
Feature and Annotation HOWTO

Brian Osborne, *Cognia Corporation* [<http://www.cognia.com>]
<brian_osborne-at-cognia.com>

This document is copyright Brian Osborne, 2003. For reproduction other than personal use please contact brian 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	1
2. The Basics	1
3. Features from Genbank	2
4. Getting Sequences	5
5. Location Objects	7
6. Other objects	7
7. Annotations from Genbank	8
8. Directly from the Sequence object	10
9. Other sequence file formats	10
10. Building your own sequences	12
11. Additional Information	13
12. Acknowledgements	15

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.

The HOWTO will discuss these objects and the differences between them. I'll also show how to get useful data from 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 touch on how objects are used in Bioperl. In the Bioperl world parsing a Genbank file doesn't give you data, it gives you an object and you can ask the object, a kind of 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. This is a very flexible and logical way to design a complex piece of software like Bioperl, since each part of the system can be created and evaluated separately.

A central idea in OOP is inheritance, which means that a child object can derive some of its capabilities or functionality from a parent object. 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 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 designed to be 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 as you'd expect but it does not have a location on the sequence, 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. You may also want to describe a feature with many locations, like a repeated sequence motif in a protein. 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/SeqIO/index.html>] for more information on these topics. Here's a bit of code, to summarize:

```
# BAB55667.gb is a Genbank file, and Bioperl knows that it
# is a Genbank file because of the '.gb' file suffix
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` [<http://doc.bioperl.org/releases/bioperl-1.4/Bio/Seq/RichSeq.html>] object - can a `PrimarySeq` [<http://doc.bioperl.org/releases/bioperl-1.4/Bio/PrimarySeq.html>] 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

I'll be focusing on the Genbank format but bear in mind that most of the code shown here will also work on other formats containing features or annotations (EMBL, Swissprot, BSML, Chado XML, GAME, KEGG, Locuslink, TIGR XML). When the entry 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="MRKHVLAASFMSLSLLVIMGDTDSKTDSSFIMDS DPRRCMRHHY
VDSISHPLYKCSSKMVLLARCEGHCSQASRSEPLVSFSTVLKQPFRSSCHCCRPQTSK
LKALRLRCSGGMRLTATYRYILSCHCEECS"
```

Features in Bioperl are accessed using their tags, either a "primary tag" or a plain "tag". Examples of primary tags and tags in this Genbank entry are shown below. You can see that in this case the primary tag is a means to access the tags and it's the tags that are associated with the data from the file.

Table 1. Tag Examples

When a Genbank file like the one above is parsed the feature data is converted into objects, specifically *Bio::SeqFeature::Generic* [<http://doc.bioperl.org/releases/bioperl-1.4/Bio/SeqFeature/Generic.html>] objects. How many? In this case 3, one for each of the primary tags.

In other parts of the Bioperl documentation one finds discussions of the "SeqFeature object", but there's more than one kind of these, as we'll see later, so what is this a reference to? More than likely it's referring to this same *Bio::SeqFeature::Generic* [<http://doc.bioperl.org/releases/bioperl-1.4/Bio/SeqFeature/Generic.html>] 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:

```
for my $feat_object ($seq_object->get_SeqFeatures) {
    print "primary tag: ", $feat_object->primary_tag, "\n";
```

```
    for my $tag ($feat_object->get_all_tags) {
        print "    tag: ", $tag, "\n";
        for 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: MRKHVLAASFMSLLVIMGDTDSKTDSSFIMSDPPRRCMRHHYVDSI
          SHPLYKCSSKMVLLARCEGHCSQASRSEPLVSFSTVLKQPFRRSSCHCC
          RPQTSKLKALRLRCSGGMRLTATYRYILSCHCEECS
```

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

```
for 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 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 com-

plete listing and description of all the Genbank, EMBL, and DDBJ feature tags.

Despite this specification some non-standard feature tags 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 your script doesn'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. Getting Sequences

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* [<http://doc.bioperl.org/releases/bioperl-1.4/Bio/PrimarySeq.html>] 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:

Table 2. Sequence retrieval methods

It's not unusual for a Genbank file to have multiple CDS or gene features (and recall that 'CDS' and 'gene' are common primary tags in Genbank format), 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:

```
use Bio::SeqIO;

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

for my $feat_object ($seq_object->get_SeqFeatures) {
    if ($feat_object->primary_tag eq "CDS") {
        print $feat_object->spliced_seq->seq, "\n";
# e.g. 'ATTATTTTCGCTCGCTTCTCGCGCTTTTGTAGATAAGGTCGCGT...'
        if ($feat->has_tag('gene')) {
            for 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 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 code 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
NIDILEQGSLIVKGKIFNDLINGIKEEIIITIQEKDQTLVLVTKKTSINLNTINVNEFP
RIRFNEKNDLSEFNQFKINYSLLVKGIIKKIFHSVSNNREISSKFNGVNFNGSNGKEIF
LEASDTYKLSVFEIKQETEPFDFILESNLLSFINSFNPEEDKSIVFYRKDNKDSFST
EMLISMDNFMISYTSVNEKFPEVNYFFEFEPETKIVVQKNELKDALQRIQTLAQNERT
FLCDMQINSELKIRAIVNNIGNSLLEEISCLKFEGYKLNISFNPSSLLDHIESFESNE
INFDFQGNISKYFLITSKSEPELKQILVPSR"
```

5. 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. Together these modules offer the user a number of useful methods to handle both exact and "fuzzy" locations, where the "start" and "end" of a particular sub-sequence are precise or themselves have start and end positions, or are not precisely defined. You'll also find methods like `union()` and `intersection()` that act on pairs of ranges. The table below is meant to illustrate some of the modules' capabilities.

Table 3. Location Examples

One type that might not be self-explanatory is 'WITHIN'. The example means "starting somewhere between positions 5 and 10, inclusive, and ending at 100". 'BETWEEN' is interesting - the example means "between 99 and 100, exclusive". A biological example of such a location would be a cleavage site, between two bases or residues, but not including them.

In their simplest form the Location objects are used to get or set start and end positions, getting the positions could look like this:

```
# polyA_signal      1811..1815
#                  /gene="NDP"
my $start = $feat_object->location->start;
my $end   = $feat_object->location->end;
```

By now you've figured out that the `location()` method returns a Location object - this object has `end()` and `start()` methods.

Another way of describing a feature in Genbank involves multiple start and end positions. These could be called "split" locations, and a very common example is the join statement in the CDS feature found in Genbank entries (e.g. "join(45..122,233..267)"). This calls for a specialized object, `SplitLocation`, which is a container for Location objects:

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

6. 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* [<http://doc.bioperl.org/releases/bioperl-1.4/Bio/Species.html>].

```
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 Perlsh shorthand:

```
# legible and long
my $species_object = $seq_object->species;
my $species_string = $species_object->species;
# Perlsh
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* [<http://doc.bioperl.org/releases/bioperl-1.4/Bio/Species.html>] for more information.

7. Annotations from Genbank

There's still quite a bit of data left in our Genbank files that's not in SeqFeature objects, and much of it is parsed into Annotation objects. Annotations, if you recall, are those values that are assigned to a sequence that have no specific location on that sequence. 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:

```
for my $key ( $anno_collection->get_all_annotation_keys ) {
    my @annotations = $anno_collection->get_Annotations($key);
    for 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');
```

And if you'd simply like all of the Annotations, regardless of key, you can do this:

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

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

Table 4. Genbank Annotations

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;
  for my $key (keys %{$hash_ref}) {
    print $key, ": ", $hash_ref->{$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 Genbank text, like KEYWORDS, end up in these `Annotation::SimpleValue` objects, the COMMENT ends up in a `Bio::Annotation::Comment` [http://doc.bioperl.org/releases/bioperl-1.4/Bio/Annotation/Comment.html] object, and references are transformed into `Bio::Annotation::Reference` [http://doc.bioperl.org/releases/bioperl-1.4/Bio/Annotation/Reference.html] 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";  
}
```

There's also `title()`, `publisher()`, `medline()`, `editors()`, `database()`, `pubmed()` and a number of other methods.

8. 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:

Table 5. RichSeq methods

These `Bio::Seq::RichSeq` [http://doc.bioperl.org/releases/bioperl-1.4/Bio/Seq/RichSeq.html] objects are created automatically when you use `SeqIO` to read from EMBL, GenBank, GAME, Chado XML, TIGR XML, Locuslink, BSML, KEGG, and SwissProt sequence files. However, it's not guaranteed that each of these formats will supply data for all of the methods above.

9. Other sequence file formats

It is worth mentioning other sequence file formats. The table below shows what sorts of objects, `Annotation` or `SeqFeature`, you'll get when you parse other sequence formats using `Bio::SeqIO`.

Table 6. Formats, SeqFeatures, and Annotations

How does one find out what data is in which object in these formats? In general the individual module documentation is not going to provide all the answers, you'll need to do some investigation yourself. Let's use an approach we used earlier to dissect a Locuslink entry in a file, "148.lf". Here's the file:

```
LOCUSID: 148  
LOCUS_CONFIRMED: yes  
LOCUS_TYPE: gene with protein product, function known or inferred  
ORGANISM: Homo sapiens  
STATUS: REVIEWED  
NM: NM_000680|4501960|na  
NP: NP_000671|4501961  
PROT: AAA93114|409029
```

```
ACCNUM: M11313|177869|na|na|na
TYPE: p
PROT: P35348|1168246
OFFICIAL_SYMBOL: ADRA1A
OFFICIAL_GENE_NAME: adrenergic, alpha-1A-, receptor
ALIAS_SYMBOL: ADRA1C
SUMMARY: Summary: Alpha-1-ARs are members of the GPCR superfamily.
CHR: 8
STS: SGC35557|8|8124|na|seq_map|epcr
COMP: 10090|Adrala|14|14 cM|11549|8|ADRA1A|ncbi_mgd
ALIAS_PROT: adrenergic, alpha-1C-, receptor
BUTTON: unigene.gif
LINK: http://www.ncbi.nlm.nih.gov/UniGene/clust.cgi?ORG=Hs&CID=52931
UNIGENE: Hs.52931
OMIM: 104221
MAP: 8p21-p11.2|RefSeq|C|
MAPLINK: default_human_gene|ADRA1A
GO: cellular component|integral to plasma membrane|P|GO:0005887|Proteome|8396931
```

First collect all the annotations:

```
use Bio::SeqIO;

my @annotations = Bio::SeqIO->new(-file => "148.ll", -format => "locuslink")->
    next_seq->annotation->get_Annotations;
```

And from this array of Annotations let's extract a hash containing the `as_text` strings as keys and the concatenated tagnames and object types as values:

```
my %tagname_type = map {$_->as_text,($_->tagname . " " . ref($_)) }
    @annotations;
```

The contents of the `%tagname_type` hash will look like the table below.

Table 7. Locuslink Annotations

The output from the script shows that Locuslink Annotations come in a variety of types, including DBLink, OntologyTerm, Comment, and SimpleValue. In order to extract the exact value you want, as opposed to the one returned by the `as_text` method, you'll need to find the desired method in the documentation for the Annotation in question.

If you were only interested in a certain type of Annotation you could retrieve it efficiently with something like this:

```
@term_annotations = map { $_->isa("Bio::Ontology::TermI"); }
    $seq_object->get_Annotations();
```

To completely parse these sequence formats you may also need to use methods that don't have anything to do with Features or Annotations per se. For example, the `display_id` method returns the LOCUS name of a Genbank entry or the ID from a SwissProt file. The `desc()` method will return the DEFINITION line of a Genbank file or the DE field in a SwissProt file. Again, this is a situation where you may have to examine a module, probably a `SeqIO::*` module, to find out more of the details.

10. Building your own 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` [<http://doc.bioperl.org/releases/bioperl-1.4/Bio/SeqFeature/Generic.html>] object is probably the best `SeqFeature` object for this purpose, in part because of its flexibility. Let's assume we have a sequence that has an interesting sub-sequence, going from position 10 to 22.

```
use Bio::SeqFeature::Generic;

# create the feature with some data, evidence and a note
my $feat = new Bio::SeqFeature::Generic(-start => 10,
                                         -end   => 22,
                                         -strand => 1,
                                         -tag    => {evidence => 'predicted',
                                                    note    => 'TATA box' } );
```

The `SeqFeature::Generic` [<http://doc.bioperl.org/releases/bioperl-1.4/Bio/SeqFeature/Generic.html>] 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;
for 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 to handle just about any sort of data.

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

```
use Bio::Seq;

# create a simple Sequence object
my $seq_obj = Bio::Seq->new(-seq => "attcccccttataaaatttttttttttgaggggtggg",
                           -display_id => "BI052" );
# then add the feature we've created to the sequence
$seq_obj->add_seqfeature($feat);
```

The `add_SeqFeature()` method will also accept an array of `SeqFeature` objects.

What if you wanted to add an `Annotation` to a sequence? You'll create the `Annotation` object, add data to it, create an `AnnotationCollection` object, add the `Annotation` to the `AnnotationCollection` along with a tag, and then add the `AnnotationCollection` to the sequence object:

```
use Bio::Annotation::Collection;
use Bio::Annotation::Comment;

my $comment = Bio::Annotation::Comment->new;
$comment->text("This looks like a good TATA box");
my $coll = new Bio::Annotation::Collection;
$coll->add_Annotation('comment',$comment);
$seq_obj->annotation($coll);
```

Now let's examine what we've created by writing the contents of `$seq_obj` to a Genbank file called "test.gb":

```
use Bio::SeqIO;

my $io = Bio::SeqIO->new(-format => "genbank",
                        -file    => ">test.gb" );
$io->write_seq($seq_obj);
```

Voila!

```
LOCUS      BIO52                      36 bp    dna      linear    UNK
DEFINITION
ACCESSION  unknown
COMMENT    This looks like a good TATA box
FEATURES             Location/Qualifiers
                10..22
                /match2="PF002534 e-3.1"
                /match1="PF000123 e-7.2"
                /author="john"
                /note="TATA box"
BASE COUNT      7 a      5 c      8 g      16 t
ORIGIN
                1 attccccctt ataaaatttt ttttttgagg ggtggg
//
```

11. 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/Graphics-HOWTO/index.html>]. The documentation for each of the individual `SeqFeature`, `Range`, `Location` and `Annotation` modules is also very useful, here's a list of them. If you have questions or comments that aren't addressed herein then write the Bioperl community at bioperl-l@bioperl.org.

SeqFeature Modules

SeqFeatureI.pm [<http://doc.bioperl.org/releases/bioperl-1.4/Bio/SeqFeatureI.html>]

SeqFeature/AnnotationAdaptor.pm

[<http://doc.bioperl.org/releases/bioperl-1.4/Bio/SeqFeature/AnnotationAdaptor.html>]

SeqFeature/FeaturePair.pm [<http://doc.bioperl.org/releases/bioperl-1.4/Bio/SeqFeature/FeaturePair.html>]

SeqFeature/Similarity.pm [<http://doc.bioperl.org/releases/bioperl-1.4/Bio/SeqFeature/Similarity.html>]

SeqFeature/Generic.pm [<http://doc.bioperl.org/releases/bioperl-1.4/Bio/SeqFeature/Generic.html>]

SeqFeature/SimilarityPair.pm [<http://doc.bioperl.org/releases/bioperl-1.4/Bio/SeqFeature/SimilarityPair.html>]

SeqFeature/PositionProxy.pm [<http://doc.bioperl.org/releases/bioperl-1.4/Bio/SeqFeature/PositionProxy.html>]

SeqFeature/Computation.pm [<http://doc.bioperl.org/releases/bioperl-1.4/Bio/SeqFeature/Computation.html>]

SeqFeature/Primer.pm [<http://doc.bioperl.org/releases/bioperl-1.4/Bio/SeqFeature/Primer.html>]

SeqFeature/Collection.pm [<http://doc.bioperl.org/releases/bioperl-1.4/Bio/SeqFeature/Collection.html>]

SeqFeature/CollectionI.pm [<http://doc.bioperl.org/releases/bioperl-1.4/Bio/SeqFeature/CollectionI.html>]

SeqFeature/SiRNA/Pair.pm [<http://doc.bioperl.org/releases/bioperl-1.4/Bio/SeqFeature/SiRNA/Pair.html>]

SeqFeature/SiRNA/Oligo.pm [<http://doc.bioperl.org/releases/bioperl-1.4/Bio/SeqFeature/SiRNA/Oligo.html>]

SeqFeature/Gene/GeneStructure.pm

[<http://doc.bioperl.org/releases/bioperl-1.4/Bio/SeqFeature/Gene/GeneStructure.html>]

SeqFeature/Gene/NC_Feature.pm

[http://doc.bioperl.org/releases/bioperl-1.4/Bio/SeqFeature/Gene/NC_Feature.html]

SeqFeature/Gene/Transcript.pm

[<http://doc.bioperl.org/releases/bioperl-1.4/Bio/SeqFeature/Gene/Transcript.html>]

SeqFeature/Gene/Exon.pm [<http://doc.bioperl.org/releases/bioperl-1.4/Bio/SeqFeature/Gene/Exon.html>]

SeqFeature/Gene/GeneStructureI.pm

[<http://doc.bioperl.org/releases/bioperl-1.4/Bio/SeqFeature/Gene/GeneStructureI.html>]

SeqFeature/Gene/Poly_A_site.pm

[http://doc.bioperl.org/releases/bioperl-1.4/Bio/SeqFeature/Gene/Poly_A_site.html]

SeqFeature/Gene/TranscriptI.pm

[<http://doc.bioperl.org/releases/bioperl-1.4/Bio/SeqFeature/Gene/TranscriptI.html>]

SeqFeature/Gene/ExonI.pm [<http://doc.bioperl.org/releases/bioperl-1.4/Bio/SeqFeature/Gene/ExonI.html>]

SeqFeature/Gene/Intron.pm [<http://doc.bioperl.org/releases/bioperl-1.4/Bio/SeqFeature/Gene/Intron.html>]

SeqFeature/Gene/Promoter.pm [<http://doc.bioperl.org/releases/bioperl-1.4/Bio/SeqFeature/Gene/Promoter.html>]

SeqFeature/Gene/UTR.pm [<http://doc.bioperl.org/releases/bioperl-1.4/Bio/SeqFeature/Gene/UTR.html>]

SeqFeature/Tools/Unflattener.pm

[<http://doc.bioperl.org/releases/bioperl-1.4/Bio/SeqFeature/Tools/Unflattener.html>]

SeqFeature/Tools/TypeMapper.pm

[<http://doc.bioperl.org/releases/bioperl-1.4/Bio/SeqFeature/Tools/TypeMapper.html>]

Annotation Modules

AnnotationI.pm [<http://doc.bioperl.org/releases/bioperl-1.4/Bio/AnnotationI.html>]

AnnotatableI.pm [<http://doc.bioperl.org/releases/bioperl-1.4/Bio/AnnotatableI.html>]

AnnotationCollectionI.pm [<http://doc.bioperl.org/releases/bioperl-1.4/Bio/AnnotationCollectionI.html>]

Annotation/AnnotationFactory.pm

[<http://doc.bioperl.org/releases/bioperl-1.4/Bio/Annotation/AnnotationFactory.html>]

Annotation/Comment.pm [<http://doc.bioperl.org/releases/bioperl-1.4/Bio/Annotation/Comment.html>]

Annotation/Reference.pm [<http://doc.bioperl.org/releases/bioperl-1.4/Bio/Annotation/Reference.html>]

Annotation/TypeManager.pm [<http://doc.bioperl.org/releases/bioperl-1.4/Bio/Annotation/TypeManager.html>]

Annotation/DBLink.pm [<http://doc.bioperl.org/releases/bioperl-1.4/Bio/Annotation/DBLink.html>]

Annotation/SimpleValue.pm [<http://doc.bioperl.org/releases/bioperl-1.4/Bio/Annotation/SimpleValue.html>]

Annotation/Collection.pm [<http://doc.bioperl.org/releases/bioperl-1.4/Bio/Annotation/Collection.html>]

Annotation/OntologyTerm.pm [<http://doc.bioperl.org/releases/bioperl-1.4/Bio/Annotation/OntologyTerm.html>]

Annotation/StructuredValue.pm

[<http://doc.bioperl.org/releases/bioperl-1.4/Bio/Annotation/StructuredValue.html>]

Location Modules

LocationI.pm [<http://doc.bioperl.org/releases/bioperl-1.4/Bio/LocationI.html>]

LocatableSeq.pm [<http://doc.bioperl.org/releases/bioperl-1.4/Bio/LocatableSeq.html>]

Location/Atomic.pm [<http://doc.bioperl.org/releases/bioperl-1.4/Bio/Location/Atomic.html>]

Location/AvWithinCoordPolicy.pm

[<http://doc.bioperl.org/releases/bioperl-1.4/Bio/Location/AvWithinCoordPolicy.html>]

Location/CoordinatePolicyI.pm

[<http://doc.bioperl.org/releases/bioperl-1.4/Bio/Location/CoordinatePolicyI.html>]

Location/Fuzzy.pm [<http://doc.bioperl.org/releases/bioperl-1.4/Bio/Location/Fuzzy.html>]

Location/FuzzyLocationI.pm [<http://doc.bioperl.org/releases/bioperl-1.4/Bio/Location/FuzzyLocationI.html>]

Location/NarrowestCoordPolicy.pm

[<http://doc.bioperl.org/releases/bioperl-1.4/Bio/Location/NarrowestCoordPolicy.html>]

Location/Simple.pm [<http://doc.bioperl.org/releases/bioperl-1.4/Bio/Location/Simple.html>]

Location/Split.pm [<http://doc.bioperl.org/releases/bioperl-1.4/Bio/Location/Split.html>]

Location/SplitLocationI.pm [<http://doc.bioperl.org/releases/bioperl-1.4/Bio/Location/SplitLocationI.html>]

Location/WidestCoordPolicy.pm

[<http://doc.bioperl.org/releases/bioperl-1.4/Bio/Location/WidestCoordPolicy.html>]

Range Modules

RangeI.pm [<http://doc.bioperl.org/releases/bioperl-1.4/Bio/RangeI.html>]

Range.pm [<http://doc.bioperl.org/releases/bioperl-1.4/Bio/Range.html>]

12. Acknowledgements

Thanks to Steven Lembark for comments and neat code discussions.