

Using Perl for Bioinformatics
Module 2: BioPerl

Jason Stajich, jason.stajich@duke.edu

August 3, 2001

Contents

1	This Document	1
2	Bioperl Basics	1
2.1	Bioperl Semantics	1
2.1.1	Bioperl Name space	1
2.1.2	Interfaces	1
3	Sequence Analysis	2
3.1	Sequence Objects	2
3.2	Sequence Annotation	2
3.3	Sequence Input/Output	3
4	Sequence Databases	3
4.1	Bioperl database interfaces	3
4.1.1	Local databases and Index files	3
4.2	Remote databases	4
4.3	SQL databases	4
5	Sequence Analysis: Parsing and Running	4
5.1	Parsing Analysis	4
5.1.1	BLAST parsing	4
5.2	Multiple Sequence Alignment Parsing	5
5.2.1	Gene prediction parsing	6
5.3	Generalize Parsing of Analysis	6
5.4	Running Analyses	7
5.5	BLAST	7
5.6	ClustalW and TCoffee	7
6	Summary	7

1 This Document

This document contains a description of the Bioperl toolkit from a software design perspective. Classes will be denoted like Interfaces are denoted *Bio::Root::RootI* while implementations are denoted **Bio::Seq**. Function names will be denoted *new()* while perl built-in functions denoted *split*.

The objects and interfaces will be described in general terms, for detailed lists of methods and their descriptions please see the documentation embedded in the module files available through the perldoc command or online at <http://bio.perl.org/Doc/Latest/Core/>.

The bioperl toolkit is constantly evolving and this document describes a snapshot of the current object model with discussion of ideas in upcoming releases. The code released in the future will be a derivative of the current code base but may not have the same limitations or conventions as the 0.7 stable branch or 0.9 developer release series.

2 Bioperl Basics

The Bioperl toolkit is one of the oldest open source bioinformatics toolkit projects and the first Open Bioinformatics Foundation Bio{*} project. It was started in 1995 by a group of scientists tired of rewriting BLAST and sequence parsers for various formats. The project has since grown to a 600+ subscriber email list, 10-15 main developers coordinated by a core group of 4 developers, and deployment in a variety of industry, academic, and governmental environments. It currently has modules focused on sequence format parsing, sequence analysis report parsing, and annotation with an object model to support most types of data formats in these realms. It is an ever growing and expanding project approaching a mature and stable 1.0 release by the end Q4 in 2001.

Being an open-source project, anyone can jump in and make contributions to improve the system. Simply sign up on the mailing list, introduce yourself and your idea of a new module or an improvement to an existing one and we'll try and make suggestions to help steer you in the direction to help sustain the consistency of the toolkit and get your ideas implemented.

This course will dive into the basics of the Bioperl Toolkit, give you experience using some of the modules, and help you approach some of your own problems with a new set of tools at your disposal. It is assumed that you understand basic perl programming, know how to "use" modules in your code, and can take a stab at writing your own module.

2.1 Bioperl Semantics

Perl is not an inherently Object Oriented (OO) language. In fact one has to be fairly careful to make perl behave like an object oriented system. This requires a diligent programmer and some fairly straightforward rules. The Bioperl package takes advantages of the OO constructs to create a consistent, well documented, and somewhat well thought out object model for interacting with biological data in the life sciences.

2.1.1 Bioperl Name space

In perl namespaces are not enforced, but help the programmer and user make sense of where things are installed. The bioperl package installs everything in the Bio:: namespace. (In perl sub-name spaces are the same as a directory structure and are separated by :: and are equivalent to the way hierarchies are built in java with '.').

2.1.2 Interfaces

For those familiar with software design, an Interface is a definition for a set of methods that a class must implement. Perl does not have an explicit system for forcing interfaces on to a class, but there are some basic tricks one can play to enforce that a class implement an interface.

In Bioperl the *Bio::Root::RootI* is the top-level Interface that all Bioperl objects derive from. It contains methods for debugging and throwing errors (**throw**, **warn**), parsing named input parameters (**_rearrange**), and the ability to build chained constructor and destructor methods (for help with OO inheritance).

Each class of data or function has its own interfaces defined. For example, Sequence data has the *Bio::PrimarySeqI* interface while annotated sequence data has the *Bio::SeqI* interface which is derived from the *Bio::PrimarySeqI* interface. Not all classes in Bioperl have a properly defined interface file as of the 0.7.x releases because sometime it takes some experience with the data to derive a proper summary object to represent it. The 1.0 release should have well defined interfaces for all major components of the toolkit.

3 Sequence Analysis

Most of the work in Bioperl has been centered around supporting sequence analysis. Therefore a well defined set of objects has been created to handle sequence, annotation, parsing and producing sequence files, alignment, and analysis of sequence data.

3.1 Sequence Objects

The *Bio::PrimarySeqI* interface defines the basic methods for interacting with sequence data. Methods such as **seq()** to get and set the sequence string, **trunc()** to return a truncated version of the sequence, **revcom** to get the reverse complement, and **translate** to get the protein for a RNA or DNA sequence. The interface is implemented by the **Bio::PrimarySeq** object which represents sequences as in-memory strings. The **Bio::Seq::LargePrimarySeq** implements the *Bio::PrimarySeqI* interface as well but stores the sequence data as a file since it is designed to handle sequences that are too large to fit completely in memory. Because of OO inheritance one can use both of these implementations identically and not worry about how the underlying methods are implemented.

The *Bio::SeqI* interface inherits from *Bio::PrimarySeqI* and defines additional methods for storing and retrieving sequence annotation in the form of sequence features (*Bio::SeqFeatureI*) and annotation objects (**Bio::Annotation**). The methods for manipulating sequence features include **top_SeqFeatures()**, **all_SeqFeatures()**, **feature_count()**. The **annotation()** provides access to get and set the associated Annotation object. The **primaryseq()** hints at the delegation pattern of the *Bio::SeqI* objects which delegate all the sequence related methods to an underlying *Bio::PrimarySeqI* object. The **species()** method allows one to associate a **Bio::Species** object with a Sequence.

The aforementioned **Bio::Seq::PrimarySeq** implements the *Bio::PrimarySeqI* interface and manages the large sequence strings as files. The **Bio::Seq::LargeSeq** is analogous to the **Bio::Seq** object in that it manages the annotation but delegates the sequence data specific methods to a **Bio::Seq::LargePrimarySeq** object.

The *Bio::Seq::RichSeqI* defines additional methods for properly storing all the related sequence data in the GenBank/EMBL data format. It is implemented by the **Bio::Seq::RichSeq** object.

3.2 Sequence Annotation

The *Bio::SeqFeatureI* interface represents the concept of an annotated sequence feature which has a location on a sequence. Most sequence features are generic to be handled by **Bio::SeqFeature::Generic**. Essentially anything that can be stored in the GFF format can be represented by this object. However, more complicated objects such as HSP's from BLAST or FASTA reports need to contain information about the Query sequence location and the Subject (Hit) location and so must be the composition of two SeqFeatures. To this end there are **Bio::SeqFeature::FeaturePair** objects for paired Features and a subclass of these called **Bio::SeqFeature::SimilarityPair** which can handle the notion that two Features have similarity information about each other. These object will continue to evolve as the Bioperl project defines more general objects to handle all types of sequence database searches.

The **Bio::Annotation** object handles the representation of DBlinks, Literature References, and miscellaneous comments. *Annotations* are different than *SeqFeatures* because annotations do not have the concept of a location, they are a description that apply to the entire sequence. Annotations are also more specialized to handle the concept of linking unique ids for a sequence to databases.

3.3 Sequence Input/Output

Input and output or IO of sequence data can be the bane of biologists trying to interact with bioinformatics tools. The *Bio::SeqIO* system was designed to make getting and storing sequences to and from the myriad of formats as easy as possible. To create a **Bio::Seq** object from a sequence file in genbank, one can do this in just a few lines of perl code using the Bioperl toolkit. Figure 3.3 demonstrates this below.

```
use Bio::SeqIO;
my $seqio = new Bio::SeqIO('-format' => 'genbank', '-file' => 'filename.gb');
my $seqout = new Bio::SeqIO('-format' => 'embl', '-file' => '>filename.embl');
while( my $seq = $seqio->next_seq) { $seqout->write_seq($seq) }
```

Figure 3.3: A 3 line sequence converter

The sequence parsing system will read rich sequence formats like EMBL or GenBank and will create the associated features that are annotated on a sequence. Additionally the full set of features may be written back out in the same or different format. Obviously formats which do not support the notion of features (FASTA, GCG, Raw) will not contain as much information as an EMBL flatfile for the same sequence. Table 3.3 lists the available formats in the Bioperl 0.7 release series.

- Ace - AceDB format
- EMBL - European Molecular Biology Laboratories sequence format
- Fasta - Bill Pearson's FASTA sequence database format
- GAME - Genomic Annotation Marker Environment an XML format for BDGP
- GenBank - NCBI sequence format
- GCG - GCG sequence format
- largefasta - for reading in FASTA files of very long sequences (>100MB)
- raw - raw sequence
- scf - SCF chromatogram format
- swiss - SwissProt format

Table 3.3: Supported sequence formats by the *Bio::SeqIO* system

4 Sequence Databases

When processing many sequences, one often needs to store and retrieve these sequences from databases rather than a collection of flatfiles. The bioperl project provides interfaces for storing and retrieving from locally indexed databases (indexing speeds up the accessing of the data but requires building a file larger than the data itself) and from remote sequence databases such as EMBL, GenBank, and SwissProt.

4.1 Bioperl database interfaces

The Bioperl DB interface is encapsulated in the *Bio::DB::SeqI* interface which describes mechanisms for retrieving a sequence from a database, either by accession number, id number, or as a *Bio::SeqIO* data stream.

4.1.1 Local databases and Index files

The *Bio::Index::Abstract* object describes how one can build local indexes of sequence data. The indexing system uses the DBM database system for building local indexes. The Index objects derived from the *Bio::DB::SeqI* interface which allows the indexes to be queried exactly like any other database.

4.2 Remote databases

One may also want to query a remote database such as NCBI's GenBank. The module **Bio::DB::GenBank** allows queries for sequences based on a single or multiple accession numbers or genbank ids. Additionally there are similar modules written to query the NCBI's GenPept database of proteins, EMBL's database of nucleotide and protein sequence, and SwissProt's database of curated protein sequences.

4.3 SQL databases

The bioperl-db project (to be renamed biosql project) is a perl layer on top of a relational database which allows access to a database of sequence data. The open-source database mysql was chosen at the first implementation but a postgres, sybase, oracle, or any other RDB could substitute. The database is intended to be able to store and fetch sequences and effectively represent annotation and related information as rich as the GenBank/EMBL format. Additional work is being done . More details available at the bioperl website and the code is available under the bioperl-db cvs module name.

5 Sequence Analysis: Parsing and Running

Parsing sequences and annotation is useful for a certain class of analysis such as looking for simple patterns or small custom written analyses, but typically one will want to rely on other software such as BLAST or ClustalW to do sophisticated analysis. Bioperl provides mechanisms for both parsing the output from these programs and for certain applications an interface for executing the programs within a perl script.

5.1 Parsing Analysis

Parsing sequence analysis is critical for linking together results from a collection of analyses and building suitable pipelines for analysis. Due to the nature of the bioperl development done by different developers solving their own specific tasks, the interfaces for analysis parsing are still evolving. The current state of the parsers are that each type is typically its own beast and it requires the users to read the documentation before beginning to use a parser. Hopefully this guide and the descriptions below will make this process easier for new users wanting to parse their results.

5.1.1 BLAST parsing

BLAST is the staple of every bioinformatician's toolchest so a suitable parser for the format is a necessity. Currently two modules attempt to address this need. An older, complex module called **Bio::Tools::Blast** is an extensible and robust object that handles a wide variety of tasks including parsing and writing HTML-ified Blast reports, and **Bio::Tools::BPlite** a simpler less sophisticated parser that handles NCBI and WuBlast results. Modules BPbl2seq and BPpsilite use objects from the BPlite system handle bl2seq and psi-blast results respectively.

The author recommends the BPlite module for those just starting out with BLAST parsing. The example in Figure 5.1.1 should give the reader a good idea of how to get started.

```
use Bio::Tools::BPlite;

my $parser = new Bio::Tools::BPlite('-file' => 'blast.report');
my $pvalue = 1e-3;
SBJCT: while( my $sobjct = $parser->nextSbjct ) {
    my ($id) = split(/\s+/, $sobjct->name);
    while( my $hsp = $sobjct->nextHSP ) {
if( $hsp->P > $pvalue) {
    # skip Sbjcts that don't meet the minimum P value
    print STDERR "skipping $id with a HSP with pvalue=", $hsp->P, "\n";
    next SBJCT;

```

```

}
print "Sbjct length is ", $hsp->subject->length, "\n";
}

# parse the ids if this a blast hit against an
# NCBI formatted library
my @ids = split(/\|/, $id);
# in NCBI libs the last one is the accession number
print "id is ", pop @ids, "\n";

}

```

Figure 5.1.1: Using **Bio::Tools::BPlite** for blast parsing

5.2 Multiple Sequence Alignment Parsing

Multiple sequence alignments are also a staple of bioinformatics research. Bioperl offers the *Bio::AlignIO* system for reading and writing MSA reports produced by a variety of sources include ClustalW and GCG. The system is analogous to the *Bio::SeqIO* system and provides a simple interface to parsing the data. The `next_aln()` method returns a **Bio::SimpleAlign** object which encapsulates the MSA alignment. One can ask for slices of the alignment at certain columns, overall statistics, or for the references to the individual sequences that make up the alignment. Future work in this area will establish an interface definition for this object and build a specialized version of the interface for sequence assemblies. Figure 5.2 demonstrates the use of the system to obtain alignments and query them. Table 5.2 describes the supported alignment formats the current *Bio::AlignIO* system.

- bl2seq
- clustalw
- fasta
- mase
- meme
- msf
- nexus
- pfam
- phylip
- prodom
- selex
- stockholm

Table 5.2: *Bio::AlignIO* supported formats

```

use Bio::AlignIO;
my $alignio = new Bio::AlignIO(-format => 'clustalw',
                               -file => 'alignment1.aln');

while( my $aln = $alignio->next_aln ) {
    print "len=", $aln->length,

```

```

" # residues=", $aln->no_residues, " ",
" percent id=", $aln->percentage_identity, "\n";
print "seqs are :\n";
foreach my $seq ($aln->each_seq) {
    print "\t '", $seq->display_id(), "'\n";
}
print "---\n";
}

```

Figure 5.2: The *Bio::AlignIO* system in use

5.2.1 Gene prediction parsing

Bioperl has also written parsers for a few standard gene prediction programs: *genscan* and *mzef*. These parsers can then produce **Bio::SeqFeature::GeneStructure** objects which can represent all the gene information returned by a prediction program. Future work is being done to write parsers for more formats and build gene objects from annotations in sequence files. For examples of using the parsers the reader is directed towards the POD and the bioperl tutorial available from the bioperl website.

5.3 Generalize Parsing of Analysis

The *Bio::SeqAnalysisParserI* interface describes a simple way to retrieve features from a parser for analyses that produce lists of features. It has a single iterator method **next_feature()** which must return a *Bio::SeqFeatureI* object. This is a limited interface which is still under development as the bioperl project is still trying to expand to handling a richer set of analysis types. Some of the objects described do not comply to the *Bio::SeqAnalysisParserI* interface but may in the near future.

For those that do comply we have created a Factory object which simplifies the code for a user. The **Bio::Factory::SeqAnalysisParserFactory** allows users to create an analysis parser with just one line of code and handle the creation of the specific parser object for you. The code in Figure 5.3 should show a good example of how to use it.

```

use Bio::Factory::SeqAnalysisParserFactory;
# initialize an object implementing this interface, e.g.
$factory = Bio::Factory::SeqAnalysisParserFactory->new();
# find out the methods it knows about
print "registered methods: ",
    join(', ', keys(%{$factory->driver_table()})), "\n";
# obtain a parser object
my $inputobj = 'seq.genscan';
my @params = ();
my $method = 'genscan';
$parser = $factory->get_parser(-input=>$inputobj,
    -params=>[@params],
    -method => $method);
# $parser is an object implementing Bio::SeqAnalysisParserI
# annotate sequence with features produced by parser
while(my $feat = $parser->next_feature()) {
    print "feature is ", $feat->gff_string(), "\n";
    # if you had the sequence object
    # $seq->add_SeqFeature($feat);
}

```

Figure 5.3: **Bio::Factory::SeqAnalysisParserFactory** in action

5.4 Running Analyses

5.5 BLAST

Support for running BLAST locally and remotely is built-in to bioperl through the modules **Bio::Tools::Run::StandAloneB** and **Bio::Tools::Run::RemoteBlast**. These modules allow one to either submit jobs to the NCBI blast queue or utilize a local blast executable and databases. Running BLAST locally requires setting some environment variables which are documented in the POD included with the module.

5.6 ClustalW and TCoffee

Multiple sequence alignment can be run locally on a set of sequences using the **Bio::Tools::Run::Alignment::Clustal** and **Bio::Tools::Run::Alignment::TCoffee** modules. These modules require the setting of an environment variable and local copies of the executables which is documented in the POD.

6 Summary

The bioperl toolkit is a collection of perl modules for bioinformatics programming. The dynamic nature of the toolkit's development may cause some of the information in this document to be out of date. Please join the mailing list if you find you want to learn more or consider joining the development team.