Feature and Annotation HOWTO

Brian Osborne

Cognia Corporation [http://www.cognia.com]

<brian_osborne-at-cognia.com>

2003-10-14

This is a HOWTO written in DocBook format that explains how to use the SeqFeature and Annotation objects of Bioperl.

Table of Contents

1. Introduction

There's no more central notion in bioinformatics than the idea that portions of protein or nucleotide sequence have specific characteristics. A given stretch of DNA may have been found to be essential for the proper transcriptional regulation of a gene, or a particular amino acid sequence may bind a particular ion. This simple idea turns out to be a bit more complicated in the bioinformatics world where there's a need to represent the actual data in all its varied forms. The promoter region may not be precisely defined down to the base pair, a transcribed region may be divided into discontinuous exons, a gene may have different numbered positions on different maps, a sequence may have a sub-sequence which itself possesses some characteristic, an experimental observation may be associated with a literature reference, and so on. This HOWTO describes aspects of Bioperl's approach. The problem is how to create software that accepts, analyzes, and displays any and all of this sequence annotation with the required attention to detail yet remains flexible and easy to use. The general names for the modules or objects that serve these purposes in Bioperl are SeqFeature and Annotation.

This HOWTO will discuss these objects and the differences between them. I'll also show how to parse files with these objects and discuss the basics of how to annotate sequence using the objects.

2. The Basics

Some Bioperl neophytes may also be new to object-oriented programming (OOP) and this notion of an object. OOP is not the subject of this HOWTO but I do want to show how objects are used in Bioperl. In the object-oriented world parsing a Genbank file doesn't give you data, it gives you objects and you can ask the object, a variable, for data. While annotating you don't create a file or database entry directly. You might create a "sequence object" and an "annotation object", then put these two together to create an "annotated sequence object". You could then tell this object to make a version of itself as a file, or pass this object to a "database object" for entry. In a sense a bit more complicated but in another way very flexible and logical, since each kind of data is treated independently.

The Bioperl authors use Perl in an object-oriented way so each module, or object, inherits at least some of its capabilities from another object, a parent. The OOP approach also allows new modules to modify or add functionality, distinct from the parent. Practically speaking this means that there's not one definitive SeqFeature or Annotation object but many, each a variation on a theme. The details of the these varieties will be discussed in other sections, but for now we could use some broad definitions that apply to all the variations.

A SeqFeature object is associated with a sequence, and can have a location on that sequence - it's a way of describing the characteristics of a specific part of a sequence. SeqFeature objects can also have features themselves, which you could call sub-features but which, in fact, are complete SeqFeature objects. SeqFeature objects can also have one or more Annotations associated with them.

An Annotation object is also associated with a sequence, of course, but it does not have a location on the sequence, so it's associated with an entire sequence. This is one of the important differences between a SeqFeature and an Annotation. Annotations also can't have SeqFeatures, which makes sense since SeqFeature objects typically have locations. The relative simplicity of the Annotation has made it amenable to the creation of a useful set of Annotation objects, each devoted to a particular kind of observation or attribute.

I mentioned locations, above. Describing locations can be complicated in certain situations, say when some feature is located on different sequences with varying degrees of precision. One location could also be shared between disparate objects, such as two different kinds of SeqFeatures. Because of these sorts of complexities and because one may want to create different types of locations the Bioperl authors elected to keep location functionality inside dedicated Location objects.

SeqFeatures and Annotations will make the most sense if you're already somewhat familiar with Bioperl and its central Seq and SeqIO objects. The reader is referred to the bptutorial [http://bioperl.org/Core/Latest/bptutorial.html], the module documentation, and the SeqIO HOWTO [http://bioperl.org/HOWTOs/html/SeqIO.html] for more information on these topics. Here's a bit of code, to summarize:

```
# BAB55667.gb is a Genbank file
use Bio::SeqIO;
my $seqio_object = Bio::SeqIO->new(-file => "BAB55667.gb );
my $seq_object = $seqio_object->next_seq;
```

Note

This object, `$seq_object`, is actually a `Bio::Seq::RichSeq` object - can a PrimarySeq object, the simple parent of all Sequence objects, have a feature or an annotation? No.

Now that we have a sequence object in hand we can examine its features and annotations.

3. Features from Genbank

When the file comes from Genbank it's easy to see where most of the features are, they're in the Feature table section, something like this:

```
FEATURES             Location/Qualifiers
     source          1..1846
                     /organism="Homo sapiens"
                     /db_xref="taxon:9606"
                     /chromosome="X"
                     /map="Xp11.4"
     gene            1..1846
                     /gene="NDP"
                     /note="ND"
                     /db_xref="LocusID:4693"
                     /db_xref="MIM:310600"
     CDS             409..810
                     /gene="NDP"
                     /note="Norrie disease (norrin)"
                     /codon_start=1
                     /product="Norrie disease protein"
                     /protein_id="NP_000257.1"
                     /db_xref="GI:4557789"
                     /db_xref="LocusID:4693"
                     /db_xref="MIM:310600"
                     /translation="MRKHVLAASFSMLSLLVIMGDTDSKTDSSFIMDSDPRRCMRHHY
                     VDSISHPLYKCSSKMVLLARCEGHCSQASRSEPLVSFSTVLKQPFRSSCHCCRPQTSK
                     LKALRLRCSGGMRLTATYRYILSCHCEECNS"
```

Features in Bioperl are accessed using their tags, either a "primary tag" or a plain "tag". Examples of primary tags in this text are "source", "gene", and "CDS". Plain tags include "organism", "note", "db_xref", and "translation".

When a Genbank file like this is parsed the feature data is converted into objects, specifically a `Bio::SeqFeature::Generic` object. In the other parts of the Bioperl documentation one finds discussions of the "SeqFeature object", but there's more than one of these, so what is this a reference to? More than likely it's referring to this same `Bio::SeqFeature::Generic` object. Think of it as the default SeqFeature object. Now, should you care what kind of object is being made? For the most part, no, you can write lots of useful and powerful Bioperl code without ever knowing these specific details.

Tip

By the way, how does one know what kind of object one has in hand? Try something like:

```
print ref($seq_object);
# results in "Bio::Seq::RichSeq"
```

The SeqFeature::Generic object uses tag/value pairs to store information, and the values are always returned as arrays. A simple way to access all the data in the features of a Seq object would look something like this:

```
foreach my $feat_object ($seq_object->get_SeqFeatures) {
   print "primary tag: ", $feat_object->primary_tag, "\n";
   foreach my $tag ($feat_object->get_all_tags) {
      print "  tag: ", $tag, "\n";
      foreach my $value ($feat_object->get_tag_values($tag)) {
         print "    value: ", $value, "\n";
      }
   }
}
```

This bit would print out something like:

```
primary tag: source
  tag: chromosome
    value: X
  tag: db_xref
    value: taxon:9606
  tag: map
    value: Xp11.4
  tag: organism
```

```
    value: Homo sapiens
primary tag: gene
  tag: gene
    value: NDP
  tag: note
    value: ND
primary tag: CDS
  tag: codon_start
    value: 1
  tag: db_xref
    value: GI:4557789
    value: LocusID:4693
    value: MIM:310600
  tag: product
    value: Norrie disease protein
  tag: protein_id
    value: NP_000257.1
  tag: translation
    value: MRKHVLAASFSMLSLLVIMGDTDSKTDSSFIMDSDPRRCMRHHYVDSI
           SHPLYKCSSKMVLLARCEGHCSQASRSEPLVSFSTVLKQPFRSSCHCC
           RPQTSKLKALRLRCSGGMRLTATYRYILSCHCEECNS
```

So to retrieve specific values, like all the database identifiers, you could do:

```
foreach my $feat_object ($seq_object->get_SeqFeatures) {
   push @ids,$feat_object->get_tag_values("db_xref")
        if ($feat_object->has_tag("db_xref"));
}
```

Important

Make sure to include that "if ($feat_object->has_tag(<tag>))" part, otherwise you'll get errors when the feature does not have the tag you're requesting.

One commonly asked question is "How do I get the sequence of a SeqFeature?" The answer is "it depends on what you're looking for". If you'd like the sequence of the parent, the sequence object that the SeqFeature is associated with, then use `entire_seq()`:

```
$seq_object = $feat_object->entire_seq;
```

This doesn't return the parent's sequence directly but rather a `Bio::PrimarySeq` object corresponding to the parent sequence. Now that you have this object you can call its `seq()` method to get the sequence string, or you could do this all in one step:

```
my $sequence_string = $feat_object->entire_seq->seq;
```

There are 2 other useful methods, `seq()` and `spliced_seq()`. Consider the following Genbank example:

```
FEATURES             Location/Qualifiers
     source          1..177
                     /organism="Mus musculus"
                     /mol_type="genomic DNA"
                     /db_xref="taxon:10090"
     tRNA            join(103..111,121..157)
                     /gene="Phe-tRNA"
```

To get the sequence string from the start to the end of the tRNA feature use `seq()`. To get the spliced sequence string, accounting for the start and end locations of each sub-sequence, use `spliced_seq()`. Here are the methods and the corresponding example coordinates:

Method	**Coordinates**
entire_seq()	1..177
seq()	103..157

Method	Coordinates
spliced_seq()	103..111,121..157

Table 1. Sequence string methods

It's not unusual for a Genbank file to have multiple CDS or gene features (and recall that 'CDS' or 'gene' could be primary tags in Bioperl), each with a number of tags, like 'note', 'protein_id', or 'product'. How can we get, say, the nucleotide sequences and gene names from all these CDS features? By putting all of this together we arrive at something like:

```
my $seqio_object = Bio::SeqIO->new(-file => $gb_file);
my $seq_object = $seqio_object->next_seq;

foreach my $feat_object ($seq_object->get_SeqFeatures) {
  if ($feat_object->primary_tag eq "CDS") {
    print $feat_object->spliced_seq->seq,"\n";
    # e.g. 'ATTATTTTCGCTCGCTTCTCGCGCTTTTTGAGATAAGGTCGCGT...'
    foreach my $val ($feat->get_tag_values('gene')) {
      print "gene: ",$val,"\n";
      # e.g. 'NDP', from a line like '/gene="NDP"'
    }
}
```

Many people wouldn't write code in the rather deliberate style I've used above. The following is a more compact code that gets all the features with a primary tag of 'CDS', starting with a Genbank file:

```
my @cds_features = grep { $_->primary_tag eq 'CDS' }
Bio::SeqIO->new(-file => $gb_file)->next_seq->get_SeqFeatures;
```

With this array of SeqFeatures you could do all sorts of useful things, such as find all the values for the 'gene' tags and their corresponding spliced nucleotide sequences and store them in a hash:

```
my %gene_sequences = map {$_->get_tag_values('gene'),
                          $_->spliced_seq->seq } @cds_features;
```

Because you're asking for a specific primary tag and tag, 'CDS' and 'gene' respectively, this would only work when there are features that looked something like this:

```
CDS             735..1829
                /gene="MG001"
                /codon_start=1
                /product="DNA polymerase III, subunit beta (dnaN)"
                /protein_id="AAC71217.1"
                /translation="MNNVIISNNKIKPHHSYFLIEAKEKEINFYANNEYFSVKCNLNK
                NIDILEQGSLIVKGKIFNDLINGIKEEIITIQEKDQTLLVKTKKTSINLNTINVNEFP
                RIRFNEKNDLSEFNQFKINYSLLVKGIKKIFHSVSNNREISSKFNGVNFNGSNGKEIF
                LEASDTYKLSVFEIKQETEPFDFILESNLLSFINSFNPEEDKSIVFYYRKDNKDSFST
                EMLISMDNFMISYTSVNEKFPEVNYFFEFEPETKIVVQKNELKDALQRIQTLAQNERT
                FLCDMQINSSELKIRAIVNNIGNSLEEISCLKFEGYKLNISFNPSSLLDHIESFESNE
                INFDFQGNSKYFLITSKSEPELKQILVPSR"
```

One last note on Genbank features. The Bioperl parsers for Genbank and EMBL are built to respect the specification for the feature tables agreed upon by Genbank, EMBL, and DDBJ (see Feature Table Definition [http://www.ncbi.nlm.nih.gov/projects/collab/FT/] for the details). Check this page if you're interested in a complete listing and description of all the Genbank, EMBL, and DDBJ feature tags.

Despite this specification some non-standard feature descriptors have crept into Genbank, like "bond". When the Bioperl Genbank parser encounters a non-standard feature like this it's going to throw a fatal exception. The

work-around is to use eval{} so you don't die, something like:

```
use Bio::SeqIO;
my $seq_object;
my $seqio_object = Bio::SeqIO->new(-file   => $gb_file,
                                   -format => "genbank");
eval { $seq_object = $seqio_object->next_seq; };
# if there's an error
print "Problem in $gb_file. Bad feature perhaps?\n" if $@;
```

4. Other objects

As an aside I should mention that certain data associated in a Genbank file is accessible both as a feature and through a specialized object. Taxonomic information on a sequence, below, can be accessed through a Species object as well as a value to the "organism" tag, and you'll get more information from the `Bio::Species object`.

```
SOURCE      human.
  ORGANISM  Homo sapiens
            Eukaryota; Metazoa; Chordata; Craniata; Vertebrata; Euteleostomi;
            Mammalia; Eutheria; Primates; Catarrhini; Hominidae; Homo.
```

You can create this Species object and use its methods or you can use the Perlish shorthand:

```
# legible and long
my $species_object = $seq_object->species;
my $species_string = $species_object->species;
# Perlish
my $species_string = $seq_object->species->species;
# either way $species_string is "Homo sapiens"
my $classification = $seq_object->species->classification;
# "sapiens Homo Hominidae Catarrhini Primates Eutheria Mammalia
# Euteleostomi Vertebrata Craniata Chordata Metazoa Eukaryota"
```

The reason that ORGANISM isn't treated only as a plain tag is that there are a variety of things one would want to do with taxonomic information, so returning just an array wouldn't suffice. See the documentation on `Bio::Species` for more information.

5. Annotations from Genbank

There's still quite a bit of data left our Genbank files that's not in a SeqFeature, and much of it is parsed into Annotation objects. In order to get access to these objects we can get an AnnotationCollection object, which is exactly what it sounds like:

```
my $io = Bio::SeqIO->new(-file => $file, -format => "genbank" );
my $seq_obj = $io->next_seq;
my $anno_collection = $seq_obj->annotation;
```

Now we can access each Annotation in the AnnotationCollection object. The Annotation objects can be retrieved in arrays:

```
foreach my $key ( $anno_collection->get_all_annotation_keys ) {
  my @annotations = $anno_collection->get_Annotations($key);
  foreach my $value ( @annotations ) {
    print "tagname : ", $value->tagname, "\n";
    # $value is an Bio::Annotation, and has an "as_text" method
    print "  annotation value: ", $value->as_text, "\n";
  }
}
```

It turns out the value of `$key`, above, and $value->tagname are the same. The code will print something like:

```
tagname : comment
  annotation value: Comment: REVIEWED REFSEQ: This record has been curated by
NCBI staff. The reference sequence was derived from X65882.1. Summary: NDP is the
genetic locus identified as harboring mutations that result in Norrie disease.
tagname : reference
  annotation value: Reference: The molecular biology of Norrie's disease
tagname : date_changed
  annotation value: Value: 31-OCT-2000
```

If you only wanted a specific annotation, like COMMENT, you could do:

```
my @annotations = $anno_collection->get_Annotations('comment');
```

The following is a list of some of the common Annotations and what they're derived from in Genbank files:

Genbank Text	Key	Object Type	Note
COMMENT	comment	Comment	
SEGMENT	segment	SimpleValue	e.g. "1 of 2"
ORIGIN	origin	SimpleValue	e.g. "X Chromosome."
REFERENCE	reference	Reference	
INV	date_changed	SimpleValue	e.g. "08-JUL-1994"
KEYWORDS	keyword	SimpleValue	
ACCESSION	secondary_accession	SimpleValue	2nd of 2 accessions

Table 2. Annotation Keys

Some Annotation objects, like Reference, make use of a `hash_tree()` method, which returns a hash reference. This is a more thorough way to look at the actual values than the `as_text()` method used above. For example, `as_text()` for a Reference object is only going to return the title of the reference, whereas the keys of the hash from `hash_tree()` will be "title", "authors", "location", "medline", "start", and "end".

```
if ($value->tagname eq "reference") {
  my $hash_ref = $value->hash_tree;
  foreach my $key (keys %{$hash_ref}) {
    print $key,": ",$ref->{$key},"\n";
  }
}
```

Which yields:

```
authors: Meitinger,T., Meindl,A., Bork,P., Rost,B., Sander,C., Haasemann,M. and
Murken,J.
location: Nat. Genet. 5 (4), 376-380 (1993)
medline: 94129616
title: Molecular modelling of the Norrie disease protein predicts a cystine knot
 growth factor tertiary structure
end: 1846
start: 1
```

Other Annotation objects, like SimpleValue, also have a `hash_tree()` method but the hash isn't populated with data and `as_text()` will suffice.

The simplest bits of text, like KEYWORDS, end up in these Annotation::SimpleValue objects, the COMMENT ends up in a `Bio::Annotation::Comment` object, and references are tranformed into `Bio::Annotation::Reference` objects. Some of these specialized objects will have specialized methods.

Take the Annotation::Reference object, for example:

```
if ($value->tagname eq "reference") {
  print "author: ",$value->authors(), "\n";
}
# as well as title(), location(), medline(), start(), and end()
```

6. Directly from the Sequence object

This is just a reminder that some of the "annotation" data in your sequence files can be accessed directly, without looking at SeqFeatures or Annotations. For example, if the Sequence object in hand is a Seq::RichSeq object then here are some useful methods:

Method	Returns
get_secondary_accessions	array
keywords	array
get_dates	array
seq_version	string
pid	string
division	string

Table 3. RichSeq methods

These `Bio::Seq::RichSeq` objects are created automatically when you use SeqIO to read from EMBL, GenBank, and SwissProt files. However, it's not guaranteed that each of these formats will supply data for all of the methods above.

7. Location Objects

There's quite a bit to this idea of location, so much that it probably deserves its own HOWTO. This is my way of saying that if this topic interests you should take a closer look at the modules that are concerned with both Location and Range. The Range modules offer the user a number of useful methods to handle "fuzzy" locations, where the "start" and "end" of a particular sub-sequence themselves have start and end positions, or are only partially defined.

In their simplest form the Location objects are used to get or set these start and end positions, or the multiple start and end positions of "split" locations, like the join statements in the CDS feature found in Genbank entries (e.g. "join(45..122,233..267)"):

```
foreach my $feature ($seqobj->top_SeqFeatures){
  if ( $feature->location->isa('Bio::Location::SplitLocationI')
                && $feature->primary_tag eq 'CDS' )  {
    foreach my $location ( $feature->location->sub_Location ) {
      print $location->start . ".." . $location->end . "\n";
    }
  }
}
```

8. Building your own annotated sequences

We've taken a look at getting data from SeqFeature and Annotation objects, but what about creating these objects when you already have the data? The `Bio::SeqFeature::Generic` object is probably the best SeqFeature object for this purpose, in part because of its flexibility.

```
use Bio::SeqFeature::Generic;
# create the feature and add additional data while initializing,
# an author and a note
my $feat = new Bio::SeqFeature::Generic(-start  => 10,
                                        -end    => 22,
                                        -strand => 1,
                                        -tag    => {author => 'john',
                                                    note   => 'TATA box' } );
```

The SeqFeature::Generic object offers the user a "tag system" for addition of data that's not explicitly accounted for by its methods, that's what the "-tag" is for, above. If you want to add your own custom data to a feature you could use the "-tag" tag or you could add values after the object has been created:

```
$feat->add_tag_value("match1","PF000123 e-7.2");
$feat->add_tag_value("match2","PF002534 e-3.1");

my @arr = $feat->get_all_tags;
foreach my $tag (@arr) {
  print $tag,":",$feat->get_tag_values($tag),"  ";
}
# prints out:
# author:john  match1:PF000123 e-7.2  match2:PF002534 e-3.1  note:TATA box
```

Since the value passed to "-tag" could be any kind of scalar, like a reference, it's clear that this approach should be able handle just about any sort of data.

Once the feature is created it can be associated with a sequence:

```
# first we want a Sequence object
my $seq_obj = Bio::Seq->new(-seq => "attccccttataaaatttttttttgaggggtggg");
# associate the sequence and the feature
$feat->attach_seq($seq_obj);
```

Once you have a feature you can add annotations to it using an AnnotationCollection object:

```
$db_link = new Bio::Annotation::DBLink();
$db_link->database('dbSNP');
$db_link->primary_id('2367');
$feat->annotation->add_Annotation('dblink',$db_link);
```

Note that the first argument to `add_Annotation()` is the tag name, 'dblink'.

What if you wanted to add an Annotation directly to a sequence? This is an operation similar to the one above. Assume you already have a sequence object, you'll create the Annotation object and simply add the object to the sequence object:

```
# first create an Ontology annotation
my $annterm = new Bio::Annotation::OntologyTerm(-label => 'ABC1',
                                                -tagname => 'Gene Name');
$seq_object->annotation->add_Annotation($annterm);
```

This is an interesting example because the OntologyTerm object was created with a tag name, so you don't need to specify it when you use the `add_Annotation()` method.

9. The different Annotation objects

There are a number of specialized Annotation objects, each designed to accomodate a particular type of data. Here is a table showing the existing types.

Description	Object
comment	Bio::Annotation::Comment

Description	Object
database identifier	Bio::Annotation::DBLink
simple fact	Bio::Annotation::SimpleValue
generic value	Bio::Annotation::SimpleValue
ontology term	Bio::Annotation::OntologyTerm
reference	Bio::Annotation::Reference

Table 4. The different Annotation objects

The best way to find out more about a particular Annotation is in the module's own documentation. You can access the complete module documentation online at http://doc.bioperl.org/.

10. Additional Information

If you would like to learn about representing sequences and features in graphical form take a look at the Graphics HOWTO [http://bioperl.org/HOWTOs/html/Graphics-HOWTO.html]. The documentation for each of the individual SeqFeature and Annotation modules is also very useful. If you have questions or comments that aren't addressed therein then write the Bioperl community at bioperl-l@bioperl.org.

11. Acknowledgements

Thanks to Steven Lembark for comments and neat code discussions.