

This page is a work-in-progress, describing each of the key areas in which you might want to work with the new BioJava3 code. It is structured in the form of use-cases and is not a comprehensive resource. Sections will be added and updated as new modules are added and existing ones developed in more detail.

Contents

- 1 Symbols and Alphabets
 - ◆ 1.1 A DNA sequence
 - ◇ 1.1.1 Construction and basic manipulation
 - ◇ 1.1.2 Reversing and Complementing DNA
 - ◇ 1.1.3 Editing the sequence
 - ◆ 1.2 A quality-scored DNA sequence
 - ◇ 1.2.1 Constructing a quality-scored DNA sequence
 - ◇ 1.2.2 Iterating over the base/score pairs
 - ◇ 1.2.3 Iterating over the bases only
 - ◇ 1.2.4 Iterating over the scores only
- 2 File parsing and converting
 - ◆ 2.1 FASTA
 - ◇ 2.1.1 Parsing a FASTA file (the easy way)
 - ◇ 2.1.2 Parsing a FASTA file (the hard way)
 - ◇ 2.1.3 Converting the FASTA sequence into DNA sequence
 - ◇ 2.1.4 Converting a DNA sequence back into FASTA
 - ◇ 2.1.5 Writing a FASTA file (the easy way)
 - ◇ 2.1.6 Writing a FASTA file (the hard way)

Symbols and Alphabets

A DNA sequence

All the examples in this section require the biojava-dna module.

Construction and basic manipulation

```
String mySeqString = "ATCGatcgATCG"; // Note that you can use mixed-case strings.
List<Symbol> mySeq = SymbolListFormatter.parseSymbolList(mySeqString);

// Is it a big list? Don't want to hold it all in memory? Use an iterator instead.
```

BioJava3:HowTo

```
Iterator<Symbol> myIterator = SymbolListFormatter.parseSymbols(mySeqString);
while (myIterator.hasNext()) {
    Symbol sym = myIterator.next();
}

// You can now use any List method, from Java Collections, to manipulate the list of bases.

// The List returned is actually a SymbolList, you can cast it to get some bio-specific
// functions that work with 1-indexed positions as opposed to Java's default 0-indexed positions

SymbolList symList = (SymbolList)mySeq;
Symbol symA = symList.get(0); // The first symbol, List-style.
Symbol symB = symList.get_bio(1) ; // The first symbol, bio-style.
if (symA==symB) { // Symbols are singletons, so == will work if they are identical including case
    System.out.println("Identical!");
}

// Instead of using equals() or == to compare symbols, use the alphabet of your choice to
// compare them in multiple ways. It will return different values depending on whether one
// is a gap and the other isn't, whether they match exactly, or if they're the same symbol
// but in a different case, etc.
Alphabet dna = DNATools.DNA_ALPHABET;
SymbolMatchType matchType = dna.getSymbolMatchType(Symbol.get("A"), Symbol.get("a"));
```

Reversing and Complementing DNA

```
// All methods in this section modify the list in-place.
List<Symbol> mySeq = SymbolListFormatter.parseSymbolList("ATCG");

// Reverse.
// Method A.
Collections.reverse(mySeq); // Using Java Collections.
// Method B.
DNATools.reverse(mySeq); // DNATools-style.

// Complement.
DNATools.complement(mySeq);

// Reverse-complement.
DNATools.reverseComplement(mySeq);

// Reverse only the third and fourth bases, 0-indexed list style?
Collections.reverse(mySeq.subList(2,4)); // Java Collections API.

// Do the same, 1-indexed bio style?
Collections.reverse(((SymbolList)mySeq).subList_bio(3,5));
```

Editing the sequence

```
// Delete the second and third bases.
List<Symbol> mySeq = SymbolListFormatter.parseSymbolList("ATCG");
mySeq.subList(1,3).clear();

// Remove only 2nd base, bio-style.
((SymbolList)mySeq).remove_bio(2);

// Get another sequence and insert it after the 1st base.
List<Symbol> otherSeq = SymbolListFormatter.parseSymbolList("GGGG");
```

```
mySeq.addAll(1, otherSeq);
```

A quality-scored DNA sequence

Constructing a quality-scored DNA sequence

```
// Construct a default unscored DNA sequence with capacity for integer scoring.
List<Symbol> mySeq = SymbolListFormatter.parseSymbolList("ATCG");
TaggedSymbolList<Integer> scoredSeq = new TaggedSymbolList<Integer>(mySeq);

// Tag all the bases with the same score of 5.
scoredSeq.setTagRange(0, scoredSeq.length(), 5);

// Tag just the 3rd base (0-indexed) with a score of 3.
scoredSeq.setTag(2, 3);

// Do the same, 1-indexed.
scoredSeq.setTag_bio(3, 3);

// Get the score at base 4, 1-indexed.
Integer tag = scoredSeq.getTag_bio(4);
```

Iterating over the base/score pairs

```
// A 1-indexed iterator and ListIterators are also available.
Iterator<TaggedSymbol<Integer>> iter = scoredSeq.taggedSymbolIterator();
while (iter.hasNext()) {
    TaggedSymbol<Integer> taggedSym = iter.next();
    Symbol sym = taggedSym.getSymbol();
    Integer score = taggedSym.getTag();
    // Change the score whilst we're at it.
    taggedSym.setTag(6); // Updates the score to 6 in the original set of tagged scores.
}
```

Iterating over the bases only

```
// Use the default iterator.
// A ListIterator is also available, as are 1-indexed iterators.
Iterator<Symbol> iter = scoredSeq.iterator();
```

Iterating over the scores only

```
// A ListIterator is also available, as are 1-indexed iterators.
Iterator<Integer> iter = scoredSeq.tagIterator();
while (iter.hasNext()) {
    Integer score = iter.next();
}
```

File parsing and converting

FASTA

The examples in this section require the biojava-fasta module. The examples that deal with converting to/from DNA sequences also require the biojava-dna module.

Convenience wrapper classes are provided to make the parsing process simpler for the most common use-cases.

Parsing a FASTA file (the easy way)

```
ThingParser<FASTA> parser = ThingParserFactory.
    getReadParser(FASTA.format, new File("/path/to/my/fasta.fa"));
while (parser.hasNext()) {
    FASTA fasta = parser.next();
    // fasta contains a complete FASTA record.
}
parser.close();
```

Parsing a FASTA file (the hard way)

```
FASTAReader reader = new FASTAFileReader(new File("/path/to/my/fasta.fa"));
FASTABuilder builder = new FASTABuilder();
ThingParser<FASTA> parser = new ThingParser<FASTA>(reader, builder);
while (parser.hasNext()) {
    FASTA fasta = parser.next();
    // fasta contains a complete FASTA record.
}
parser.close();
```

Converting the FASTA sequence into DNA sequence

```
List<Symbol> mySeq = SymbolListFormatter.parseSymbolList(fasta.getSequence());
```

Converting a DNA sequence back into FASTA

```
FASTA fasta = new FASTA();
fasta.setDescription("My Description Line");
fasta.setSequence(SymbolListFormatter.formatSymbols(mySeq));
```

Writing a FASTA file (the easy way)

```
ThingParser<FASTA> parser = ThingParserFactory.
    getWriteParser(FASTA.format, new File("/path/to/my/fasta.fa"), fasta);
parser.parseAll();
```

```
parser.close();
```

Writing a FASTA file (the hard way)

```
FASTAEmitter emitter = new FASTAEmitter(fasta);  
FASTAWriter writer = new FASTAFileWriter(new File("/path/to/new/fasta.fa"));  
ThingParser<FASTA> parser = new ThingParser<FASTA>(emitter, writer);  
parser.parseAll();  
parser.close();
```