Unité Mixte de Recherche 7585 CNRS - LPNHE - IN2P3
Universités Paris VI et Paris VII
4 place Jussieu
Tour 43 rez-de-chaussée
75252 Paris Cedex 05
Tél +033 144 27 63 13
Fax +033 144 27 46 38
http://www-lpnhep.in2p3.fr

NeCTAr Project

Software documentation

Nicolas Roche, Jean-François Huppert

February 15, 2016

NeCTAr Project

Software documentation

Nicolas Roche, Jean-François Huppert

LPNHE

February 15, 2016

Résumé

This is the Java software documentation for NeCTAr.

Mots clés: NeCTAr, software

Responsables: Mr Pascal VINCENT, Mr François LEGRAND

Annotations:

Contents

Introduction

- **Projet Description**

  NECTAR means New Electronics for CTA Readout.
  It is a French project to build integrated Electronics for CTA. Mostly readout and data transfer electronics.

  You will find here the software and his documentation:

  - Tools we use.
  - The code we produce.
  - External documentations.

  Thanks reading me.

- **TODO List:**

  - activate GEDEK without ping (Java cannot ping without root privileges)

    ```
    Le GEDEK n'a pas besoin d'être pingé pour être activé.
    Le ping est de notre point de vue une solution simple afin que le GEDEK connaisse
    l'adresse MAC/IP hôte vers lequel il doit envoyer ses données.

    Cette information est contenue dans les registres de l'IP (interface Reg_*)
    @2 (Destination MAC Address (32 LSB)),
    @3 (Destination MAC Address (16 MSB))
    et @4 (Destination IP Address).
    Vous pouvez donc les modifier afin de les configurer avec les valeurs qui vous
    conviennent.
    ```

  - implement write/read to the virtals registers of the GEDEK:

    ```
    reg 0x00: OK
    reg 0x01: OK
    reg 0x02: OK
    reg 0x42: OK
    reg 0xff: OK
    ```

  - tell that servelet isn't possible without a servelet server (like tomcat).

- **LPNHE team**

name	job	know-how	contact
Pascal Vincent *	Physics	Dev. low level	vincentp@lpnhe.in2p3.fr
Jean-Paul Tavernet	Physics	PM & Tests	jean-paul.tavernet@lpnhe.in2p3.fr
Julien Bolmont	Physics	PM	bolmont@lpnhe.in2p3.fr
Patrick Nayman *	Electronique	cards conception	nayman@in2p3.fr
François Toussenel *	Electronique	monitoring/assembly	francois.toussenel@lpnhe.in2p3.fr
Pascal Corona	Electronique	drawer tests/debuging	pascal.corona@lpnhe.in2p3.fr
Jean-François Huppert	Computer science	Dev. bas niveau	jean-francois.huppert@lpnhe.in2p3.fr
Richard Randriatoamanana *	Computer science	management	richard.randria@lpnhe.in2p3.fr
Nicolas Roche	Computer science	Dev.	01.44.27.41.98 nicolasf.roche@gmail.com

* : officer in charge.

- **Contacts**

name	entity	know-how	contact

Part I

Products

You will find here explanations about the JAVA software.
It consist on a tool 'oriented efficiency' for the electronic team.

- Here are the functional specifications:

  1. We must use JAVA technology for code portability.
  2. We must use APPLET technology to pull the over someone's eyes (plug and play).
  3. So, the client should be able connect form the internet even if it only have one ETHERNET card.
  4. We must develop a **single** client (no serveur application nor client/router configuration at all).
  5. We are using a private network *(we will provide a switch/router that make ip routage/masquerading)*.

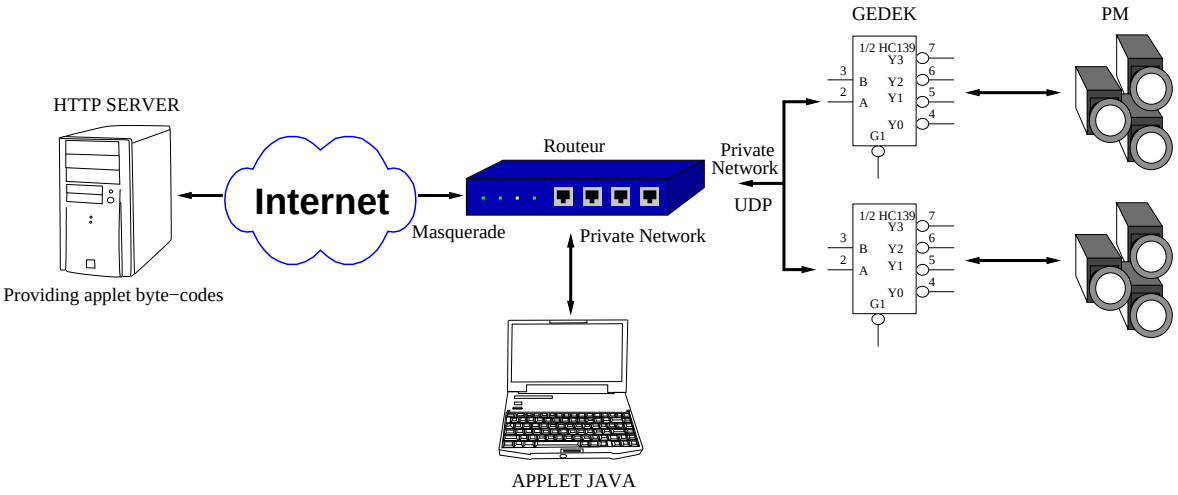

Figure 1: Functional specifications

- Here are the headlines we are working on:

  1. We only use JAVA technology for code portability (no regression tests).
  2. We use WEBSTART so as to launch the client from a browser but not to use the browser's JVM.
  3. We develop a single client (no serveur application nor router configuration at all).
  4. The client is connected to the private network and use *masquerading* to connect to the internet.

- Here are the millstones :

  1. Main functionnalities:
     (a) Capture data files from the GEDEK board.
     (b) Develop a **Main** abstract class using a **Log** classe and that contain all static data.
     (c) Make choice for the media to parse (socket or file).
     (d) Make the parser store data in JAIDA memory.
  2. Develop the GUI:
     (a) Display embeded plots using JAIDA.
     (b) Display the logs in a windows.
  3. WebStart:
     (a) Run the application from an HTML page served by an HTTP server.

(b) Sign the JAR file to give permission on the client side.

(c) embebed default configuration in the JAR file.

4. (todo) Find portables solution:

   (a) (todo) try the soft on the WINDOWS operating system.

   (b) (todo) write configuration and log file in temporary directory depending on the OS.

   (c) (todo) Configuring Virtual IP Address in Windows XP/Vista/Seven usiong VLAN.

   (d) (todo) Configuring the switch using a DHCP client.

5. Know bug and limitation

   (a) The Java virtual machine memory may required to be increase when we will use many drawers.

   (b) The GUI application freeze sometime (Threading priority ?)

   (c) According to *Programmer en Java 5ème édition Java 5 et 6, Claude Delannoy, Eyrolles p318*: we should not use thread priority in portable developpement. So we use the `Trhead.sleep()` function.

Chapter 1

User interface

1.1 Graphical user interface

1.1.1 Where to store the global variables

In order to enable multiples entry points from our sources (for instance one main function for a GUI and on other for a command-line application), we define a static class that old the main independents objects.

```
package nectar;
import javax.swing.*;

public class Main {
    static final String confFile = "etc/nectar.xml";
    static UI_Camera camera = null;
    static public JPanel gui = null;
    static Semaphore semSelect = null;
}
```

As it, we may develop several GUI that derive from the same interface (wich as to be defined). For now we use casting to call the GUI's functions from the other objects :

```
((GUI_GraphicalEmbryo) Main.jPanel).plotEvtCounter(counter);
```

1.1.2 Entry point for GUIs

Here is the `Main` class. It give us a first `JPanel` empty object and the `matchPanel` function that allow to fill it with any GUI object.

```
package nectar;
import javax.swing.*;
import java.awt.*;

public class Main extends javax.swing.JFrame {

    // add a JPanel into a container (build the GUI object tree)
    static void matchPanel(Container target, JPanel pluggin) {
        GroupLayout logPanelLayout = new GroupLayout(target);
        target.setLayout(logPanelLayout);
        logPanelLayout.setHorizontalGroup(
            logPanelLayout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
            .addComponent(pluggin, javax.swing.GroupLayout.DEFAULT_SIZE, javax.swing.GroupLayo
            );
        logPanelLayout.setVerticalGroup(
            logPanelLayout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
            .addComponent(pluggin, javax.swing.GroupLayout.DEFAULT_SIZE, javax.swing.GroupLayo
            );
    }

    public Main() {
        initComponents();

        if (Main.gui != null) {
            Main.matchPanel(getContentPane(), Main.gui);
        }
        setVisible(true);
    }

    private void initComponents() {
        setDefaultCloseOperation(javax.swing.WindowConstants.EXIT_ON_CLOSE);
        javax.swing.GroupLayout layout = new javax.swing.GroupLayout(getContentPane());
        getContentPane().setLayout(layout);
```

```
        layout.setHorizontalGroup(
            layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
            .addGap(0, 785, Short.MAX_VALUE)
        );
        layout.setVerticalGroup(
            layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
            .addGap(0, 427, Short.MAX_VALUE)
        );
        pack();
    }

    // Creates the main windows
    public static void main(String args[]) {
        java.awt.EventQueue.invokeLater(new Runnable() {
            public void run() {
                if (Main.gui == null) {
                    UT_GraphicalEmbryo.ut(); // the default entry point.
                }
                new Main();
            }
        });
    }
}
```

The `Main.main()` function will be our entry point call by all the GUI application (unit tests too). For instance :

```
public class GUI_Log extends JPanel {
    ...
    // Unitary test for GUI_Log
    public static void main(String args[]) {
        Main.jPanel = new GUI_Log();
        Main.main(null);
    }
```

1.2 input/output API

1.2.1 Introduction

This page give the API to talk to the GEDEK card using UDP sockets :

- port **1024**: DAQ or input from the GEDEK card
- port **1040**: SLC or order to the GEDEK card
- port **1056**: internal GEDEK register bank
- port **1072**: NeCTAr slow control chip

remarque:

- Convertir un hexadécimal en décimal en ligne de commande

```
$ printf "%i\n" 0x0400
$ echo "ibase=16; obase=A; 400" | bc
1024
```

- Convertir un décimal en hexadécimal en ligne de commande

```
$ printf "%x\n" 1040
$ echo "ibase=10; obase=16; 1040" | bc
410
```

1.2.2 Serializer

7 blocks are used to configure the slow control of the board. All the following words are 32 bits.

CntrlSlc (ok)

CntrlSlc is used to program the threshold for the pixel, the pixel sum and the common mode voltage. It is 24 bytes long.

```
CntrlSlc    = AAAA-AAAA
              0000-7E0C
              Data
              AAAA-AAAA

Data        = Thrshld1
              Thrshld2
              VMC

Thrshld1    = 0000-00XX (8 bits)
Thrshld2    = 0000-00YY (8 bits)
VMC         = 0000-0ZZZ (12 bits)
```

Here is a test file: cntrlSlc.bin.

- Thrshld1 = 170
- Thrshld2 = 85
- VMC = 4095

CntrlNectarReg

CntrlNectarReg is used to program the internal SAM registers. It is 32 bytes long.

```
CntrlNectarReg = AAAA-AAAA
                 0000-7E3E
                 Data
                 AAAA-AAAA

Data           = CDR1
                 CDR2
                 TestFCR
                 CDR6
                 CDR7

CDR1           = XXXX-XXXX
CDR2           = XXXX-XXXX
TestFCR        = XXXX-XXXX
CDR6           = XXXX-XXXX
CDR7           = XXXX-XXXX
```

Here is a test file: cntrlNectarReg.bin.

- CDR1 = 1111-1111
- CDR2 = 2222-2222
- TestFCR = FFFF-FFFF
- CDR6 = 6666-6666
- CDR7 = 7777-7777

CntrlNectarDac

CntrlNectarDac is used to program the 32 internal SAM DAC's. It is 80 bytes long.

```
CntrlNectarDac = AAAA-AAAA
                 0000-7E3A
                 Data
                 AAAA-AAAA

Data           = MemNum
                 DACL{16}

MemNum         = 0000-000X [0:15]
DACLx          = XXXX-XXXX
```

Here is a test file: cntrlNectarDac.bin.

- MemNum = 2
- DACLx = xxxx-xxxx

CntrlNectarNd

CntrlNectarNd is used to define the delay pointer **Nd**. It is 16 bytes long.

```
CntrlNectarNd = AAAA-AAAA
                0000-7E3C
                Nd
                AAAA-AAAA

Nd            = 0000-YXXX
Y             = NeCTAr number (F=broadcast)
 XXX          = NdValue [0:1023]
```

Here is a test file: cntrlNectarNd.bin.

- Y = F
- Nd = 9

CntrlReadBack

CntrlReadBack is 16 bytes long.

```
CNTRLRealBack = 0000-AAAA
                0000-7E40
                0000-0Y00
                0000-AAAA

Y           = 1: DAC, 2: Nd, 3: NeCTAr chip reg
```

Here is a test file: cntrlReadBack.bin.

- Y = 1

CntrlIntReg

CntrlIntReg is used to program an read back the internal GEDEK core registers. In case of read back, the message does not include the data. It is 16 or 36 bytes long.

```
CNTRLIntReg = AAAA-AAAA
              0000-7E50
              0000-000Y
              Data{0,1}
              AAAA-AAAA

Y           = 1: write the following registers, 0: read back
Data        = XXXX-XXXX  FPGA Board MAC Address (32 LSB)
            = XXXX-XXXX  FPGA Board IP Address
            = XXXX-XXXX  Destination MAC Address (32 LSB)
            = 0000-XXXX  Destination MAC Address (16 MSB)
            = XXXX-XXXX  Destination IP Address
```

Here are 2 test files for test:

- cntrlIntReg0.bin :
  - Y = 0
- cntrlIntReg1.bin :
  - Y = 1
  - FPGA MAC 32LSB = 75:d6:34:3f
  - FPGA IP = 127.0.0.1
  - Destination MAC 32 LSB = 75:d6:34:3f
  - Destination MAC 16 MSB = 00:04
  - Destination IP = 127.0.0.1

CntrlDAQ

CntrlDAQ is used to define the DAQ parameters. It is 16 bytes long.

```
CntrlDAQ    = AAAA-AAAA
              0000-7E30
              Data
              AAAA-AAAA

Data        = 0000-XXXY
XXX         = Nf
   Y        = unused QSampls T0 ToT

Nf          = "valeure Nf codée sur 10 bits" [0:1023]
unused      = 1 bit (inutilisé)
QSampls     = 1 bit (0:Sample 1:Charge)
T0          = 1 bit
ToT         = 1 bit
```

T0 et Tot: demande d'ajouter l'info T0 aux blocs DAQ. Ie:

- une fenêtre est composée de Nf samples
- T0 donne le numéreau de sample du maximum du signal dans la fenêtre
- ToT donne le nombre de samples au dessus du seuil (threshold) dans la fenêtre

Here is a test file: cntrlDaq.bin.

- Nf = 1023
- Qsampls = 1
- T0 = 1
- ToT = 1

1.2.3 Parser

The following blocks are used to receive data in either charge mode or sampling mode. In order to get a correct *IPNum* word, it is mandatory to launch at least one time the **CntrlIntReg Y=0** block to read the internal registers.

DAQ IntReg

DAQIntReg is used to read back the internal GEDEK core registers. It is 32 bytes long.

```
DAQIntReg   = BBBB-BBBB
              XXXX-XXXX  FPGA Board MAC Address (32 LSB)
              XXXX-XXXX  FPGA Board IP Address
              XXXX-XXXX  Destination MAC Address (32 LSB)
              0000-XXXX  Destination MAC Address (16 MSB)
              XXXX-XXXX  Destination IP Address
              0000-000X  Host detected status [0:1]
              BBBB-BBBB
```

Here is a test file: daqIntReg.bin.

- FPGA MAC 32LSB = 75:d6:34:3f
- FPGA IP = 127.0.0.1

- Destination MAC 32 LSB = 75:d6:34:3f
- Destination MAC 16 MSB = 00:04
- Destination IP = 127.0.0.1
- Status = 1

DAQ charge

Data in charge mode is 48 bytes long:

```
DAQCharge   = AAAA-AAAA
              0000-EEE0
              FPGA's IPNum
              EvtCounter
              DataBlock
              AAAA-AAAA

FPGA IPNum  = XXXX-XXXX
EvtCounter  = 0000-XXXX
DataBlock   = PM{7}
PM          = Ch_Charges
Ch_Charges  = YYYY-ZZZZ

YYYY        = Data2
     ZZZZ   = Data1
```

Here is a test file: daqCharge.bin.

- FPGA IPNum = 127.0.0.1
- EvtCounter = 1
- DataBlocks = 0X0Y-0X0Y ({PM}{Channel}):
  `0000-0001 0100-0101 0200-0201...`

DAQ sample

Data in sampling mode is (5 + 14*Nf)*4 bytes long:

```
DAQSample   = 0000-AAAA
              0000-EEE3
              FPGA's IPNum
              EvtCounter
              DataBlock
              0000-AAAA

FPGA IPNum  = XXXX-XXXX
EvtCounter  = 0000-XXXX
DataBlocks  = PM{7}
PM          = Ch_Sample{2}
Ch_Sample   = Data{Nf}
Data        = 0000-XXXX
```

Here is a test file: daqSample.bin.

- FPGA IPNum = 127.0.0.1
- EvtCounter = 1
- NF = 2
- Data = 0000-XYZZ ({PM}{Channel}{NF})

Chapter 2

Modules

2.1 Tests

2.1.1 Introduction

Non regression tests are needed in all serious project. They consist to automate units tests by sending them alway the same input data and comparing the output to the expected good one.

2.1.2 Unary tests

Each JAVA file should contain a single class. In all theses class we define a `main` function that permit to test it without the global context. Under NETBEANS you select the file you want to test and then you press `maj-F6`. Else:

```
$ java -cp dist/nectar.jar nectar.ConfXml
```

2.1.3 Non regression tests

We need to run the tests under ANT, using externals tools like JDIFF (We decided to use only *oriented-java* tools). For now, it is too heavy to import such a tools into the developpement environment (and to stay portable). It should be something like:

```
$ java -cp dist/nectar.jar nectar.ConfXml < inputTest1 > outputTest1
$ [ -z "$(diff -q outputTest1 expectedTest1)" ] && echo "test ok" || echo 'test failed'
```

2.2 Log

2.2.1 Howto log

A Logger object is added to the main Class Nectar. We must add it because if we make our own class, the logging messages will display this new class instead of the original calling line.

`UI_Log` and `GUI_Log` will both log into the log file and the log window if it is allocated.

```
package nectar;
import java.io.IOException;
import java.util.logging.FileHandler;
import java.util.logging.Level;
import java.util.logging.Logger;
import java.util.logging.SimpleFormatter;

public class Nectar {
    Logger logger;

    Nectar() {
        // This block configure the logger with handler and formatter
        logger = Logger.getLogger("Nectar");
        FileHandler fh;

        try {
            fh = new FileHandler("/tmp/NectarFile.log", true);
            logger.addHandler(fh);
            logger.setLevel(Level.ALL);
            SimpleFormatter formatter = new SimpleFormatter();
            fh.setFormatter(formatter);

        } catch (SecurityException e) {
            e.printStackTrace();
        } catch (IOException e) {
            e.printStackTrace();
        }
    }

    public static void main(String[] args) {
        Nectar all = new Nectar();
        all.logger.log(Level.SEVERE,"testing SEVERE log level");
        all.logger.log(Level.WARNING,"testing WARNING log level");
        all.logger.log(Level.INFO,"testing INFO log level");
        all.logger.log(Level.CONFIG,"testing CONFIG log level");
        all.logger.log(Level.FINE,"testing FINE log level");
        all.logger.log(Level.FINER,"testing FINER log level");
        all.logger.log(Level.FINEST ,"testing FINEST log level");
    }
}
```

Note that you cannot log to a file using a non-signed applet.

2.2.2 Modify format

```
    /**
    * MyCustomFormatter formats the LogRecord as follows:
    * date   level   localized message with parameters
    */
    static public class MyCustomFormatter extends Formatter {

public MyCustomFormatter() {
            super();
```

```
}

public String format(LogRecord record) {
            // Create a StringBuffer to contain the formatted record
            // start with the date.
            StringBuffer sb = new StringBuffer();

            // Get the date from the LogRecord and add it to the buffer
            /*Date date = new Date(record.getMillis());
            sb.append(date.toString());
            sb.append(" ");*/

            // Get the calling function
            sb.append(record.getSourceClassName() + ".");
            sb.append(record.getSourceMethodName() + ": ");

            // Get the level name and add it to the buffer
            sb.append(record.getLevel().getName());
            sb.append("\t");

            // Get the formatted message (includes localization
            // and substitution of paramters) and add it to the buffer
            sb.append(formatMessage(record));
            sb.append("\n");

            return sb.toString();
}
    }
...
    MyCustomFormatter f = new MyCustomFormatter();
    FileHandler       fh = new FileHandler(logFile, true);
    fh.setFormatter(f);
    logger.addHandler(fh);
```

2.2.3 Log into a windows

```
    static class MyWindowHandler extends Handler {
        /**
        * This is the overridden publish method of the abstract super class
        * Handler. This method writes the logging information to the associated
        * Java window. This method is synchronized to make it thread-safe. In case
        * there is a problem, it reports the problem with the ErrorManager, only
        * once and silently ignores the others.
        *
        * @record the LogRecord object
        *
        */
        public synchronized void publish(LogRecord record) {
            String message = null;
            //check if the record is loggable
            if (!isLoggable(record))
                return;
            try {
                message = getFormatter().format(record);
                System.out.println(":) " + message);
            } catch (Exception e) {
                System.out.println("error: " + e.getMessage());
            }
```

```
        }

        public void close() {}
        public void flush() {}
    }
...
    MyWindowHandler wh = new MyWindowHandler();
    logger.addHandler(wh);
```

Chapter 3

Libraries

3.1 UDP Sockets

3.1.1 Include

UDP are include in the JAVA languages classes :

```
import java.net.*;
```

3.1.2 ICMP client

ICMP ECHO REQUESTs is used if the privilege can be obtained, otherwise it will try to establish a TCP connection on port 7 (Echo) of the destination host.

```
String host;
boolean status;
status = InetAddress.getByName(host).isReachable(timeOut);
```

comments: the GEDEX wait for an ICMP ECHO REQUEST in order to set the internals registers @2 (Destination MAC Address (32 LSB)), @3 (Destination MAC Address (16 MSB)) and @4 (Destination IP Address).

3.1.3 UDP client

```
String          host;
int             port;
byte[]          buffer = new byte[1024];
InetAddress     address = InetAddress.getByName(host);
DatagramSocket  socket = new DatagramSocket();
DatagramPacket  packet = new DatagramPacket(buffer, buffer.length, address, port);

socket.send(packet);
socket.close();
```

3.1.4 UDP server

```
String          host;
int             port;
byte[]          buffer = new byte[1024];

DatagramSocket serverSocket = new DatagramSocket(port);

while(1) {
   DatagramPacket serverPacket = new DatagramPacket(buffer, buffer.length);
   serverSocket.receive(serverPacket);
}

serverSocket.close();
```

3.2 XML configuration file

We use the JDOM library in order to parse the XML files.

```
JDOM is an open source Java-based document object model for XML that was designed specifically
Java platform so that it can take advantage of its language features.
JDOM integrates with Document Object Model (DOM) and Simple API for XML (SAX), supports XPath
```

3.2.1 Include

```
import org.jdom.input.*;
import org.jdom.*;
import org.jdom.xpath.*;
```

Comments: we add the *jdom.jar* and the *jaxen.jar* files to the environment.

3.3 JAIDA data analysys

3.3.1 Include

We use the JAIDA library.

```
import hep.aida.*;
```

Comments: we add all the */lib/*.jar* files to the environment. Outside NETBEANS you should do :

```
$ export CLASSPATH=`find .../freeHep/jaida-3.3.0-6/lib -name '*.jar' -exec echo -n "{}:" \;`
$ javac Histogram.java
$ java Histogram
```

You will find exemples here.

3.3.2 Embedded the plotter

By default JAIDA will build a pop-up windows using the `plotter.show();` call. The following exemple will embed the plotter in a `JPanel` object.

```
package nectar;
import javax.swing.*;
import java.util.logging.Level;
import hep.aida.*;
import hep.aida.ref.plotter.PlotterUtilities;
import java.awt.BorderLayout;
import org.freehep.swing.popup.GlobalMouseListener;
import org.freehep.swing.popup.GlobalPopupListener;

public class GUI_PM extends JPanel {

   UI_PM pm = null;
   static IPlotter plotter = null;

   public GUI_PM()
   {
      super(new BorderLayout());

      plotter = UI_PM.af.createPlotterFactory().create("je veux pas de pop-up");
      plotter.createRegions(1,2,0); // grid for 2 plots
      plotter.setParameter("plotterWidth","400");
      plotter.setParameter("plotterHeight","600");

      // Now embed the plotter
      add(PlotterUtilities.componentForPlotter(plotter), BorderLayout.CENTER);

      // Enable the context menu like in the popup windows (bug)
      // (http://bugs.freehep.org/browse/JAIDA-165)
      GlobalMouseListener gml = new GlobalMouseListener(this);
      gml.addMouseListener(new GlobalPopupListener());
   }

   void plotLastSample() {
       // plots
       if (pm != null) {
           for (int channel = 0; channel<2; channel++)
               plotter.region(channel).plot(pm.dpsLast[channel], "mode=replace");
```

```
        }

        // plots title
        plotter.region(0).setTitle("Event sample (low gain)");
        plotter.region(1).setTitle("Event sample (high gain)");
        // plotter.refresh(); ne sert à rien.
    }

    public static void main(String args[]) {
         UI_Log.allocate();
         UI_Log.logger.log(Level.INFO,"This is the GUI_PM class.");

         GUI_PM gui_pm = new GUI_PM();
         gui_pm.pm = new UI_PM();
         gui_pm.pm.addValuesExemple1();
         gui_pm.plotLastSample();

         Main.jPanel = gui_pm;
         Main.main(null);
     }
}
```

3.3.3 2D plot and 1D histogram

Statics objetcs needed

```
package nectar;
import hep.aida.*;
import java.util.logging.Level;
import java.util.Random;

...
    final static IAnalysisFactory af = IAnalysisFactory.create();
    final static ITree tree = af.createTreeFactory().create();
    final static IHistogramFactory hf = af.createHistogramFactory(tree);
    final static IDataPointSetFactory dpsf = af.createDataPointSetFactory(tree);
    final static IFitFactory fitF = af.createFitFactory ();
    final static IFunctionFactory funcF = af.createFunctionFactory (tree);
```

1D histogram

```
public class UI_PM {
    IHistogram1D  1Dhisto = null;

    UI_PM() {
        // binning of 1 and range of [0:maxDataValue]
        1Dhisto = hf.createHistogram1D("Charge Hard (low gain)", maxDataValue, 0, maxDataValue
    }

    public void addCharge(int value) {
        1Dhisto.fill(value);
    }
}
```

2D plot

```
public class UI_PM {
```

```
    IDataPointSet 2Dgraph = null;

    UI_PM() {
        2Dgraph = dpsf.create("sample","Event sample (low gain)",1);
        for ( i = 0; i<windowSize; i++ ) {
            2Dgraph.addPoint();
        }
    }

    public void addSample(int[] sample) {
        int j, chargeSoftValue;

        for ( j=0; j<windowSize; j++ ) {
            2Dgraph.point(j).coordinate(0).setValue(sample[j]);
        }
    }
}
```

Fit

```
public class UI_PM {
    IFitter fitter = null;
    IFitResult fitted = null;

    UI_PM() {
        fitter = fitF.createFitter("chi2");
    }

    public void fit(){
       int i,j;
/*
       double[] fPars;
       double[] fParErrs;
       String[] fParNames;
*/
       fitted = fitter[i].fit(chargeSoft[i],"g");
/*
       fPars     = fittedSoft[i].fittedParameters();
       fParErrs  = fittedSoft[i].errors();
       fParNames = fittedSoft[i].fittedParameterNames();

       for(j=0; j< fittedSoft[i].fittedFunction().numberOfParameters(); j++ )
           System.out.println("Soft= " + fParNames[j]+" : "+fPars[j]+" +- "+fParErrs[j]);
*/
   }
}
```

Part II

Miscellaneous

Chapter 1

Operating Systems

1.1 Windows

1.1.1 Remote access

Remote desktop (Virtual Network Computing)

This allow you to connect remotely to the WINDOWS computeur:

- Download, install and launch, a VNC server computer.
- Install and launch a VNC client on the client computer (for instance you may use *gtkvncviewer* on LINUX or use the link above that give also a client for WINDOWS).

SSH client & X serveur

This only concern WINDOWS computeur used as client (personnal computer) and allow to run {LI|U}NIX graphical applications.

- Download and install and launch an X server
- Download, install and launch an SSH client
- Configure your SSH client to use
  - Compression: `PuTTY >>> SSH >>> Enable compression`
  - X11 forwarding: `PuTTY >>> SSH >>> X11 >>> Enable X11 forwarding`
- Now you should be able to connect to a {LI|U}NIX machine that export display to your monitor (try `$ xeyes`).

1.1.2 Java Runtime Environment & Development Kit

- Download and install the last JDK. **rq**: The Java Development Kit (JDK) contain the Java Runtime Environment (JRE).
- Now you are able to run the Java executable and also to compile it:
  - double click on the *nectar\dist\nectar.jar* file.

1.1.3 Integrated development environment

- Please download and install the last JDK.
- Download and install NETBEANS.

1.2 Fedora

1.2.1 Running Nectar software

Java Runtime Environment & Development Kit

You should be already be able to compile and run Java executables using openJDK, but we prefer to install the sunJDK.

- Download and install the last JDK' RPM. **rq**: The Java Development Kit (JDK) contain the Java Runtime Environment (JRE).

```
# chmod +x jdk-6u19-linux-i586-rpm.bin
# ./jdk-6u19-linux-i586-rpm.bin

rq:  # rpm --erase jdk-1.6.0_19-fcs          (will remove it)
```

- You may want to make the SUN JDK the default one.

```
$ rpm -ql jdk-1.6.0_19-fcs | grep bin/javac    (the sun jdk)

$ which java
$ ls -l /usr/bin/java*           (link pointing to open jdk)
# cd /usr/bin
# rm -f java*                                (replacing links)
# find /usr/java/jdk1.6.0_19/bin/java* -exec ln -s {} . \;

$ /etc/alternatives/java -version                  (testing)
$ java -version
```

Hello World

```
$ cat > hello.java
class HelloWorldApp {
    public static void main(String[] args) {
        System.out.println("Hello World!"); // Display the string.
    }
}
^D

$ javac hello.java
$ java HelloWorldApp
```

Nectar application

- double click on the *nectar/dist/nectar.jar* file.
- or

```
$ cd nectar/dist
$ java -jar nectar.jar
```

1.2.2 Integrated development environment

- Download and install the last JDK (like above).

```
# chmod +x jdk-6u19-linux-i586-rpm.bin
# ./jdk-6u19-linux-i586-rpm.bin
```

- Download and install NETBEANS.

```
# yum install netbeans
```

1.3 Scientific Linux

1.3.1 Running the Nectar application

Java Runtime Environment & Development Kit

You should be already be able to compile and run Java executables using openJDK, but we prefer to install the sunJDK.

- Download and install the last JDK' RPM. **rq**: The Java Development Kit (JDK) contain the Java Runtime Environment (JRE).

```
# chmod +x jdk-6u19-linux-i586-rpm.bin
# ./jdk-6u19-linux-i586-rpm.bin

rq:  # rpm --erase jdk-1.6.0_19-fcs          (will remove it)
```

- You may want to make the SUN JDK the default one.

```
$ rpm -ql jdk-1.6.0_19-fcs | grep bin/javac    (the sun jdk)

$ which java
$ ls -l /usr/bin/java*           (link pointing to open jdk)
# cd /usr/bin
# rm -f java*                                (replacing links)
# find /usr/java/jdk1.6.0_19/bin/java* -exec ln -s {} . \;

$ /etc/alternatives/java -version                  (testing)
$ java -version
```

Hello World

```
$ cat > hello.java
class HelloWorldApp {
    public static void main(String[] args) {
        System.out.println("Hello World!"); // Display the string.
    }
}
^D

$ javac hello.java
$ java HelloWorldApp
```

Nectar application

- double click on the *nectar/dist/nectar.jar* file.
- or

```
$ cd nectar/dist
$ java -jar nectar.jar
```

1.3.2 Integrated development environment

- Download and install the last JDK (like above).

```
# chmod +x jdk-6u19-linux-i586-rpm.bin
# ./jdk-6u19-linux-i586-rpm.bin
```

- Download and install NETBEANS.

```
# yum install netbeans
```

1.4 Debian

1.4.1 Running Nectar software

Java Runtime Environment & Development Kit

- Install the Java Development Kit (JDK).

```
# apt-get install ant sun-java6-jdk
```

Hello World

```
$ cat > hello.java
class HelloWorldApp {
    public static void main(String[] args) {
        System.out.println("Hello World!"); // Display the string.
    }
}
^D

$ javac hello.java
$ java HelloWorldApp
```

Nectar application

- double click on the *nectar/dist/nectar.jar* file.
- or

```
$ cd nectar/dist
$ java -jar nectar.jar
```

1.4.2 Integrated development environment

- Download and install the last JDK (like above).
- Download and install NETBEANS.

```
# apt-get install netbeans-ide
```

Chapter 2

Tools

2.1 Cvs

2.1.1 Administration

CVS configuration files are in the CVS module *CVSROOT*. In order to update it you fist have to check-out it, modify it and finaly to commit the changes. Because CVS is not implemented as a serveur (it is served via `xinetd`), there is no need to explicitely reload the configuration.

```
$ cvs -d cvs -d :pserver:roche@lpnp90.in2p3.fr:/home/cvsroot login
$ cvs co CVSROOT
$ cd CVSROOT
...
$ commit
```

Howto change passwd

- This php code function will crypt a new password.
- Next you will have to put it in the *CVSROOT/passwd* file:

  ```
  huppert:$1$wPmUCqO0$GzVFOzLeXV872qr0PPpRN/:hess
  ```

Howto add user priviledges

The priviledges are store in the *CVSROOT/avail* file:

```
unavail                          ;; lock whole repository
avail   |denauroi, roche          ;; allow total access
avail   |roche, huppert      |nectar
```

comment: only `roche` and `denauroi` are allow to commit the *CVSROOT* module.

2.2 Tcpdump

2.2.1 Sniffing the private network

- We see that the GEDEX once powered will send data to the default IP address `192.168.1.207`.

```
# tcpdump tcpdump -i eth0 -nn
14:27:43.126030 IP 192.168.1.18.1234 > 192.168.1.207.1234: UDP, length 1104
...

# tcpdump tcpdump -i eth0 -nn -x -c1
```

- However it also use the default destination MAC address `00:19:66:32:2e:2a`.

```
# tcpdump tcpdump -i eth0 -nn -e
14:29:12.115233 00:07:ed:a1:b2:c4 > 00:19:66:32:2e:2a, ethertype IPv4 (0x0800), length 11
```

So, the GEDEX doesn't make ARP query to make the corespondance between MAC and IP.

- Sniffing bytes on a special port:

```
# tcpdump tcpdump -i eth0 -nn -X port 1235
16:27:52.233635 IP 192.168.1.207.41889 > 192.168.1.18.1235: UDP, length 4
        0x0000:  4500 0020 f29c 4000 4011 c3fe c0a8 01cf  E.....@.@.......
        0x0010:  c0a8 0112 a3a1 04d3 000c 844f 0004 ff00  ...........O....
```

tcpdump -i eth0 -nn -X port 1235

note: TCPDUMP set network interfaces into promiscuous mode.

```
# tail /var/log/syslog
Jun  4 01:42:30 narval kernel: eth2: Promiscuous mode enabled.
Jun  4 01:42:30 narval kernel: device eth2 entered promiscuous mode
```

2.3 NetCat

`netcat` is a more performant `telnet` application:

- it allow to use UDP transport layer.
- it allow to capture response without using `sleep` (but we don't need it here).

2.3.1 Listenning and sending

There are two differents implémentation under RED-HAT and DEBIAN families.

- using FEDORA:

```
$ nc -u -l 1235
                                        $ nc -u 127.0.0.1 1235
                                        coucou
coucou
```

- using DEBIAN:

```
$ nc -u -l -p 1235
                                        $ nc -u 127.0.0.1 1235
                                        coucou
coucou
```

2.3.2 Sending and receiving bytes

```
$ nc -u -l 1235 < hexaQuery
                                    $ nc -u 127.0.0.1 1235 > hexaResponse
```

Building an hexa file

- We first have to build an empty file:

  ```
  $ dd if=/dev/zero of=/hexaQuery bs=4 count=4
  ```

  We use data block of 32 bits.

- Next we have to edit it:

  ```
  $ hexedit hexaQuery
  ^Q
  (or)
  $ hexcurse hexaQuery
  ^S ^Q
  ```

  As Java is using big-endian and PC little-endian, we have to translate each 4 bytes as : abcd => dcba. IE: `67 45 23 01` should be translate as `01 23 45 67`.

Reading an hexa file

```
$ hexdump hexaResponse
```

Warning: on PC, `hexdump` and `tcpdump` permute each block of 2 bytes (I don't know why). IE: `23 01 67 45` should be translate as `01 23 45 67`.

2.3.3 Testing the GEDEK

Please consider this code: there is only 5 register working on the 256 allowed ?!

```
$ ./go
] Monitoring UDP Port 0x04d3 for registers
reg 0x00: OK
reg 0x01: OK
reg 0x02: OK
reg 0x42: OK
reg 0xff: OK
] Done !
```

- Capture the NECTAR outputs:

  ```
  $ nc -u -l 1234 > outData
  ```

- (todo) Sending NECTAR order:

  ```
  $ nc -u 192.168.1.18 1234 < inOrder
  ```

- Updating GEDEK virtuals registers:
  Build the *inRegister* query file with these bytes:
  `00 FF 08 01  78 56 34 12`

  ```
  $ nc -u 1235 < putRegister
  ```

- Reading GEDEK virtuals registers:
  Build the *inRegister* query file with these bytes:
  `00 FF 04 01`

  ```
  $ nc -u -l 1235 < getRegister
  ```

  ```
  $ nc -u -l 1235 > outData

  $ hexdump outData
   0000 0104 3412 7856
  ```

2.4 Netbeans

2.4.1 Testing

New project

```
File > New Project > Java > Java Application

// TODO code application logic here
System.out.println("Hello World!");

shift-F11
```

Web Start

Enable Java Web Start

With Java Web Start, the user can launch a Java application by clicking an HTML link to a JNLP file for this application inside a web browser. The JNLP file, which is a special configuration file, instructs Java Web Start to download, cache, and run the Java application. To run applications with Java Web Start, it is enough to have a compatible version of the Java Runtime Environment (JRE) installed on the client machine. The installation of the Java Development Kit (JDK) is not required.

To enable your Java application to run with Java Web Start, you need to configure the properties of how the IDE should build the project. When Java Web Start is enabled in project properties, the IDE automatically creates a JNLP file and an HTML page with the link to the JNLP file, together with the JAR file. Configuring the Project to Enable Java Web Start

At first, we configure the project to make it Java Web Start enabled and test its execution locally.

1. Right-click the project node and choose Properties.
2. Under Categories, choose Web Start and select the Enable Web Start checkbox.
3. Leave the Local Execution option selected as the Codebase as we will first run the application locally.
4. The Codebase Preview field shows the path to local application files.
5. Ensure that the Self-signed checkbox is selected.
6. The application JAR file will be signed by a certificate that is generated automatically when the project is built. With the self-signed certificate, the application will be able to access the same resources from the computer as a regular application that is running locally. For example, the self-signed certificate allows an application to access local files and the network.Modifying the JNLP File
7. (Optional). In the Project Properties dialog box, select the Applications panel and change the application title and vendor name.
8. Click OK to close the Project Properties dialog box.

Modifying the JNLP File

To launch the application from the web, you need to provide a link to the applications source file on the web in the JNLP file.

1. Right-click the project node, choose Properties and select Web Start under Categories.
2. Choose User Defined as the Codebase.

3. In the Codebase Preview field, enter the URL where you will be uploading the source files. For example, `http://lpnp90.in2p3.fr/~roche/nectari/dist`.

4. Click OK in the Project Properties window.

5. Right-click the project node and choose Clean and Build.

Uploading the Source Files

Just copy the `/dist` directory to the web served directories.

```
$ rm -fr /htdocs/nectari/dist
$ cp -fr cvs/nectar/dist /htdocs/nectari/.
```

Modify the `codebase` in the *launch.jnlp* file and eventualy remove the *.jar* library that cannot be signed :

```
<?xml version="1.0" encoding="UTF-8"?>
<jnlp spec="1.0+" codebase="http://lpnp90.in2p3.fr/~roche/nectari/dist/" href="launch.jnlp">
    <information>
        <title>nectar</title>
        <vendor>Herman</vendor>
        <description>nectar</description>
        <description kind="short">nectar</description>
        <homepage href=""/>
        <icon href="shot0000.jpg" kind="default"/>
        <offline-allowed/>
    </information>
    <security>
        <all-permissions/>
    </security>
    <resources>
        <j2se version="1.5+" java-vm-args="-Djava.security.policy=applet.policy"/>
        <jar href="nectar.jar" main="true" download="eager"/>
        <jar href="lib/aida-3.3.jar" download="eager"/>
...
        <jar href="lib/xercesImpl-2.6.2.jar" download="eager"/>
    </resources>
    <application-desc main-class="nectar.Main">
    </application-desc>
</jnlp>
```

2.4.2 Configuration

Adding the SUN JDK

- `Tools >>> Java Platforms >>> Add Platform...`
- Open to the */usr/java/jdk1.6.0_19/default* directory.
- Click the `Next >` button that was just enabled.
- Make the `Platform Sources` point to */usr/lib/jvm/java-6-sun-1.6.0.12/bin/*
- Click the `Finish` button.

Adding JDK's Javadoc

- test : clic on a word of the code and then do `ctrl-shift espace`. You should read : `Javadoc not found`
- Download JDK documentation from here
- In NetBeans IDE :
  1. Choose Tools > Java Platform Manager from the main window.
  2. Select the platform to which you want to add Javadoc in the left panel of the dialog box.
  3. In the Javadoc tab, click Add ZIP/Folder and specify the location of the Javadoc files.
  4. Click Close.

note: the javadoc of JAIDA is available as a `.jar` file in the */cvs/nectar/lib* directory.

Getting the sources and libraries from CVS

- Check out: In NetBeans IDE, choose `Team >>> CVS >>> Checkout` from the main menu. The Checkout wizard opens.
- enter this CVS Root: *:pserver:roche@lpnp90.in2p3.fr:2401/home/cvsroot*
- ask for the *nectar* module.
- uncompress the */NetBeansProjects/nectar/lib/jdom-1.1.1.tar.gz* lib files for instance.

Make the SUN JDK the default one

- Right-click the project's root node in the Projects window and choose **Properties**.
- In the Project Properties dialog box, select the **Libraries** node in the left pane.
- Choose the desired Java platform in the Java Platform combo box.

Configuring libraries

1. From the ant subdirectory of the just-unpacked library, run `$ ant javadoc` to generate the Javadocs locally. You can skip this step, but then you'll be missing the API documentation.
2. In NetBeans, select the Library Manager item from the Tools menu
3. Click on the *New Library...* button and enter the library name NETBEANS prompt for.
4. With the Classpath tab selected, click on the *Add JAR/Folder...* button and select the *.jar* files from the *build* and the *lib* directories.
5. With the Sources tab selected, click on the *Add JAR/Folder...* button and select the source directory for this library.
6. With the Javadoc tab selected, click on the *Add ZIP/Folder...* button and select the javadoc directory for this library.

At this point, you have complete the configuration of the libraries and you should be able to compile.

- compile: Maj+F11
- run: F6

Configuring the running JVM with more

- Right-click the project's root node in the Projects window and choose **Properties**.
- In the Project Properties dialog box, select the **Run** node in the left pane.
- Insert the following options for initial and maximum memory size of the JVM into the **VM Option** box :

```
-Xms512m -Xmx1024m
```

2.4.3 Best practices

Unitary tests

Each *.java* file contain a *static void main* function and is executable alone. You have to click on the file and then press `Maj+F6`.

Without NETBEANS you should do:

```
$ ls src/nectar/*.java
$ java -cp dist/nectar.jar nectar.ConfXml
```

Packaging

```
$ cd nectar
$ ant dist                      # compile software
$ java -jar dist/nectar.jar     # test software
$ tar -zcf nectar.tgz dist/     # archive software
```

Comment: so as to add the XML configuration file you will have to add the folowing target to the *build.xml* file:

```
<target name="-post-jar">
    <copy file="nectar.xml" flatten="true" todir="${dist.dir}"/>
</target>
```

2.5 Jar

2.5.1 Introduction

The JAR files (Java ARchive) allow to archive files in a JAVA specific format which is indeed portable. The JAR files are in fact just some ZIP files renamed (we can `unzip` them). The advantage of JAR archives is that tey can be run like `.exe` files under WINDOWS.

2.5.2 Build from scratch

- File *MANIFEST.MF*:

```
Main-Class: ClassePrincipale
Class-Path: \Archives\liquidlnf.jar // Le fichier JAR contenant le Look \& Feel Liquid
```

- Command:

```
$ jar cvmf MANIFEST.MF Programme.jar *.class
$ unzip Programme.jar
```

2.5.3 Netbeans

Including libraries

If we enable the WEBSTART option, the `.jar` libraries files will load even if they are not included into the project JAR file.
To do it, you must right click on the project node and choose “proporties” and then “web start”.

Including ressources

In order to include the `etc/` directory (and the `nectar.xml` configaration file) into the sources.
To do it, we have to right click on the project node and choose “proporties” and then “sources” an add the `etc/` directory as a “source package folder”. **Note** that the `nectar.xml` is not prefixed by the project name nor by `etc/`.

```
try {
    // use the user configuration file (local copy)
    doc = builder.build(new FileInputStream("etc/" + Main.confFile));
    UI_Log.logger.info("LOCAL configuration file loaded");

} catch (Exception f) {
    UI_Log.logger.info("Cannot load local configuration file... try to look into the JAR");

    // use the default configuration file (into the jar)
    ClassLoader cl = Thread.currentThread().getContextClassLoader();
    InputStream jarFile = cl.getResourceAsStream("nectar.xml");
    if (jarFile == null) {
        UI_Log.logger.severe("Cannot load default configuration from JAR file");
        return;
    }
    else {
        UI_Log.logger.info("DEFAULT configuration file loaded");
        doc = builder.build(jarFile);
    }
}
```

2.6 Applet

rq: We prefer to use WEB START. Java Web Start provides a platform-independent, secure, and robust deployment technology. It enables developers to deploy full-featured applications to end-users by making the applications available on a standard web server. With any web browser, end-users can launch the applications and be confident they always have the most-recent version :

- It use the native JVE instead of the JVE of the browser.
- It is easy to use with NETBEANS and trully simplify the librairies distribution and signature by certificate.
- It seems to do what we really wanted to do using the APPLET.
- It was advised to us by the JAVA computers scientist of the LPSC laboratory.

2.6.1 Introduction

An applet is a program written in the Java programming language that can be included in an HTML page, much in the same way an image is included in a page. You will find here some notes about the Netbeans tutorial.

2.6.2 Is-it really working ?

... sometime yes. Into NETBEANS do:

1. Open a new project called "JavaApplication1"
2. Create a new **JAppletForm** file (from the **Swing GUI Forms** cathégory and call it *NewJApplet.java*
3. Add a color Chooser component to the pannel so as to have this code:

```
public class NewJApplet extends javax.swing.JApplet {

    /** Initializes the applet NewJApplet */
    public void init() {
        try {
            java.awt.EventQueue.invokeAndWait(new Runnable() {
                public void run() {
                    initComponents();
                }
            });
        } catch (Exception ex) {
            ex.printStackTrace();
        }
    }

    private void initComponents() {
        jColorChooser1 = new javax.swing.JColorChooser();

        javax.swing.GroupLayout layout = new javax.swing.GroupLayout(getContentPane());
        getContentPane().setLayout(layout);
        layout.setHorizontalGroup(
            layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
            .addGroup(javax.swing.GroupLayout.Alignment.TRAILING, layout.createSequentialGroup()
                .addContainerGap(javax.swing.GroupLayout.DEFAULT_SIZE, Short.MAX_VALUE)
                .addComponent(jColorChooser1, javax.swing.GroupLayout.PREFERRED_SIZE,
                  javax.swing.GroupLayout.DEFAULT_SIZE, javax.swing.GroupLayout.PREFERRED_SIZE)
                .addContainerGap())
        );
        layout.setVerticalGroup(
            layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
            .addGroup(javax.swing.GroupLayout.Alignment.TRAILING, layout.createSequentialGroup()
                .addContainerGap(javax.swing.GroupLayout.DEFAULT_SIZE, Short.MAX_VALUE)
                .addComponent(jColorChooser1, javax.swing.GroupLayout.PREFERRED_SIZE,
                  javax.swing.GroupLayout.DEFAULT_SIZE, javax.swing.GroupLayout.PREFERRED_SIZE))
        );
    }
    private javax.swing.JColorChooser jColorChooser1;
}
```

4. Test this selected file with `Maj-F6`
5. Build the *dist/JavaApplication1.jar* with `F11`

Outside of NETBEANS:

1. Copy the JAR archive int */htdocs*

2. Create the *applet1.html* file:

```
<applet
 code=NewJApplet.class
 archive="JavaApplication1.jar">
 width=350 height=200
</applet>
```

3. Now you should be able to load at this.

2.6.3 Signing the JAR file

Please try this second applet (doing like above) so as to test:

- That your browser ask you for accepting our certificat.
- That you cannot write to a file if you refuse it
- That you can write to a file by accepting it

1. testing IO

```
import java.io.*;

public class NewJApplet extends javax.swing.JApplet {

    /** Initializes the applet NewJApplet */
    public void init() {
        try {
            java.awt.EventQueue.invokeAndWait(new Runnable() {
                public void run() {
                    initComponents();
                }
            });
        } catch (Exception ex) {
            ex.printStackTrace();
        }
    }

    private void initComponents() {

        jToggleButton1 = new javax.swing.JToggleButton();
        jScrollPane1 = new javax.swing.JScrollPane();
        jTextArea1 = new javax.swing.JTextArea();

        jToggleButton1.setText("test");
        jToggleButton1.addActionListener(new java.awt.event.ActionListener() {
            public void actionPerformed(java.awt.event.ActionEvent evt) {
                jToggleButton1ActionPerformed(evt);
            }
        });

        jTextArea1.setColumns(20);
        jTextArea1.setRows(5);
        jScrollPane1.setViewportView(jTextArea1);

        javax.swing.GroupLayout layout = new javax.swing.GroupLayout(getContentPane());
        getContentPane().setLayout(layout);
        layout.setHorizontalGroup(
```

```
            layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
            .addGroup(layout.createSequentialGroup()
                .addContainerGap()
                .addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEA
                    .addComponent(jToggleButton1)
                    .addComponent(jScrollPane1, javax.swing.GroupLayout.PREFERRED_SIZE, 32
                      javax.swing.GroupLayout.PREFERRED_SIZE))
                .addContainerGap(206, Short.MAX_VALUE))
        );
        layout.setVerticalGroup(
            layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
            .addGroup(layout.createSequentialGroup()
                .addContainerGap()
                .addComponent(jToggleButton1)
                .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)
                .addComponent(jScrollPane1, javax.swing.GroupLayout.PREFERRED_SIZE, 155,
                  javax.swing.GroupLayout.PREFERRED_SIZE)
                .addContainerGap(151, Short.MAX_VALUE))
        );
    }

    private void jToggleButton1ActionPerformed(java.awt.event.ActionEvent evt) {
          try {
              jTextArea1.append("testing...");
              BufferedWriter fichier = new BufferedWriter(new FileWriter("/tmp/tata.log")
              fichier.write("bonjour tout le monde");
              fichier.newLine();
              fichier.close();
     } catch (Exception e){
System.out.println(e.toString());
        jTextArea1.append(e.toString());
     }
     jTextArea1.append("done");
    }

    private javax.swing.JScrollPane jScrollPane1;
    private javax.swing.JTextArea jTextArea1;
    private javax.swing.JToggleButton jToggleButton1;
}
```

2. `keytool` is delivered with Sun's development kit.

   ```
   $ keytool -genkey -keyalg rsa -alias yourkey
   ```

   Follow the instructions and type in all needed information.

3. Now we make the certificate:

   ```
   $ keytool -export -alias yourkey -file yourcert.crt
   ```

4. Now we have to sign the applet:

   ```
   $ jarsigner yourapplet.jar yourkey
   ```

Part III

Annexes

- Les milestones ("jalons") avec les deadlines

Lecture de la carte via Ethernet	avril 2010
Affichage des graphiques 2D pour les électroniciens	avril 2010
Soft Java exécutable depuis un navigateur Web	avril 2010
Soft Java entièrement exécutable et compilable depuis n'importe quelle architecture	avril 2010

- Liste de vos taches à faire (avancement)

Apprentissage et développement du GUI Java	en cours
Documentation & formation	en cours
Code multiplatforme	à faire
Installation d'un second serveur multiboot	à faire
Soft Java exécutable depuis un navigateur Web	à valider

- Liste de vos taches faîtes

Lecture de la carte via Ethernet
Enquête sur les environnement de développement Java
Appropriation des outils de compilation des projets Java
Test des processus d'exécution des archives Java
Installation d'un premier serveur multiboot
Affichage des graphiques 2D pour les électroniciens

- Difficultés éventuelles rencontrées

Temps perdu à trouver un PC de développement
Absence de cahier des charges et d'argumentation technique
Défaut de complicité entre l'informatique et l'électronique : vision linéaire du projet
Manque de motivation dans l'apprentissage d'un nouvel environnement de développement
Toujours pas intégré au sein de l'équipe Hess (aucune vision d'ensemble)

- Les formations

vaccins hépatite A et typhoïde	fait le 7/4/9

- Vos prochaines réunions/missions du projet

Le mardi à 10H30 toutes les semaines.

Bibliography

[1] Interface nectar ethernet v1. LPNHE-Paris.

[2] Interface nectar ethernet v2. LPNHE-Paris.

[3] *Programmer en Java*. Eyrolles, 2009. 5e edition Java 5 et 6.

[4] *GEDEK Documentation*. Advanced Logic Synthesis for Electronics, 2009.