Swissknife — ‘lazy parsing’ of SWISS-PROT entries

Henning Hermjakob, Wolfgang Fleischmann and Rolf Apweiler

EMBL Outstation Hinxton, The European Bioinformatics Institute (EBI), Wellcome Trust Genome Campus, Hinxton, Cambridgeshire CB10 1SD, UK

Received on April 1, 1999; revised on May 17, 1999; accepted on May 18, 1999

Abstract

Summary: We present Swissknife, a set of Perl modules which provides a fast and reliable object-oriented interface to parsing and modifying files in SWISS-PROT format.

Availability: The Swissknife modules are available at ftp://ftp.ebi.ac.uk/pub/software/swissprot/

Contact: hhe@ebi.ac.uk

Introduction

The combination of SWISS-PROT and TrEMBL (Bairoch and Apweiler, 1999) forms a comprehensive protein database which combines the high quality of annotation in SWISS-PROT with the completeness of TrEMBL. Use of these databases for bioinformatics often involves extracting partial information from a huge number of entries in SWISS-PROT text format. In addition, the creation of in-house databases in SWISS-PROT format requires the ability to modify or create entries in SWISS-PROT format.

SRS (Etzold et al., 1996) provides a powerful query interface for text-based biological databases, but it is not very suitable for modifying entries in SWISS-PROT format. Scripting languages like Perl (Wall and Schwartz, 1991) allow fast and easy parsing of text-based databases, but ad-hoc developed parsers tend to neglect details of the formatting of the parsed database, and it is often difficult to maintain a multitude of specialised parsing statements for a given database.

To overcome these difficulties, we have developed Swissknife, a set of Perl modules which provides a complete, fast and reliable object-oriented interface to parsing and modifying files in SWISS-PROT format. Swissknife has now been used internally for more than a year by the SWISS-PROT group at the EBI and therefore provides a well-tested environment for the handling of entries in SWISS-PROT format.

The interface of Swissknife closely follows the interface of the PrEMBL modules (Pocock et al., 1998), a set of Perl modules to parse and edit files in EMBL (Stoesser et al., 1999) format, but the implementation differs to account for the format differences between SWISS-PROT and EMBL, and to implement the fast ‘lazy parsing’ concept.

# Make Swissknife available
use SWISS::Entry;

# Read an entire record at a time
$/ = "\n\n";

# Initialisation of the hash holding
# new mass spectrometry data [omitted]

# Main loop over all entries
while (<>){
    $entry = SWISS::Entry->fromText($_);
    ...

    # Add mass spec data
    if ($massHash{$entry->AC}{
        $entry->CCs->add('MASS SPECTROMETRY',
                        $massHash{$entry->AC});
    }

    # Delete all keywords
    # starting with 'Phospho'.
    $entry->KWs->del('Phospho.*');

    # Output the modified entry
    print $entry->toText;
}

Fig. 1. Usage example.

Figure 1 shows a usage example of Swissknife. If there is new mass spectrometry data available for the entry, it is added in a new comment block. Additionally, all keywords starting with ‘Phospho’ are deleted.

Swissknife ‘lazy parsing’

A parser which completely parses complex SWISS-PROT entries tends to be relatively slow, and it is computationally expensive to completely parse an entry just to get the list of accession numbers. Therefore we have developed the concept of ‘lazy parsing’ for Swissknife. When an entry is
read from the flat file, it is only stored as an unstructured text (Fig. 2a). If a particular subobject is requested, e.g. the accession number (\$entry\rightarrow\text{AC}), the text is parsed, the AC object is created and stored as a subobject of \$entry (Figs 2b and 2c). All subsequent accesses are then redirected to the subobject instead of to the textual representation. When the entry is written, the modified contents of the subobjects are written back into the text representation first.

The concept of ‘lazy parsing’ has allowed us to implement a complete interface to entries in SWISS-PROT format while retaining a high parsing speed if only few items of the entry are requested. A full parse of 243 SWISS-PROT sample entries requires 2.52 s on a Pentium 266, while extracting the accession numbers requires only 0.61 s (Table 1).

Memory consumption is nearly constant if entries are parsed sequentially as in Fig. 1. A run on the 77 977 entries of SWISS-PROT Release 37 resulted in an increase in memory consumption from 3.6 Mbyte (entry 0) to 4.3 Mbyte (entry 77 977). If entries are stored, the additional memory consumption depends on the degree of parsing. Storing 10 240 partially parsed entries by adding the statement \$store\{$entry\rightarrow\text{AC}\}=$entry to the above program resulted in a memory consumption of 3.80 kb per entry, compared to 2.21 kb per entry in the ASCII file.

**Table 1.** Execution time of different commands for a test set of 243 SWISS-PROT entries

<table>
<thead>
<tr>
<th>Purpose</th>
<th>Command</th>
<th>Time (s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Read entry</td>
<td>$entry\rightarrow\text{fromText}</td>
<td>0.40</td>
</tr>
<tr>
<td>Get accession number</td>
<td>$entry\rightarrow\text{AC}</td>
<td>0.61</td>
</tr>
<tr>
<td>Get DR lines</td>
<td>$entry\rightarrow\text{DRs}</td>
<td>0.82</td>
</tr>
<tr>
<td>Full parse</td>
<td>$entry\rightarrow\text{fullparse}</td>
<td>2.52</td>
</tr>
</tbody>
</table>

**Future development**

As Swissknife is used in the production of SWISS-PROT and TrEMBL, it will continuously be updated to reflect SWISS-PROT format changes. In addition, we are working on interface improvements and new input methods. Internally we are already using additional Swissknife modules to access the relational SWISS-PROT database implementation.

**Acknowledgements**

This work was supported in part by grant number B104-CT97-2099 of the European Commission. The Swissknife development has been based on the example of the PrEMBL modules (Pocock et al., 1998).

**References**


