Netica-J Manual

Version 3.06 and Higher

Java Version of Netica API

Norsys Software Corp

Netica-J Reference Manual
Version 3.06
August 15, 2005

Published by:

Norsys Software Corp.
3512 West 23rd Avenue
Vancouver, BC,
CANADA
V6S 1K5
www.norsys.com

1. Introduction

This reference manual is for Netica-J, the Java version of the Netica API Programmer's Library. It is meant to be used in conjunction with the onscreen Netica-J javadocs reference (see below). Netica-J is a set of Java classes and an accompanying Java Native Interface (JNI) library that allow a Java developer to use the Netica API Programmer's Library.

This manual assumes that you are familiar with the Java programming language. It also assumes familiarity with Bayes nets (Bayesian belief networks) or influence diagrams. Questions and comments about material in this manual may be sent to **netica-j@norsys.com**.

1.1. Netica-J Features

- A complete implementation of the Netica API in Java.
- A clean object-oriented design.
- A comprehensive installation guide and user's manual.
- Sample applications to get you started.
- Source code for dozens of powerful utility functions in "Extra" classes: NetEx, NodeEx, and NodeListEx.
- High performance, relying on the same ultra-reliable industrial-strength computation engine in Netica's C-API (with hundreds of fielded applications over the past eight years).
- Written by Java developers for Java developers. For instance, it:
  - Possesses a complete and thorough javadocs.
  - Uses Java's exception handling mechanism in the natural way.
  - Supports event listening by any Java object for events such as the creation, deletion, duplication, etc. of Nets or Nodes
  - Supports user data fields for any Serializable Java object
  - Supports multiple threads
  - Supports standard Java I/O streams
  - Supplies AWT classes for displaying Bayesian Net components
- Thoroughly tested and reliable.
- Comes at no extra cost with Netica API for other languages. Works with your existing Netica-API license, with the same rights, restrictions, and support arrangements.
- Download it from: http://www.norsys.com/netica-j.html

1.2. Netica Programmer's Library

The Netica Programmer's Library, also known as the Netica API, is described thoroughly in the **Netica API Programmer's Library Reference Manual, C Version,** which is available for free download from http://www.norsys.com/netica_api.html **.** If you have never programmed Netica before, you may want to read the first chapters of that manual first, to familiarize yourself with the general operation of Netica before venturing to use Netica-J.

1.3. Files Included

The following files are included in the distribution of Netica-J, the Java version of Netica API:

Directory	File	Description
doc	• NeticaJ_Man.pdf	• this file
	• javadocs\	• the javadocs directory for Netica-J
	• LicAgree.txt	• a legal document relating to the use of Netica API
bin	• NeticaJ.jar	• the Java class library that defines Netica-J
	• NeticaJ.dll	• the Java-to-Native interface library (Windows only)
	(libNeticaJ.so)	" " " (Unix/Linux only)
	(libNeticaJ.jnilib)	" " " (Mac OS X only)
	• Netica.dll	• the native Netica API library (Windows only)
	(libnetica.a)	" " " (Mac OS X only)
src/neticaEx/	• NetEx.java	• A class containing useful Net methods
	• NodeEx.java	• " " Node "
	• NodeListEx.java	• " " NodeList "
src/neticaEx/ aliases	• Net.java	• A convenience class that renames NodeEx as Node
	• Node.java	• " " " NetEx as Net
	• NodeList.java	• " " " NodeListEx as NodeList
demo	• Demo.java	• a sample application to test your Netica-J installation
	• compile.bat (.sh)	• a sample batch file for compiling Demo.java (.bat for Windows, .sh for Unix/Linux/MacOSX)
	• run.bat (.sh)	• a sample batch file for running Demo.class (.bat for Windows, .sh for Unix/Linux/MacOSX)
examples	• BuildNet.java	• demonstrates building a Bayes net from scratch
	• DoInference.java	• demonstrates doing inference
	• SimulateCases.java	• demonstrates creating case instances that statistically derive from a given net
	• LearnCPTs.java	• demonstrates learning from cases
	• LearnLatent.java	• demonstrates EM Learning
	• ClassifyData.java	• demonstrates Naive Bayesian Classification of real-world medical data
	• MakeDecision.java	• demonstrates building a decision net and choose an optimal decision with it
	• DrawNet.java	• demonstrates use of the gui package for drawing nets
	• NetViewer.java	• demonstrates use of the gui package for editing nets and their findings
	• TestNet.java	• demonstrates testing the performance of a learned net with the net tester tool
	• ChestClinic.dne	• an example net file required by SimulateCases/LearnCPTs/TestNet.java
	• BreastCancer.dne	• an example net file required by ClassifyData.java
	• ChestClinic.cas	• a case file required by TestNet.java
	• LearnLatent.cas	• a case file required by LearnLatent.java
	• BreastCancer.cas	• a case file required by ClassifyData.java
	• compile.bat (.sh)	• a sample batch file for compiling all the java files in this directory
	• run.bat (.sh)	• a sample batch file for running all the java programs in this directory, after they have been compiled

The Netica-J directory structure

The **doc**\ directory contains manuals, javadocs, license agreements, and any other documentation.
The **bin**\ directory contains the Netica-J runtime software without which Netica-J will not function.
The **src**\ directory contains source software that is distributed with Netica-J. You are free to examine, compile, or copy from these source files. We suggest that you leave the original files unmodified. These functions may change in future version of Netica.

The **demo**\ directory contains a simple program that should be compiled and run after installation to establish that your Netica-J system is correctly installed and ready to use.
The **examples**\ directory contains assorted sample data and source code that you may examine, copy, and edit freely.

1.4. License Agreement

Before using Netica-J, make sure you accept the license agreement that is included with this package as file **LicAgree.txt.**

If you have purchased the C-version of Netica API, you are entitled to the same rights and privileges with this Java version, and the license password you were issued will work identically with Netica-J. The same applies in reverse. Indeed, the same Netica API license will work for any member of the Netica API family (Java, C, Basic, etc.), for the same version of that API.

The license-password is provided to the constructor for Environ. For example:

```
Environ env = new Environ ("your unique license string");
```

If you do not have a license, then you can simply supply `null` to the Environ constructor, in which case Netica-J will be fully functional, except for the size of the Nets it can manage, and the number of cases it can learn from at one time.

1.5. Getting Started

Recommended Installation steps:

1. A Java-2 platform is required. There are many suppliers, for example SUN Microsystems at http://java.sun.com/products/

2. Download Netica-J from the Norsys ftp site: ftp://norsys.com/pub/users/norsys/dl/NeticaJ_Win.zip (or the version for your OS/platform)

3. Unzip it, and it will form a directory called NeticaJ_306 (or the current version number).

4. Test your installation with the Demo application provided.

   a) Change to the **demo**\ directory and at the command line, type: `compile.bat` (`compile.sh` on Unix/Linux/MacOS X). Or click on the compile.bat icon. This will compile Demo.java and create Demo.class.

   b) At the command line, type: `run.bat` (`run.sh`). Or click on the run.bat icon. This will run Demo.class.

   c) If it displays a welcome message, and does simple probabilistic inference without declaring any errors, then your installation was probably successful.

5. Now that you have the example program running, you can duplicate the **demo**\ directory, replace Demo.java with your own source files, and you are ready to build your own application. Don't forget to replace "`null`" in "`new Environ(null)`" with your own license password, if you want to have the full functionality of Netica.

6. Demo.java, is a good starting point for developing your own applications. You may wish to "cut-and-paste" from it. Similar examples showing how to build a net from scratch, do inference, generate cases, and learn from cases are provided in the **examples**\ directory.

7. If you are familiar with the Hugin or JavaBayes systems and would like information on equivalent Netica functions, contact Norsys.

1.6. Complete Javadocs Reference

For javadocs-style documentation for Netica-J, simply point your browser at the **index.html** file in the **docs\javadocs** directory. The javadocs very thoroughly document every class and every function of the Netica API. You will find it an invaluable companion during development.

1.7. Customized Installation Considerations

Using Java IDEs (Eclipse,JBuilder, JDeveloper, Forte, etc.)

You must inform your IDE of the locations of the three library files: **NeticaJ.dll** (**libNeticaJ.so**), **NeticaJ.jar**, and **Netica.dll** (**libnetica.a**). Assuming Netica-J was installed at the following location on your filesystem:

Windows: `C:\NeticaJ_306`
Unix/Linux/MacOSX: `/home/NeticaJ_306`

1) NeticaJ.dll(libNeticaJ.so/.jnilib) must appear on the java library path. Typically this is done with a -D option to the JVM. For example:

Windows: `java -Djava.library.path=C:\NeticaJ_306\bin`
Unix/Linux/MacOSX: `java -Djava.library.path=/home/NeticaJ_306/bin`

2) NeticaJ.jar must appear on the java CLASSPATH. For example:

Windows: `java -classpath C:\NeticaJ_306\bin\NeticaJ.jar`
Unix/Linux/MacOSX: `java -classpath /home/NeticaJ_306/bin/NeticaJ.jar`

3) **Windows Only**: Netica.dll must appear on the Windows execution "path, so that Windows can find it. For example:

Windows: `set PATH=C:\NeticaJ_306\bin;%PATH%`

1.8. Feedback welcome

We at Norsys have worked hard to make Netica-J a very high quality and robust package that is easy and natural to use. If you have any ideas for how it can be improved, please send your suggestions to: **netica-j@norsys.com**

1.9. Join the Netica-J Mailing List

If you would like to be notified of version updates and other news regarding Netica-J, please visit https://www.norsys.com/mailing_list.html?interests=Netica-J and supply us with your e-mail address. Mailings are infrequent.

2. Netica-J Package Design and Usage

This section outlines programming principles and issues as they relate to Netica-J's operation and organization. If you are an experienced Java developer or are planning a sizeable development effort with Netica-J, you will definitely want to read and understand this section before beginning your design.

2.1. The "Ex" classes NetEx, NodeEx, and NodeListEx

The "Ex" classes inherit from their parent class (NodeEx extends Node, NetEx extends Net, etc).

They do not offer any additional constants or instance variables. All they offer is additional functionality (the "Ex" stands for "Extra", "Example", "External", "Experimental", and "Excellent!"). These are utilities and convenience methods that were deemed useful, but not basic enough to belong in the base class. Some of them are "Ex" methods because they are more useful in source code form, so that you can customize them to your needs.

Because their Java source is included, the "Ex" classes are a good place to look for coding examples. Indeed, many of the coding examples found in the javadocs are taken from the "Ex" classes.

The "Ex" classes may change in future versions; methods may be added, removed or modified. For this reason, you may want to keep copies of the Ex classes for future reference, or you may want to copy out any methods you need to form your own extensions of the parent classes.

Since the "Ex" classes contain so many useful methods, many users will want to use the "Ex" classes in place of the more basic parent classes. See Section 3, Inheritance, below, for considerations when doing this.

The "Ex" classes are meant to be supported by the entire community of Netica-J users, so please feel welcome to submit additional methods that you have found useful, or to suggest improvements to the ones already there.

Some of the "Ex" class methods are static, while others are not. The basic criterion of choosing to make a method static was whether that method could be thought of as a "standalone-utility" that would be useful to have around even when you didn't have an "Ex" object present. Since none of the "Ex" classes define new state data, it is a trivial exercise to convert a static method to be non-static or vice versa, should you prefer the alternate.

Because the "Ex" classes are so useful, many developers will want to use them directly. To make this easy, their compiled classes have been included in the NeticaJ.jar distribution. All you need do is `import norsys.neticaEx.*;` and you are ready to use them without the need to compile your own versions of them.

Finally, as a convenience, we also supply in the **norsys.neticaEx.aliases** package, three wrapper classes for NetEx, NodeEx, and NodeListEx, that are named Net, Node, and NodeList, respectively. They allow you to use the base class names and still use the Ex classes. See **demo\Demo.java** and **examples\BuildNet.java** for examples of how to use these convenience classes.

2.2. Java objects and native object peers

Since Netica-J is a JNI API, many of the Java objects created are “proxies” of their native or “peer” counterpart objects internal to the core Netica binary. This is true of Environ, Net, Node, NetTester, NeticaError, Sensitivity, and Streamer. The remaining Java classes (General, NeticaEvent, NeticaException, NeticaListener, NodeList, State, User, Util, and VisualNode) do not have peer equivalents. (Those of you familiar with the Netica C API may be surprised that NodeList does not represent the native peer (nodelist_bn). This approach was chosen because nodelist_bn is simply a convenience collection and Java already has excellent collection classes that Java developers prefer to use.)

The existence of peer relationships is usually transparent to the Java developer. Netica-J was designed to give the developer as much as possible the sense he/she is working in a 100% pure Java environment.

The only situations where you need to know about peer objects is when considering finalization and the cleanup of native resources (discussed in the *Finalizers* section, below), or when working in a Model View Controller (MVC) environment where things could be happening to the native model objects, and the Java environment is presenting but one view on that model. This can happen, for instance, if Netica-J is communicating with peer objects inside Netica Application. A user of Netica Application could delete a native node via the GUI, and the Java environment would then find that its Node object had been disconnected from its peer. Netica-J has a standard Java Publish-and-Subscribe mechanism (using NeticaEventListeners) for Java objects to be made aware of such occurrences on the native side of the universe (see the *Event Handling* section, below).

2.3. Exception Handling

Exception handling in Netica-J works in the normal Java way. If a method encounters an unexpected situation it cannot resolve, a NeticaException is thrown. The vast majority of Netica-J methods are able to throw a NeticaException. The toString() method of NeticaException details the reason for the Exception. Hence, your typical try-catch block will look something like this:

```
try {
      // call Netica-J methods
}
catch (NeticaException e) {
      e.printStackTrace();
}
```

If you are familiar with the Netica C API, you will find that Netica-J’s exception handling mechanism makes coding much more convenient and straightforward, since you no longer need to actively check if an error has occurred. Netica-J looks after that for you, and will throw a NeticaException automatically if any “serious” (“show stopper”) error occurs. By “serious” we mean any errors of severity level ERROR_ERR or XXX_ERR which means the requested operation was not completed.

Note that this means that WARNING_ERR and lower warnings do not result in a NeticaException being thrown, so in those cases where such warnings can occur, you can actively call the static method `NeticaError.getWarnings` after the method call, to determine if a

warning has occurred and, if so, what the warning was about. See the javadocs for `NeticaError.getWarnings` for examples of this.

It is okay to call `NeticaError.getWarnings` only once in awhile, since warnings will cumulate until the next `getWarnings` invocation, whereupon they are cleared from the warnings list.

2.4. Inheritance of the Node and Net classes

Advanced users will want to create their own specialized Node and Net classes. To make this task easier, and avoid the need for copy constructors, we have supplied you with a means to inform Netica-J what class you would like it to use when constructing a Net or Node (for example, when Netica-J is reading a net in from a file). The static methods:

```
Net.setConstructorClass (String className)
```
and
```
Node.setConstructorClass (String className)
```

have been supplied for this purpose. All they require is that your Net or Node extension have a default constructor. See their javadocs pages for examples.

Some users will want to use the words "Net" and "Node" for their own net and node classes, that inherit from `norsys.netica.Net` and `norsys.netica.Node`, respectively. The supplied files in **src\neticaEx\aliases** have examples of this. Although, overloading the terms "Net" and "Node" like this is not difficult, namespace conflicts may arise. In general, if you explicitly import your Node or Net class, the Java compiler will use those as the default classes.

2.5. Event Handling

If you wish your program to receive events, Netica-J has the ability to call your program when such events occur.

Any Java object can choose to listen to Netica events by simply implementing the NeticaEventListener interface and asking the node or net that generates the events to add itself to that node or net's listener list. The methods `Node.addListener` and `Net.addListener` are supplied for this purpose. Since Node and Net objects are already NeticaEventListeners, they each possess an `eventOccurred(NeticaEvent)` method. If you should choose to override this method, it is important that you call the base class method (`super.eventOccurred(event);`) in your method, so that this node or net will still be able to handle delete events properly.

Future versions of Netica-J will include more types of events. If you have a request, please let us know.

2.6. Multi-threading

If you are running Netica-J within a single process and are not creating more than one thread in that process, you needn't be bothered with this issue. The large majority of users will fall in this category. However, if you are operating in a concurrent usage environment, then please read on.

Netica-J's approach to thread-safety is to share the responsibility of thread-safety with the developer. Netica-J guarantees thread-safety whenever two threads are operating on two separate nets, and also when two threads are operating on separate nodes, or sets of nodes (where the sets share no member nodes), within the same net. However, in the case where two threads can operate on the same node simultaneously, Netica-J does not guarantee thread safety, and so in this case only, it is the responsibility of the developer to synchronize such code. This compromise was made for efficiency reasons, since it is very likely you will need to synchronize such code blocks anyhow, in order to avoid situations where one thread is changing a node that another thread is simultaneously using.

2.7. Finalizers & Memory Management

For large networks and large node tables, Netica can consume large amounts of memory. Often Java developers cease to worry about memory management, as the JVM's garbage collector will automatically collect Java objects that can no longer be referenced. However, the Java specification does not require that a JVM actually call the garbage collector whenever a Java object reference goes out of scope. It may or may not do so, and it may choose to do so on its own schedule. Accordingly, you may want to actively call the `delete()` or `finalize()` methods on resource-hungry objects when you are done with those objects, rather than wait for the JVM to free them.

Note that ever since Netica-J 2.21, `Net.delete()` has been merged into `Net.finalize()` so that calling `Net.finalize()` will now free all of a net's native resources.

This is not the case for Node. `Node.finalize()` and `Node.delete()` remain distinct since you typically do not want a node's native resources to be freed just because the node is out of scope. Call `Node.delete()`when you want to free its native resources.

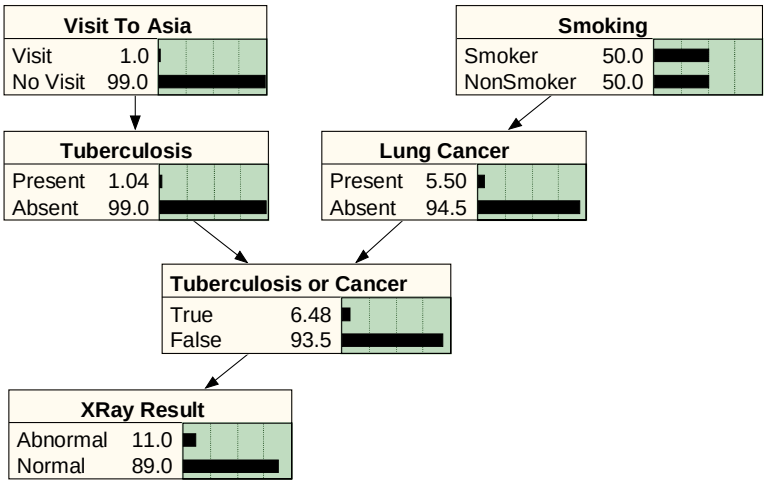
Note, if you do override the `finalize()` method of any Netica-J class, be certain that you always call the base class finalizer method (`super.finalize();`) as your last instruction, so that Netica-J can do its own housekeeping upon the Java object being collected. For example, if your class extends norsys.netica.Streamer, and you need to override the `finalize()` method to perform special close-down handling of files and such, then your finalize method would look something like this:

```
/**
 *  overrides Streamer.finalize().
 */
public void finalize() throws NeticaException {
    . . .  your own finalization logic . . .
    super.finalize();
}
```

3. Sample Software

3.1. Building and Saving a Net

The following program can be found in the **examples** directory of your Netica-J installation. It builds the following Bayes net in memory:



For additional information on this topic, see the *Building and Saving Nets* chapter in the **C API Manual.**

```
/*
 *  BuildNet.java
 *
 *  Example use of Netica-J to construct a Bayes net and save it to file.
 */
import norsys.netica.*;
import norsys.neticaEx.aliases.Node;

public class BuildNet {
    public static void main (String[ ] args){
        try {
            Node.setConstructorClass ("norsys.neticaEx.aliases.Node");
            Environ env = new Environ (null);

            Net net = new Net();
            net.setName ("ChestClinic");

            Node  tuberculosis    = new  Node  ("Tuberculosis",    "present, absent",      net);
            Node  smoking         = new  Node  ("Smoking",         "smoker, nonsmoker",    net);
            Node  cancer          = new  Node  ("Cancer",          "present, absent",      net);
            Node  tbOrCa          = new  Node  ("TbOrCa",          "true, false",          net);
            Node  xRay            = new  Node  ("XRay",            "abnormal, normal",     net);
            Node  visitAsia       = new  Node  ("VisitAsia",       "visit, no_visit",      net);
```

```
            visitAsia.setTitle ("Visit to Asia");
            cancer.setTitle ("Lung Cancer");
            tbOrCa.setTitle ("Tuberculosis or Cancer");

            visitAsia.state("visit").setTitle ("Visited Asia within the last 3 years");

            tuberculosis.addLink (visitAsia);       // puts link from visitAsia to tuberculosis
            cancer.addLink (smoking);
            tbOrCa.addLink (tuberculosis);
            tbOrCa.addLink (cancer);
            xRay.addLink (tbOrCa);

            visitAsia.setCPTable (0.01, 0.99);
            smoking.setCPTable (0.5,  0.5);

                                        // VisitAsia     present    absent
            tuberculosis.setCPTable     ("visit",        0.05,      0.95);
            tuberculosis.setCPTable     ("no_visit",     0.01,      0.99);

                                    // Smoking        present    absent
            cancer.setCPTable       ("smoker",        0.1,       0.9);
            cancer.setCPTable       ("nonsmoker",     0.01,      0.99);

                                   // TbOrCa      abnormal     normal
            xRay.setCPTable        ("true",         0.98,        0.02);
            xRay.setCPTable        ("false",        0.05,        0.95);

            tbOrCa.setEquation ("TbOrCa (Tuberculosis, Cancer) = Tuberculosis || Cancer");
            tbOrCa.equationToTable (1, false, false);

            Streamer stream = new Streamer ("ChestClinicBuilt.dne");
            net.write (stream);


            net.finalize();  // free resources immediately and safely; not strictly necessary,
                             // but a good habit
        }
        catch (Exception e){
            e.printStackTrace();
        }
    }
}
```

After this program executes you may want to look at the `ChestClinicBuilt.dne` file it creates using a text editor.

3.2. Performing Probabilistic Inference

The following program can be found in the **examples** directory of your Netica-J installation. It reads back into memory the ChestClinicBuilt.dne file created above and then does probabilistic inference on the net. For additional information on this topic, see the *Probabilistic Inference* chapter in the **C API Manual.**

```
/*
 *  DoInference.java
 *
 *  Example use of Netica-J for doing probabilistic inference.
 */
import norsys.netica.*;

public class DoInference {
    public static void main (String[ ] args){
        try {
            Environ env = new Environ (null);

            // Read in the net created by the BuildNet.java example program.
            Net net = new Net (new Streamer ("ChestClinicBuilt.dne"));

            Node visitAsia      = net.getNode ("VisitAsia");
            Node tuberculosis   = net.getNode ("Tuberculosis");
            Node xRay           = net.getNode ("XRay");

            net.compile();

            double belief = tuberculosis.getBelief ("present");
            System.out.println ("\nThe probability of tuberculosis is " + belief);

            xRay.finding().enterState ("abnormal");
            belief = tuberculosis.getBelief ("present");
            System.out.println ("\nGiven an abnormal X-ray,\n" +
                                "the probability of tuberculosis is " + belief);

            visitAsia. finding().enterState ("visit");
            belief = tuberculosis.getBelief ("present");
            System.out.println ("\nGiven an abnormal X-ray and a visit to Asia,\n" +
                                "the probability of tuberculosis is " + belief + "\n");


            net.finalize();
        }
        catch (Exception e){
            e.printStackTrace();
        }
    }
}
```

3.3. Creating Cases

The following program can be found in the **examples** directory of your Netica-J installation. It reads back into memory the ChestClinic.dne file created from the BuildNet example above and then creates a case file of cases that follow the probability distribution of that file.

```
/*
 *  SimulateCases.java
 *
 *  Example use of Netica-J for generating random cases that follow
 *  the probability distribution given by a Bayes net.
 */
import java.io.File;
import norsys.netica.*;

public class SimulateCases {

    public static void main (String[ ] args){
        int numCases = 200;
        System.out.println ("Creating " + numCases + " random cases...");

        try {
            Environ env = new Environ (null);

            // Read in the net created by the BuildNet.java example program.
            Net  net = new Net (new Streamer ("ChestClinicBuilt.dne"));
            NodeList nodes = net.getNodes();

            (new File ("ChestClinic.cas")).delete();     // "ChestClinic.cas" may exist from a
previous run
            Streamer caseFile = new Streamer ("ChestClinic.cas");

            net.compile();

            for (int n = 0;  n < numCases;  n++) {
                net.retractFindings();
                int res = net.generateRandomCase (nodes, 0, 20);
                if (res >= 0)
                        net.writeFindings (caseFile, nodes,  n, -1.0);

            net.finalize();
            }
        }
        catch (Exception e){
            e.printStackTrace();
        }
    }
}
```

3.4. Learning CPTs from Cases

The following program can be found in the **examples** directory of your Netica-J installation. It learns the CPTables (not structure) for a net structurally identical to ChestClinicBuilt.dne from the case file (ChestClinic.cas) generated by the SimulateCases program above. For additional information on this topic, see the *Learning From Cases* chapter in the **C API Manual.**

```
/*
 *  LearnCPTs.java
 *
 *  Example use of Netica-J for learning the CPTs of a Bayes net
 *  from a file of cases.
 */
import java.io.File;
import norsys.netica.*;

public class LearnCPTs {

    public static void main (String[ ] args) {
        System.out.println ("Running Netica-J LearnCPTs example...");

        try {
            Environ env = new Environ (null);

            // Read in the net created by the BuildNet.java example program.
            Net net = new Net (new Streamer ("ChestClinicBuilt.dne"));
            NodeList nodes = net.getNodes();
            int numNodes = nodes.size();

            // Remove CPTables of nodes in net, so new ones can be learned.
            for (int n = 0;  n < numNodes;  n++) {
                Node node = (Node) nodes.get (n);
                node.deleteTables();
            }
            // Read in the case file created by the SimulateCases.java
            // example program, and learn new CPTables.

            Streamer caseFile = new Streamer ("ChestClinic.cas");
            net.reviseCPTsByCaseFile (caseFile, nodes, 1.0);

            net.write (new Streamer ("Learned_ChestClinic.dne"));


            net.finalize();
        }
        catch (Exception e){
            e.printStackTrace();
        }
    }
```

```
}

        /*=====================================
         * This alternate way can replace the net.reviseCPTsByCaseFile
         * line above, if you need to filter or adjust individual cases.
         */
        long[ ] casePosn = new long[1];
        casePosn [0] = Net.FIRST_CASE;
        while (true) {
            net.retractFindings();
            net.readCase (casePosn, caseFile, nodes, null, null);
            if (casePosn[0] == Net.NO_MORE_CASES)  break;

            net.reviseCPTsByFindings (nodes, 1.0);
            casePosn[0] = Net.NEXT_CASE;
        }
        /*=====================================*/
```

3.5. Drawing Nodes and Nets

Version 3 of Netica-J includes several Java-SWING components for displaying Netica nets and nodes. The color, layout, and general appearance of the displayed components is very similar to the style used in the Netica™ application program (see http://www.norsys.com/netica.html for details, to purchase, or to download a size-restricted free version.)

With the exception of norsys.netica.VisualNode, all of the classes relevant to graphical display can be found in the **norsys.netica.gui** package. The two most important classes in this package are NodePanel and NetPanel. Each of these is a javax.swing.JPanel that displays assorted graphical components (e.g., JLabels) within itself. You typically have full access to these subcomponents, and so can change colors, fonts, and borders, attach event listeners, set visibility, etc., just as you would with any AWT/SWING component.

The philosophy behind the development of the **gui** package is to enable you to very easily and rapidly add graphical displays to your Netica-J programs. For instance, the following tiny program is all that is needed to display a net.

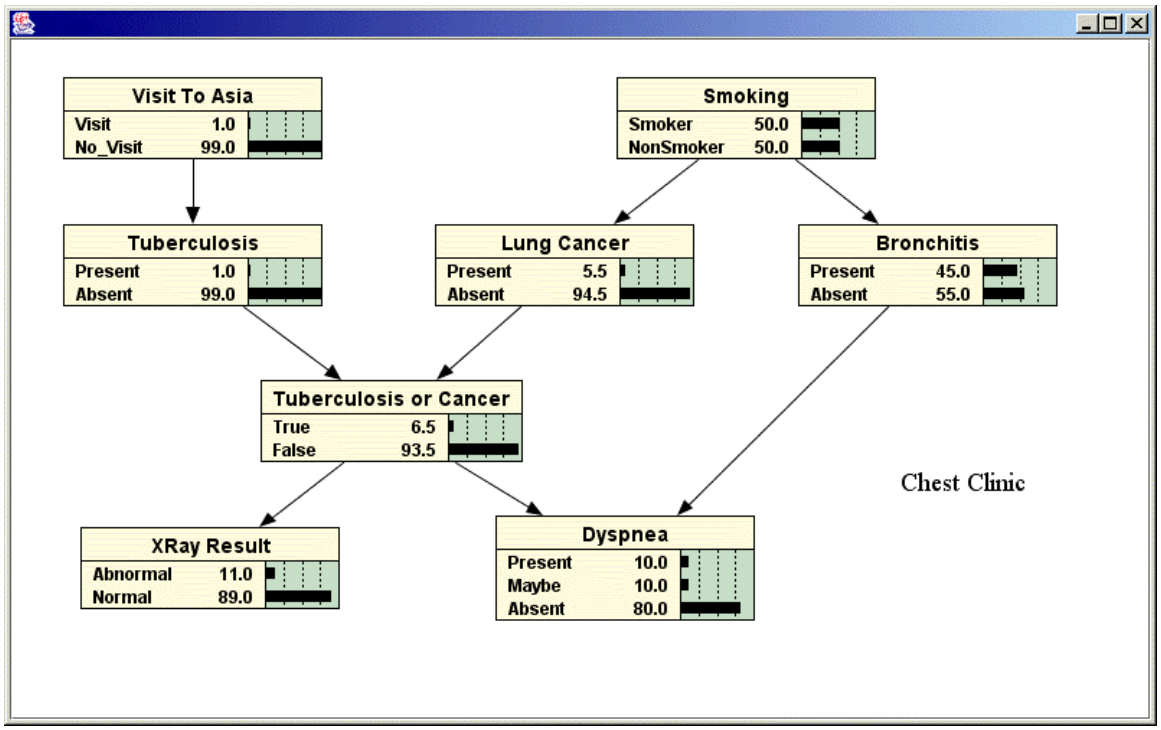
```
import norsys.netica.*;
import norsys.netica.gui.*;
import javax.swing.*;

class DrawNet extends JFrame{

   public DrawNet( String netName ) throws Exception {
        Net net = new Net( new Streamer( netName ) );
        net.compile();  //optional
        NetPanel netPanel = new NetPanel( net, NodePanel.NODE_STYLE_AUTO_SELECT );
        getContentPane().add( new JScrollPane(netPanel ) ); //adds the NetPanel to ourself
        setDefaultCloseOperation( JFrame.EXIT_ON_CLOSE );
        setSize( 800, 500 ); // or supply getPreferredSize();
        show();
   }
```

```
    public static void main( String[] args ) {
        try {
            Environ env = new Environ( null );
            DrawNet dn = new DrawNet( args[0] );
        }
        catch ( Exception e ) {
            e.printStackTrace();
        }
    }
}
```

You call the program with: `java DrawNet SomeNet.dne` (or .neta) and it will draw the net in a fashion similar to the way that Netica™ does, with the nature nodes drawn using the popular "belief bar" style. If you'd prefer another style, then simply replace NodePanel.NODE_STYLE_AUTO_SELECT in the NetPanel constructor call with the style of your choice. Here is the window that `DrawNet Asia.dne` creates:



Relationship of Netica net visual properties and the gui package

Netica files (.dne and .neta files) may optionally contain visual information as to the size, position, and visual style of the nodes they contain. This visual information is used both by the Netica™ application and the Netica-J gui package in order to help decide where and how to display the net components. The general policy of a Netica graphical application is to attempt to make sense of and employ the visual information that is present, but if it is not able to do so, it will ignore that information.

Node Position

If a Netica file has been saved without visual information, then all of the Nodes are by default given a position of (0,0). To add position information, you may use the Netica™ application to display the net and then position the Nodes as desired, or from within the Java API you may use Node.visual().setPosition().

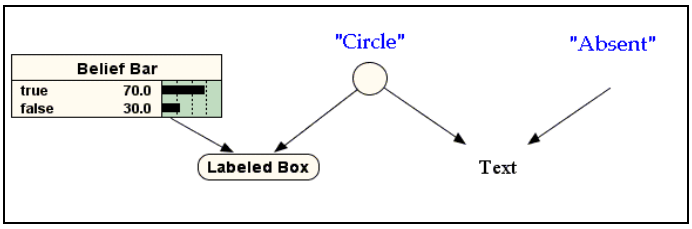
Once a NodePanel has been created, it is a Java component that can moved anywhere (e.g., using java.awt.Component.setLocation()) without affecting the Node.visual() position data. If you want to keep the displayed position in sync with the Netica visual() position, you must either manage this yourself, or else confine yourself to moving the component by calling NodePanel.moveBy().

Node Style

If a Node does not contain any style information (introduced using Netica Application or by calling Node.visual().setStyle()), then NeticaJ will apply certain default styles when creating NodePanels (NodePanel.createNodePanel()) for that node. Conversely, if the node does contain style information, that information will be treated as the preferred style when NeticaJ is given an option in deciding what variety of NodePanel to create.

Drawing Nodes

You do not require the NetPanel class to draw nodes. You may draw the nodes directly using any of several NodePanels that are supplied:



The NodePanel class itself is an abstract base class that manages a number of common functions of all NodePanels, such as hooking up to the Netica Node that is represented and discovering it's title (or name, if it lacks a title), managing event listeners, and managing display modes (hi-lighted, normal, or grayed).

Event Handling

Besides being standard JComponents and hence being able to partake of all the standard Java AWT/SWING events, NetPanel and NodePanel objects are also norsys.netica.gui.RecursingEventListeners. The RecursingEventListener interface provides two very convenient methods: **addListenerToAllComponents**(java.util.EventListener eventListener) and **removeListenerFromAllComponents**(java.util.EventListener eventListener). These will recursively run through all the subcomponents of the NetPanel or NodePanel and attach the EventListener to those subcomponents. Thus, with a single command, you can add an event listener to all the components within a NetPanel.

NetViewer

Included in this distribution, in the **examples** directory, is a reasonably sophisticated program, NetViewer.java. This program allows you to select a net from a list, whereupon it draws the selected net, and then allows you to edit the net and to enter finding information as well. It illustrates how to attach mouse events to nodes, to parts of nodes (e.g., the belief-bars rows), and even to links. If you need to build a more sophisticated graphical application for selecting nodes, entering findings, and such, you may wish to use this program as a starting point.

Miscellaneous Useful Features

- The NetPanel class supports the concept of a selection set, which is just a NodeList of nodes in the "hi-lited state" (see NodePanel.getDisplayMode). Also, you may choose to display a subset of the nodes in a net using the setSubnet method.

Feedback Wanted

We would appreciate hearing whether you find the gui package useful and how you might like to see it evolve.