Netica-J Manual

Version 2.12

Java Version of Netica API

Netica-J Reference Manual
Version 2.12
January 22, 2002

Published by:

Norsys Software Corp.
2315 Dunbar Street
Vancouver, BC, Canada
V6R 3N1
www.norsys.com

1. Introduction

This reference manual is for Netica-J, the Java version of the Netica API Programmer’s Library. Netica-J is a set of Java classes and accompanying Java Native Interface (JNI) libraries 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 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 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,** which is available for free download from **ftp://ftp.norsys.com/pub/users/norsys/dl/** as either **NeticaAPIMan.ps** (PostScript version) or **NeticaAPIMan.rtf.zip**. If you have never programmed Netica before, you will likely 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.

1.2. Files Included

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 (.so)	• the Java-to-Netica interface library (.dll for Windows, .so for Unix/Linux)
	• Netica.dll (.so)	• the native Netica API library (.dll for Windows, .so for Unix/Linux)
src	• NetEx.java	• A class containing useful Net methods
	• NodeEx.java	• “ “ Node “
	• NodeListEx.java	• “ “ NodeList “
demo	• Demo.java	• a sample application to test your Netica-J installation
	• Net.java	• a sample class that extends norsys.netica.Net and is used by Demo.java
	• Node.java	• a sample class that extends norsys.netica.Node and is used by Demo.java
	• compile.bat (.sh)	• a sample batch file for compiling Demo.java (.bat for Windows, .sh for Unix/Linux)
	• run.bat (.sh)	• a sample batch file for running Demo.class (.bat for Windows, .sh for Unix/Linux)
examples	• BuidNet.java	• demonstrates building a Bayes net from scratch
	• DoInference.java	• demonstrates doing inference
	• Learning.java	• demonstrates learning from cases
	• AsiaEx.dne	• an example net file required by Learning.java

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, but we suggest that you not edit them. 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 program files that you may examine and edit freely.

1.3. 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 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.

1.4. 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_212 (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. This will compile Demo.java and create Demo.class.

   b) At the command line, type: run.bat. 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. A similar example showing how to generate and learn from cases is provided in \examples\Learning.java. This and other useful sample software can be found in the **\examples** directory. The compile.bat and run.bat files from the \demo directory can be copied to the \examples directory and used on Learning.java by changing the "rem" statements within them.

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

1.5. 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.

1.6. Customized Installation Considerations

<u>Using Java IDEs (VisualAge, JBuilder, JDeveloper, Forte, etc.)</u>

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

Windows: `C:\NeticaJ_212`
Unix/Linux: `/home/NeticaJ_212`

1) NeticaJ.dll(.so) 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_212\bin`
   Unix/Linux: `java -Djava.library.path=/home/NeticaJ_212/bin`

2) Netica-J.jar must appear on the java CLASSPATH. For example:
   Windows: `java -classpath C:\NeticaJ_212\bin\Netica-J.jar`
   Unix/Linux: `java -classpath /home/NeticaJ_212/bin/Netica-J.jar`

3) Netica.dll(.so) must appear on the Windows(Unix/Linux) execution "path", so that Windows(Unix/Linux) can find it. For example:
   Windows: `set PATH=C:\NeticaJ_212\bin;%PATH%`
   Unix/Linux: `setenv path /home/NeticaJ_212/bin:$path`

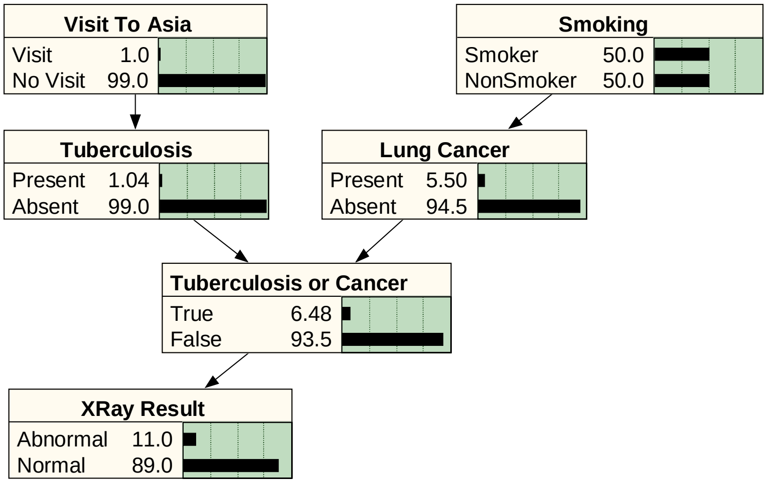
1.7. Feedback welcome

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**

2. Sample Software

2.1 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 *Chapter 3, Building and Saving Nets*, in the **C API Manual.**

```
/*
 *  BuildNet.java
 *
 *  Example use of Netica-J for constructing a Bayes net and
 *  saving it to file.
 */
import NodeEx;
import norsys.netica.*;

public class BuildNet {

  public static void main (String[] args){
    try {
       Node.setConstructorClassName ("NodeEx");
       Environ env = new Environ (null);

       Net net = new Net();
       net.doc().setName("AsiaEx");

       NodeEx tuberculosis = new NodeEx ("Tuberculosis","present,absent",  net);
       NodeEx smoking      = new NodeEx ("Smoking",     "smoker,nonsmoker",net);
       NodeEx cancer       = new NodeEx ("Cancer",      "present,absent",  net);
       NodeEx tbOrCa       = new NodeEx ("TbOrCa",      "true,false",      net);
       NodeEx xRay         = new NodeEx ("XRay",        "abnormal,normal", net);
       NodeEx visitAsia    = new NodeEx ("VisitAsia",   "visit,no_visit",  net);
       visitAsia.docState("visit").setTitle ("Visited Asia within last 3 years");

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

       visitAsia.setCPTable (0.01, 0.99);
```

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

        smoking.setCPTable (0.5, 0.5);

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

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

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

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

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

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

2.2 Performing Probabilistic Inference

The following program can be found in the **\examples** directory of your Netica-J installation. It reads back into memory the AsiaEx.dne file created above and then does probabilistic inference on the net. For additional information on this topic, see *Chapter 2, Probabilistic Inference*, in the **C API Manual.**

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

public class DoInference {

  public static void main (String[] args){
    try {
        Node.setConstructorClassName ("NodeEx");
        Environ env = new Environ (null);

        Streamer stream = new Streamer ("AsiaEx.dne");
        NetEx net = new NetEx(stream);

        NodeEx  visitAsia    = (NodeEx) net.getNode( "VisitAsia"    );
```

```
      NodeEx  tuberculosis = (NodeEx) net.getNode( "Tuberculosis" );
      NodeEx  xRay         = (NodeEx) net.getNode( "XRay"         );

      net.compile();

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

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

      visitAsia.enterFinding ("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.delete();
    }
    catch (Exception e) {
      e.printStackTrace();
    }
  }
}
```

2.3 Learning from Cases

The following program can be found in the **\examples** directory of your Netica-J installation. It reads back into memory the AsiaEx.dne file created above and then learns CPTables (not structure). For additional information on this topic, see *Chapter 7, Learning From Cases*, in the **C API Manual.**

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

public class Learning {

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

    try {
      Environ env = new Environ (null);
      Net.setConstructorClassName ("NetEx");
      Node.setConstructorClassName ("NodeEx");

      Streamer netFile      = new Streamer ("AsiaEx.dne");
      NetEx      origNet    = new NetEx (netFile);
      NodeList origNodes    = origNet.getNodes();

      NetEx    learnedNet   = new NetEx ();
```

```
        //-- duplicate all the nodes from origNet into learnedNet
        NodeList learnedNodes = origNet.duplicateNodes (origNodes, learnedNet);
        int      numNodes     = learnedNodes.size();

        //-- remove CPTables of nodes in learned net, so new ones can be learned.
        for (int n=0; n < numNodes; n++) {
            NodeEx node = (NodeEx) learnedNodes.get (n);
            node.deleteTables();
        }

        (new File("temp.cases")).delete(); // in case "temp.cases" existed from
                                           // a previous run
        Streamer caseFile = new Streamer ("temp.cases");

        origNet.compile();

        int numCases = 120;
        for (int n=0; n < numCases; n++) {
            origNet.retractFindings();
            int res = origNet.generateRandomCase (origNodes, 0, 20);
            if (res >= 0) {
               origNet.writeCase (origNodes, caseFile, n, -1.0);
            }
        }
        //-- So, now we read back the cases and try to learn a net with CPTables
        //-- similar to origNet.

        long[] casePosn = new long[1];
        casePosn[0] = Net.FIRST_CASE;
        while (true) {
            learnedNet.retractFindings();
            learnedNet.readCase (casePosn, caseFile, learnedNodes, null, null);
            if (casePosn[0] == Net.NO_MORE_CASES) break;

            learnedNet.reviseCPTsByFindings (learnedNodes, 1.0);
            casePosn[0] = Net.NEXT_CASE;
        }

        netFile  = new Streamer ("Learned_AsiaEx.dne");
        learnedNet.write (netFile);

        origNet.delete();
        learnedNet.delete();
     }
     catch (Exception e) {
        e.printStackTrace();
     }
   }
 }
```

3. Netica-J Package Design and Usage

This section outlines theoretical 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.

3.1. 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", “Experimental”, “Example”, “External”, 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 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 welcomed 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.

3.2. Java objects and native object peers

- Since Netica-J is a JNI API, many of the Java objects created are reflections of their native or “peer” counterpart objects. This is true of Environ, Net, Node, NetTester, NeticaError, Sensitivity, and Streamer. The remaining Java classes (Doc, NeticaEvent, NeticaException, NeticaListener, NodeList, NodeVisual, and Util) 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 section 3.6, Finalizers, below), or when working in a Model View Controller (MVC) environment where things could be happening to the 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. There exists a full and complete standard Java Publish-and-Subscribe mechanism (using NeticaEventListeners) for Java objects to be made aware of such happenings on the native side of the universe (see section 3.5, Event Handling, below).

3.3. Exception Handling

Exception handling in Netica-J works exactly as you would expect. 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 C API to Netica, 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 to see if an error has occurred after every API function call. Netica-J looks after that for you, and will throw a NeticaException automatically if any “serious” error occurs. By “serious” we mean any errors of type ERROR_ERR or XXX_ERR. These are “show stopper” errors that probably mean you have made an error and need to correct the situation.

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 getWarning() invocation, whereupon they are cleared from the warnings list.

3.4. 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 the you with a means by which you can inform Netica-J of what class you would like it to use when constructing a Net or Node, say when Netica-J is reading a net in from file. The static methods:
  `Net.setConstructorClassName( String className)` and
  `Node.setConstructorClassName( 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 “Node” and “Net” for their own Net and Node classes, that inherit from norsys.netica.Net and norsys.netica.Node, respectively. The supplied files in \demo have examples of this. Although, overloading the terms “Net” and “Node” like this is not difficult, namespace conflicts will arise, and you will have to deal with ambiguity issues at compile-time and sometimes run-time. In general, if you explicitly import your Node or Net class, the Java compiler will reserve that term as the abbreviated name of your class.

3.5. 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 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. `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.

3.6. 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 has taken an approach to thread-safety that we believe makes the most sense. This 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. Netica-J's design team decided that to have extra locking mechanisms operating within Netica-J to prevent this, would add little real protection, and would significantly decrease runtime performance.

3.7. 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 JVM cannot automatically collect memory that was allocated by the native Netica system, and large networks and node tables fall in this category. Therefore, it is the responsibility of the developer to supply finalizers that will automatically free native resources whenever the Java object is to be garbage collected, if this is the desired behavior. In a single-threaded system, this is typically always the desired behavior. In multi-threaded systems, especially if remote processes are involved, you may not be the owner of the native resources, and so freeing them may not be desirable. You will have to decide these issues yourself. **Most users will want to override the default finalizer in their Net extension classes, or, if they are not using extension classes, then will want to call net.delete() when they are done using a net object.**

If you do wish to override Net.finalize(), be certain 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, the following will delete the native Netica net upon the corresponding Java Net object being garbage collected. If your class extends norsys.netica.Net, you will likely want to just cut-and-paste it into that class:

```
/**
 *  overrides Net.finalize().
 */
public void finalize() throws NeticaException {
    delete();
    super.finalize();
}
```

Please be aware that the Java specification does not require that a JVM actually call the garbage collector whenever a Java object reference goes out of scope, so you may want to actively call the delete() methods on resource-hungry objects anyhow, rather than wait for the JVM to free them, since it may wait a long time before doing so, or even never actually get around to doing so.