

Written by:
Kitty ter Stege
Thomas Kelder (t.a.j.kelder@student.tue.nl)
Gijs Huisman

2004, Technical University of Eindhoven

Tutorial for Creating a BioPAX Pathway with Jena.....	1
<i>Introduction</i>	<i>1</i>
<i>Setting up a java program to use Jena.....</i>	<i>2</i>
<i>Setting up a BioPAX pathway.....</i>	<i>2</i>
Create a new ontology model.....	2
Define the namespaces.....	2
Create a new ontology and import BioPAX.....	3
<i>Adding pathway data to the ontology model.....</i>	<i>4</i>
Creating an individual in Jena.....	4
Adding properties to the individuals.....	6
<i>Writing the pathway data in XML syntax</i>	<i>8</i>
Appendix A	9
create_testpathway.java	9
Frame1.java.....	10
Biopax.java	14
Appendix B.....	16
testpathway.owl	16

Tutorial for Creating a BioPAX Pathway with Jena

Introduction

This document is written for those who know want to get a practical introduction about how to use BioPAX. BioPAX is a collaborative effort to create a data exchange format for biological pathway data. The specifications of how the data has to be stored are written in OWL (Ontology Web Language). OWL is a language for describing ontologies, it can both define the structure of data and store the data in a document. OWL is written in XML syntax. If you don't know what an ontology is, we recommend you read the introduction of the "OWL guide" which can be found at: <http://www.w3.org/TR/2004/REC-owl-guide-20040210/#Introduction>. To handle OWL documents, there are several software packages available like SWOOP or Protegé in which you can create and edit OWL documents in a GUI (graphical user interface). However, when you want to write your own software package that can handle BioPAX pathways or want to implement BioPAX into your existing program, these packages cannot be used. This is where you need Jena, a Java API (a collection of Java classes) for handling OWL documents. Jena is a very extensive package, but for the purpose of creating a pathway in BioPAX format, a very small part of it is used.

To read this document a bit of Java knowledge is necessary and it is useful first to take a look at the document "An Introduction to RDF and the Jena RDF API" that can be found at: http://jena.sourceforge.net/tutorial/RDF_API/.

Setting up a java program to use Jena

The first thing to do before you can use Jena is to install the classes. This can be done by importing the “lib” directory of the downloaded jena.zip (<http://prdownloads.sourceforge.net/jena/Jena-2.1.zip?download>) into your Java IDE (Integrated Development Environment), or adding this directory to the java class path.

The Jena classes that you use have to be imported in your own java program. The Jena classes we use in this tutorial are imported by the following statement:

```
1 // Import the Jena Classes
2 import com.hp.hpl.jena.rdf.model.*;
3 import com.hp.hpl.jena.ontology.*;
4 import com.hp.hpl.jena.util.*;
```

Setting up a BioPAX pathway

Create a new ontology model

In Jena, an ontology is represented as a model. This model contains resources like classes, properties and individuals. To these resources, data can be added. Also relations can be defined between the different resources. To create a pathway, first the model containing the pathway has to be created:

```
5 // Create an empty OWL model
6 biopaxmodel = ModelFactory.createOntologyModel
7             (OntModelSpec.OWL_MEM,null);
```

The first parameter of the method “createOntologyModel” defines the specification of the model (there are alternatives to OWL which Jena can also handle). The OWL specification is stored in the class OntModelSpec. In the second parameter a “base model” can be given, relative to which the new model is created, but we don’t use that option here.

Define the namespaces

The next step is to define the namespaces you want to use in your new model.

Namespaces are required when using OWL, since the OWL convention demands unique names for each variable used in any OWL document. So by placing the namespace name in front of the variable name it becomes a unique variable name.

A namespace is always in the form of an URI, it is convenient to use a URI ending with the name of your document so in our case it could be something like:

```
http://somewhere/testpathway#
```

The namespace of the biopax-level1 OWL has to be added too, since we are going to use variables defined in the bioxpax-level1 ontology.

Since namespaces are often used, we add them by making a prefix for every namespace which makes the output more readable (line 13 and 15).

```
8 // Declare the variables which contain the URIs of the namespaces
9 String biopaxString = "http://www.biopax.org/release/biopax-
10                       level1.owl#";
11 String namespaceString = "http://somewhere/testpathway#";
```

```
12 // Set the namespaceprefix for this pathway
13 biopaxmodel.setNsPrefix("bp",biopaxString );
```

```
14 // Set the namespaceprefix for the biopax-level1 OWL
15 biopaxmodel.setNsPrefix("", namespaceString);
```

The result in xml syntax representation looks like this:

```
<rdf:RDF
  xmlns:bp="http://www.biopax.org/release/biopax-level1.owl#"
  xmlns="http://somewhere/testpathway#" >
```

An xml tag which uses the biopax-level1 namespace now looks like this:

```
<bp:protein>
  .....
</bp:protein>
```

The first parameter of the method “setNsPrefix” is the prefix of the namespace and the second parameter is the namespace itself. For the namespace of this document, we use an empty prefix. This results in the fact that when a variable in our document contains no prefix, it is a variable within the document namespace and when a prefix is used, the variable belongs to the prefix’s namespace.

The namespaces of OWL and RDF (with prefixes “owl:” and “rdf:”) are added by Jena automatically.

Create a new ontology and import BioPAX

The model is still empty and the first thing to be added is a new ontology in which the pathway will be created:

```
16 // Define the URI of the owl file
17 String namespaceFileString = "http://somewhere/testpathway.owl";
18 // create the pathway ontology
19 Ontology ont = biopaxmodel.createOntology(namespaceFileString);
```

You see that also the ontology has to be given as an URI, pointing to the file which contains the ontology.

The biopax-level1 OWL now has to be imported into the pathway ontology:

```
20 // Define the URI of the biopax-level1 owl file
21 String biopaxFileString = "http://www.biopax.org/release/biopax-
22 level1.owl";
```

```
23 // import the biopax-level1 ontology into the pathway ontology
24 ont.addImport(biopaxmodel.createResource(biopaxFileString));
25 biopaxmodel.addLoadedImport(biopaxFileString);
```

In xml syntax the import looks like this:

```
<owl:Ontology rdf:about="">
  <owl:imports rdf:resource="http://www.biopax.org/release/biopax-
    level1.owl"/>
</owl:Ontology>
```

Adding pathway data to the ontology model

Now we have created an ontology model with the correct structure we can start to add the pathway data to it. The pathway data can come from another pathway storage format that has to be converted to BioPAX or it can be new data that has to be stored as a BioPAX pathway. In this example, the data is stored in three string arrays with three proteins. These proteins consist of a name and a reference to a protein database. The reference consists of the database name and the ID of the protein, corresponding to that database (table 1).

Protein name	Database name	Protein ID
Enamelysin	Swiss-Prot	P023244
Metalloelastase	RefSeq	NP_304845
Collagenase	Swiss-Prot	P308934

Table 1

This data has to be stored in the right classes of the BioPAX model. An overview of classes and their structure can be found in the document “biopax-level1-diagram.pdf”, which is in the BioPAX zip file (www.biopax.org/download). It is important to know which data has to be in which classes before writing the program.

In this example, first an individual in the class “pathway” has to be created. This is the pathway to which the proteins can be added. The individual protein contains a property “NAME” which stores the protein name shown in table 1. The database name and ID has

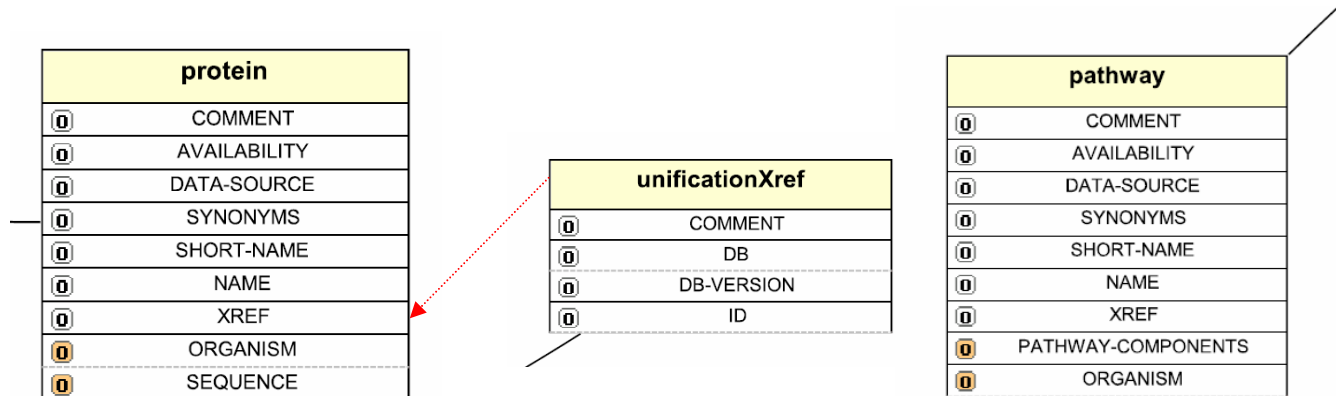


Figure 1

to be stored in the properties “DB” and “ID” of an individual of the class “unificationXref”. To define that a reference relates to a protein, the property “XREF” of the protein has to contain a link to the individual in the class “unificationXref” corresponding to that protein (red arrow in figure 1).

Creating an individual in Jena

In Jena an individual is created by using the method “createIndividual”. The URI which names the individual must be constructed by using the pathway namespace:

```
String individualURI=namespaceString + IndividualName;
```

In order to be able to create a individual a resource is needed, the resource is the class to which the individual belongs and exists in the biopax-level1 namespace:

```
String classURI=biopaxString + classname;
```

This also holds for properties. Properties of the individuals also belong to the biopax-level1 namespace.

The piece of code needed to create an individual stays the same for every individual, only the class and the individualname changes. Therefore it is better to make your own Java class, containing a method that creates an individual with given name and class:

```
26 public class Biopax {
27     public static String biopaxString =
28         "http://www.biopax.org/release/biopax-level1.owl#";
29     public static String namespaceString =
30         "http://somewhere/testpathway.owl#";
31
32     public static void createBiopaxIndividual(String individualName,
33 String individualClass, OntModel model) {
34         // Get the resource of the class where the individual has to be
35         placed
36         Resource res = model.getResource(biopaxString+individualClass);
37         // Add the individual to the class
38         model.createIndividual(namespaceString + individualName,res);
39     }
```

The method “createBiopaxIndividual” creates an individual in our testpathway for a given individual name, class and model.

If we want to create the individual “pathway1” in the class “pathway”, we use this method:

```
40 // Create an individual in #pathway for the testpathway
41 individualName = "pathway1";
42 individualClass = "pathway";
43 Biopax.createBiopaxIndividual(individualName,individualClass,
44                               biopaxmodel);
```

In XML syntax, the result is:

```
<bp:pathway rdf:ID="pathway1">
</bp:pathway>
```

Adding proteins to the pathway is pretty much the same. For every protein, an individual has to be created in the class “protein”:

```
45 // Declare the string arrays with the protein data
46 String[] proteinName = { "Enamelysin", "Metalloelastase",
47                           "Collagenase" };
48 String[] proteinDB = { "SWISS-PROT", "RefSeq", "SWISS-PROT" };
49 String[] proteinID = { "P023244", "NP_304845", "P308934" };
50 // Create and individual in the class "protein" for every protein
51 for (int i = 0; i < proteinName.length; i++) {
52     // Create the name of the individual
53     individualName = "Protein" + i;
54     individualClass = "protein";
55     Biopax.createBiopaxIndividual(individualName,individualClass,
56     biopaxmodel);
57 }
```

Now the only individuals that have to be added are the references to the database:

```
58 // Create an individual in the class "unificationXref" for every
59 protein
60 for (int i = 0; i < proteinName.length; i++) {
61     // Create the name of the individual
62     individualName = "Xref" + i;
63     individualClass = "unificationXref";
64     Biopax.createBiopaxIndividual(individualName,individualClass,
```

```

65     biopaxmodel);
66     }

```

Adding properties to the individuals

In the previous paragraph we created a set of individuals. There hasn't been stored any data yet. The data should be stored in the properties of the individual. To create these properties and fill them with data, we create another method in our Java class "Biopax":

```

67 public static void addBiopaxProperty(String propertyLiteral, String
68 propertyName, String individualName, OntModel model) {
69     // Get the individual for which the property has to be added
70     Individual ind = model.getIndividual(namespaceString+
71         individualName);
72     // Get the property
73     Property prop = model.getProperty(biopaxString+propertyName);
74     // Add the literal to the property
75     ind.addProperty(prop,propertyLiteral);
76 }

```

The Jena method `addProperty` adds the data to the given property of the individual it works on. The individual can be obtained by the method `getIndividual`, which needs the URI of the individual. The property is obtained by the method `getProperty`, which also needs an URI. Note that the property, like the classes, is again a part of the "biopax-level1" namespace and not of the "testpathway" namespace.

The property "NAME" of the pathway individual created in the previous paragraph can be added:

```

78 // Add the property "NAME" to the pathway individual
79     propertyName = "NAME";
80     Biopax.addBiopaxProperty(pathwayName,propertyName,individualName,
81         biopaxmodel);

```

In XML syntax, the result is:

```

<bp:pathway rdf:ID="pathway1">
  <bp:NAME>testpathway</bp:NAME>
</bp:pathway>

```

Also, the property "NAME", stored in the string array "proteinName" can now be added to the protein individuals:

```

82 // Add the property "NAME" to every protein
83     propertyName = "NAME";
84     for (int i = 0; i < proteinName.length; i++) {
85         // Create the name of the individual
86         individualName = "Protein" + i;
87         // Add the properties to the Protein individuals
88         Biopax.addBiopaxProperty(proteinName[i],propertyName,
89             individualName,biopaxmodel);
90     }

```

Of course, in the real program, it is easier to put the creation of the individuals and adding of the properties in the same for loop.

The properties to the "Xref" individuals, stored in the string arrays "proteinDB" and "proteinID", can be added the same way:

```

91 // Create an individual in #unificationXref for every protein
92 for (int i = 0; i < proteinName.length; i++) {

```

```

93     // Add the properties to the Xref individuals
94     propertyName = "DB";
95     Biopax.addBiopaxProperty(proteinDB[i],propertyName,
96     individualName,biopaxmodel);
97     propertyName = "ID";
98     Biopax.addBiopaxProperty(proteinID[i],propertyName,
99     individualName,biopaxmodel);
100    }

```

The data stored in the properties “NAME”, “ID” and “DB” are all just strings or numbers, called “literals”. However, the data in the property “XREF” of the individuals in the class “protein”, we need a link to an individual. This can be done by changing the second input argument of the method “addProperty” to a Resource. To do this we make another method in the class Biopax:

```

101 public static void addBiopaxLinkedProperty(String propertyName,
102 String linkedresourceName, String individualName, OntModel model)
103 {
104     // Get the individual for which the property has to be added
105     Individual ind = model.getIndividual(namespaceString+
106     individualName);
107     // Get the property
108     Property prop = model.getProperty(biopaxString+propertyName);
109     // Get the resource which has to be added to the property
110     Resource res = model.getResource(biopaxString+
111     linkedresourceName);
112     Resource res = model.getResource(namespaceString+
113     linkedresourceName);
114     // Link the properties
115     ind.addProperty(prop, res);
116 }

```

For the individual “protein0”, the result of this action in XML syntax is:

```

<bp:protein rdf:ID="Protein0">
  <bp:XREF>
    <bp:unificationXref rdf:ID="Xref0">
      <bp:DB>SWISS-PROT</bp:DB>
      <bp:ID>P023244</bp:ID>
    </bp:unificationXref>
  </bp:XREF>
  <bp:NAME>Enamelysin</bp:NAME>
</bp:protein>

```

So far the document contains information about three proteins. However, no information about the relations between these proteins is added. These relationships are needed to create a complete pathway. With a few modifications to the methods in the Java class we presented, a complete pathway can be build.

Writing the pathway data in XML syntax

With the actions in the previous paragraphs, we created an ontology model of a pathway in Jena. However, this model has to be stored as a text file with the XML syntax. In Jena, this is done by the method “write”. This method uses an “PrintWriter” to write the data to a file. A `RDFWriter` is a Jena class that creates the XML syntax.

An example of writing a model to a file:

```
117 // Declare the filename
118 String fileName = "testpathway.owl";

119 // create an outputstream for the file to be saved
120 PrintWriter out = null;
121 try {
122     out = new PrintWriter(new FileOutputStream(fileName,false));

123     RDFWriter writer = biopaxmodel.getWriter("RDF/XML-ABBREV") ;
124     writer.setProperty("xmlbase",namespaceFileString) ;
125     // Save the OWL model
126     writer.write(biopaxmodel,out,namespaceString);
127 }
128 catch (FileNotFoundException ex) {
129     JOptionPane.showMessageDialog(null,"Could not create and/or
130                                     save the file: "+fileName);
131 }
132 out.close() ;
```

The input for the method “getWriter” defines the syntax of the output. Before writing the file, the property “xmlbase” has to be added, which in XML syntax looks like this:

```
xml:base="http://somewhere/testpathway.owl"
```

The “try/catch” construct is used to produce an error message in case the file cannot be written.

Appendix A

Here you find an example program which creates the testpathway as described in this document. The program is a bit more complicated, because it has a GUI, but the code contains all the examples we discussed. The code is stored in three different files. The Biopax class is in the file “Biopax.java”. The code that generates the pathway is in the file “Frame1.java”.

create_testpathway.java

```
package testpathway;

import javax.swing.UIManager;
import java.awt.*;

public class create_testpathway {
    private boolean packFrame = false;

    //Construct the application
    public create_testpathway() {
        Frame1 frame = new Frame1();
        //Validate frames that have preset sizes
        //Pack frames that have useful preferred size info, e.g. from their
        layout
        if (packFrame) {
            frame.pack();
        }
        else {
            frame.validate();
        }
        //Center the window
        Dimension screenSize = Toolkit.getDefaultToolkit().getScreenSize();
        Dimension frameSize = frame.getSize();
        if (frameSize.height > screenSize.height) {
            frameSize.height = screenSize.height;
        }
        if (frameSize.width > screenSize.width) {
            frameSize.width = screenSize.width;
        }
        frame.setLocation((screenSize.width - frameSize.width) / 2,
        (screenSize.height - frameSize.height) / 2);
        frame.setVisible(true);
    }
    //Main method
    public static void main(String[] args) {
        try {
            UIManager.setLookAndFeel(UIManager.getSystemLookAndFeelClassName());
        }
        catch(Exception e) {
            e.printStackTrace();
        }
        new create_testpathway();
    }
}
```

Frame1.java

```
package testpathway;

import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
import java.io.*;

/////Import the Jena Libraries
import com.hp.hpl.jena.rdf.model.*;
import com.hp.hpl.jena.ontology.*;
import com.hp.hpl.jena.util.*;
import com.hp.hpl.jena.rdf.arp.*;
import com.borland.jbcl.layout.*;
import com.hp.hpl.jena.vocabulary.*;

public class Frame1 extends JFrame {
    ///// Declare some global variables
    public String biopaxString = "http://www.biopax.org/release/biopax-
level1.owl#";
    public String biopaxFileString =
"http://www.biopax.org/release/biopax-level1.owl";
    public OntModel biopaxmodel;
    public String namespaceString = "http://somewhere/testpathway#";
    public String namespaceFileString =
"http://somewhere/testpathway.owl";

    private JPanel contentPane;
    private JTextArea disptext = new JTextArea();
    private JButton newpathway = new JButton();
    private JButton addproteins = new JButton();
    private JScrollPane jScrollPane1 = new JScrollPane(disptext);
    private JButton saveOWL = new JButton();

    //Construct the frame
    public Frame1() {
        enableEvents(AWTEvent.WINDOW_EVENT_MASK);
        try {
            jbInit();
        }
        catch(Exception e) {
            e.printStackTrace();
        }
    }
    //Component initialization
    private void jbInit() throws Exception {

//setIconImage(Toolkit.getDefaultToolkit().createImage(Frame1.class.getR
esource("[Your Icon]"));
        contentPane = (JPanel) this.getContentPane();
        contentPane.setLayout(null);
        this.setSize(new Dimension(412, 432));
        this.setTitle("Biopax!!");
        newpathway.setBounds(new Rectangle(5, 5, 390, 27));
        newpathway.setActionCommand("newpathway");
        newpathway.setText("create a new BioPax Pathway");
        newpathway.addActionListener(new java.awt.event.ActionListener() {
            public void actionPerformed(ActionEvent e) {
                newpathway_actionPerformed(e);
            }
        });
        disptext.setEditable(false);
        disptext.setLineWrap(true);
        addproteins.setBounds(new Rectangle(5, 37, 390, 27));
        addproteins.setToolTipText("");
        addproteins.setActionCommand("addproteins");
    }
}
```

```

addproteins.setText("Add pathway data");
addproteins.addActionListener(new java.awt.event.ActionListener() {
    public void actionPerformed(ActionEvent e) {
        addproteins_actionPerformed(e);
    }
});
jScrollPane.setBorder(BorderFactory.createLineBorder(Color.black));
jScrollPane.setBounds(new Rectangle(5, 97, 384, 301));
saveOWL.setBounds(new Rectangle(4, 69, 390, 25));
saveOWL.setActionCommand("saveOWL");
saveOWL.setText("save the pathway");
saveOWL.addActionListener(new java.awt.event.ActionListener() {
    public void actionPerformed(ActionEvent e) {
        saveOWL_actionPerformed(e);
    }
});
contentPane.add(jScrollPane, null);
contentPane.add(newpathway, null);
contentPane.add(addproteins, null);
contentPane.add(saveOWL, null);
}
//Overridden so we can exit when window is closed
protected void processWindowEvent(WindowEvent e) {
    super.processWindowEvent(e);
    if (e.getID() == WindowEvent.WINDOW_CLOSING) {
        System.exit(0);
    }
}

/////This is what happens when you press the button "Create a new
BioPax Pathway"
void newpathway_actionPerformed(ActionEvent e) {
    // Create an empty OWL model
    biopaxmodel =
ModelFactory.createOntologyModel(OntModelSpec.OWL_MEM,null);
    disptext.append(">> empty model 'biopaxmodel' created\n");

    // Set the namespaceprefix for biopax-level1
    biopaxmodel.setNsPrefix("bp",biopaxString );
    biopaxmodel.setNsPrefix("",namespaceString);

    // create the pathway ontology
    Ontology ont = biopaxmodel.createOntology(namespaceFileString);
    disptext.append(namespaceString+" ontology created\n");
    // import the biopax-level1 ontology into the pathway ontology
    ont.addImport(biopaxmodel.createResource(biopaxFileString) );
    biopaxmodel.addLoadedImport(biopaxFileString);
    disptext.append(biopaxFileString+" ontology sucessfully imported in
"+namespaceString+" ontology\n");
}

/////This is what happens when you press the button "Add pathway
data"
void addproteins_actionPerformed(ActionEvent e) {
    // First start with some example data
    String pathwayName = "testpathway";
    String[] proteinName = {
"Enamelysin", "Metalloelastase", "Collagenase" };
    String[] proteinDB = { "SWISS-PROT", "RefSeq", "SWISS-PROT" };
    String[] proteinID = { "P023244", "NP_304845", "P308934" };

    String individualName;
    String individualClass;
    String propertyName;
    String linkedresourceName;

    // Create an individual in #pathway for the testpathway
    individualName = "pathway1";

```

```

        individualClass = "pathway";

Biopax.createBiopaxIndividual(individualName, individualClass, biopaxmodel
);
    disptext.append(">> individual: "+individualName+" added to class:
"+individualClass+"\n");
    // Add the property "NAME" to the pathway individual
    propertyName = "NAME";

Biopax.addBiopaxProperty(pathwayName, propertyName, individualName, biopaxm
odel);
    disptext.append(">> literal "+pathwayName+" added to property "
+propertyName+" of individual
"+individualName+"\n");

    // Create an individual in #unificationXref for every protein
    for (int i = 0; i < proteinName.length; i++) {
        // Create the name of the individual
        individualName = "Xref" + i;
        individualClass = "unificationXref";

Biopax.createBiopaxIndividual(individualName, individualClass, biopaxmodel
);
    disptext.append(">> individual: "+individualName+" added to class:
"+individualClass+"\n");
    // Add the properties to the Xref individuals
    propertyName = "DB";

Biopax.addBiopaxProperty(proteinDB[i], propertyName, individualName, biopax
model);
    disptext.append("literal: "+proteinDB[i]+" added to property:
"+propertyName+"\n");
    propertyName = "ID";

Biopax.addBiopaxProperty(proteinID[i], propertyName, individualName, biopax
model);
    disptext.append("literal: "+proteinID[i]+" added to property:
"+propertyName+"\n");
    }

    // Create and individual in #protein for every protein
    for (int i = 0; i < proteinName.length; i++) {
        // Create the name of the individual
        individualName = "Protein" + i;
        individualClass = "protein";

Biopax.createBiopaxIndividual(individualName, individualClass, biopaxmodel
);
    disptext.append(">> individual: "+individualName+" added to class:
"+individualClass+"\n");
    // Add the properties to the Protein individuals
    propertyName = "NAME";

Biopax.addBiopaxProperty(proteinName[i], propertyName, individualName, biop
axmodel);
    disptext.append("literal: "+proteinName[i]+" added to property:
"+propertyName+"\n");
    // Now the property XREF has to be set (linked to the
unificationXref of this protein)
    propertyName = "XREF";
    linkedresourceName = "Xref"+i;

Biopax.addBiopaxLinkedProperty(propertyName, linkedresourceName, individua
lName, biopaxmodel);
    disptext.append(">> property: "+linkedresourceName+" linked to
property: "+
                    propertyName+" of individual:
"+individualName+"\n");
    }

```

```

}

void saveOWL_actionPerformed(ActionEvent e) {
    // Let the user define the filename
    String fileName = JOptionPane.showInputDialog("Give the filename");

    // create an outputstream for the file to be saved
    PrintWriter out = null;
    try {
        // set append to false!!! --> FileOutputStream(fileName, FALSE)!!
        out = new PrintWriter(new FileOutputStream(fileName,false));

        RDFWriter writer = biopaxmodel.getWriter("RDF/XML-ABBREV") ;
        writer.setProperty("xmlbase",namespaceFileString) ;
        // Save the OWL model
        writer.write(biopaxmodel,out,namespaceString);
        disptext.append(">> file saved");
    }
    catch (FileNotFoundException ex) {
        JOptionPane.showMessageDialog(null,"Could not create and/or save
the file: "+fileName);
    }
    out.close() ;
}
}

```

Biopax.java

```
package testpathway;
////////Import the Jena Libraries
import com.hp.hpl.jena.rdf.model.*;
import com.hp.hpl.jena.ontology.*;
import com.hp.hpl.jena.util.*;
import com.borland.jbcl.layout.*;

public class Biopax {
    public static String biopaxString =
"http://www.biopax.org/release/biopax-level1.owl#";
    public static String namespaceString =
"http://somewhere/testpathway.owl#";

    ////////// 'createBiopaxIndividual' creates an individual in the biopax
ontology
    ////////// input: String individualName: the name (not the property NAME)
of the individual to be created
    ////////// String individualClass: the name of the Class where the
individual has to be created
    ////////// OntModel model: the model in which the individual has to
be created

public static void createBiopaxIndividual(String individualName, String
individualClass, OntModel model) {
    // Get the resource of the class where the individual has to be
placed
    Resource res = model.getResource(biopaxString+individualClass);
    // Add the individual to the class
    model.createIndividual(namespaceString + individualName,res);
    // Tell the people what action the class performed
    System.out.println(">> individual: "+individualName+" added to
class: "+individualClass);
}

    ////////// 'addBiopaxProperty' adds a property to a given individual and
adds a literal to that property
    ////////// input:
    ////////// String propertyName: the name of the property to be
added
    ////////// String individualName: the name (not the property NAME)
of the individual to be created
    ////////// OntModel model: the model in which the individual has to
be created
public static void addBiopaxProperty(String propertyLiteral, String
propertyName, String individualName, OntModel model) {
    // Get the individual for which the property has to be added
    Individual ind =
model.getIndividual(namespaceString+individualName);
    // Get the property
    Property prop = model.getProperty(biopaxString+propertyName);
    // Add the literal to the property
    ind.addProperty(prop,propertyLiteral);
    System.out.println(">> literal: "+propertyLiteral+" added to
property: "+
        propertyName+" of individual: "+individualName);
}

    ////////// 'addBiopaxLinkedProperty' adds a property to a given individual
and links it to a resource
    ////////// input:
    ////////// String propertyName: the name of the property to be
added
    ////////// String linkedresourceName: the resource to where the
property has to be linked
    ////////// String individualName: the name (not the property NAME)
of the individual to be created
```

```

//////// OntModel model: the model in which the individual has to
be created
public static void addBiopaxLinkedProperty(String propertyName, String
linkedresourceName, String individualName, OntModel model) {
    // Get the individual for which the property has to be added
    Individual ind =
model.getIndividual(namespaceString+individualName);
    // Get the property
    Property prop = model.getProperty(biopaxString+propertyName);
    // Get the resource which has to be added to the property
    //Resource res = model.getResource(biopaxString+linkedresourceName);
    Resource res =
model.getResource(namespaceString+linkedresourceName);
    // Link the properties
    ind.addProperty(prop,res);
    System.out.println(">> property: "+linkedresourceName+" linked to
property: "+
                                propertyName+" of individual: "+individualName);
}
}

```

Appendix B

This appendix contains the output file generated by the previous program.

testpathway.owl

```
<rdf:RDF
  xmlns:rss="http://purl.org/rss/1.0/"
  xmlns:bp="http://www.biopax.org/release/biopax-level1.owl#"
  xmlns:jms="http://jena.hpl.hp.com/2003/08/jms#"
  xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
  xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
  xmlns:owl="http://www.w3.org/2002/07/owl#"
  xmlns:vcard="http://www.w3.org/2001/vcard-rdf/3.0#"
  xmlns:daml="http://www.daml.org/2001/03/daml+oil#"
  xmlns:dc="http://purl.org/dc/elements/1.1/"
  xmlns="http://somewhere/testpathway#"
  xml:base="http://somewhere/testpathway.owl">
  <owl:Ontology rdf:about="">
    <owl:imports rdf:resource="http://www.biopax.org/release/biopax-
level1.owl"/>
  </owl:Ontology>
  <bp:protein rdf:ID="Protein1">
    <bp:NAME>Metalloelastase</bp:NAME>
    <bp:XREF>
      <bp:unificationXref rdf:ID="Xref1">
        <bp:ID>NP_304845</bp:ID>
        <bp:DB>RefSeq</bp:DB>
      </bp:unificationXref>
    </bp:XREF>
  </bp:protein>
  <bp:protein rdf:ID="Protein0">
    <bp:XREF>
      <bp:unificationXref rdf:ID="Xref0">
        <bp:DB>SWISS-PROT</bp:DB>
        <bp:ID>P023244</bp:ID>
      </bp:unificationXref>
    </bp:XREF>
    <bp:NAME>Enamelysin</bp:NAME>
  </bp:protein>
  <bp:unificationXref rdf:ID="Xref2">
    <bp:DB>SWISS-PROT</bp:DB>
    <bp:ID>P308934</bp:ID>
  </bp:unificationXref>
  <bp:protein rdf:ID="Protein2">
    <bp:NAME>Collagenase</bp:NAME>
    <bp:XREF rdf:resource="#Xref2"/>
  </bp:protein>
  <bp:pathway rdf:ID="pathway1">
    <bp:NAME>testpathway</bp:NAME>
  </bp:pathway>
</rdf:RDF>
```