

INTERNSHIP REPORT

# BioParser: Easier Mining of Online Bioinformatics Databases

An internship report presented in  
partial fulfillment of the requirement  
of the Master of Science  
in Computational Biosciences

**Carol Barner**

*Computational Biosciences Program  
Arizona State University*

**Dr. Rosemary Renaut**

*Internship Advisor*

*Department of Mathematics and Statistics  
Arizona State University*

**Mr. John Pearson**

*Internship Sponsor*

*Bioinformatics Unit*

*Translational Genomics Research Institute*

*Dr. Eric Kostelich, Committee Member*

*Dr. Martin Wojciechowski, Committee Member*

Internship:

**May, 2005 : April, 2006**

Technical Report Number: **06-12**

**April 25, 2006**

## Table of Contents

Abstract.....	3
Goals of Project.....	4
1 Introduction .....	6
1.1 Significance.....	9
2 Methods .....	10
2.1 Object-Oriented Perl Programming.....	10
2.2 Damian Conway's Parse::RecDescent Parsing Program.....	11
2.3 CVS, the Concurrent Versioning System .....	14
2.4 Use of a Testing Harness .....	14
2.5 Automatic Generation of Documentation .....	15
2.6 Systematic Organization and Naming of files .....	16
2.7 UNIX operating system and VPN access .....	17
3 Results and Significance.....	18
3.1 Homologene .....	18
3.2 dbEST .....	19
3.3 Significance of the Results.....	23
4 Conclusions and Future Directions.....	24
4.1 Conclusions .....	24
4.2 Future Directions .....	24
References.....	26

## Abstract

**Motivation:** Many sophisticated online bioinformatics databases exist, such as [GenBank](#) (1) and [Swiss-Prot](#) (2). They provide excellent websites for researchers to search and view individual records containing information about genes, DNA sequences, or proteins. However, if a researcher wishes to select specific information from hundreds or thousands of records, data mining becomes very cumbersome and lengthy. It is usually necessary to dig through individual web pages and copy and paste the desired embedded data. Also, retrieval of only the desired records may not be possible, depending on how extensive a query system is provided by the website programmers.

Another available route is to get direct access to the online data by downloading a text copy of all the data. However, these are not standardized, and are often difficult to understand and use. In addition to frequent updates being made to the actual data, the data file formats are changed frequently.

**Results:** [TGen's BioParser project](#) (3) parses and converts these huge text files into data objects that are saved to disk for later use. These object files may then be quickly and easily searched to obtain exactly the desired information fields from the desired records (objects). While other programming projects have provided partial access to some databases, BioParser is unique in providing easy, quick access to all fields in many online databases. It offers reliable, commercial-quality support that will respond quickly to the frequent file changes. BioParser has been extended for parsing the Homologene and dbEST databases.

## Goals of Project

BioParser will provide direct, easy access to all fields of all records of online data from most popular online bioinformatics databases. BioParser will cope with the frequent changes to the data and the data file formats by providing frequent updates to the parsed databases, often weekly.

The BioParser project uses object-oriented, recursive-descent parsing in the Perl computer programming language to parse, organize, and label all the fields (information items) in online Bioinformatics databases, so that desired information items can be easily and quickly searched for and retrieved.

Initial design and major programming of BioParser was already done by John Pearson.

It will be very useful internally as a research tool for TGen scientists.

The goal for this internship project was to help extend Release Version 1 of this project by programming additional modules to parse more databases. Modules for the Homologene and dbEST databases were programmed.

A feasibility study (4) on the marketing of BioParser was conducted. This was a separate project performed by a team of ASU Computational Biosciences graduate students for HSM 591, a Business course for Computational Biosciences students taught by Dr. Bradford Kirkman-Liff. The study concluded that BioParser could be successfully marketed.

The report states:

*“Our industry survey indicates that other tools have been made available which parse biological databases. The alternative software packages either do not cover the breadth with respect to the number of databases parsed, or lack the depth of parsing all fields in a database*

*as BioParser does. BioParser offers a unique combination of broad database coverage and ease of database searching, along with ongoing commercial-quality support” (4, page 4).*

The report recommended offering already-parsed database downloads via the Internet. Sample programs for accessing the files would also be provided. Academics and nonprofit researchers could obtain the downloads without cost, while commercial and governmental users would be charged an annual license fee. The resulting revenue would provide funding for maintaining the ongoing parsing operations, as well as additional revenue for other internal TGen projects.

# 1 Introduction

For years, scientists and bioinformatics programmers have struggled with how best to mine the specific data they need from various online databases. While each database offers a web-based search and retrieval system, these often are limited to searching on specific fields. Also, they often return entire database records as their search results, while the researcher may only desire one or two specific items from each record. Thus, the laborious task arises of copying and pasting those desired specific items from each returned record.

Recognizing these limitations, each database generally offers the ability to download its full set of data files in text format. This facilitates individualized programming for selection of desired information. However, this data mining job is an arduous one, often beyond the programming abilities and desires of many researchers who are interested in analyzing the data, and not in spending many hours programming and executing programs just to selectively retrieve relevant data.

A number of different programming solutions have been offered as free solutions to this problem. However, they are all of two types. Either they are parser programs limited to retrieving many fields from one or two databases (4, page 4), or they are limited to retrieving only commonly selected fields from several databases, such as [BioPerl](#) (5). Further, they all require users to download the latest version of the complete database text file and then spend many hours of computer time running a particular program. They often require considerable knowledge of programming in Perl or some other language in order to specify the desired information. Virtually all of them are offered as free programs provided by unpaid volunteers who have limited time to help users or to modify their code in response to changes in the database format.

BioParser is a sophisticated system of programs written in the [Perl programming language](#) (6). It is designed to “parse” online biological databases. Parsing refers to the BioParser programs going through the data files one character at a time, and recognizing each piece of data as a specific field. Some representative examples are:

- a GenBank ID for a biological sequence
- authors of academic papers about the sequence
- lines containing the DNA sequence itself
- the scientific name of the organism whose DNA was analyzed

There is great variability in the contents of individual records in one database, and each online database uses its own file format. These formats are frequently changed without notice. Often additional fields will be added, or an archaic field retired. Many records lack specific fields of information.

BioParser incorporates a free, powerful tool for parsing, [Damian Conway's Recursive Descent Parser](#) (7, 8). BioParser code creates a grammar defining the structure of each data file, then passes it to Conway's `Parse::RecDescent` module to create a parsing module for one online database. This can then be used to parse new releases of the online database.

When the database format is changed without notice, BioParser will encounter the change and stop, thus signaling the change to the BioParser staff programmers. This is a major advantage over other program systems which usually look only for certain fields, and do not discover changes in the file format. Or if they do run into a change that stops their program, the free support offered by volunteers will often not get the program going again for some time, perhaps weeks or months.

Once a database format change is discovered by a BioParser parsing run stopping, the BioParser programming staff must then update the grammar code for that database to accommodate the changed file format. Normally that will be a simple, quick job for the TGen programmers who are skilled in writing Parse::RecDescent grammar code. However, it would be a very difficult job for research scientists or even bioinformatics programmers who are not familiar with this type of programming.

After BioParser has recognized all the items in a data record, these are stored on disk in a Perl object format. Each data item is labeled with a descriptive name. This makes it very easy for scientists to retrieve exactly the data items that they desire. Sample Perl programs for doing so will be provided to users, along with lists of all the label names of the data items. To utilize BioParser in their everyday work, a programmer or research scientist will merely have to:

- Download the latest BioParser object version of a desired database
- Change the sample code by removing names of undesired items of data and inserting names of data fields they desire. Additional selection criteria may also be easily specified, such as searching for only human information.
- Run the changed code. It will quickly produce a file or report containing exactly the desired information. This is much faster than the hours required for a computer program to parse through a downloaded database text file.

John Pearson, Head of the Bioinformatics Unit at TGen, has created the overall design and structure of the system of BioParser programs. He pioneered use of Damian Conway's Recursive Descent Parser for parsing the bioinformatics database files, as well as designing a consistent set of object-oriented Perl programs for parsing, testing code, and storing the

parsed data objects. The programs are implemented in Unix, but could easily be converted to Windows. [CVS \(Concurrent Versioning System\)](#) is used to manage them (9). There is an organized, sophisticated set of testing programs that can be easily called upon to test parse sample data from all the databases which have been added to the system.

This report details the development of modules that parse the [Homologene](#) (10) and [dbEST](#) (11) databases. This involved study and application of the sophisticated programming techniques used in BioParser. The project required use of object-oriented Perl, Damian Conway's Recursive Descent Parser, techniques for handling large database files, the CVS Concurrent Versioning System to control versions and automatically generate documentation, a testing harness, Unix, hashes in Perl, and analysis of the structure of major online bioinformatics database files.

## **1.1 Significance**

This project advances BioParser closer to commercial release by programming 2 additional database modules. When released, BioParser will be a valuable tool for scientists who wish to do bioinformatics research in the online databases. Revenue from it will provide funding for additional TGen projects.

## 2 Methods

A project of this size needs powerful software tools and much systematic organization.

The major tools and strategies of the BioParser project are:

- Object-Oriented Perl Programming
- Damian Conway's Parse::RecDescent Parsing Program
- CVS, the Concurrent Versioning System
- Use of a Testing Harness
- Automatic Generation of Documentation
- Systematic Organization and Naming of files
- UNIX operating system and VPN access

### ***2.1 Object-Oriented Perl Programming***

BioParser is written in the [Perl programming language](#) (6). Perl is an ideal choice for this work because it offers regular expressions for parsing strings of information. It is also a powerful, object-oriented programming language that integrates well with the underlying UNIX operating system. Further, the parsing program chosen as the best one for this project is already written in Perl. Parsers generated with it are Perl objects, specifically, blessed hash references. Lastly, when the sample programs are distributed to bioinformaticians and scientific researchers, many of them will already be familiar with Perl, as it is the most popular programming language in this academic area.

Use of object oriented programming in these programs greatly reduces the coding burden. John Pearson programmed hierarchies of objects, ranging from generic ones that work for all the databases down to specific ones for each individual database. This allows data objects to be individually generated, then handled generically for common tasks such as output to data files.

## **2.2 Damian Conway's *Parse::RecDescent* Parsing Program**

The key tool in the BioParser project is Damian Conway's recursive descent parser, [Parse::RecDescent](#) (7, 8). This is available at no cost as an Internet download. This parser generator, written in the Perl programming language, takes in a "grammar" written in a style similar to Perl regular expressions. From this, Conway's code generates parser code. The parser then follows the rules of the grammar as it attempts to recognize the various items in a data file. When items have been recognized, they are returned in a Perl hash. Then the programmer can specify what to do with the information. Typically it is written to another file in a labeled object format.

`Parse::RecDescent` is better than other parsers because it requires only one program to be written, which contains the grammar and descriptions of each token. Most other parsers require three separate programs; a lexer to split the data into labeled tokens, a parser to digest the grammar, and a custom program to import and analyze the results from the other two (8).

The term "grammar" is used because this process is analogous to an English student using definitions of the possible parts of a sentence to parse English sentences. A simplified English sentence grammar might define that a sentence consists of a subject and a predicate, followed by a sentence-ending punctuation mark such as a period or question mark. The subject must always contain a noun, and it might contain from zero to many adjectives that describe the noun. It might also contain zero or one definite articles, placed at the beginning of the subject. The predicate must contain a verb, and it might also contain from zero to many adverbs, as well as zero or one direct object. A direct object must contain a noun, and might

also contain zero to many adjectives. It might also contain zero or one definite articles, placed at the beginning of the direct object.

In the English sentence below, a student might recognize the following parts:

*The quick brown fox jumped over the lazy dog.*

Sentence:

Subject: *The quick brown fox*

Definite article: *the*

Adjectives: *quick, brown*

Noun: *fox*

Predicate: *jumped over the lazy dog*

Verb: *jumped*

Adverb: *over*

Direct Object: *the lazy dog*

Definite article: *the*

Adjective: *lazy*

Noun: *dog*

Ending Punctuation Mark: *period*

Here is the Perl code to create a parser and to parse this sentence:

```
#!/usr/bin/perl -w
use Parse::RecDescent;
my $grammar = q{

sentence:      <skip:'[ \t,]*'>
               subject
               predicate
               '.'
               {print "valid sentence recognized.\n"}

subject:       definite_article(?)
               adjective(s?)
               noun
               {print "valid subject recognized.\n"}

definite_article: 'a'|'A'|'The'|'the'
                 {print "definite article recognized.\n"}
adjective:     'slow'|'quick'|'lazy'|'hard-working'|'brown'
                 {print "adjective recognized.\n"}
noun:          'fox'|'dog'|'cat'|'fish'
                 {print "valid noun recognized.\n"}

predicate:     verb
               adverb(s?)
               direct_object(?)
               {print "valid predicate recognized.\n"}
```

```

verb:          'sprinted' | 'jumped'
               {print "valid verb recognized.\n"}
adverb:       'over' | 'under' | 'quickly' | 'slowly'
               {print "valid adverb recognized.\n"}
direct_object: definite_article(?)
               adjective(s?)
               noun
               {print "direct object recognized.\n"}

};
my $parser = new Parse::RecDescent($grammar);
undef $/;
my $text = "The quick, brown fox jumped over the lazy dog.";
$parser->sentence($text);

```

One can readily appreciate some of the difficulties in writing a computer program to recognize the parts of an English sentence. Definite articles might or might not be present. Adjectives and adverbs might occur zero, one, or multiple times. The direct object portion is similar to the subject, and might be confused with it. The direct object might be present in a sentence, or it might not. Inside the direct object, items such as a definite article and adjectives might or might not be present.

Furthermore, the grammar rules presented above are insufficient to parse more complex English sentences such as this. Reality is much more complex.

Note that the parsing layout above starts by attempting to recognize an entire sentence. It then “descends” in the parsing process. It knows that if it can recognize a subject, predicate, and ending period, it has the required pieces to recognize a sentence. It then descends further by attempting to recognize the parts of a sentence subject. If the parser can descend through all the layers in all the sentence parts and succeed in recognizing everything in the input sentence, it reports back that it has recognized a valid sentence. If it fails anywhere in the parsing process, it must back up and attempt to recognize the failed sentence section using a later definition in the grammar (8).

The parsing of a text file from an online bioinformatics database is analogous to the process described above. In online bioinformatics database file records, attempts are made to simplify parsing by providing fixed formats and/or labels on specific data items. However, major or minor sections of a record might or might not be present, and might repeat many times. For example, a list of authors who have written about this sequence might include zero, one, or many names. It is often difficult for a computer program to recognize when a section with optional repeats has ended and an optional new section has begun.

Because the format of each online database is unique, a separate grammar must be written for each one.

### ***2.3 CVS, the Concurrent Versioning System***

CVS (9) is an excellent system for keeping files straight when there are multiple programmers working on a large project. It supports a shared central server area that contains all finished code. It also supports code under construction in individual work areas assigned to different programmers. It keeps track of versions, notifies programmers of version conflicts, and attempts to reconcile conflicts. There is a process to “check out” files of code, so CVS knows which files are being worked on by each programmer.

### ***2.4 Use of a Testing Harness***

Use of a testing harness ensures methodical, thorough testing of all modules. The first testing hurdle that a new RDParse parsing program for another database must overcome is compiling correctly in Perl. Secondly, a test program is written for each program module and placed in the t testing directory. For example, the test program for Homologene’s RDParse is

named `bio_parser_homologene_rdparsers.t`. It just attempts to create a few objects which are passed up the program chain.

Thirdly, the `bpr_parse.pl` program is passed the name of the database, such as `homologene`, and the name of the test data input file on the command line. It knows to find the `RDParsers.pm` in the `homologene` directory, and the test data input file in the `/t/data` directory. This program then calls on the parsing grammar in `RDParsers.pm`, which is passed to Damian Conway's `Parse::RecDescent`. This generates code which then attempts to parse the data file. If all goes well, a hash containing all the labeled fields from each record is returned. If problems occur, debugging options and settings are available to facilitate debugging.

When a new program set for a database has been successfully debugged, it is committed to the shared CVS main directory for the project. At this point, the program is tested for object generation and the grammar is tested each time a system-wide makefile compilation is done.

## ***2.5 Automatic Generation of Documentation***

John Pearson implemented the standard [Perl "pod" documentation system](#) (12) to automatically generate documentation for each program. Pod stands for plain old documentation, which are basic text files generated from specially marked comment text embedded in the source code. The pod documentation system was also greatly extended through the `bpr_pod2html.pl` program, written by John Pearson. This program provides automatic generation of very nice-looking html web page documentation from the standard pod, augmented with some custom markers for extra features such as larger, bold type.

## **2.6 Systematic Organization and Naming of files**

A system for organizing and naming files makes it possible for the human programmers to keep track of the various types of program and data files. Further, systematic placement and naming facilitates the passing of objects to generic code, where they can be handled without duplicating the same code for different databases. For example, each online database has its own directory, named for that database, such as Homologene, placed within the \lib\Bio\Parser directory. Inside each database directory is a set of files:

- FileParser.pm reads the data file, extracts one record, and passes it to RDParse.pm.
- Record.pm is a file of support subroutines for FileParser.
- RDParse.pm parses the one data record one character at a time, recognizing the various data labels and items. It returns a Perl hash containing the labeled data items.
- SerialDatabase.pm is the program module which outputs the hashed record to a data file, in easily readable object format.

There are also organized directories and file naming conventions for test data, the testing harness programs, and the generic object-oriented programs that are accessed by the individual database modules. Besides making it easy for programmers to find programs in the large set of programs, this organization also enables the generic Perl modules to handle many tasks. It is sufficient that a given module be passed the name of the database being parsed, in order to identify the appropriate directory and file name required.

## **2.7 UNIX operating system and VPN access**

The [UNIX operating system](#) (13) is well designed for shared, secure access to huge projects. John Pearson and the TGen network support staff provided individual work areas for development and testing of code, plus shared areas in which finished code could be placed and accessed by all BioParser workers. When BioParser generated huge numbers of data object files, these were successfully handled by the UNIX server. They would have overwhelmed other operating systems.

TGen provided remote access to their Unix servers using [VPN](#), a Virtual Private Network (14). This allowed work from home computers. Use of the VPN protocol ensured that the files and the server would remain secure.

## 3 Results and Significance

### 3.1 Homologene

The first database considered is relatively simple. [Homologene](#) (10) is a database that groups similar genes in different species. A sample Homologene record is shown below:

```
41  9606 675  BRCA2      4502451  NP_000050.1
41  9598 452526  LOC452526 55639693  XP_509619.1
41  9615 474180  BRCA2      57104978  XP_543141.1
41  10090 12190  Brca2      6857765  NP_033895.1
41  10116 497682  LOC497682 62658293  XP_579511.1
41  9031 374139  BRCA2      45383586  NP_989607.1
41  7165 1270068  1270068  31199573  XP_308734.1
```

The first item in each line is the Homologene record ID. When a line begins with a different number, it signals the beginning of a new record, and thus the beginning of a new gene group. Group number 41 consists of genes from 7 different species, one on each line. The second item in each line is the taxonomy ID, which identifies the species listed on this line. The third item in each line is the gene ID. This is a unique number assigned to each gene sequence in a species.

The fourth item is the gene symbol. This consists of letters which identify the gene in this organism. The fifth item is the protein identifier number. This is used for looking up records on this gene in the Swiss-Prot database. The sixth and final item on each line is the accession number. This is another ID number used to locate a record on this gene in GenBank.

### 3.2 dbEST

The database [dbEST](#) (11) includes much more data for each record, and with much more variability. A sample record from dbEST is shown below. This database presents a difficulty to the parser because there are many fields which can appear each 0, 1, or many times. Thus, a parsing technique that utilizes the spaces embedded in the data file, as well as the embedded labels, to successfully recognize and parse records was used.

||

#### IDENTIFIERS

dbEST Id: 30612150  
EST name: BW974700  
GenBank Acc: BW974700  
GenBank gi: 71959383

#### CLONE INFO

Clone Id: PBL010097E04 (5')  
DNA type: cDNA

#### PRIMERS

PolyA Tail: Unknown

#### SEQUENCE

```
GAGACTCTTGAGAAGGCAGCTACGGAGGCTGCAGAGGTCTGGCAGGCCATGGAGGAGCCC  
CCTTTGCGAGAGGAGGAGGAGGAAGGGGACGAGGCGGGGCCCGAGGGGGCTCTGGGCAAG  
AGCCCCCTTCCAGCTGACAGCCGAAGACGTATATGACATCTCTTACGTGATGGGCCGAGAG  
CTGATGGCCCTGGGCAGCGACCCCCGGGTGACACAGCTGCAGTTCAAGATCGTCCGTGTT  
CTGGAGATGCTGGAGACGCTGGTGAATGAGGGCAACTTGACGGTGGAGGAGCTGAGAATG  
GAGCGGGACAACCTCAGGACGGAGGTGGAGGGGCTGCGGAGAGAGGGCTCCGCGGCCGGC  
GGAGAGGTGAACCTGGGACCAGACAAAATGGTGGTTGACCTGACAGATCCCAACCGACCA  
CGCTTTACTCTGCAGGAGCTGAGGGATGTGCTACAGGAGCGCAACAACTCAAGTCGCAG  
CTGCTGGTGGCACAGGAGGAGCTGCAGTGCTATAAGAGTGGCCTGATTCCACCAAGAGAA  
GGCCCAGGAGGAAGAAGAGAAAAAGATACTCTGGTTGCTCGGGCCAACAATGCCAGGAGT  
AACAAGGAGGAGAAGACAATCATAAGGAAGCTGTTCTCTTTCCGGATCAGGGAAGCAGACA  
TAGATCTGAGGCCACGACTAAATTCTCAGACTCAGAAAACAGCTCACAAAGACAACCTCC  
AAAATCATCTCTCAGTGCCACGCGTACCCACTGCACATGCTGCTTTGTTTCTCCTCAAAG  
CTGTCTGAGGAGGAAGGGGAAACGTTTTCTCCCTAGCTGCAGAACTGGACACCCTTGAAG  
GCTGGGCCAGAGCAGA
```

Entry Created: Aug 8 2005

Last Updated: Aug 8 2005

#### COMMENTS

EST project with full-length enriched cDNA libraries carried out in Animal Genome Research Program (Japan) by National Institute of Agrobiological Sciences and STAFF-Institute Single pass sequencing of clones derived from oligo-capped

cdNA library  
Vector sequences were eliminated by RepeatMasker version  
2002/07/13 and crossmatch version 0.990319  
Low quality bases were trimmed based on the quality values

LIBRARY

dbEST lib id: 15103  
Lib Name: full-length enriched swine cdNA library, adult peripheral  
blood mononuclear cell  
Organism: Sus scrofa  
Tissue type: peripheral blood mononuclear cell  
Develop. stage: adult

SUBMITTER

Name: Hirohide Uenishi  
Lab: Animal Genome Laboratory, Genome Research Department  
Institution: National Institute of Agrobiological Sciences  
Address: 2 Ikenodai, Tsukuba, Ibaraki 305-8602, Japan  
Tel: +81-29-838-8627  
Fax: +81-29-838-8627  
E-mail: huenishi@affrc.go.jp

CITATIONS

Medline UID: 14681463  
Title: PEDE (Pig EST Data Explorer): construction of a database for  
ESTs derived from porcine full-length cdNA libraries  
Authors: Uenishi,H., Eguchi,T., Suzuki,K., Sawazaki,T., Toki,D.,  
Shinkai,H., Okumura,N., Hamasima,N., Awata,T.  
Citation: Nucleic Acids Res. 32 (1): D484-D488 2004

MAP DATA

||

The normal style of BioParser grammars is to use a special skip command provided in `Parse::RecDescent` at the beginning of each parser grammar. It automatically skips all spaces and tabs. However, the format of this database is sufficiently complex, with so many items that could be present 0, 1, or many times, that the parser was unable to complete parsing. An example of this is the additional lines that might be present 0, 1, or many times after the `Lib Name:` line. This is shown in red in the data example, which has 1 additional line in this case. With the spaces removed, the parser was unable to distinguish between additional library name lines and the `Organism` line.

Changing the skip command to only remove tabs, and leave in all spaces means that the parser code has to individually cope with every space. The parser code to recognize the dbEST library name counts manual spaces inserted after `Lib Name:`

```
lib_name:          'Lib Name:          ' WL EOL
```

This allows use of an approach whereby the search is for 16 leading spaces at the beginning of a library name continuation line:

```
lib_line:          space16 WL EOL
```

The `space16` token is defined as a string of 16 spaces. Similarly, `WL` (a Whole Line of text) is defined as a string of characters that does not contain an end-of-line character. The search ends when the parser encounters the end-of-line character. `EOL` is defined as an end-of-line character.

```
space16:          '                  '
WL:               /^[^\n]*/          { $item{__PATTERN1__} }
EOL:              /\n/              { '<EOL>' }
```

Use of this coding technique enables the parser to recognize the end of the additional library name lines, because the line after the library name lines starts with 'Organism:' and not 16 spaces.

The general strategy in the parser grammar code is to return the recognized items as part of a Perl hash. After each item is identified, Perl code is written between curly brackets { }. This defines what actions should be executed in Perl upon recognition of this item. The BioParser action strategy is to insert each item into a large hash, the address of which becomes the return from the entire parsing operation. Here is some typical code from the dbEST grammar, with line numbers added to facilitate this discussion:

```
1   lib_name:   'Lib Name:   ' WL EOL
2             lib_line(s?)
3             {my $libname = $item{'WL'};
4             if (@{$item[4]} > 0) {
5                 $libname .= ' '.join(" ", @{$item[4]});
6             }
7             $est{'lib_name'} = $libname;
8             }
9
10  lib_line:   space16 WL EOL
11            { $item{'WL'} }
12
13  organism:   'Organism:   ' WL EOL
14            { $est{'organism'} = $item{'WL'} }
```

The first line of code, as discussed previously, defines a lib\_name as consisting of the literal text, "Lib Name:" followed by 7 spaces, then a Whole Line of text string that is terminated by an EOL, then an EOL. But the definition of lib\_name continues onto line 2. It may include from 0 to many lib\_line continuation lines. These are defined in line 10. Line 11 specifies that if a

lib\_line continuation line is recognized, the Whole Line string (WL) should be returned. It is returned back up to line 2. In line 3, the Whole Line of text from the primary lib\_name line is assigned to the variable \$libname. In line 4, a check is made if any additional lib\_lines were found. These would be the fourth item recognized in the lib\_name definition. If the \$item[4] position in the @item array is found to be nonempty, then the additional lib\_line is joined to the main \$libname with a space between. Finally, in line 7 the main Perl hash, called \$est in this case, is given the assembled library name text, with the label of 'lib\_name' in the hash.

Lines 13 and 14 show a much simpler definition of a data field, organism. It is simple because there can only be exactly one organism line. The text from it is extracted into WL and inserted into the Perl hash with an 'organism' label.

### ***3.3 Significance of the Results***

This project provided code for 2 more databases of the 13 that are planned to be parsed for commercial release. BioParser promises be a very useful commercial scientific tool someday, perhaps within a year (4).

## **4 Conclusions and Future Directions**

### ***4.1 Conclusions***

Program modules for two more databases in the BioParser set of programs, Homologene and dbEST, have been completed. The sophisticated parsing techniques developed previously have been further extended to handle the complex variability in dbEST data records.

### ***4.2 Future Directions***

The BioParser Feasibility Study (4, page 5) recommended that the remaining code modules be finished in 2006, using the help of interns from ASU's Computational Biosciences program. A full Business Plan should be prepared. This could be done by a summer intern from the ASU MBA program who is specializing in Biotechnology Business.

For an investment of approximately \$15,000 it could then be commercially marketed by a company set up as a for-profit spinoff from the nonprofit TGen (4, page 21). This initial investment figure is so low because TGen will provide personnel and office space. The money would purchase a separate network server, advertising materials, sales travel, website registration programming, legal services and registrations.

Eventually the profits from sale of annual BioParser licenses to commercial firms and government agencies should provide significant revenues to TGen. These could be used to fund other internal TGen research projects.

Also, as the number of paying customers grows, additional databases could be programmed. Full-time staff and facilities would eventually be established for supporting BioParser and its customers.

## References

- 1) GenBank: <http://www.ncbi.nlm.nih.gov/Genbank/>
- 2) Swiss-Prot: <http://www.ebi.ac.uk/swissprot/>
- 3) BioParser: <http://bioinformatics.tgen.org/brunit/software/bioparser/>
- 4) Barner, Carol, Gholba, Sumeda, Goldberg, Loretta, Gupta, Pankaj, and Revollo, Julio, *Venture Feasibility Study for BioParser*, prepared for Professor Bradford Kirkman-Liff, HSM 591, December, 2005.
- 5) BioPerl: <http://www.bioperl.org>
- 6) Perl: <http://www.perl.com/>
- 7) Parse::RecDescent: <http://search.cpan.org/~dconway/Parse-RecDescent-1.94/lib/Parse/RecDescent.pod>
- 8) Parse::RecDescent tutorial: <http://search.cpan.org/src/DCONWAY/Parse-RecDescent-1.94/tutorial/tutorial.html>
- 9) CVS : [http://ximbiot.com/cvs/wiki/index.php?title=Main\\_Page](http://ximbiot.com/cvs/wiki/index.php?title=Main_Page)
- 10) Homologene : <http://www.ncbi.nlm.nih.gov/entrez/query.fcgi?db=homologene>
- 11) dbEST: <http://www.ncbi.nlm.nih.gov/dbEST/>
- 12) Perl pod : <http://perldoc.perl.org/perlpod.html>
- 13) UNIX: <http://www.unix.org/>
- 14) VPN: <http://www.vpnlabs.com/>