLERLRLTTTEMETDPGRMAKDASGKPVKLKRSKSFDDL
```

```
TTVELAPEPKASRIVVDDEGTEADDEETHPPEERQKTEVRRRRPPKKPSKSPRPSKPKKP
KKPDSAYIPSILAILVVSLIVGIL
>110KD_PLAKN [ 110KD_PLAKN 110 kDa antigen (PK110) (Fragment) ]
FNSNMLRGSVCEEDVSLMTSIDNMIEEIDFYEKEIYKGSHSGGVIKGMDYDLEDDENDED
EMTEQMVEEVADHITQDMIDEVAHHVLDNITHDMAHMEEIVHGLSGDVTQIKEIVQKVNV
AVEKVKHIVETEETQKTVEPEQIEETQNTVEPEQTEETQKTVEPEQTEETQNTVEPEQIE
ETQKTVEPEQTEEAQKTVEPEQTEETQKTVEPEQTEETQKTVEPEQTEETQKTVEPEQTE
ETQKTVEPEQTEETQKTVEPEQTEETQKTVEPEQTEETQNTVEPEPTQETQNTVEP
>12KD_FRAAN [ 12KD_FRAAN Auxin-repressed 12.5 kDa protein. ]
MVLLDKLWDDIVAGPQPERGLGMLRKVPQPLNLKDEGESSKITMPTTPTTPVTPTTPISA
RKDNVWRSVFHPGSNLSSKTMGNQVFDSPQPNSPTVYDWMYSGETRSKHHR
```

The command 'seq_os' takes two arguments, the file name with sequence IDs and additional information (see above), and the sequence object name:

```
yabby> seq_os sprot_test.dat.os test

 Reading the file 'sprot_test.dat.os' ...
 3 sequences to process.

 All done.
 [ seq_os: 'test.seq' exists, overwritten ]
```

The original sequence object was overwritten, but the difference is only in that the comments are changed to contain the information provided in the file 'sprot_test.dat.os':

```
yabby> print test.seq

>104K_THEPA [ 104K_THEPA 104 kDa microneme/rhoptry antigen
... precursor (p104) Theileria parva. ]
MKFLILLFNILCLFPVLAADNHGVGPQGASGVDPITFDINSNQTGPAFLTAVEMAGVKYL
QVQHGSNVNIHRLVEGNVVIWENASTPLYTGAIVTNNDGPYMAYVEVLGDPNLQFFIKSG
DAWVTLSEHEYLAKLQEIRQAVHIESVFSLNMAFQLENNKYEVETHAKNGANMVTFIPRN
GHICKMVYHKNVRIYKATGNDTVTSVVGFFRGLRLLLINVFSIDDNGMMSNRYFQHVDDK
YVPISQKNYETGIVKLKDYKHAYHPVDLDIKDIDYTMFHLADATYHEPCFKIIPNTGFCI
TKLFDGDQVLYESFNPLIHCINEVHIYDRNNGSIICLHLNYSPPSYKAYLVLKDTGWEAT
THPLLEEKIEELQDQRACELDVNFISDKDLYVAALTNADLNYTMVTPRPHRDVIRVSDGS
EVLWYYEGLDNFLVCAWIYVSDGVASLVHLRIKDRIPANNDIYVLKGDLYWTRITKIQFT
QEIKRLVKKSKKKLAPITEEDSDKHDEPPEGPGASGLPPKAPGDKEGSEGHKGPSKGSDS
SKEGKKPGSGKKPGPAREHKPSKIPTLSKKPSGPKDPKHPRDPKEPRKSKSPRTASPTRR
PSPKLPQLSKLPKSTSPRSPPPPTRPSSPERPEGTKIIKTSKPPSPKPPFDPSFKEKFYD
DYSKAASRSKETKTTVVLDESFESILKETLPETPGTPFTTPRPVPPKRPRTPESPFEPPK
```

```
DPDSPSTSPSEFFTPPESKRTRFHETPADTPLPDVTAELFKEPDVTAETKSPDEAMKRPR
SPSEYEDTSPGDYPSLPMKRHRLERLRLTTTEMETDPGRMAKDASGKPVKLKRSKSFDDL
TTVELAPEPKASRIVVDDEGTEADDEETHPPEERQKTEVRRRRPPKKPSKSPRPSKPKKP
KKPDSAYIPSILAILVVSLIVGIL
>110KD_PLAKN [ 110KD_PLAKN 110 kDa antigen (PK110) (Fragment)
... Plasmodium knowlesi. ]
FNSNMLRGSVCEEDVSLMTSIDNMIEEIDFYEKEIYKGSHSGGVIKGMDYDLEDDENDED
EMTEQMVEEVADHITQDMIDEVAHHVLDNITHDMAHMEEIVHGLSGDVTQIKEIVQKVNV
AVEKVKHIVETEETQKTVEPEQIEETQNTVEPEQTEETQKTVEPEQTEETQNTVEPEQIE
ETQKTVEPEQTEEAQKTVEPEQTEETQKTVEPEQTEETQKTVEPEQTEETQKTVEPEQTE
ETQKTVEPEQTEETQKTVEPEQTEETQKTVEPEQTEETQNTVEPEPTQETQNTVEP
>12KD_FRAAN [ 12KD_FRAAN Auxin-repressed 12.5 kDa protein.
... Fragaria ananassa (Strawberry). ]
MVLLDKLWDDIVAGPQPERGLGMLRKVPQPLNLKDEGESSKITMPTTPTTPVTPTTPISA
RKDNVWRSVFHPGSNLSSKTMGNQVFDSPQPNSPTVYDWMYSGETRSKHHRim}
```

2.1.1 Swiss-Prot related commands

Several commands exist for the manipulation of Swiss-Prot data files.

The command 'sprot_fetch' fetches a single sequence from the Swiss-Prot data file based on the sequence unique ID. For example, the Swiss-Prot file 'sprot_test.dat' contains 18 sequences, including the sequence with the ID '110KD_PLAKN'. The command 'sprot_fetch' could be used to extract this sequence from the database file:

```
yabby> sprot_fetch sprot_test.dat 110KD_PLAKN plakn

 Sequence '110KD_PLAKN' found.
 [ Saving as 'plakn.seq' ]

yabby> print plakn.seq

>110KD_PLAKN [ 110 kDa antigen (PK110) (Fragment). Plasmodium knowlesi. ]
FNSNMLRGSVCEEDVSLMTSIDNMIEEIDFYEKEIYKGSHSGGVIKGMDYDLEDDENDED
EMTEQMVEEVADHITQDMIDEVAHHVLDNITHDMAHMEEIVHGLSGDVTQIKEIVQKVNV
AVEKVKHIVETEETQKTVEPEQIEETQNTVEPEQTEETQKTVEPEQTEETQNTVEPEQIE
ETQKTVEPEQTEEAQKTVEPEQTEETQKTVEPEQTEETQKTVEPEQTEETQKTVEPEQTE
ETQKTVEPEQTEETQKTVEPEQTEETQKTVEPEQTEETQNTVEPEPTQETQNTVEP
```

The command 'seq_sprot_os' fetches the organism name from the Swiss-Prot data file (Swiss-Prot field "OS") for a set of sequences saved in the workspace. This is useful when the organism name is lost in the sequence manipulation pipeline. For example, when the Swiss-Prot database is convereted into the

FASTA format with the program "sreformat" from the Sean Eddy's SQUID toolkit, the organism names are not copied to the FASTA file.

The command 'seq_sprot_os' requires two arguments, the name of the Swiss-Prot database file and the name of an existing sequence object. To illustrate this, we first load the test set of sequences ('sprot_test.fas') in the workspace, under the name 'test':

```
yabby> seq_load sprot_test.fas test

 Reading the file 'sprot_test.fas' ..
 3 sequence(s) found.

yabby> print test.seq

>104K_THEPA [ 104K_THEPA 104 kDa microneme/rhoptry antigen precursor (p104) ]
MKFLILLFNILCLFPVLAADNHGVGPQGASGVDPITFDINSNQTGPAFLTAVEMAGVKYL
QVQHGSNVNIHRLVEGNVVIWENASTPLYTGAIVTNNDGPYMAYVEVLGDPNLQFFIKSG
DAWVTLSEHEYLAKLQEIRQAVHIESVFSLNMAFQLENNKYEVETHAKNGANMVTFIPRN
GHICKMVYHKNVRIYKATGNDTVTSVVGFFRGLRLLLINVFSIDDNGMMSNRYFQHVDDK
YVPISQKNYETGIVKLKDYKHAYHPVDLDIKDIDYTMFHLADATYHEPCFKIIPNTGFCI
TKLFDGDQVLYESFNPLIHCINEVHIYDRNNGSIICLHLNYSPPSYKAYLVLKDTGWEAT
THPLLEEKIEELQDQRACELDVNFISDKDLYVAALTNADLNYTMVTPRPHRDVIRVSDGS
EVLWYYEGLDNFLVCAWIYVSDGVASLVHLRIKDRIPANNDIYVLKGDLYWTRITKIQFT
QEIKRLVKKSKKKLAPITEEDSDKHDEPPEGPGASGLPPKAPGDKEGSEGHKGPSKGSDS
SKEGKKPGSGKKPGPAREHKPSKIPTLSKKPSGPKDPKHPRDPKEPRKSKSPRTASPTRR
PSPKLPQLSKLPKSTSPRSPPPPTRPSSPERPEGTKIIKTSKPPSPKPPFDPSFKEKFYD
DYSKAASRSKETKTTVVLDESFESILKETLPETPGTPFTTPRPVPPKRPRTPESPFEPPK
DPDSPSTSPSEFFTPPESKRTRFHETPADTPLPDVTAELFKEPDVTAETKSPDEAMKRPR
SPSEYEDTSPGDYPSLPMKRHRLERLRLTTTEMETDPGRMAKDASGKPVKLKRSKSFDDL
TTVELAPEPKASRIVVDDEGTEADDEETHPPEERQKTEVRRRRPPKKPSKSPRPSKPKKP
KKPDSAYIPSILAILVVSLIVGIL
>110KD_PLAKN [ 110KD_PLAKN 110 kDa antigen (PK110) (Fragment) ]
FNSNMLRGSVCEEDVSLMTSIDNMIEEIDFYEKEIYKGSHSGGVIKGMDYDLEDDENDED
EMTEQMVEEVADHITQDMIDEVAHHVLDNITHDMAHMEEIVHGLSGDVTQIKEIVQKVNV
AVEKVKHIVETEETQKTVEPEQIEETQNTVEPEQTEETQKTVEPEQTEETQNTVEPEQIE
ETQKTVEPEQTEEAQKTVEPEQTEETQKTVEPEQTEETQKTVEPEQTEETQKTVEPEQTE
ETQKTVEPEQTEETQKTVEPEQTEETQKTVEPEQTEETQNTVEPEPTQETQNTVEP
>12KD_FRAAN [ 12KD_FRAAN Auxin-repressed 12.5 kDa protein. ]
MVLLDKLWDDIVAGPQPERGLGMLRKVPQPLNLKDEGESSKITMPTTPTTPVTPTTPISA
RKDNVWRSVFHPGSNLSSKTMGNQVFDSPQPNSPTVYDWMYSGETRSKHHR
```

The Swiss-Prot example file 'sprot_test.dat' contains 18 sequences which include the tree from the file 'sprot_test.fas'. The command 'seq_sprot_os' could be used to fetch the OS field from the Swiss-Prot file, and insert this in the sequence comment:

```
yabby> seq_sprot_os sprot_test.dat test

 3 sequences to process.
 Printing dot per processed sequence:
 ...
 All done.
 [ seq_sprot_os: 'test.seq' exists, overwritten ]
```

This manipulation has happened in place, and the original sequence object was overwritten. the difference is of course only in the comment field:

```
yabby> print test.seq

>104K_THEPA [ 104K_THEPA 104 kDa microneme/rhoptry antigen precursor
... (p104) Theileria parva. ]
MKFLILLFNILCLFPVLAADNHGVGPQGASGVDPITFDINSNQTGPAFLTAVEMAGVKYL
QVQHGSNVNIHRLVEGNVVIWENASTPLYTGAIVTNNDGPYMAYVEVLGDPNLQFFIKSG
DAWVTLSEHEYLAKLQEIRQAVHIESVFSLNMAFQLENNKYEVETHAKNGANMVTFIPRN
GHICKMVYHKNVRIYKATGNDTVTSVVGFFRGLRLLLINVFSIDDNGMMSNRYFQHVDDK
YVPISQKNYETGIVKLKDYKHAYHPVDLDIKDIDYTMFHLADATYHEPCFKIIPNTGFCI
TKLFDGDQVLYESFNPLIHCINEVHIYDRNNGSIICLHLNYSPPSYKAYLVLKDTGWEAT
THPLLEEKIEELQDQRACELDVNFISDKDLYVAALTNADLNYTMVTPRPHRDVIRVSDGS
EVLWYYEGLDNFLVCAWIYVSDGVASLVHLRIKDRIPANNDIYVLKGDLYWTRITKIQFT
QEIKRLVKKSKKKLAPITEEDSDKHDEPPEGPGASGLPPKAPGDKEGSEGHKGPSKGSDS
SKEGKKPGSGKKPGPAREHKPSKIPTLSKKPSGPKDPKHPRDPKEPRKSKSPRTASPTRR
PSPKLPQLSKLPKSTSPRSPPPPTRPSSPERPEGTKIIKTSKPPSPKPPFDPSFKEKFYD
DYSKAASRSKETKTTVVLDESFESILKETLPETPGTPFTTPRPVPPKRPRTPESPFEPPK
DPDSPSTSPSEFFTPPESKRTRFHETPADTPLPDVTAELFKEPDVTAETKSPDEAMKRPR
SPSEYEDTSPGDYPSLPMKRHRLERLRLTTTEMETDPGRMAKDASGKPVKLKRSKSFDDL
TTVELAPEPKASRIVVDDEGTEADDEETHPPEERQKTEVRRRRPPKKPSKSPRPSKPKKP
KKPDSAYIPSILAILVVSLIVGIL
>110KD_PLAKN [ 110KD_PLAKN 110 kDa antigen (PK110) (Fragment)
... Plasmodium knowlesi. ]
FNSNMLRGSVCEEDVSLMTSIDNMIEEIDFYEKEIYKGSHSGGVIKGMDYDLEDDENDED
EMTEQMVEEVADHITQDMIDEVAHHVLDNITHDMAHMEEIVHGLSGDVTQIKEIVQKVNV
AVEKVKHIVETEETQKTVEPEQIEETQNTVEPEQTEETQKTVEPEQTEETQNTVEPEQIE
ETQKTVEPEQTEEAQKTVEPEQTEETQKTVEPEQTEETQKTVEPEQTEETQKTVEPEQTE
ETQKTVEPEQTEETQKTVEPEQTEETQKTVEPEQTEETQNTVEPEPTQETQNTVEP
>12KD_FRAAN [ 12KD_FRAAN Auxin-repressed 12.5 kDa protein.
... Fragaria ananassa (Strawberry). ]
MVLLDKLWDDIVAGPQPERGLGMLRKVPQPLNLKDEGESSKITMPTTPTTPVTPTTPISA
RKDNVWRSVFHPGSNLSSKTMGNQVFDSPQPNSPTVYDWMYSGETRSKHHR
```

Note that all sequences from the sequence object must be present in the Swiss-Prot data file, or this

command will fail (in the example above, all sequences from the 'test.seq' object must be present in the file 'sprot_test.dat'). On the other hand, the Swiss-Prot file can contain additional sequences. The sequences are searched by the sequence ID.

The usability of commands 'sprot_fetch' and 'seq_sprot_os' is limited if the Swiss-Prot file is large and/or the sequence object is large. For example, the UniProt release 12.4 (23-Oct-2007) contains 5,275,429 sequences, and the combined Swiss-Prot file is approx 13 Gb in size (swiss-prot + TrEMBL). The command 'seq_sprot_os' cannot cope well with the file of this size.

Consider for example a sequence object Omp85 with 1,000 sequences, which was derived in some way from the UniProt database (for example, the list of hits from the hidden Markov model search of the combined UniProt data file). To associate organism names with the Omp85 sequences, one would need to search the entire UniProt for each of the 1,000 sequences in the Omp85. The command 'seq_sprot_os' is not optimized to cope with such a large computational task. However, this task can be accomplished in two steps, by using the commands 'sprot_os' and 'seq_os'. First, the command 'sprot_os' can be used to extract the organism names from the UniProt file:

```
yabby> sprot_os uniprot.dat uniprot.dat.os

 Processing the database file 'uniprot.dat' ..
 Printing a dot per 10000 processed sequences:
 ..........................................................
 ..........................................................
 ..........................................................
 ..........................................................
 ..........................................................
 ..........................................................
 ..........................................................
 ..........................................................
 ..........................................

 All done.
```

In the above example, 'uniprot.dat' is the combined UniProt data file (swiss-prot + TrEMBL) in the Swiss-Prot format. The organism names are saved in the file 'uniprot.dat.os'. This file lists sequence IDs and the first OS field as given in the Swiss-Prot file, one line per sequence:

```
A0AQI4_9ARCH uncultured archaeon.
A0AQI5_9ARCH uncultured archaeon.
A0AQI7_9ARCH uncultured archaeon.
A0AQI8_9ARCH uncultured archaeon.
A0B530_METTP Methanosaeta thermophila (strain DSM 6194 / PT)
A0B531_METTP Methanosaeta thermophila (strain DSM 6194 / PT)
A0B532_METTP Methanosaeta thermophila (strain DSM 6194 / PT)
A0B533_METTP Methanosaeta thermophila (strain DSM 6194 / PT)
```

```
A0B534_METTP Methanosaeta thermophila (strain DSM 6194 / PT)
A0B535_METTP Methanosaeta thermophila (strain DSM 6194 / PT)
...
```

The command 'seq_os' (explained above) can be used to insert the organism names in the sequence object, by using the intermendiate file created by the command 'sprot_os':

```
yabby> seq_os uniprot.dat.os Omp85

 Reading the file 'uniprot.dat.os' ...
 1000 sequences to process.

 All done.
 [ seq_os: 'Omp85.seq' exists, overwritten ]
```

Similarly as with the command 'sprot_test.dat', the insertion of comments was in place, and the sequence object 'Omp85.seq' was overwritten.

2.2 Housekeeping commands

Several commands are dedicated to the general manipulation of Yabby objects. These include 'what' and 'print', and already seen 'help, 'dump', 'restore', 'delete', 'flush', 'license', 'exit' and 'quit'.

The 'help' command is probably the most useful. By itself it prints the list of currently available commands:

```
yabby> help

 39 commands ready.

 blast          blast_info     blastg         delete
 dump           emboss_needle  exit           flush
 help           hmm_score2seq  hmm_score_extract license
 mol2seq        mol_crd        mol_load       motif_cmp
 motif_load     motif_meme     pdb_conv       pdb_model
 pfam_fetch     print          quit           restore
 seq2id         seq_comment    seq_compl      seq_genbank
 seq_info       seq_letter     seq_letterc    seq_load
 seq_op         seq_pattern    seq_pick       seq_strip
 seq_unique     sprot_split    what

 For info about a particular command try: 'help COMMAND'
```

If followed by a command name, it prints the information about the command:

```
yabby> help seq_load


 Loads sequence(s) from the database file.

 Usage:
        seq_load [ options ] DBA_FILE OBJ_NAME

 Where DBA_FILE is the name of the database file. OBJ_NAME is
 the internal Yabby name for the sequence(s).

 Options:

 -a -- append sequences to an already existing sequence
  object OBJ_NAME
 -f -- the file format is FASTA (default)
 -b -- the file format is BLOCKS

 Notes:
 1. Only BLOCKS format as given by MEME output was tested.
```

The command 'delete' deletes an object, and the command 'flush' deletes all objects from the workspace:

```
yabby> what

    object(s)      type
  ------------------------------
    hits           seq
                   hmm_score

yabby> delete hits.seq

 [ 'hits.seq' deleted ]

yabby> flush

 [ workspace flushed ]

yabby> what

    object(s)      type
  ------------------------------
```

```
   [ none ]
```

The command 'dump' stores the session onto a file:

```
yabby> dump mysession

 Yabby session archived as 'mysession.tar.gz'
```

The session can be restored with the command 'restore':

```
yabby> restore mysession

 Yabby session 'mysession' restored
```

Note that the commands 'dump'/'restore' depend on unix commands 'tar' and 'gzip' being available for execution by Perl.

2.3 Working with sequence motifs

Sequence motifs are handled as 'motif' objects. Motifs could be loaded from FASTA files, or from BLOCKS files:

```
yabby> motif_load -b m2.blocks m2

 Reading the file 'm2.blocks' ..
 11 sequence(s) found in the motif 'm2'.
```

Motifs can be printed:

```
yabby> what

    object(s)      type
  -------------------------------
    m2             motif

yabby> print m2.motif

>Sthermophile [ from BLOCKS file 'm2.blocks' ]
PMAESGVTIRSDSEQYSRHEEQSVSPPSSSS
>Panserina [ from BLOCKS file 'm2.blocks' ]
PMAESGVTIRSDSEQYSSPEELSTSPPSSSS
```

```
>Cglobosum [ from BLOCKS file 'm2.blocks' ]
PMAESGITIPSDSEQYSRHGDQSTSPPSSSS
>Ncrassa [ from BLOCKS file 'm2.blocks' ]
PLAESGVTISSDSEQYSAPESASPQSPSSSS
>Oclavigerum [ from BLOCKS file 'm2.blocks' ]
PMAESGVTMHSDSEQYSGAEELSASLESSHS
>Afumigatus [ from BLOCKS file 'm2.blocks' ]
ELYDSGLTVPSDSEIYSANHEVSSPMSASSS
>Gzea [ from BLOCKS file 'm2.blocks' ]
HLAESGVTMHSDIELYSAGDDLSSPPSSNSP
>Aoryzae [ from BLOCKS file 'm2.blocks' ]
ELYESGLTVRSDSENYSANNELSESTSSSPL
>Tlanuginosus [ from BLOCKS file 'm2.blocks' ]
ELSDSALTVPSDSENYSANNEFSSSPSASNS
>Pnodorum [ from BLOCKS file 'm2.blocks' ]
PLTASGLTIPTDSESYSAPADSPSPSPPSSS
>Apullulans [ from BLOCKS file 'm2.blocks' ]
RDIYDSMTMQSDSETYDQPDISSPSSPSSDS
```

The command 'motif_cmp' compares two motifs. Two motifs are identical if they have the same number of sequences, the sequences have the same ID, and the sequences themselves are identical as strings:

```
yabby> motif_cmp m2 m2_second

 Motifs 'm2' and 'm2_second' contain the same sequence IDs.
 Comparing the sequences...
 Motifs 'm2' and 'm2_second' are identical.
```

The command 'motif_meme' extracts the motifs from the plain text output of the motif detection program MEME [2]. To extract the motif 1 from the file 'meme.out' use:

```
yabby> motif_meme meme.out 1 m1

 Reading MEME output 'meme.out' ..
 Motif 1 saved as 'm1'.

yabby> what

    object(s)      type
  ------------------------------
    m1             motif
```

In the internal representation 'motif' type object is identical to the 'seq' type object.

2.4 Working with the HMMER output

HMMER is a powerful software for the profile hidden Markov model search of sequence databases [3, 4]. The 'search step' in HMMER search pipeline is performed by the program HMMPFAM which is a part of the HMMER software suite. Yabby's commands 'hmm_score_extract' and 'hmm_score2seq' can aid in the extraction of the search results from the HMMPFAM output files.

As an example, consider the HMMPFAM search performed on the genome sequence 'LmjFmockup.pep'. The output of this search was redirected to the file 'hmmpfam.out'. To extract the hits with the E-value above 0.01, and store these hits under the name 'hits':

```
yabby> hmm_score_extract -E 0.01 -s hits hmmpfam.out

 Processing HMMPFAM search output file

  No     Sequence                                E-score
 ------------------------------------------------------
 (   1)  LmjF05.1190@All                         2.70e-03
 (   2)  LmjF05.1190@GlycogenStarch              4.20e-03
 (   3)  LmjF05.0920@All                         7.50e-03
yabby> what

    object(s)      type
  ------------------------------
    hits           hmm_score
```

The command 'hmm_score_extract' with the '-s hits' flag has created the object of the type 'hmm_score' in the workspace. This is merely a list containing the sequence name, the model the hit was recorded to, and the E-value:

```
LmjF05.1190@All 0.0027
LmjF05.1190@GlycogenStarch 0.0042
LmjF05.0920@All 0.0075
```

To get to the actual sequences, one needs to go to the original sequence database on which the HMMPFAM search was executed, match the sequence IDs, and extract the sequences. This can be achieved with the command 'hmm_score2seq'

```
yabby> hmm_score2seq LmjFmockup.pep hits.fas hits

 found 3 sequences to extract
 Processing the database 'LmjFmockup.pep'
 Sequences written to 'hits.fas'
```

This command takes the name of the sequence database on which the search was executed (LmjF-mockup.pep), the name of the output file where sequence are to be saved (hits.fas), and the name of the Yabby 'hmm_score' object where the hits were extracted.

The output file contains sequence hits in the FASTA format, which can be loaded and manipulated with sequence commands:

```
yabby> seq_load hits.fas  hits

 Reading the file 'hits.fas' ..
 3 sequence(s) found.

yabby> print hits.seq

>LmjF05.1190@All [ LmjF05.1190@All [ (score: 2.70e-03, model: All)
MSNKKQTWANAHSQLWTSSVARSAGSRTQSEVSSIASTNRDRSLLDDGDGYHQPPSHVCA
QELRHQAYQPNDKTLAQVSDILESYGAIACDADIPAVLLEVVKELERTKRDNNFKDLLLR
EYSDTVQRRFGLYGEESPPSIAEVSARLRDGAPARPVAAPNPAWLEQPLNELWRTVCDGM
QACFEKNADLSAPVLLPEARRTKANVSQLLHTSCDVLSQLAKEYTTAKAAVMKKMERTVA
ASQSFRQLTLSTALSELEAQQELKGSAAEDGGASAMDGSHTVGSALQSLIDVIPASLQIH
.... extra output deleted ....
```

2.5 Running NCBI BLAST from Yabby

BLAST (Basic Local Alignment Search Tool) refers to a heuristic algorithm widely used to find regions of similarity between protein/DNA sequences. BLAST is often used to denote the computer program, an actual implementation of the BLAST algorithm. Two such implementations are in wide use: NCBI BLAST and WU-BLAST (from the Washington University). In addition, the term BLAST is often used to denote the Web interface for blast search, such as the one provided by NCBI [5].

It is generally less known that NCBI BLAST executables can be downloaded [6] and run locally, which has some distinct advantages. The Yabby command 'blast' allows one to run NCBI BLAST search locally, through the Yabby interface.

The command 'blast' depends on NCBI BLAST being installed locally. The location of the NCBI BLAST installation is set in the Yabby library 'yabby_blast.pm':

```
$BLASTALL = "/usr/local/bin/blastall";
```

For NCBI BLAST to work one also needs to set up the file '.ncbirc' in the user's home directory that will give the location of blast data files (such as substitution matrices):

```
[NCBI]
   Data=/usr/local/blast/2.2.9/data
```

The database to be searched need first to be indexed for BLAST search with the program 'formatdb' which is provided in the NCBI BLAST package. For example, to format the database 'LmjFwholegenome.pep' (L.major genome in FASTA fomat),

```
$ /usr/local/bin/formatdb -i LmjFwholegenome.pep -p T -o T
```

This would create several additional files in the same directory, such as 'LmjFwholegenome.pep.phr', 'LmjFwholegenome.pep.psq' etc.

Yabby can execute BLAST search against multiple sequences. For example, to BLAST six sequences from the file 'cad.fas':

```
yabby> seq_load cad.fas cad

 Reading the file 'cad.fas' ..
 6 sequence(s) found.

yabby> blast -E 5.0 tmp/LmjFwholegenome_20050601_V5.2.pep cad

 6 sequence(s) found in the object 'cad'

 Now running BLAST ..
 BLASTing sequence 1 of 6 (Q45153_BACFI)
 Query sequence 'Q45153_BACFI'
 Database 'tmp/LmjFwholegenome_20050601_V5.2.pep'
 Found 2 hits above the threshold (E=5.00)
 The best hit: 'LmjF36.2210'
 E-score = 1.93e+00

 BLASTing sequence 2 of 6 (Q53650_STAAU)
 Query sequence 'Q53650_STAAU'
 Database 'tmp/LmjFwholegenome_20050601_V5.2.pep'
 Found 1 hits above the threshold (E=5.00)
 The best hit: 'LmjF04.0510'
 E-score = 3.57e+00

 BLASTing sequence 3 of 6 (Q97PJ0_STRPN)
 Query sequence 'Q97PJ0_STRPN'
 Database 'tmp/LmjFwholegenome_20050601_V5.2.pep'
 Found 1 hits above the threshold (E=5.00)
 The best hit: 'LmjF13.1210'
 E-score = 2.11e+00

 BLASTing sequence 4 of 6 (P95773_STALU)
```

```
 Query sequence 'P95773_STALU'
 Database 'tmp/LmjFwholegenome_20050601_V5.2.pep'
 No BLAST hits above the threshold (E=5.00) found.

 BLASTing sequence 5 of 6 (Q9JXN6_NEIMB)
 Query sequence 'Q9JXN6_NEIMB'
 Database 'tmp/LmjFwholegenome_20050601_V5.2.pep'
 No BLAST hits above the threshold (E=5.00) found.

 BLASTing sequence 6 of 6 (Q9CE92_LACLA)
 Query sequence 'Q9CE92_LACLA'
 Database 'tmp/LmjFwholegenome_20050601_V5.2.pep'
 No BLAST hits above the threshold (E=5.00) found.
```

The BLAST search was executed for each sequences in the 'cad' set in turn, creating six 'blast' objects:

```
yabby> what

    object(s)      type
  ------------------------------
    cad            seq
    cad_1          blast
    cad_2          blast
    cad_3          blast
    cad_4          blast
    cad_5          blast
    cad_6          blast
```

These can be examined later with the command 'blast_info':

```
yabby> blast_info cad_1

 Query sequence 'Q45153_BACFI'
 Database 'tmp/LmjFwholegenome_20050601_V5.2.pep'
 Found 2 hits above the threshold (E=5.00)
 The best hit: 'LmjF36.2210'
 E-score = 1.93e+00
```

2.6 EMBOSS 'needle' commands

The command 'emboss_needle' shows the sequences from the output of the program 'needle' from the EMBOSS suite of programs [7]. The input to the program 'needle' are two sequences, and the program

calculates the global sequence alignment (Needleman-Wunsch algorithm). Typically two files are provided as the input, each containing one sequence in FASTA format. If the second file contains multiple sequences, the global sequence alignment will be calculate between the first sequence and each sequence in turn from the second set. If the second set contains many sequences, the output file will be large, and it may be difficult to see the best matches. These can be extracted with the command 'emboss_needle':

```
yabby> emboss_needle needle.out 3

 Processing the file 'needle.out' ..

 (1) Q45153_BACFI:LmjF05.1170, Similarity: 26.6
 (2) Q45153_BACFI:LmjF05.1120, Similarity: 16.4
 (3) Q45153_BACFI:LmjF05.1040, Similarity: 15.0
```

The second argument is the number of top-scoring sequences to be printed out.

The purpose of the command 'emboss_needl2' is to find out what is the similarity between the two sets of sequences. Consider the following problem: given two sets of sequences, set A and set B, we need to find out how similar are the sequences in the set B to the sequences in the set A.

The command emboss_needl2 will take one sequence at a time from the query set A, and run the EMBOSS program 'needle' to find the similarity after the best alignment between this sequence and each sequence in the set B. emboss_needl2 will then extract the ID of the most similar sequence, as reported by 'needle', and it will move to the next query sequence from the query set A set until all query sequences are exausted.

Consider the following example: the set A is 'Ecoli057.fas' and the set B is 'Ecoli_lab.fas', both given as FASTA files. To find out the similarity of sequences in Ecoli057.fas to sequences in Ecoli_lab.fas, we first load the query set in yabby:

```
yabby> seq_load Ecoli057.fas Ecoli057

 Reading the file 'Ecoli057.fas' ..
 9 sequence(s) found.
```

The second set 'Ecoli_lab.fas' is required as FASTA file. Then:

```
yabby> emboss_needl2 Ecoli057 Ecoli_lab.fas sim_list

 'Ecoli057' contains 9 sequence(s)
 Working on sp|P36546|OMPX_ECOLI
Needleman-Wunsch global alignment.
 Working on tr|Q8X8K6|Q8X8K6
Needleman-Wunsch global alignment.
```

```
 Working on sp|P39170|UP05_ECOLI
Needleman-Wunsch global alignment.
 Working on tr|Q8XDF1|Q8XDF1
Needleman-Wunsch global alignment.
 Working on sp|P58603|OMPT_ECO57
Needleman-Wunsch global alignment.
 Working on tr|Q8XAS0|Q8XAS0
Needleman-Wunsch global alignment.
 Working on tr|Q8X592|Q8X592
Needleman-Wunsch global alignment.
 Working on tr|Q8X7N5|Q8X7N5
Needleman-Wunsch global alignment.
 Working on tr|Q8X8F2|Q8X8F2
Needleman-Wunsch global alignment.
yabby>
```

The argument to the command 'emboss_needl2' are the query sequence object, the FASTA file of the database set, and the name of the object to be created which contains the information about similarities This is 'needl2' object:

```
yabby> what

    object(s)      type
  ------------------------------
    Ecoli057       seq
    sim_list       needl2
```

One can print the 'needl2' object:

```
yabby> print sim_list.needl2

OMPX_ECOLI OMPX_ECOLI 100.0
Q8X8K6 OMPG_ECOLI 100.0
UP05_ECOLI UP05_ECOLI 100.0
Q8XDF1 OMPF_ECOLI 99.4
OMPT_ECO57 OMPT_ECOLI 99.7
Q8XAS0 NMPC_ECOLI 80.3
Q8X592 MIPA_ECOLI 99.6
Q8X7N5 PHOE_ECOLI 99.7
Q8X8F2 YFEN_ECOLI 32.1
```

Or sorted by percent similarity:

```
yabby> print -s sim_list.needl2

Q8X8F2 YFEN_ECOLI 32.1
Q8XAS0 NMPC_ECOLI 80.3
Q8XDF1 OMPF_ECOLI 99.4
Q8X592 MIPA_ECOLI 99.6
OMPT_ECO57 OMPT_ECOLI 99.7
Q8X7N5 PHOE_ECOLI 99.7
OMPX_ECOLI OMPX_ECOLI 100.0
Q8X8K6 OMPG_ECOLI 100.0
UP05_ECOLI UP05_ECOLI 100.0
```

The above table lists two sequence IDs (ID_query and ID_dba) and the percent similarity as reported by the program 'needle' after the best global alignment. The above table contains nine rows, since the query set ('Ecoli057.fas') contains nine sequences. Each line refers to one sequence from the 'Ecoli057.fas', and given are the best matching sequence from the second set ('Ecoli_lab.fas'), and percent similarity. We see that the sequence 'Q8X8F2' from 'Ecoli057.fas' has the least similarity to any sequence from the 'Ecoli_lab.fas' set.

The table produced by 'emboss_needl2' can be written to a file:

```
yabby> print -f needl2.out -s sim_list.needl2

 'sim_list.needl2' written to the file 'needl2.out'
```

2.7 Stand-alone commands

Stand-alone commands do not interact with other commands. They are, in effect, independent program scripts. Nevertheless it is often useful to collect such independent scripts under Yabby to simplify their maintenance and ensure long-term retention. An example of such command is 'pdb_model'.

Another example of a stand-alone command is 'pdb_model', which splits the Protein Data Bank (PDB) file with multiple models. The output is a series of PDB files, one per model:

```
yabby> pdb_model 2C7H.pdb RBP_

 working on model  1 (creating 'RBP_1.pdb')
 working on model  2 (creating 'RBP_2.pdb')
 working on model  3 (creating 'RBP_3.pdb')
 working on model  4 (creating 'RBP_4.pdb')
 working on model  5 (creating 'RBP_5.pdb')
.... extra output deleted ....
```

where '2C7H.pdb' is the original PDB file with multiple models, and 'RBP_' is the prefix used for individual model files.

2.8 Extending Yabby

The one-to-one correspondence between Yabby commands and Yabby command scripts makes it very clear from where the particular funcionality comes from. This also encapsulates a functionality within a particular script file, and small size of individual script files makes it easy to understand, modify, and maintain such scripts. The complete decoupling between command scripts ensures that it is hard to break existing commands when adding new commands.

Any functional Perl script placed in the Yabby library directory will become available for execution as Yabby command. Consider the file 'hello.pl' with the following content:

```
print "Hello World!\n";
```

If placed in the the Yabby library directory, it becomes immediately available as the command 'hello':

```
$ yabby

 - YABBY version 0.1 -

 [ 36 command(s) ready ]

yabby> hello

Hello World!

yabby>
```

If already running, Yabby will need to be restarted before the command 'hello' becomes available.

2.8.1 Creating a new Yabby command: seq_letter

Although a simple perl script would work if just dropped in the Yabby library, to create a Yabby command would require some additional effort. Consider creating a very simple command which operates on the sequence object, and prints the first letter of the first sequence stored in the sequence object. The command will be named 'seq_letter'.

We start by creating the file 'seq_letter.pl' in the Yabby library:

```
# seq_letter.pl
```

```
use yabby_sys;
use yabby_seq;

use Getopt::Std;
```

All Yabby commands rely on the module 'yabby_sys'. The module 'yabby_seq' contains functions for the manipulation of sequences. The Perl module 'Getopt::Std' is required to enable processing of single-character command switches.

The next step is to create the usage string:

```
$USAGE = "
 Prints the first letter from the first sequence of a sequence
 object.

 Usage:
        seq_letter OBJ_NAME

 where OBJ_NAME is the name of an existing sequence object.

 Notes:

 1. This command is merely an example to demonstrate how new
 Yabby commands can be developed.
";
```

The usage string will be printed when 'help seq_letter' or 'seq_letter help' is issued.

We add the initialization commands:

```
# options
# initialization
@argl = sys_init( @ARGV );
check_call( @argl, [ 1 ] );
$obj_name = $argl[0];
```

In the case of this command there are no options. The function 'sys_init' is used to process arguments to all Yabby commands in a standard way. The function 'check_call' is used to check the number of arguments. In this case, exactly one argument must be given, the name of an existing sequence object. Finally, in the last line the name of sequence object is associated to the variable 'obj_name'. In the next step the sequence object is loaded in the memory:

```
requirements( $obj_name, $SEQUENCE );
$xmldoc = load_ip_xml( $obj_name, $SEQUENCE );
```

The function 'requirements()' checks whether the sequence object named 'obj_name' exists. If not, it will terminate the script with an appropriate error message. If this step is passed, the function 'load_ip_xml()' loads the sequence object from the workspace. Finally, we fetch the first sequence and print its first letter:

```
$seq_hash = xml2seq( $xmldoc );
$seq_item = $seq_hash->{1};
printf " The first letter of the first sequence is: '%s'\n",
  substr( $seq_item->{$DBA_SEQUENCE}, 0, 1 );
```

The complete listing of the modified command script 'seq_letter' is given below:

```
# seq_letter.pl

use yabby_sys;
use yabby_seq;

use Getopt::Std;

$USAGE = "
 Prints the first letter from the first sequence of a sequence
 object.

 Usage:
        seq_letter OBJ_NAME

 where OBJ_NAME is the name of an existing sequence object.

 Notes:

 1. This command is merely an example to demonstrate how new
 Yabby commands can be developed.
";

# options
# initialization
@argl = sys_init( @ARGV );
check_call( @argl, [ 1 ] );
$obj_name = $argl[0];

# requirements
requirements( $obj_name, $SEQUENCE );
$xmldoc = load_ip_xml( $obj_name, $SEQUENCE );

# body
```

```
$seq_hash = xml2seq( $xmldoc );
$seq_item = $seq_hash->{1};
printf " The first letter of the first sequence is: '%s'\n",
  substr( $seq_item->{$DBA_SEQUENCE}, 0, 1 );
```

When executed, the command gives the following output:

```
yabby> seq_load tom20.fas tom20

 Reading the file 'tom20.fas' ..
 3 sequence(s) found.

yabby> seq_letter tom20

 The first letter of the first sequence is: 'M'
```

Consider what happens when no arguments are given:

```
yabby> seq_letter

 seq_letter:: ERROR: exactly 1 argument(s) required [ 0 supplied ]
 [ command 'seq_letter' failed ]
```

This error was raised by the function 'check_call'. If the sequence object does not exist:

```
yabby> seq_letter ttt

 seq_letter:: missing requirement: 'ttt.seq'

 [ command 'seq_letter' failed ]
```

The last error was raised by the function 'requirements()'.

2.8.2 Extending seq_letter to work on all sequences

It would be easy to extend the command seq_letter to print the first letter of all sequences from the sequence object. This would require modifying the body of the command:

```
# body
$seq_hash = xml2seq( $xmldoc );
$keys = get_seq_keys( $seq_hash );
```

```
printf " '%s' contains %d sequence(s)\n", $obj_name, $#{$keys}+1;

for $key ( @$keys ) {
  $seq_item = $seq_hash->{$key};
  printf " Sequence '%s', the first letter is '%s'\n",
  $seq_item->{$DBA_SEQID}, substr( $seq_item->{$DBA_SEQUENCE}, 0, 1 );
}
```

The output of 'seq_letter' is:

```
yabby> seq_letter tom20

 'tom20' contains 3 sequence(s)
 Sequence 'A.thaliana2', the first letter: 'M'
 Sequence 'O.sativa', the first letter: 'M'
 Sequence 'L.esculentum', the first letter: 'M'
```

2.8.3 Adding a command switch

Consider how one could add an optional switch to print the last letter of each sequence, instead of the first letter.

To add the switch '-t':

```
# options
getopts('t');

if ( defined( $opt_t ) ) {
  $opt_t_flag = 1;
} else {
  $opt_t_flag = 0;
}
```

To extend the command body to take into account possible '-t' switch:

```
$seq_hash = xml2seq( $xmldoc );
$keys = get_seq_keys( $seq_hash );
printf " '%s' contains %d sequence(s)\n", $obj_name, $#{$keys}+1;

for $key ( @$keys ) {
  $seq_item = $seq_hash->{$key};
  $seq_id = $seq_item->{$DBA_SEQID};
  $sequence = $seq_item->{$DBA_SEQUENCE};
```

```
  if ( $opt_t_flag ) {
    $lett = substr( $seq_item->{$DBA_SEQUENCE}, -1, 1 );
  } else {
    $lett = substr( $seq_item->{$DBA_SEQUENCE}, 0, 1 );
  }

  printf " Sequence '%s', the letter is '%s'\n", $seq_id, $lett;
}
```

The possible use of the modified 'seq_letter':

```
yabby> seq_letter tom20

 'tom20' contains 3 sequence(s)
 Sequence 'A.thaliana2', the letter is 'M'
 Sequence 'O.sativa', the letter is 'M'
 Sequence 'L.esculentum', the letter is 'M'

yabby> seq_letter -t tom20

 'tom20' contains 3 sequence(s)
 Sequence 'A.thaliana2', the letter is 'R'
 Sequence 'O.sativa', the letter is 'R'
 Sequence 'L.esculentum', the letter is 'R'
```

The full listing of the command 'seq_letter' is:

```
# seq_letter.pl

use yabby_sys;
use yabby_seq;

use Getopt::Std;

$USAGE = "
 Prints the first sequence letter from the sequence object.

 Usage:
        seq_letter [ options ] OBJ_NAME

 where OBJ_NAME is the name of an existing sequence object.
```

```
 Options:

 -t -- print the last letter instead of the first

 Notes:

 1. This command is merely an example to demonstrate how new
 Yabby commands can be developed.
";

# options
getopts('t');

if ( defined( $opt_t ) ) {
  $opt_t_flag = 1;
} else {
  $opt_t_flag = 0;
}

# initialization
@argl = sys_init( @ARGV );
check_call( @argl, [ 1 ] );
$obj_name = $argl[0];

# requirements
requirements( $obj_name, $SEQUENCE );
$xmldoc = load_ip_xml( $obj_name, $SEQUENCE );

# body
$seq_hash = xml2seq( $xmldoc );
$keys = get_seq_keys( $seq_hash );

printf " '%s' contains %d sequence(s)\n", $obj_name, $#{$keys}+1;

for $key ( @$keys ) {

  $seq_item = $seq_hash->{$key};
  $seq_id = $seq_item->{$DBA_SEQID};
  $sequence = $seq_item->{$DBA_SEQUENCE};

  if ( $opt_t_flag ) {
    $lett = substr( $seq_item->{$DBA_SEQUENCE}, -1, 1 );
  } else {
    $lett = substr( $seq_item->{$DBA_SEQUENCE}, 0, 1 );
```

```
  }

  printf " Sequence '%s', the letter is '%s'\n", $seq_id, $lett;
}
```

2.8.4 A command that creates an object

The command 'seq_letter' is very simple in that it merely prints something on the output screen; it does not modify an existing object nor does it creates any new Yabby objects. Consider a command that actually creates a sequence object. An example for this may be a command that chops the first letter from each sequence in the sequence object, and saves the result as a new sequence object. This requires two names to be given on the input, the name of an existing sequence object, and the name of the sequence object that will be created:

```
seq_letterc OBJ_NAME OBJ_NAME_NEW
```

The new command is named 'seq_letterc' to distinguish it from the previous 'seq_letter'.

To build the command 'seq_letterc' we create the file 'seq_letterc.pl' in the Yabby library:

```
# seq_letters.pl

use yabby_sys;
use yabby_seq;

use Getopt::Std;

$USAGE = "
 Chops the first sequence letter from the sequence object, and
 saves the result as a new sequence object.

 Usage:
        seq_letters OBJ_NAME OBJ_NAME_NEW

 where OBJ_NAME is the name of an existing sequence object.

 Notes:

 1. This command is merely an example to demonstrate how new
 Yabby commands can be developed.
";
```

Two arguments are now required:

```
# options
# initialization
@argl = sys_init( @ARGV );
check_call( @argl, [ 2 ] );
$obj_name = $argl[0];
$obj_name2 = $argl[1];
```

We can use the same construct to check for requirements and fetch the sequence object:

```
# requirements
requirements( $obj_name, $SEQUENCE );
$xmldoc = load_ip_xml( $obj_name, $SEQUENCE );

# body
$seq_hash = xml2seq( $xmldoc );
$keys = get_seq_keys( $seq_hash );

printf " '%s' contains %d sequence(s)\n", $obj_name, $#{$keys}+1;
```

The new command requires a new sequence object to be created, and populated with sequences from the existing sequence object (minus the first letter). We first initialize a new sequence object:

```
$seq_hash2 = {};
```

The same construct as in 'seq_letter' could be used to loop through the sequences. Now, we also create a new sequence for each existing sequence, chop the first letter, and joing the result to the new sequence object:

```
for $key ( @$keys ) {

  $seq_item = $seq_hash->{$key};

  $seq_item_new = {};
  $seq_item_new->{$DBA_SEQID} = $seq_item->{$DBA_SEQID};
  $seq_item_new->{$DBA_COMMENT} = $seq_item->{$DBA_COMMENT};
  $seq_item_new->{$DBA_SEQUENCE} = $seq_item->{$DBA_SEQUENCE};

  # chop the first letter from the sequence
  substr( $seq_item_new->{$DBA_SEQUENCE}, 0, 1 ) = "";

  $seq_hash2->{$key} = $seq_item_new;
}
```

We need to save the newly created sequence hash 'seq_hash2' as the new sequence object. In the first step, we convert the sequence hash object into an XML document:

```
$xmldoc = seq2xml( $seq_hash2 );
```

This is the reverse of the above command 'xml2seq()', which converts an XML document representing a sequence object into a sequence hash, the form all commands employ to represent the sequence object in memory. The functions 'xml2seq()' and 'seq2xml()' are specific for sequence objects, and are defined in the package 'yabby_seq.pm'. Finally, we store the XML sequence object in the workspace:

```
printf " Saving chopped sequences as '%s'\n", $obj_name2;
save_ip_xml( $xmldoc, $obj_name2, $SEQUENCE, $WARN_OVERW );
```

The function 'save_ip_xml()' sends the XML representation of an object to the workspace. This function is the reverse analog of the previously used function 'load_ip_xml()', which fetches the XML representation of an object from the workspace. These two functions are defined in the package 'yabby_sys.pm'.

The arguments to the function 'save_ip_xml()' are are follows: the XML representation of an object ('xmldoc'), the name of the object to be created ('obj_name2'), the type of the object to be created ('SEQUENCE' in this case), and the flag which determines whether the warning will be issued is the object with the same name and the same type already exists in the workspace and will be overwritten ('WARN_OVERW').

The command 'seq_letterc' could be used as follows:

```
yabby> seq_load tom20.fas tom20

 Reading the file 'tom20.fas' ..
 3 sequence(s) found.

yabby> what

    object(s)      type
  ------------------------------
    tom20          seq

yabby> seq_letterc tom20 tom20_chop

 'tom20' contains 3 sequence(s)
 Saving chopped sequences as 'tom20_chop'

yabby> what

    object(s)      type
```

```
  ------------------------------
    tom20          seq
    tom20_chop     seq

yabby> print tom20.seq

>A.thaliana2 [ A.thaliana2 ]
MEFSTADFERFIMFEHARKNSEAQYKNDPLDSENLLKWGGALLELSQFQPIPEAKLMLND
AISKLEEALTINPGKHQALWCIANAYTAHAFYVHDPEEAKEHFDKATEYFQRAENEDPGN
DTYRKSLDSSLKAPELHMQFMNQGMGQQILGGGGGGGGGGMASSNVSQSSKKKKRNTEFT
YDVCGWIILACGIVAWVGMAKSLGPPPPAR
>O.sativa [ O.sativa ]
MDMGAMSDPERMFFFDLACQNAKVTYEQNPHDADNLARWGGALLELSQMRNGPESLKCLE
DAESKLEEALKIDPMKADALWCLGNAQTSHGFFTSDTVKANEFFEKATQCFQKAVDVEPA
NDLYRKSLDLSSKAPELHMEIHRQMASQASQAASSTSNTRQSRKKKKDSDFWYDVFGWVV
LGVGMVVWVGLAKSNAPPQAPR
>L.esculentum [ L.esculentum ]
MDMQSDFDRLLFFEHARKTAETTYATDPLDAENLTRWAGALLELSQFQSVSESKKMISDA
ISKLEEALEVNPQKHDAIWCLGNAYTSHGFLNPDEDEAKIFFDKAAQCFQQAVDADPENE
LYQKSFEVSSKTSELHAQIHKQGPLQQAMGPGPSTTTSSTKGAKKKSSDLKYDVFGWVIL
AVGLVAWIGFAKSNMPXPAHPLPR

yabby> print tom20_chop.seq

>A.thaliana2 [ A.thaliana2 ]
EFSTADFERFIMFEHARKNSEAQYKNDPLDSENLLKWGGALLELSQFQPIPEAKLMLNDA
ISKLEEALTINPGKHQALWCIANAYTAHAFYVHDPEEAKEHFDKATEYFQRAENEDPGND
TYRKSLDSSLKAPELHMQFMNQGMGQQILGGGGGGGGGGMASSNVSQSSKKKKRNTEFTY
DVCGWIILACGIVAWVGMAKSLGPPPPAR
>O.sativa [ O.sativa ]
DMGAMSDPERMFFFDLACQNAKVTYEQNPHDADNLARWGGALLELSQMRNGPESLKCLED
AESKLEEALKIDPMKADALWCLGNAQTSHGFFTSDTVKANEFFEKATQCFQKAVDVEPAN
DLYRKSLDLSSKAPELHMEIHRQMASQASQAASSTSNTRQSRKKKKDSDFWYDVFGWVVL
GVGMVVWVGLAKSNAPPQAPR
>L.esculentum [ L.esculentum ]
DMQSDFDRLLFFEHARKTAETTYATDPLDAENLTRWAGALLELSQFQSVSESKKMISDAI
SKLEEALEVNPQKHDAIWCLGNAYTSHGFLNPDEDEAKIFFDKAAQCFQQAVDADPENEL
YQKSFEVSSKTSELHAQIHKQGPLQQAMGPGPSTTTSSTKGAKKKSSDLKYDVFGWVILA
VGLVAWIGFAKSNMPXPAHPLPR
```

Since no safeguards were put against overwriting an existing sequence object, the command 'seq_letterc' could be used twice to chop sequences from the N-terminus in place,

```
yabby> seq_letterc tom20 tom20
```

```
 'tom20' contains 3 sequence(s)
 Saving chopped sequences as 'tom20'
 [ seq_letterc: 'tom20.seq' exists, overwritten ]

yabby> seq_letterc tom20 tom20

 'tom20' contains 3 sequence(s)
 Saving chopped sequences as 'tom20'
 [ seq_letterc: 'tom20.seq' exists, overwritten ]
```

2.8.5 Creating a new object type

Yabby supports two internal representations of data in the workspace: simple lists and XML objects. To create a new object type one must first decide whether the data in question can be effectively represented with a two-dimensional table. Two-dimensional tables can be effectively represented with two-dimensional lists, where each item in the list corresponds to the table row. Lists are simple and efficient in Perl, and if suitable this is the preferred representation of data. However, XML format is much more powerful and flexible, and is the preferred solution when the data has a complex internal structure.

A sequence object uses XML to store data into the workspace. A Yabby commands store sequence objects in memory in the form of a hash (sequence hash). All sequence commands use the standard function 'load_ip_xml()' to fetch the XML representation of a sequence object from the workspace. Subsequently the object is converted into a more efficient structure, a sequence hash, with the function 'xml2seq()'. After the sequences are manipulated, or a new sequence object is created, the result may be stored in the workspace. In this case, the reverse transformation process takes place: the sequence hash is convereted into an XML object with the function 'seq2xml()', and the resulting XML object is saved in the workspace with the function 'save_ip_xml()'.

The functions 'load_ip_xml()' and 'save_ip_xml()' can be used to load/store the XML representation of any object in the workspace. The analogous functions for objects represented as a two-dimensional lists are 'load_ip()' and 'save_ip()'.

As an example we will create a new object type named "seqid" to represent the list of sequence IDs. To demonstrate the use of this object we will also create the command 'seq2id' which creates the "seqid" object from an existing sequence object.

The first step is to create the script 'seq2id.pl' in the Yabby library, and we start as previously:

```
# seq2id.pl

use yabby_sys;
use yabby_seq;

use Getopt::Std;
```

```
$USAGE = "
 Converts a sequence object into seqid object.

 Usage:
        seq2id OBJ_NAME

 where OBJ_NAME is the name of an existing sequence object.

 Notes:

 1. This command is merely an example to demonstrate how a
 new object type can be implemented.
";

# options
# initialization
@argl = sys_init( @ARGV );
check_call( @argl, [ 1 ] );
$obj_name = $argl[0];

# requirements
requirements( $obj_name, $SEQUENCE );
$xmldoc = load_ip_xml( $obj_name, $SEQUENCE );

# body
$seq_hash = xml2seq( $xmldoc );
$keys = get_seq_keys( $seq_hash );
```

With this we have the sequence hash in memory. In 'yabby_seq.pm' we add the name of the "seqid" object:

```
$SEQUENCE = "seq";
$MOTIF = "motif";
$SEQID = "seqid";
```

The new object will be represented as a one-dimensional list, where each sequence ID string will be one element of this list. This is equivalent to a two-dimensional table, where there is only one column in the table. Next we create the variable 'seqid_obj' to hold the 'seqid' object, fill this object with the values from the sequence hash, and store the object in the workspace by using the standard function:

```
$seqid_obj = [];
```

```
for $key ( @$keys ) {

  $seq_item = $seq_hash->{$key};
  $seq_id = $seq_item->{$DBA_SEQID};

  push @$seqid_obj, [ $seq_id ];
}

print " Saving '$obj_name.$SEQID'\n";
save_ip( $seqid_obj, $obj_name, $SEQID, $WARN_OVERW );
```

The complete listing for the command script 'seq2id':

```
# seq2id.pl

use yabby_sys;
use yabby_seq;

use Getopt::Std;

$USAGE = "
 Converts a sequence object into seqid object.

 Usage:
        seq2id OBJ_NAME

 where OBJ_NAME is the name of an existing sequence object.

 Notes:

 1. This command is merely an example to demonstrate how a
 new object type can be implemented.
";

# options
# initialization
@argl = sys_init( @ARGV );
check_call( @argl, [ 1 ] );
$obj_name = $argl[0];

# requirements
requirements( $obj_name, $SEQUENCE );
$xmldoc = load_ip_xml( $obj_name, $SEQUENCE );
```

```
# body
$seq_hash = xml2seq( $xmldoc );
$keys = get_seq_keys( $seq_hash );

printf " '%s' contains %d sequence(s)\n", $obj_name, $#{$keys}+1;

$seqid_obj = [];

for $key ( @$keys ) {

  $seq_item = $seq_hash->{$key};
  $seq_id = $seq_item->{$DBA_SEQID};

  push @$seqid_obj, [ $seq_id ];
}

print " Saving '$obj_name.$SEQID'\n";
save_ip( $seqid_obj, $obj_name, $SEQID, $WARN_OVERW );
```

And below is the output when the command is executed on an example sequence object:

```
yabby> seq_load tom20.fas tom20

 Reading the file 'tom20.fas' ..
 3 sequence(s) found.

yabby> seq2id tom20

 'tom20' contains 3 sequence(s)
 Saving 'tom20.seqid'

yabby> what

    object(s)      type
  ------------------------------
    tom20          seq
                   seqid
```

Of course, at this point nothing can be done with the 'seqid' object. It cannot be even printed on the screen:

```
yabby> print tom20.seqid
```

```
print:: ERROR: printing the property 'seqid' not yet implemented
[ command 'print' failed ]
```

This is because the printing of each particular object type has to be enabled in the 'print' command. However, one can view the 'seqid' object directly from the filesystem, since 'tom20.seqid' is just a simple text file, containing sequence IDs listed one per row:

```
yabby> cat .yabby/tom20.seqid
A.thaliana2
O.sativa
L.esculentum
```

2.8.6 Some additional explanations and programming conventions

All objects that are represents with two dimensional tables are initialized as annonymous arrays:

```
$seqid_obj = [];
```

When filled with values, each element of this array is a reference to anonymous array. In this case containing only one element 'seq_id':

```
push @$seqid_obj, [ $seq_id ];
```

This format is very flexible. If the data table needs to contain more than one element per row, these would be simply added to the anonymous reference representing a table row:

```
push @$seqid_obj, [ $seq_id, $colum2_elem, $colum3_elem, ... ];
```

Consider how the object 'mol' is built, which is internally represented with a two-dimensional table:

```
push @$mol, [ $resi_num, $resi_name, $atom_name, $segmentID ];
```

The function 'save_ip()' expects the reference to an anonymous array, where each element of this array is a reference to another anonymous array which may contain one or more elements. This holds for any object represented by a list, and functions 'save_ip()' and 'load_ip()' are used to store/fetch the object from the workspace. For objects represented in XML, a special converter function needs to be written akin to 'xml2seq()' and 'seq2xml()'.

Command reference

3.1 General commands

General commands are related to the basic Yabby functions, such as control of the workspace, and are not related to any specific area of application.

3.1.1 `delete`

Deletes objects from the workspace.

Usage:

```
delete OBJECT.PROPERTY
```

Options:

None

Notes:

1. If PROPERTY equals '*' (no quotes) all properties of OBJECT will be deleted.

Examples:

1. `yabby> delete sp.seq`

   `[ 'sp.seq' deleted ]`

Requirements: *None*

Creates objects: *None*

System scripts: *None*

3.1.2 dump

Dumps the current workspace to a file, which later can be restored with 'restore'.

Usage:

```
dump SESSION_NAME
```

Options:

None

Notes:

1. This command saves the current Yabby session in the file SESSION_NAME.tar.gz in the current directory.
2. Currently works only on system that have GNU tar command.

Examples:

1.
```
yabby> dump tmpsession

 Yabby session archived as 'tmpsession.tar.gz'
```

Requirements: *None*

Creates objects: *None*

System scripts: *None*

3.1.3 flush

Deletes everything from the workspace.

Usage:

```
flush
```

Options:

None

Notes:

None

Examples:

1.
```
yabby> flush

 [ workspace flushed ]
```

Requirements: *None*

Creates objects: *None*

System scripts: *None*

3.1.4 `license`

Prints the Yabby license.

Usage:

```
license
```

Options:

None

Notes:

None

Examples:

1. `yabby> license`

```
                    GNU GENERAL PUBLIC LICENSE
                       Version 2, June 1991

 Copyright (C) 1989, 1991 Free Software Foundation, Inc.
                          675 Mass Ave, Cambridge, MA 02139, USA
 Everyone is permitted to copy and distribute verbatim copies
 of this license document, but changing it is not allowed.

                             Preamble

  The licenses for most software are designed to take away your
freedom to share and change it.  By contrast, the GNU General Public
....
```

Requirements: *None*

Creates objects: *None*

System scripts: *None*

3.1.5 help

Prints help pages.

Usage:

```
help [ COMMAND ]
```

Options:

None

Notes:

1. If no argument is given the list of all available commands will be printed. If a command name is given the help about a particular command will be printed.

Examples:

1.
```
yabby> help quit

 Exits Yabby
```

Requirements: *None*

Creates objects: *None*

System scripts: *None*

3.1.6 print

Prints an object on the terminal screen or to a file.

Usage:

```
print [ options ] OBJ_NAME.PROPERTY
```

Options (general):

-f FILE_NAME – print to a file instead on the terminal screen.

Options (seq objects only):

-l – print the sequence as the three-letter code.

-c – print the sequences as the CSV table.

-n N – print N residues per line (both when printing one- and three-letter codes).

-t N – truncate each sequence at N letters when writing

Notes:

1. Currently supported objects for printing are 'seq', 'motif', and 'blastg'.

Examples:

1. Print the sequence object tom20.seq

```
yabby> print tom20.seq

>A.thaliana [ A.thaliana ]
MDTETEFDRILLFEQIRQDAENTYKSNPLDADNLTRWGGVLLELSQFHSISDAKQMIQEA
ITKFEEALLIDPKKDEAVWCIGNAYTSFAFLTPDETEAKHNFDLATQFFQQAVDEQPDNT
HYLKSLEMTAKAPQLHAEAYKQGLGSQPMGRVEAPAPPSSKAVKNKKSSDAKYDAMGWVI
LAIGVVAWISFAKANVPVSPPR
>O.sativa [ O.sativa ]
MDMGAMSDPERMFFFDLACQNAKVTYEQNPHDADNLARWGGALLELSQMRNGPESLKCLE
DAESKLEEALKIDPMKADALWCLGNAQTSHGFFTSDTVKANEFFEKATQCFQKAVDVEPA
NDLYRKSLDLSSKAPELHMEIHRQMASQASQAASSTSNTRQSRKKKKDSDFWYDVFGWVV
LGVGMVVWVGLAKSNAPPQAPR
>L.esculentum [ L.esculentum ]
MDMQSDFDRLLFFEHARKTAETTYATDPLDAENLTRWAGALLELSQFQSVSESKKMISDA
ISKLEEALEVNPQKHDAIWCLGNAYTSHGFLNPDEDEAKIFFDKAAQCFQQAVDADPENE
LYQKSFEVSSKTSELHAQIHKQGPLQQAMGPGPSTTTSSTKGAKKKSSDLKYDVFGWVIL
AVGLVAWIGFAKSNMPXPAHPLPR
>S.tuberosum [ S.tuberosum ]
MEMQSEFDRLLFFEHARKSAETTYAQNPLDADNLTRWGGALLELSQFQPVAESKQMISDA
TSKLEEALTVNPEKHDALWCLGNAHTSHVFLTPDMDEAKVYFEKATQCFQQAFDADPSND
LYRKSLEVTAKAPELHMEIHRHGPMQQTMAAEPSTSTSTKSSKKTKSSDLKYDIFGWVIL
AVGIVAWVGFAKSNMPPPPPPPPQ
>P.taeda [ P.taeda ]
MEEMAIPQSDFDRLLFFEGARKAAENTYAVNPEDADNLTRWGGALLELSQFQQGPDCVKM
VKDAVSKLEEALKISPNKHDTLWCLGNAHTSHAFLIPEHEVAKIYFKMASKYFQQAVEQD
PTNELYRKSLELTEKAPELHLEVHKQILNPQSVAAGSSTVSNLKGSRKKKSSDLKYDIMG
WIVLAVGIAAWVGMAKSHVPPPPML
>G.max [ G.max ]
MDLQQSEFDRLLFFEHARKAAEAEYEKNPLDADNLTRWGGALLELSQFQSFPESKKMTQE
AVSKLEEALAVNPKKHDTLWCLGNAHTSQAFLIPDQEEAKVYFDKAAVYFQQAVDEDPSN
ELYRKSLEVAAKAPELHVEIHKQGFGQQQQAAATAGSSTSASTNTQKKKKSSDLKYDIFG
WIILAVGIVAWVGFAKSNLPPPPPPPPR
>Z.mays [ Z.mays ]
MEMGGMSDAERLFFFEMACKNSEVAYEQNPNDADNLTRWGGALLELSQVRTGPDSLKLLE
DAEAKLEEALQIDPNKSDALWCLGNAQTSHGFFTPDNAIANEFFTKATGCFQKAVDVEPA
NELYRKSLDLSMKAPELHLEIQRQMVSQAATQASSASNPRQSRKKKDNDFWYDVCGWVIL
GAGIVAWVGLARASMPPPTPPAR
```

2. Print the sequence object tom20.seq to a file 'tom20.txt'

```
yabby> print -f tom20.txt tom20.seq
```

```
'tom20.seq' written to the file 'tom20.txt'
```

Requirements: The object to be printed must exist in the workspace.

Creates objects: *None*

System scripts: _print_seq.pl, _print_motif.pl, _print_blastg.pl

3.1.7 restore

Restores Yabby session saved with the command 'dump'

Usage:

```
restore SESSION_NAME
```

Where SESSION_NAME is the archive name used when dumping the session.

Options:

None

Notes:

1. The session will be restored from a file named SESSION_NAME.tar.gz.
2. Relies on GNU tar and gzip commands. These must be in the executable path.

Examples:

1.
```
yabby> restore tmpsession

 Yabby session 'tmpsession' restored
```

Requirements: *None*

Creates objects: *None*

System scripts: *None*

3.1.8 what

Shows objects currently in the workspace.

Usage:

```
what
```

Options:

None

Notes:

None

Examples:

1. `yabby> what`

```
  objects        properties
-------------------------------
  tom20          seq
```

Requirements: *None*

Creates objects: *None*

System scripts: *None*

3.2 Sequence commands

3.2.1 seq_comment

Modifies sequence comments to add a unique number.

Usage:

`seq_comment OBJ_NAME`

where OBJ_NAME is the name of an existing sequence object.

The sequence object with modified comments overwrites OBJ_NAME. The comment for each sequence is modified to prepend N- where N is the order of the sequence. For example, if the sequence comment was Q9JXN6, and was first in the sequence object, its comment will be modified to 1-Q9JXN6.

Options: *None*

Notes: *None*

Examples:

1. `yabby> seq_comment cad3`

```
Comments modified in 3 sequence(s).
```

Requirements: seq

Creates objects: *None*

System scripts: *None*

3.2.2 seq_compl

Calculates sequence complement (for DNA sequences).

Usage:

`seq_compl [ options ] OBJ_NAME OBJ_NAME_NEW`

Where OBJ_NAME is the name of an existing sequence object, the complement sequence will be saved under the name OBJ_NAME_NEW. The sequence OBJ_NAME must be a DNA sequence.

Options:

-r – Calculate the reverse complement.

Notes: *None*

Examples:

1. `yabby> seq_compl dna dna_c`

```
'dna' contains 1 sequence(s)
Working on 'chr01'
```

Requirements: seq

Creates objects: seq

System scripts: *None*

3.2.3 seq_genbank

Fetch a sequence from GenBank.

Usage:

`seq_genbank [ options ] IDENTIFIER OBJ_NAME`

Fetch a sequence from GenBank using an identifier such as the sequence accession number (default), version, GI number or unique ID (locus) and save the sequence under the name OBJ_NAME.

Options:

-a ACCESSION_NUMBER (default) – fetch sequence by accession number

-v VERSION – fetch sequence by version number

-g GI_NUMBER – fetch sequence by GI number

-i UNIQUE_ID – fetch sequence by unique ID

Notes:

1. This command requires Bioperl's Bio::DB module.

Examples:

1. `yabby> seq_fetch -a J00522 mig`

   ```
   Saving 'J00522' as 'mig'
   ```

Requirements: *None*

Creates objects: seq

System scripts: *None*

3.2.4 seq_info

Prints information about the sequence object.

Usage:

`seq_info [ options ] OBJ_NAME`

Where OBJ_NAME is the name of an existing sequence object.

Options:

-l – long output. When multiple sequences are present, print the number of residues for each sequence. By default, only a short summary is printed.

-n – print only the number of residues in each sequence

Notes: *None*

Examples:

1. `yabby> seq_info cad3`

   ```
   'cad3' contains 3 sequence(s)
     min number of residues: 192 (sequence 'Q53650_STAAU')
     max number of residues: 193 (sequence 'Q97PJ0_STRPN')
   ```

2. `yabby> seq_info -l cad3`

   ```
   'cad3' contains 3 sequence(s)
    1 -> Q53650_STAAU, 192 residues
    2 -> Q97PJ0_STRPN, 193 residues
    3 -> P95773_STALU, 192 residues
   ```

3. `yabby> seq_info -n cad3`

```
 'cad3' contains 3 sequence(s)
192
193
192
```

Requirements: seq

Creates objects: *None*

System scripts: *None*

3.2.5 seq_load

Loads sequence(s) from the database file.

Usage:

`seq_load [ options ] DBA_FILE OBJ_NAME`

Where DBA_FILE is the name of the database file. OBJ_NAME is the internal Yabby name for the sequence(s).

Options:

-a – append sequences to an already existing sequence object OBJ_NAME

-f – the file format is FASTA (default)

-b – the file format is BLOCKS

Notes:

1. Only BLOCKS format written by MEME [2] output was tested.

Examples:

1. `yabby> seq_load cad3.fas cad3`

```
Reading the file 'cad3.fas' ..
3 sequence(s) found.
```

Requirements: *None*

Creates objects: seq

System scripts: *None*

3.2.6 seq_op

Calculates union/intersection/difference of two sequence objects by using the sequence IDs.

Usage:

`seq_op [ options ] SEQ1_OBJ SEQ2_OBJ OBJ_NAME`

Calculates union/intersection/difference of two sequence objects SEQ1_OBJ and SEQ2_OBJ, and stores the result as the sequence object OBJ_NAME.

Options:

-u – calculate the union (default)

-i – calculate the intersection

-d – calculate the difference

Notes:

1. The calculation will fail if there are duplicate sequences in one set. For example, if two sets of sequences have no sequence in common, but one set of sequences contains two copies of the sequence 'F36.5845', the intersection of the two sets will contain this sequence.

Examples:

1.
```
yabby> seq_op -i cad cad3 cadi

 Found 6 sequence(s) in 'cad'
 Found 3 sequence(s) in 'cad3'
  INTERSECTION contains 3 sequence(s)
  DIFFERENCE contains 3 sequence(s)
  UNION contains 6 sequence(s)
 [ Saving INTERSECTION as 'cadi' ]
```

Requirements: seq

Creates objects: seq

System scripts: *None*

3.2.7 seq_os

Inserts additional information in sequence comments, where additional information is read from an external file.

Usage:

`seq_os FILE_NAME OBJ_NAME`

Where FILE_NAME is the file which contains pairs (sequence ID, additional information), and OBJ_NAME is the name of an existing sequence object. FILE_NAME must include all sequence IDs present in the OBJ_NAME.

Options:

None

Notes:

1. The object OBJ_NAME will be overwritten with altered comments to include the additional information for each sequence.

Examples:

1. Insert organism names in 21 sequences stored under the name 'cam.seq'.

```
yabby> seq_os sprot_test.dat.os cam

 Reading the file 'sprot_test.dat.os' ...
 21 sequences to process.

 All done.
 [ seq_os: 'cam.seq' exists, overwritten ]
```

Requirements: *seq*

Creates objects: *seq* (overwrites the original seq object)

System scripts: *None*

3.2.8 seq_pattern

Searches for letter pattern in a sequence object.

Usage:

`seq_pattern [ options ] PATTERN OBJ_NAME`

Searches for pattern PATTERN in sequences OBJ_NAME. Where OBJ_NAME is the name of an existing sequence object.

Options:

-c – Match the comment not the sequence.

-s NAME – extract the matching sequences and save under the name NAME.

-n – Print the matching sequences and the residue position where the pattern matches.

Notes: *None*

Examples:

1. yabby> seq_pattern IDY cad3

   ```
   'IDY' matches in 'Q53650_STAAU'
   'IDY' matches in 'Q97PJ0_STRPN'
   3 sequences examined, 2 match(es) found
   ```

2. yabby> seq_pattern -c STAA cad3

   ```
   'STAA' matches in 'Q53650_STAAU'
   3 sequences examined, 1 match(es) found
   ```

3. yabby> seq_pattern -s IDY_matches IDY cad3

   ```
   'IDY' matches in 'Q53650_STAAU'
   'IDY' matches in 'Q97PJ0_STRPN'
   3 sequences examined, 2 match(es) found
   Saving matches as 'IDY_matches'
   ```

Requirements: seq

Creates objects: (with option -s) seq

System scripts: *None*

3.2.9 seq_pick

Extracts a subset of sequences from the sequence object.

Usage:

`seq_pick [ options ] OBJ_NAME OBJ_NAME_NEW`

where OBJ_NAME is the name of an existing sequence object, and OBJ_NAME_NEW is the name of the sequence object to be created.

Options:

-n RANGE – extract the sequence by sequence number. The parameter RANGE can be a single integer, in which the sequence with this sequence number will be extracted. Alternatively, RANGE can contain two integers separated by a colon such as N:M. In this case the sequences which have the sequence number between N and M will be extracted (inclusive).

-q SEQID – pick a sequence with the sequence ID SEQID

-l MIN:MAX – pick sequences whose length is between MIN and MAX

Notes: *None*

Examples:

1. `yabby> seq_pick -q Q53650_STAAU cad3 s1`

   ```
   Fetching the sequence 'Q53650_STAAU'
   Saving the extracted sequence as 's1'
   ```

2. `yabby> seq_pick -n 2 cad3 s2`

   ```
   Fetching the sequence 2 ('Q97PJ0_STRPN')
   Saving the extracted sequence as 's2'
   ```

Requirements: seq

Creates objects: seq

System scripts: *None*

3.2.10 seq_sprot_os

Inserts the organism name into sequence comments by using the local Swiss-Prot database file.

Usage:

`seq_sprot_os DBA_FILE OBJ_NAME`

Where DBA_FILE is the local database file in the Swiss-Prot format, and OBJ_NAME is the name of an existing sequence object. The object OBJ_NAME will be overwritten with altered comments to include the organism name for each sequence.

Options: *None*

Notes:

This command overwrites the original sequence.

Examples:

1. Fetch the organism name (OS) field from the Swiss-Prot file, 'sprot_test.dat', and insert this in the sequence comment of three sequences saved as 'aqr1':

   `yabby> seq_sprot_os sprot_test.dat aqr1`

   ```
   3 sequences to process.
   Printing dot per processed sequence:
   ```

```
...
All done.
[ seq_sprot_os: 'aqr1.seq' exists, overwritten ]
```

Requirements: *seq*

Creates objects: *seq* (overwrites the original seq object)

System scripts: *None*

3.2.11 seq_strip

Strips a portion of a sequence.

Usage:

`seq_strip begin:end OBJ_NAME OBJ_NAME_NEW`

Where OBJ_NAME is the name of an existing sequence object, and begin:end are the first and last residue positions to strip (inclusive). The resulting object will be saved under the name OBJ_NAME_NEW.

If more than one sequence is present in the sequence object, all will be stripped and saved under the new name.

In stripped sequences, IDs are set to ORIGINALID_begin:end.

Options: *None*

Notes: *None*

Examples:

1. `yabby> seq_strip 21:40 cad3 cad3portion`

```
'cad3' contains 3 sequence(s)
stripping 'Q53650_STAAU'
stripping 'Q97PJ0_STRPN'
stripping 'P95773_STALU'
```

Requirements: seq

Creates objects: seq

System scripts: *None*

3.2.12 seq_uniprot

Fetch a sequence from SwissProt.

Usage:

```
seq_uniprot [ options ] LOCATION IDENTIFIER OBJ_NAME
```

Fetch a sequence from SwissProt using an identifier such as the sequence accession number (default) or unique ID and saves the sequence under the name OBJ_NAME.

Options:

-l LOCATION – possible values (australia,canada,china,korea,switzerland,taiwan,us)

-a ACCESSION_NUMBER – fetch sequence by accession number (default)

-i UNIQUE_ID – fetch sequence by unique ID

Notes:

1. This command requires Bioperl's Bio::DB module.

Examples:

1.
```
yabby> seq_uniprot -a P43780 -l australia miga

Saving 'P43780' as 'miga'
```

Requirements: *None*

Creates objects: seq

System scripts: *None*

3.2.13 seq_unique

Finds unique sequences in one sequence object relative to another by comparing sequence strings.

Usage:

```
seq_unique SEQ1_OBJ SEQ2_OBJ OBJ_NAME
```

This command will find the unique sequences in SEQ1_OBJ compared to SEQ2_OBJ, and store these unique sequences as OBJ_NAME.

Options: *None*

Notes:

1. Sequences present in the first sequence object and not present in the second second object are calculated. Therefore the order of sequence objects given in the argument matters.

2. This command compares sequence letters as opposed to sequence IDs (compared to seq_op). Two sequences are identical if they are an exact letter-by-letter match.

Examples:

1.
```
yabby> seq_unique cad cad3 caduniqincad

 3 unique sequences found.
 Saving sequences as 'caduniqincad'
```

Requirements: seq

Creates objects: seq

System scripts: *None*

3.3 Swiss-Prot related commands

3.3.1 sprot_fetch

Fetches a sequence from a Swiss-Prot local file database by its ID.

Usage:

```
sprot_fetch DBA_FILE SPROT_ID OBJ_NAME
```

Where DBA_FILE is the database is the Swiss-Prot format, and SPROT_ID is the Swiss-Prot sequence ID, and OBJ_NAME is the name under which the sequence will be saved in the workspace.

Options: *None*

Notes: *None*

Examples:

1. Fetch the sequence '110KD_PLAKN' from the Swiss-Prot file 'sprot_test.dat':

```
yabby> sprot_fetch sprot_test.dat 110KD_PLAKN plakn

 Sequence '110KD_PLAKN' found.
 [ Saving as 'plakn.seq' ]
```

Requirements: *None*

Creates objects: *seq*

System scripts: *None*

3.3.2 sprot_os

Extracts sequence ID and organism name from the Swiss-Prot sequence database file.

Usage:

```
sprot_os DBA_FILE OUT_FILE
```

Where DBA_FILE is the name of the sequence database file in the Swiss-Prot format, and OUT_FILE is the name of the output file to be created, containing pairs (sequence ID, organism name).

Options: *None*

Notes:

1. This command is designed to work with the command 'seq_os'.

Examples:

1. Extract the organism names (Swiss-Prot OS field) from the file 'uniprot.dat', and save as the file 'uniprot.dat.os':

```
yabby> sprot_os uniprot.dat uniprot.dat.os

 Processing the database file 'uniprot.dat' ..
 Printing a dot per 10000 processed sequences:
 ..........................................................
 ..........................................................
 ..........................................................
 ..........................................................
 ..........................................................
 ..........................................................
 ..........................................................
 ..........................................................
 ..........................................

 All done.
```

Requirements: *None*

Creates objects: *None*

System scripts: *None*

3.4 HMMER commands

3.4.1 hmm_score_extract

Fetches hits from the HMMER (HMMPFAM) search output file.

Usage:

```
hmm_score_extract [ options ] HMMPFAM_OUT
```

Where HMMPFAM_OUT is the HMMPFAM output file.

Options:

-E CUTOFF – Define cutoff for acceptable E values (default: 0.01)

-s HMM_ITEM – Save the scores under the name HMM_ITEM

-d – Turn debuggin on. This will create the file hmm_scores.opt_d_flag with raw scores.

Notes: *None*

Examples:

1.
```
yabby> hmm_score_extract -E 0.01 -s hits hmmpfam.out

Processing HMMPFAM search output file

 No     Sequence                                E-score
-------------------------------------------------------
(   1)  LmjF05.1190@All                         2.70e-03
(   2)  LmjF05.1190@GlycogenStarch              4.20e-03
(   3)  LmjF05.0920@All                         7.50e-03
```

Requirements: *None*

Creates objects: (with option -s) hmm_score

System scripts: *None*

3.4.2 hmm_score2seq

Fetches and saves sequences whose ID's are given in the HMM scores objects created by the command 'hmm_score_extract'.

Usage:

```
hmm_score2seq [ options ] DBA_FILE OUT_FILE OBJ_NAME
```

Where DBA_FILE is the sequence database, OUT_FILE is the output file with sequences (to be created), and OBJ_NAME is the name under which the scores were saved with 'hmm_score_extract'.

Options:

-w WIDTH – Set the width of the sequence string per line written to the OUT_FILE (default: width=60)

-m MODEL – Extract only sequences that matched a particular HMM model. If this option is not activated all matches are written to the output file and sequence IDs are written as SEQIDPFAM_MODEL. If this option is activated the sequence IDs are written only as SEQID.

-n NUMBER – Crop sequences extracted to at most NUMBER of sequences.

-c – Do NOT embed the matching model and maching score into sequence comment.

Notes:

1. Sequence database file DBA_FILE must be in FASTA format.

Examples:

1.
```
yabby> hmm_score2seq LmjFmockup.pep hits.fas hits

found 3 sequences to extract
Processing the database 'LmjFmockup.pep'
Sequences written to 'hits.fas'
```

Requirements: hmm_score

Creates objects: *None*

System scripts: *None*

3.5 Sequence motifs commands

3.5.1 motif_load

Loads sequence motif.

Usage:

```
motif_load [ options ] DBA_FILE OBJ_NAME
```

Where DBA_FILE is the name of the database file. OBJ_NAME is the internal YABBY name for this motif.

Options:

-f – the file format is FASTA (default)

-b – the file format is BLOCKS

Notes:

1. A 'motif' object is internally identical to the 'sequence' object.
2. Only BLOCKS format as given by MEME output was tested.

Examples:

1. `yabby> motif_load -b m2.blocks m2`

   ```
   Reading the file 'm2.blocks' ..
   11 sequence(s) found in the motif 'm2'.
   ```

Requirements: *None*

Creates objects: motif

System scripts: *None*

3.5.2 motif_cmp

Compares two sequence motifs.

Usage:

`motif_cmp MOTIF1_OBJ MOTIF2_OBJ`

Compares two motifs. Two motifs are identical if they have the same number of sequences, the sequences have the same ID, and the sequences themselves are identical as strings.

Options: *None*

Notes: *None*

Examples:

1. `yabby> motif_cmp m2 m2_second`

   ```
   Motifs 'm2' and 'm2_second' contain the same sequence IDs.
   Comparing the sequences...
   Motifs 'm2' and 'm2_second' are identical.
   ```

Requirements: motif

Creates objects: *None*

System scripts: *None*

3.5.3 motif_meme

Extracts motifs from MEME text output files.

Usage:

```
motif_meme MEME_OUTPUT MOTIF_NUMBER OBJ_NAME
```

Reads the output file MEME_OUTPUT, extracts motif number MOTIF_NUMBER and saves motif as OBJ_NAME.

Options: *None*

Notes: *None*

Examples:

1.
```
yabby> motif_meme meme.out 1 m1

Reading MEME output 'meme.out' ..
Motif 1 saved as 'm1'.
```

Requirements: *None*

Creates objects: motif

System scripts: *None*

3.6 BLAST commands

3.6.1 blast

Runs NCBI BLAST against a database.

Usage:

```
blast [ options ] DBA_FILE OBJ_NAME
```

Where DBA_FILE is the sequence database and OBJ_NAME is the name of the sequence object which contains the query sequence. If OBJ_NAME contains more than one sequence, all will be used in turn as a query sequence.

Options:

-E E_VALUE – Sets the expectation value for BLAST (default=0.01)

Notes:

1. This command runs the NCBI program 'blastall'
2. The full PATH to 'blastall' is defined in yabby_blast.pm

Examples:

1. yabby> blast -E 5.0 LmjFmockup.pep cad3

```
3 sequence(s) found in the object 'cad3'

Now running BLAST ..
BLASTing sequence 1 of 3 (Q53650_STAAU)
Query sequence 'Q53650_STAAU'
Database 'LmjFmockup.pep'
Found 3 hits above the threshold (E=5.00)
The best hit: 'LmjF05.1170'
E-score = 2.22e+00

BLASTing sequence 2 of 3 (Q97PJ0_STRPN)
Query sequence 'Q97PJ0_STRPN'
Database 'LmjFmockup.pep'
Found 1 hits above the threshold (E=5.00)
The best hit: 'LmjF05.0950'
E-score = 2.92e+00

BLASTing sequence 3 of 3 (P95773_STALU)
Query sequence 'P95773_STALU'
Database 'LmjFmockup.pep'
No BLAST hits above the threshold (E=5.00) found.
```

Requirements: *None*

Creates objects: *None*

System scripts: *None*

3.6.2 blast_info

Prints the information about the BLAST search previously generated by the 'blast' command.

Usage:

```
blast_info OBJ_NAME
```

Where OBJ_NAME is the name of an existing blast object.

Options: *None*

Notes: *None*

Examples:

1. yabby> blast_info cad3_1

```
Query sequence 'Q53650_STAAU'
Database 'LmjFmockup.pep'
Found 3 hits above the threshold (E=5.00)
The best hit: 'LmjF05.1170'
E-score = 2.22e+00
```

Requirements: blast

Creates objects: *None*

System scripts: *None*

3.6.3 blastg

Runs NCBI BLAST against a database and saves the best hit for each sequence.

Usage:

`blastg [ options ] DBA_FILE OBJ_NAME`

Where DBA_FILE is the sequence database and OBJ_NAME is the name of the sequence object which contains sequences to be blasted. Each sequence is in turn blasted against the database, and the top hit is stored as BLASG object. This object contains the list of:

SEQ_ID DBA_SEQ_ID E_VALUE

Where SEQ_ID is the sequence ID, DBA_SEQ_ID is the best hits database sequence ID, and E_VALUE is the E-value of the match.

Options:

-E E_VALUE – Sets the expectation value for BLAST (default=0.01)

Notes:

1. This command is experimental. Originally it was used to blast one genome against another, to obtain a quick estimate of the similarity between two genome.

Examples:

1. yabby> blastg -E 5.0 LmjFmockup.pep cad3

```
3 sequence(s) found in the object 'cad3'

Now running BLAST ..
BLASTing sequence 1 of 3 (Q53650_STAAU)
 top hit LmjF05.1170, E-score = 2.22e+00
```

```
BLASTing sequence 2 of 3 (Q97PJ0_STRPN)
 top hit LmjF05.0950, E-score = 2.92e+00
BLASTing sequence 3 of 3 (P95773_STALU)
 top hit None, E-score = -1.00e+00
```

Requirements: *None*

Creates objects: *None*

System scripts: *None*

3.7 Protein three-dimensional structure commands

3.7.1 mol_load

Loads a molecule from the PDB file.

Usage:

`mol_load [ options ] PDB_FILE OBJ_NAME`

Loads the molecule from the PDB_FILE and saves it under the name OBJ_NAME.

Options:

-u – require strict Protein Data Bank format when reading the PDB file. This amounts to requiring that residue names have maximum of three characters (columns 18,19, and 20) of an ATOM or HETATM record. By default this behavior is disabled, and four letter residue names are expected (columns 18,19,20 and 21). This is safe in most cases as the extra column 21 is actually not defined in the strict Protein Data Bank format.

Notes: *None*

Examples:

1.
```
yabby> mol_load 1BT0.pdb bto

661 atoms found in the molecule 'bto'
```

Requirements: *None*

Creates objects: mol

System scripts: *None*

3.7.2 mol2seq

Creates the amino acid sequence from a molecule.

Usage:

`mol2seq OBJ_NAME`

where OBJ_NAME is the name of an existing 'mol' object.

Options:

-f – print to a file.

Notes:

1. If no argument is given all will be printed.

Examples:

1.
```
yabby> mol2seq bto

 WARNING: a non-sequential residue:  201 ZN
 WARNING: residue 'ZN' does not have one letter code
...snip...
 WARNING: residue 'HOH' does not have one letter code
 153 residues found in the molecule 'bto'
```

Requirements: mol

Creates objects: seq

System scripts: *None*

3.7.3 pdb_conv

Converts Protein Data Bank coordinate files into XPLOR/CHARMM PDB format, with possible filtering

Usage:

`pdb_conv [ options ] PDB_INPUT PDB_OUTPUT`

This command reads Protein Data Bank file PDB_INPUT and writes XPLOR/CHARMM PDB file PDB_OUTPUT.

Options:

-u – require strict Protein Data Bank format when reading the PDB file. This amounts to requiring that residue names have maximum of three characters (columns 18,19, and 20) of an ATOM or HETATM record. By default this behavior is disabled, and four letter residue names are expected (columns 18,19,20 and 21). This is safe in most cases as the extra column 21 is actually not defined in the strict Protein Data Bank format

-f FORMAT – use the format FORMAT when writing the PDB output file. Allowed formats are 'xplor' (default) or 'charmm'. The difference between the two is subtle, and occurs only for residues which have a residue number greater than 999

-l ALT_LOC – write atoms with the alternative location field equal to ALT_LOC, together with those without ALT_LOC label. In some structures only a subset of atoms is found in two alternative locations, and therefore only a subset of atoms has ALT_LOC fields set to distinguish the two positions

-m CHAIN_ID – write only atoms with the chain ID equal to CHAIN_ID

-i SEGID – replace the segment name with SEGID

-h – discard hydrogens, i.e. all atoms whose names begin with either the letter 'H' (case insensitive) or a number

-e – discard HEATM records (by default HEATM records are included, and rewritten as ATOM records)

-r RBEGIN:OFFSET – Add offset OFFSET to reside numbers starting with the residue number RBEGIN

Notes: *None*

Examples:

1.

Requirements: *None*

Creates objects: *None*

System scripts: *None*

3.7.4 pdb_model

Splits Protein Data Bank file with multiple models into multiple XPLOR PDB files

Usage:

```
pdb_model [ options ] PDB_INPUT MODEL_ROOT
```

PDB_INPUT is the Protein Data Bank file which contains multiple models (separated with MODEL/ENDMDL statements), and MODEL_ROOT is the root name for the CHARMM/XPLOR PDB files to be created (one file per model). The final file names will be MODEL_ROOT_1.pdb, MODEL_ROOT_2.pdb, etc. until all models are exausted.

Options:

-u – require strict Protein Data Bank format when reading the PDB file. This amounts to requiring that residue names have maximum of three characters (columns 18,19, and 20) of an ATOM or HETATM record. By default this behavior is disabled, and four letter residue names are expected (columns 18,19,20 and 21). This is safe in most cases as the extra column 21 is actually not defined in the strict Protein Data Bank format

-f FORMAT – use the format FORMAT when writing the PDB output file. Allowed formats are 'xplor' (default) or 'charmm'. The difference between the two is subtle, and occurs only for residues which have a residue number greater than 999

-l ALT_LOC – write atoms with the alternative location field equal to ALT_LOC, together with those without ALT_LOC label. In some structures only a subset of atoms is found in two alternative locations, and therefore only a subset of atoms has ALT_LOC fields set to distinguish the two positions

-m CHAIN_ID – write only the molecule with the chain ID equal to CHAIN_ID

-i SEGID – replace the segment name with SEGID

-h – discard hydrogens, i.e. all atoms whose names begin with either the letter 'H' (case insensitive) or a number

-e – discard HEATM records (by default HEATM records are included, and rewritten as ATOM records)

-r RBEGIN:OFFSET – Add offset OFFSET to reside numbers starting with the residue number RBEGIN

Notes: *None*

Examples:

1. The file 1C3T.pdb contain 20 structures of ubiquitin, available from the RCSB Protein Data Bank.

```
yabby> pdb_model 1C3T.pdb ubq_

 working on model  1 (creating 'ubq_1.pdb')
 working on model  2 (creating 'ubq_2.pdb')
 working on model  3 (creating 'ubq_3.pdb')
 working on model  4 (creating 'ubq_4.pdb')
 working on model  5 (creating 'ubq_5.pdb')
 working on model  6 (creating 'ubq_6.pdb')
 working on model  7 (creating 'ubq_7.pdb')
 working on model  8 (creating 'ubq_8.pdb')
 working on model  9 (creating 'ubq_9.pdb')
 working on model 10 (creating 'ubq_10.pdb')
 working on model 11 (creating 'ubq_11.pdb')
 working on model 12 (creating 'ubq_12.pdb')
 working on model 13 (creating 'ubq_13.pdb')
 working on model 14 (creating 'ubq_14.pdb')
 working on model 15 (creating 'ubq_15.pdb')
 working on model 16 (creating 'ubq_16.pdb')
 working on model 17 (creating 'ubq_17.pdb')
 working on model 18 (creating 'ubq_18.pdb')
 working on model 19 (creating 'ubq_19.pdb')
 working on model 20 (creating 'ubq_20.pdb')
```

```
 20 models found

yabby>
```

Requirements: *None*

Creates objects: *None*

System scripts: *None*

3.8 EMBOSS commands

3.8.1 emboss_needle

Extracts sequences which have the highest similarity from he output of the EMBOSS program 'needle'.

Usage:

`emboss_needle NEEDLE_OUTPUT NN`

Where NEEDLE_OUTPUT is the output file of the program 'needle', and NN is the number of highest alignments to report (as given by the 'Similarity' line in the needle output).

Options: *None*

Notes: *None*

Examples:

1.
```
yabby> emboss_needle needle.out 3

Processing the file 'needle.out' ..

(1) Q45153_BACFI:LmjF05.1170, Similarity: 26.6
(2) Q45153_BACFI:LmjF05.1120, Similarity: 16.4
(3) Q45153_BACFI:LmjF05.1040, Similarity: 15.0
```

Requirements: *None*

Creates objects: *None*

System scripts: *None*

3.8.2 emboss_needl2

For each sequence in a set finds the best matching sequence from the database by running the EMBOSS program needle. Uses the information on sequence similarity reported by needle.

Usage:

```
emboss_needl2 SEQ_QUERY SEQ_DBA OBJ_NAME
```

Where SEQ_QUERY is an existing yabby sequence object, SEQ_DBA is the FASTA database file with sequences to be compared, and OBJ_NAME is the name of the 'needl2' object to be created.

Options:

None - Uncomment this and delete line below if no notes

Notes:

1. This command requires the EMBOSS program 'needle'

Examples:

1.
```
yabby> emboss_needl2 ec057 Ecoli_lab.fas ec

 'ec057' contains 9 sequence(s)
 Working on sp|P36546|OMPX_ECOLI
Needleman-Wunsch global alignment.
 Working on tr|Q8X8K6|Q8X8K6
Needleman-Wunsch global alignment.
 Working on sp|P39170|UP05_ECOLI
Needleman-Wunsch global alignment.
 Working on tr|Q8XDF1|Q8XDF1
Needleman-Wunsch global alignment.
 Working on sp|P58603|OMPT_ECO57
[... further output deleted ...]
```

Requirements: seq (for SEQ_QUERY)

Creates objects: needl2

System scripts: *None*

3.9 Miscellaneous commands

3.9.1 pfam_fetch

Fetches an entry from the PFAM database file.

Usage:

`pfam_fetch PFAM_CODE PFAM_DBA FILE_NAME`

Where PFAM_CODE is the PFAM accession code, PFAM_DBA is the PFAM database, and FILE_NAME is the file to save the entry. For example, to fetch the entry PF00293 from the PFAM database Pfam-A.seed, and save it as PF00293.seed:

pfam_fetch PF00293 Pfam-A.seed PF00293.full

Options: *None*

Notes: *None*

Examples:

1.

Requirements: *None*

Creates objects: *None*

System scripts: *None*

3.9.2 sprot_split

Splits large sequence database written in SWISS-PROT format into smaller files. This command is useful when a large SWISS-PROT file needs to be reformatted with an external programs (such as 'sreformat'), which have a limitation on the file size.

Usage:

`sprot_split [ options ] DBA_FILE`

Where DBA_FILE is the name of the sequence database file in the SWISS-PROT format. The output files are named DBA_FILE.1, DBA_FILE.2, etc.

Options:

-n NLINES – split the database into files approx NLINES each (default: NLINES = 20000000).

Notes: *None*

Examples:

1.
```
yabby> sprot_split uniprot_trembl.dat
Reading the database file 'uniprot_trembl.dat' ..
The number of lines in the database: 204071661
-> Creating 'uniprot_trembl.dat.1'
-> Creating 'uniprot_trembl.dat.2'
-> Creating 'uniprot_trembl.dat.3'
-> Creating 'uniprot_trembl.dat.4'
```

```
-> Creating 'uniprot_trembl.dat.5'
-> Creating 'uniprot_trembl.dat.6'
-> Creating 'uniprot_trembl.dat.7'
-> Creating 'uniprot_trembl.dat.8'
-> Creating 'uniprot_trembl.dat.9'
-> Creating 'uniprot_trembl.dat.10'
-> Creating 'uniprot_trembl.dat.11'
```

Requirements: *None*

Creates objects: A number of smaller sequence database file.

System scripts: *None*

Bibliography

[1] Stajich JE, Block D, Boulez K, Brenner SE, Chervitz SA, Dagdigian C, Fuellen G, Gilbert JG, Korf I, Lapp H, Lehvaslaiho H, Matsalla C, Mungall CJ, Osborne BI, Pocock MR, Schattner P, Senger M, Stein LD, Stupka E, Wilkinson MD, and Birney E. The Bioperl toolkit: Perl modules for the life sciences. *Genome Res*, 12(10):1611–8, 2002.

[2] Bailey TL, Williams N, Misleh C, and Li WW. MEME: discovering and analyzing DNA and protein sequence motifs. *Nucleic Acids Res*, 34:W369–73, 2006.

[3] Eddy SR. Profile hidden Markov models. *Bioinformatics*, 14(9):755–63, 1998.

[4] HMMER. http://hmmer.janelia.org/.

[5] NCBI BLAST Web interface. http://www.ncbi.nlm.nih.gov/BLAST/.

[6] NCBI BLAST programs download. ftp://ftp.ncbi.nih.gov/blast/.

[7] Rice P, Longden I, and Bleasby A. EMBOSS: the European Molecular Biology Open Software Suite. *Trends Genet*, 16(6):276–7, 2000.

Index