

Running BioJava in Forte for Java

Daniel Quest and Alexander Churbanov

June 27, 2002

Abstract

This document discusses proper installation of BioJava in Forte environment. Since a significant time has been spent to figure out the most optimal way to install the software, this document could be of significant interest to those, who are planning to work with Forte and BioJava in the nearest future.

1 Forte: A necessity for BioJava

Forte is a very powerful compiler and editor for Java. It can be found at <http://www.sun.com/software/sundev/jde/buy/index.html>.

The *Forte Community Edition* is freely distributed by *Sun*.

BioJava is software package oriented for bioinformatics freely distributed under GNU lesser licence. The package is written entirely in Java and provides an interface for various file formats parsing (GenBank, EMBL, Fasta, Blast), basic sequences manipulation, HMM profiles, statistical routines and various other functionalities.

There are several demo applications available from <http://www.biojava.org>. Also, there are tutorial notes from the 2002 BioJava bootcamp <http://www.biojava.org/bootcamp/>.

Here is the summary of what has been done in Java Forte to run a BioJava package successfully.

First of all you need to unpack the package and make sure that all important `xml` files are in the right place. Once you compile project with `ant`, it puts these files into a right place automatically. To debug source code you will need

First upload and unpack the most recent BioJava package version from <http://www.biojava.org>

Second compile using `ant`: `>> ant build.xml`. In Forte for java, we can mount directory `/biojava#.#/` into the current project (see below for detailed explanations), then find `build.xml` file in the `Filesystems` directory, right click on it and then choose `Execute`. It will compile the project and put the source and classes into `ant-build` directory.

Third If extended debugging is not needed, simply add `biojava.jar`, `jakarta-regeexp.jar`, `bytecode.jar` and `xerces.jar` to the workspace.

Fourth for the debugging purposes copy the following `xml` files from `/biojava#.#/ant-build/classes` to `/biojava#.#/ant-build/src`), otherwise your source code would not debug properly

```
org/biojava/bio/proteomics/ProteaseManager.xml
org/biojava/bio/seq/TranslationTables.xml
org/biojava/bio/seq/io/FeatureQualifier.xml
org/biojava/bio/symbol/AlphabetManager.xml
org/biojava/bio/symbol/ResidueProperties.xml
org/biojava/utils/query/QueryBuilder.xml
org/demos/dp/fakepromoter.xml
org/demos/files/fakepromoter.xml
```

In order to start working on a BioJava related project in Forte we need to set up project space properly. To do this we need

1. Create the project by going to **Project > Project Manager**
2. Open explorer **View > Explorer**
3. In the **Filesystems** tab Right click on **Filesystems** folder
4. Choose **Mount > Local directory** and then choose one containing **org** folder with the source code, it is **/biojava#.#/ant-build/src/main**
5. Right click on the **Filesystems** and choose **bytecode.jar, jakarta-regexp.jar, xerces.jar** to mount as **jar**
6. You may need to add something to the project by right clicking on the folders, JARs, ZIPs and files in **Filesystems** tab and choosing **Tools > Add to project**
7. Look on the Right click menu in both **Filesystems** and **Project** tabs - there are lots of useful options
8. For the purposes of debugging, **you have to** install JDK source following the instructions below

1.1 JDK 1.3.x

Sources of JDK 1.3.x are stored in file **src.jar** of the JDK's base directory. The JAR-file contains files

```
src/
src/java/
src/java/applet/
src/java/applet/Applet.java
...
src/java/awt
...
```

that is, all files are in subdirectory **src** within the JAR-file. If you mounted file **src.jar** and the debugger tried to find a source code for **java.applet.Applet**, it would not find it because it would try to open file **java/applet/Applet.java** which is not present in the JAR-file.

To make the sources available to the debugger, you have to: - unpack file **src.jar** (run **jar xf src.jar** in the JDK's base directory) - mount directory **src** (in Forte)

1.2 JDK 1.4.x

Sources of JDK 1.4.x are stored in file `src.zip` of the JDK's base directory. Unlike JDK 1.3.x, the JAR-archive no longer contains directory `src` so you do not need to unpack the ZIP file.

To make the sources available to the debugger, just: - mount directory `src.zip` (in Forte)

2 Flat File Parser Implementation

Here is an example of the Genbank Parser we used to populate the NMRecord Database using BioJava packages. Remember to set all of the appropriate `CLASSPATH` variables and include all needed jars (covered in section 1) when using this parser. Also if a database connection is desired, JDBC connection needs to be set

```
/**
 * This file has been modified to parse mRNA files (NM_##### records)
 */
package seq;

import java.io.*;

import org.biojava.bio.*; import org.biojava.bio.symbol.*; import
org.biojava.bio.seq.*; import org.biojava.bio.seq.io.*; import
org.biojava.bio.seq.io.*; import java.util.StringTokenizer;

public class TestGenbankNM {
    public static void main(String [] args) {
        try {
            if(args.length != 1) {
                throw new Exception("Use: TestGenbank genbankFile");
            }

            File genbankFile = new File(args[0]);
            SequenceFormat gFormat = new GenbankFormat();
            BufferedReader gReader = new BufferedReader(new
InputStreamReader(new FileInputStream(genbankFile)));
            SequenceBuilderFactory sbFact = new
GenbankProcessor.Factory(SimpleSequenceBuilder.FACTORY);
            Alphabet alpha = DNATools.getDNA();
            SymbolTokenization rParser = alpha.getTokenization("token");
            // Here we actually parse the file
            SequenceIterator seqI = new StreamReader(gReader, gFormat,
rParser, sbFact);
            // Now we start iterating with sequences
            int counter = 1;
            while(seqI.hasNext()) {
                //Create record to populate it
                Record rec = new Record();
                String type, symbList;
                //Get a sequence
                Sequence seq = seqI.nextSequence();
            }
        }
    }
}
```

```

//Get rid of non-mRNA records

if(seq.getAnnotation().getProperty("TYPE").toString().compareTo("mRNA")
!= 0)
    continue;

//Record the sequence itself
rec.setSequence(seq.seqString());
// Iterate through features
java.util.Iterator theIterator = seq.features();
try {
    while(theIterator.hasNext()) {
        Feature theFeature = (Feature)theIterator.next();
        // Do the casting of the sequence
        StrandedFeature castedFeature =
(StrandedFeature)theFeature;
        // Type of the feature (CDS, ...)
        type = castedFeature.getType();
        // The type of the feature => type
        // The sequence => symbList

        if(type.compareTo("CDS") == 0) {
            //Extract the sequence for CDS carefully...
            // List of symbols
            SymbolList sl = castedFeature.getSymbols();
            // Sequence of nucleotides for the subFeature
            symbList = sl.seqString();
            rec.setCDS(symbList);
            // Extract the location for the feature
            Location source = castedFeature.getLocation();
            int cdsStart = source.getMin();
            int cdsEnd = source.getMax();
            // Test whether we deal with positive or complementary
strand
            if(castedFeature.getStrand() == castedFeature.POSITIVE) {
                rec.setUTR5(seq.subStr(1, cdsStart - 1));
                rec.setUTR3(seq.subStr(cdsEnd + 1, seq.length()));
            } else {
                rec.setUTR3(seq.subStr(1, cdsStart - 1));
                rec.setUTR5(seq.subStr(cdsEnd + 1, seq.length()));
            }
            // Get annotation for the feature
            Annotation featureAnnotation =
theFeature.getAnnotation();
            // Here we get the keys
            java.util.Set theKeys = featureAnnotation.keys();
            // And here we get the values
            java.util.Iterator keyIterator = theKeys.iterator();
            while(keyIterator.hasNext()) {

```

```

        String key = keyIterator.next().toString();
        if(key.compareTo("protein_id") == 0) {
            StringTokenizer tokens = new
StringTokenizer(featureAnnotation.getProperty(key).toString(), "
");
            rec.setProteinID(tokens.nextToken());
        } else if(key.compareTo("translation") == 0) {
rec.setProtein(featureAnnotation.getProperty(key).toString());
        }
    }
}
}
}
}
catch(IndexOutOfBoundsException e) {
    System.err.println("Error: while getting CDS region. Index
out of bounds for sequence " + seq.getName() + ".\nSequence is not
processed.");
    continue;
}
catch (ClassCastException cce) {
    System.err.println("Error: CDS Feature is not Stranded.
Sequence " + seq.getName() + " is not processed.");
    continue;
}
// Get annotation for the whole sequence
Annotation theAnnotation = seq.getAnnotation();
java.util.Set theKeys = theAnnotation.keys();
java.util.Iterator theOtherIterator = theKeys.iterator();
while(theOtherIterator.hasNext()) {
    String key = theOtherIterator.next().toString();
    String value = theAnnotation.getProperty(key).toString();
    if(key.compareTo("DIVISION")==0){
        rec.setDivision(value);
    } else if(key.compareTo("ORGANISM")==0){
        rec.setScientificName(value);
    } else if(key.compareTo("VERSION")==0){
        StringTokenizer tokens = new StringTokenizer(value,
". ");
        tokens.nextToken();

rec.setVersion(Integer.parseInt(tokens.nextToken()));
    } else if(key.compareTo("ACCESSION")==0) {
        StringTokenizer tokens = new StringTokenizer(value,
" ");
        rec.setAccession(tokens.nextToken());
    } else if(key.compareTo("GI")==0) {
        rec.setGI(Integer.parseInt(value));
    } else if(key.compareTo("CIRCULAR")==0) {
        rec.setShape(value);

```

```

    } else if(key.compareTo("SOURCE")==0) {
        rec.setOrganismName(value);
    } else if(key.compareTo("TYPE")==0) {
        rec.setSequenceType(value);
    } else if(key.compareTo("DEFINITION")==0) {
        rec.setDefinition(value);
    } else if(key.compareTo("DIVISION")==0) {
        rec.setDivision(value);
    } else if(key.compareTo("MDAT")==0) {
        rec.setDate(value);
    } else if(key.compareTo("SIZE")==0) {
        rec.setSequenceLength(Integer.parseInt(value));
    } else if(key.compareTo("LOCUS")==0) {
        rec.setLocusName(value);
    }
}
// System.out.println("Rec# =>" + counter++);
// Put it into database
rec.populateAll();
}
System.out.println("I am done!!!");
}
catch (Throwable t) {
    t.printStackTrace();
    System.exit(1);
}
}
}
}

```