PRI Name	eWON Java interface user guide		
Access	OEM, DIST		
Since revision	5.2		
PRI Number	PRI-0006-0	**Build**	70
Mod date	17. Aug 2007		

Preliminary Reference Information

eWON Java interface user guide

JAVA ETK: 1.3

Summary:

Explains how to use the con.ewon.ewonitf eWON Java library, that provides specific mechanisms for managing the eWON in JAVA

Table of content

Intoduction: JAVA & eWON interactions

The eWON JAVA toolkit designed around the J2ME (JAVA Micro Edition) technology is described in the “eWON Java Toolkit User Guide”.

The J2ME profile provides a language and a framework for developing your applications. You will find for example:

- The Java language implementation including:
  - multithreading
  - synchonization (locking)
- java.lang: classes for basic types, floats, runtime, ... manipulations
- java.io: stream operations
- java.util: Calendar, Hashtable, Stack, Vector,...
- java.microedition.io: connection management interface (socket, files, ...)
- java.microedition.file: Flash file system interface (this is JSR-075)

The only thing the eWON programmer will miss is an interface to the eWON specific features like:

- IO Server tags
- Custom web site
- Scheduled actions management
- Event log
- ...

Therefore a specific eWON package called **com.ewon.ewonitf** is provided with a number of classes providing access to these eWON features.

This document is divided in a number of chapters, with each chapter describing an eWON feature and its JAVA interface with a small application example.

IMPORTANT: This document will introduce examples provided in the Test project.

This document is not supposed to be a complete reference guide.

You will need to check the javadoc (index.html, provided in the javaetk\docs directory) and the Basic user manual that provides additional information about the eWON features.

Know which JavaETK version you are running

In your Java application

The current version of the JavaETK in the eWON is readable via the **System.getProperty** function.

This is a standard System class function and a special **ewonitf.version** field has been added to report the current JavaEtk version.

```
System.out.println("ewonitf Version: "+System.getProperty("ewonitf.version"));
```

The JavaEtk version is not updated each time the eWON firmware is updated. The version is increased to keep track of Java toolkit bug fixes or when new functions are added to the toolkit.

If an application written for a newer toolkit is ran on an older firmware it will work as long as the new functions are not called. If an non existing function is called, it will produce an exception.

In the eWON Web interface

From the eWON web interface, it is also possible to know which toolkit version is present via the status file (Files Transfer → estat.htm → JavaVersion)

In the Java documentation

For newer functions and classes, they are marked in the documentation as “since version X.Y”

SysControlBlock: Reading and writing configuration

Class: com.ewon.ewonitf.SysControlBlock

This class provides function equivalent to the Basic:

setsys XXX,”load”, getsys ..., setsys ... and setsys XXX,”save”

Test class

The Test class for the SysControlBlock class is:

com.ewon.ewontest.TestSysControlBlock

Example

The following example show how to read and set the “identification” field of the **config.txt** block.

```
public static void readWriteSysBlock()
{
SysControlBlock SCB;

   try
   {
      SCB = new SysControlBlock(SysControlBlock.SYS);
      System.out.println("Value: "+SCB.getItem("identification"));
      SCB.setItem("identification","My eWON");
      SCB.saveBlock();
   }
   catch (Exception e)
   {
       System.err.println("Exception:"+e.getMessage());
   }
```

Remark: in this example, the modification is not saved to flash.

IOManager: Read/Write tags IO. Manage led and button

Class: com.ewon.ewonitf.IOManager

This class is used to access the eWON tags, the user led and front button.

Test class

The Test class for the IOManager class is:

com.ewon.ewontest.TestIOManager

Example:

This example is an excerpt from the TestIOManager, it show how to read or write a tag value:

```
public static void readWriteTagTest()
{
  float TagValue;

    try
    {
      TagValue = IOManager.readTag("MyTag");
      System.out.println("Tag Value"+ Float.toString(TagValue));

      IOManager.writeTag("MyTag", (float)123.0);
    }
    catch (Exception e)
    {
    }
    finally
    {
        IOManager.userLedJvmCtrl(false);
    }
}
```

The example here is also an excerpt from the TestIOManager, it shows how to let java take control of the user led and how to report the eWON from button status with the user led.

```
try
{
  //****** Test3: switch led with button 10 times
  //-- Let JAVA take control of the user led
  IOManager.userLedJvmCtrl(true);

  OldButton = IOManager.readButton();
  System.out.println("Button is" + ((OldButton==true)?"Pushed":"Released") );

  int i=0;
  while (i<10)
  {
     Button =  IOManager.readButton();
     if (Button != OldButton)
     {
         i++;
         if (Button)
           IOManager.setUserLed(IOManager.COLOR_RED);
         else
           IOManager.setUserLed(IOManager.COLOR_GREEN);
         OldButton=Button;
         System.out.println("Button is"+ ((OldButton)?"Pushed":"Released") );
     }
  }
}
catch (Exception e)
{
  System.out.println("Error occured: "+e.getMessage());
}
finally
{
    IOManager.userLedJvmCtrl(false);
}
```

TagControl: Map eWON Tag to monitor and manage tags

Class: com.ewon.ewonitf.TagControl

This class is used to map a tag and provides utility function to set or read tag value and alarm status.

An instance of this object is created by specifying one of the

- tag name
- tag id
- tag ndx
- another tag control
- Nothing, but then you can adjust the tag relation later or even update it.

Test class

The Test class for the TagControl class is:

com.ewon.ewontest.TestTagControl

Example:

This example is an excerpt from the TestTagControl.

This function gives a few examples of the methods available for reading/writing tag values, log the tag in the historical files (irc files), acknowledge the alarms.

```
public class TestTagControl {

    public static void readWriteTest()
    {
       try
       {
          TagControl TC = new TagControl("MyTag");
          int TagId = TC.getTagId();
          double D = TC.getTagValueAsDouble();
          System.out.println("Value D: "+ Double.toString(D));
          System.out.println("Name: "+ TC.getTagName());
          TC.setTagValueAsInt(0);
          TC.logTag();
          TC.setTagValueAsInt(10);
          TC.logTag();
          System.out.println("Value Alarm status: "+ Integer.toString(TC.getAlarmStatus()));
          System.out.println("Value Alarm type: "+ Integer.toString(TC.getAlarmType()));
          TC.ackAlarm("adm");
       }
       catch (Exception e)
       {
           System.err.println("Exception:"+e.getMessage());
       }
    }
```

PRI Name	eWON Java interface user guide		
Access	OEM, DIST		
Since revision	5.2		
PRI Number	PRI-0006-0	Build	70
Mod date	17. Aug 2007		

DefaultEventHandler: Manage eWON events form Java

Class: com.ewon.ewonitf.DefaultEventHandler

Events can be managed by polling status at regular interval but as usual there are 2 drawbacks to this method:

- The system spends time to check value
- When the delay between checks is increased, the load is reduced (good) but the latency is increased (bad).

In order to provide a more fluent method, the javaetk provides an event management mechanism.

The default event handler has a 'run' function that will run forever, this function will call events 'listener' you have defined when the specific events occur.

There are different events listeners for deferent event types.

The generic mechanism is described by this small example:

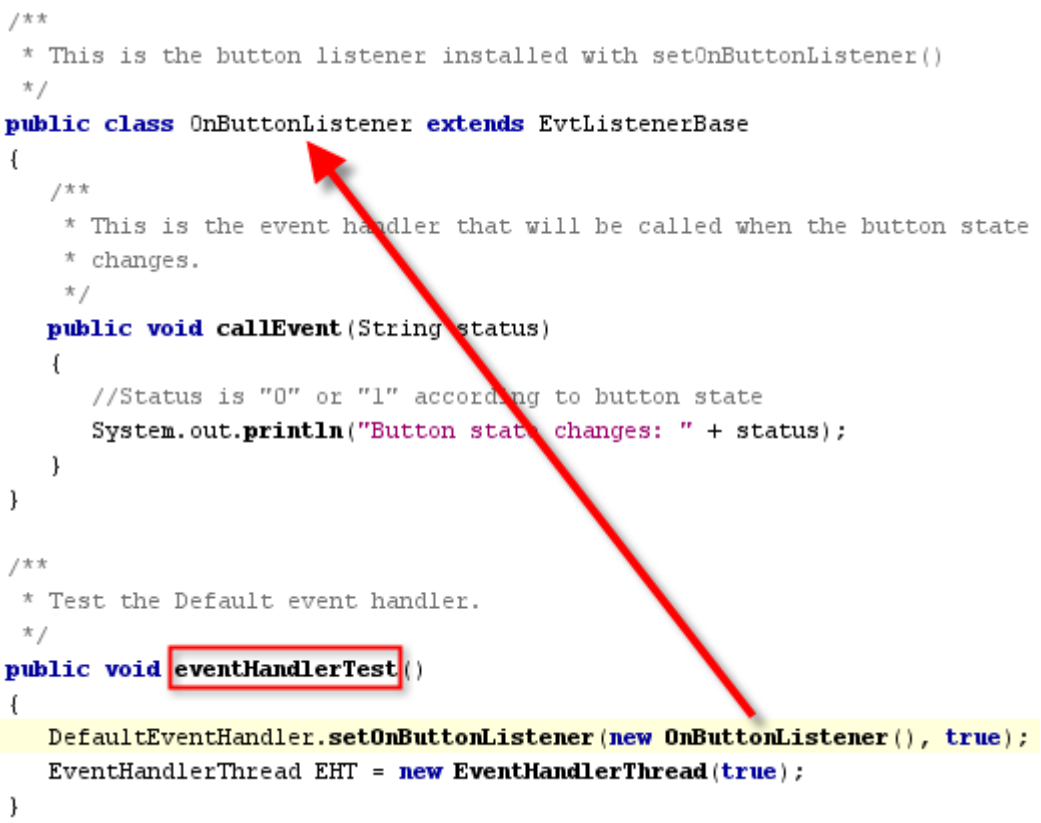
```
/**
 * This is the button listener installed with setOnButtonListener()
 */
public class OnButtonListener extends EvtListenerBase
{
   /**
    * This is the event handler that will be called when the button state
    * changes.
    */
   public void callEvent(String status)
   {
      //Status is "0" or "1" according to button state
      System.out.println("Button state changes: " + status);
   }
}

/**
 * Test the Default event handler.
 */
public void eventHandlerTest()
{
   DefaultEventHandler.setOnButtonListener(new OnButtonListener(), true);
   try
   {
      DefaultEventHandler.runEventManager();
   }
   catch (Exception ex)
   {
      System.err.println("DefaultEventHandler "+ex.getMessage());
   }
}
```

In this example, the eventHandlerTest function installs the Button change handler and then runs the EventManager.

As you can see a “listener” is an instance of a “listener” class, this “listener” class derive from the EvtListenerBase object and defines the **callEvent** function, this function will be called when the runEventHandler will detect the specific type of event.

The **runEventManager** function will never return, and will block the JVM, so its a good idea to call it in its own thread.

Then following example shows the same application with the event manager handler called in a thread using the EventManagerThread object.



```
/**
 * This is the button listener installed with setOnButtonListener()
 */
public class OnButtonListener extends EvtListenerBase
{
   /**
    * This is the event handler that will be called when the button state
    * changes.
    */
   public void callEvent(String status)
   {
      //Status is "0" or "1" according to button state
      System.out.println("Button state changes: " + status);
   }
}

/**
 * Test the Default event handler.
 */
public void eventHandlerTest()
{
   DefaultEventHandler.setOnButtonListener(new OnButtonListener(), true);
   EventHandlerThread EHT = new EventHandlerThread(true);
}
```

Create Web forms

JAVA web forms are output when the following URL is called:

http://ewon_ip/rcgi.bin/jvmForm?formName=NNNNN&timeout=TTTTT

Where NNNNN is the form name

And TTTTT is the maximum amount of time the web server will wait for the JAVA to produce the output.

The kind of from output can be produced by 2 different methods:

- Produce the output directly on a web stream directed to the web browser.
- Call the **produceAstPage** function to produce a template output that uses SSI tags.

In any case the web form is installed using:

addWebFormListener(String formName, EvtWebFormListener webFormListener)

or

setDefaultWebFormtListener(EvtWebFormListener webFormListener)

(see javadoc)

Produce output directly using a stream

This is an example of handler that outputs the form result on a stream and show a number of interesting functions.

```
public class MyFormWebFormListener extends EvtWebFormListener
{
   public void callForm(String formName)
   {
      System.out.println("Form: "+formName+" is called.");

      //log the formName obtained from web var in the RT log.
      System.out.println(getWebVar("formName","???"));
      //set the TestVar web var.
      setWebVar("TestVar","abcd");

      //configure output as text/plain mime type (default is html)
      setWebHeader("text/plain");

      //create a stream to output the form content.
      PrintStream ps = new PrintStream(this);
      ps.print("Hello web browser, here I am: "
               +getWebVar("formName","???")
               +getWebVar("TestVar","???"));
   }
}
```

getWebVar: this function returns the value of a form parameters passed with POST or through the URL params.

SetWebVar: this function can change or set the value of a web parameters

setWebHeader: use this function if you want to specify a different mime type for the form output.

PrintStream: this is an object for formating output (standard), you can pass the EvtWebFormListener object as parameter, as it is an output stream whose output data will be sent to the client web browser.

Produce the output using a template

This is an example of handler that outputs the form result based on a **astA.shtm** template.

The following form listener has been created:

```
public class MyForm2WebFormListener extends EvtWebFormListener
{
   public void callForm(String formName)
   {
      produceAstPage("/usr/astA.shtm");

   }
}
```

And installed with:

```
DefaultEventHandler.addWebFormListener("MyForm2",
                                       new MyForm2WebFormListener());
```

If this is the **/usr/astA.shtm** template content:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
<html>
  <head>
    <title>Test AST 1</title>
  </head>
  <body>
   Hello, this is the body starting<br>
   <%#jvmAst,astData%>
   <br>...And here is the end.
  </body>
</html>
```

And the following AST (active server template) listener has been created:

```
public class WebAstDataListener extends EvtWebAstListener
{
   public void callAst(String astName)
   {
      try
      {
         String Output = " Ast Data called " +
                         getWebVar("formName","???");
         write(Output.getBytes());
      }
      catch (IOException ex)
      {}
   }
}
```

and installed with:

```
DefaultEventHandler.addWebAstListener("astData", new WebAstDataListener());
```

Then, when

http://ewon_ip/rcgi.bin/jvmForm?formName=MyForm2

is called, it will result in the following output:

Hello, this is the body starting
Ast Data called MyForm2
...And here is the end.

The <%#jvmAst,astData%> tag in the template page produced with **produceAstPage** has generated a call to the **WebAstDataListener** which has produced the “Ast Data called” output in place of the tag.

Manage Tag events (value change and alarm)

There are 4 events handler for managing tag events:

- Specific tag, value changed
- Default tag, value changed
- Specific tag, Alarm

- Default tag, Alarm

You need to extend a class from the corresponding event listener and then pass an instance of your class to the event listener configuration function.

Tag value change listener

This is an example of tag value change listener

```
public class MyTagValueListener extends EvtTagValueListener
{
   public void callTagChanged()
   {
      try
      {
         System.out.println("Tag changed: "+ getTagName() +
                            " Change Val:" + Double.toString(getChangedValueAsDouble()) +
                            "CurrentValue: " + Double.toString(getTagValueAsDouble()));
      }
      catch (Exception ex)
      {

      }
   }
}
```

Your callTagChanged function will be called when a tag value has changed.

You should take note that the **EvtTagValueListener** is itself extending the **TagControl** class, when the listener is called, the instance is preset to the tag that changed and you can use the listener itself to monitor or change the tag.

The current value of the Tag may already have changed by the time your event listener is called, but the value that triggered the change event is saved and the EvtTagValueListener has specific functions to retrieve that value (and even the value type, as it can also change).

To define the listener for all tags:

```
MyTagValueListener evtTagValueListener = new MyTagValueListener();

DefaultEventHandler.setDefaultTagValueListener(evtTagValueListener);
```

The evtTagValueListener.callTagChanged will be called for any tag that changes. When called, the getTagName from the TagControl class can be used to detect which tag triggered the event.

To define the listener for a specific tag:

```
MyTagValueListener evtTagValueListener = new MyTagValueListener();

evtTagValueListener.setTagName("OtherTag");
DefaultEventHandler.addTagValueListener(evtTagValueListener);
```

In this example the listener is only defined for the tag which has the name “OtherTag”

Tag Alarm Listener

This is an example of tag alarm listener

```
public class MyTagAlarmListener extends EvtTagAlarmListener
{
   public void callTagChanged()
   {
      try
      {
         System.out.println("Alarm changed: "+ getTagName() +
                            " Alarm Status:" +Integer.toString(getAlarmStatus()) +
                            " Alarm Type:" +Integer.toString(getAlarmType()) +
                            "CurrentValue: " + Double.toString(getTagValueAsDouble()));
      }
      catch (Exception ex)
      {

      }
   }
}
```

The same principle as for the Tag value change event listener applies. It also extends the TagControl class and the Alarm type and level at the time the event is triggered are also saved and available as described in this example via specific methods.

Event listener are defined as for Tag value change, please refere to that example.

Receive SMS using

The event handler can have a listener for checking reception of SMS.

It is installed using the setOnSmsListener.

```
DefaultEventHandler.setOnSmsListener(new OnSmsListener(), true);
```

Reception of the SMS itself is described in the ModemManager class and the complete example is also available in TestDefaultEventHandler.

See also javadoc.

Check eWON Button events

The event handler can have a listener for checking any change to the eWON from button state.

It is installed using the setOnButtonListener

```
DefaultEventHandler.setOnButtonListener(new OnButtonListener(), true);
```

The complete example is also available in TestDefaultEventHandler.

See also javadoc.

Check PPP, VPN, WAN connect/disconnect events

The event handler can have a listener for checking any change in the PPP, WAN or VPN connection state. A specific event listener is available for any of these events.

It is installed using the setOnPppListener, setOnWanListener or setOnWanListener

```
DefaultEventHandler.setOnPppListener(new OnNetListener("PPP"), true);
DefaultEventHandler.setOnVpnListener(new OnNetListener("VPN"), true);
DefaultEventHandler.setOnWanListener(new OnNetListener("WAN"), true);
```

The complete example is also available in TestDefaultEventHandler.

See also javadoc.

Test class

The Test class for the TestDefaultEventHandler class is:

com.ewon.ewontest.TestDefaultEventHandler

There are more examples presented in this class.

Loader: Load data in the eWON (configuration for ex.)

Class: com.ewon.ewonitf.Loader

The loader class can be used to transfer any stream of data in one of the eWON ftp root file (as far as this file is writable of course).

Example: transfer a /usr/MyFile1.txt to /config.txt

Test class

The Test class for the Loader class is:

com.ewon.ewontest.TestLoaderExporter

Example

For example: if you have a file called **/usr/MyTest0.txt**

containing:

```
:System
Identification:My eWON
Information:Some info about this eWON
```

And you want to apply this data to eWON configuration.

Then you can call the following code to change the **config.txt** accordingly.

```
try
{
   Loader loader = new Loader("/config.txt");
   loader.LoadFrom("file:////usr/MyTest0.txt");
   loader.close();
}
catch (IOException ioe)
{
   System.err.println("Error loaderLoadFromTest: "+
                      ioe.toString());
}
```

As you can see the loader object defines which eWON file will be loader. File writable through FTP can be defined here (except ewonfwr.edf)

The Loader instance is an OutputStream where data can be written or the **LoadFrom** method can be called to define the input stream source.

Exporter: Transfer an Export Block Descriptor to an output stream

Class: com.ewon.ewonitf.Exporter

The exporter class can be used to transfer the content of an export block descriptor to an Input stream. The stream can also be directly transferred to a user file.

Example: transfer a $dtEV content to /usr/event_info.txt

NOTE Exporter is a stream, so the content **must not** be written to a /usr file but can be processed in line.

Test class

The Test class for the Exporter class is:

com.ewon.ewontest.TestLoaderExporter

Example

For example: if you want to transfer the content of a $dtEV export block descriptor to the /usr/Export0.txt file, you could use the following code:

```
try
{
   Exporter exporter = new Exporter("$dtEV");
   exporter.ExportTo("file:////usr/Export0.txt");
   exporter.close();
}
catch (IOException ioe)
{
   System.err.println("Error exporterExportFromTest: "+
                      ioe.toString());
}
```

The exporter object is an output stream pointing to an export block descriptor, in this example the stream content is copied until end of file to the /usr/Export0.txt file.

ModemManager: Read SMS, reset modem, ...

Class: com.ewon.ewonitf.ModemManager

The ModemManager class provides a number of functions related to the modem like:

- Get the number of SMS in the queue
- Read an SMS from the queue
- Reset the modem
- Check the modem reset status.

Test class

The Test class for the ModemManager class is:

com.ewon.ewontest.TestModemManager

Example

The TestModemManager class provides examples detailed below on how to reset the modem and read an SMS.

Read an SMS

The following example reads and logs the number of SMS pending. Then it tries to read an SMS, an exception is thrown if the is no SMS pending.

```
int count = ModemManager.getSmsCount();
System.out.println("Sms: "+count);
try
{
   SmsMessage smsMessage = ModemManager.readSms();
   System.out.println("SmsNdx: "+smsMessage.getSmsNdx());
   System.out.println("SmsDateTime"+
                      new Date(smsMessage.getSmsDataTime()).toString());
   System.out.println("SmsDateTime"+
                      Long.toString(smsMessage.getSmsDataTime()));
   System.out.println("SmsFrom: "+smsMessage.getSmsFrom());
   System.out.println("SmsMessage: "+smsMessage.getSmsMessage());
}
catch (Exception e)
{
   System.out.println("Error receiveSmsTst: "+e.toString());
}
```

The readSMS function returns an SmsMessage object which is used to get the Sms properties.

Reset the Modem

The following example resets the modem and waits until the reset operation is completely done.

```
try
{
   ModemManager.resetModem();
   while (ModemManager.getModemInitStatus() !=
           ModemManager.INIT_FINISHED)
   {
      Thread.sleep(500);
   }
   System.out.println("Reset modem done");
}
catch (Exception e)
{
   System.out.println("Error resetModemTest: "+e.toString());
}
```

ScheduledActionManager: Sendmail, PutFTP etc...

Class: com.ewon.ewonitf.ScheduledActionManager

This class is used to generate “scheduled actions” like SendMail, PutFTP, NTPSync etc...

The class provides also a number of method for reading and managing the list of actions like:

- Get the number of actions in log
- Read one action in log by its ID
- Read one action in log by its index
- Read the whole log.
- Clear pending actions

Test class

The Test class for the ScheduledActionManager class is:

com.ewon.ewontest.Test ScheduledActionManager

Example:

The examples here cover 2 cases:

- Create a scheduled action
- Read the scheduled action log

Send an Email

The following example show how to send an Email. The function will post the Email in the Scheduled action manager queue **AND it will wait until the operation is finished**.

This is quite different from Basic where the function returned immediately. The rest of the JVM will continue to run, but the current thread will be suspended until the operation ends (success or error). The function returns the operation's result.

```
try
{
   int res =
   ScheduledActionManager.SendMail("joe@users.com",     //from
                                   "smith@johns.com",   //to
                                   "",                  //cc
                                   "TestMail",          //subject
                                   "Content");          //content
   System.out.println("Send mail result: "+new Integer(res).toString());
}
catch (EWException e)
{
   System.out.println("Error sendMailTest: "+e.toString());
}
```

Read and clear scheduled action log

This example show how to read the “scheduled action log” at once with **readActionLog**, display its content by using the **ScheduledAction** object and then remove pending actions from the log by using the **clearPendingActions** function.

```
ScheduledAction [] scheduledActionList =
                                   ScheduledActionManager.readActionLog();
int actionLogSize = scheduledActionList.length;
int i=0;
while ((actionLogSize--)>0)
{
   ScheduledAction scheduledAction = scheduledActionList[i++];
   System.out.println("Id: "+scheduledAction.getId()+" --------------------");
   System.out.println("Type: "+scheduledAction.getType());
   System.out.println("TypeStr: "+scheduledAction.getTypeAsString());
   System.out.println("Status: "+scheduledAction.getStatus());

   System.out.println("Start"+
                      new Date(scheduledAction.getStartDateTime()).toString());
   System.out.println("End"+
                      new Date(scheduledAction.getEndDateTime()).toString());
}
//Clear the log
ScheduledActionManager.clearPendingActions();
```

RuntimeControl: Java watchdog, reseet and reboot

Class: com.ewon.ewonitf.RuntimeControl

This class provides the following features:

- Watchdog management for JVM: You setup the watchdog and then have to call a “refresh” function before the timeout expires or the eWON is rebooted.
- Reset the eWON: smooth with Shutdown or rough with Reboot.
- Restart the JVM: define a new JVM command line and restart it from the Java program.
- Unlock the used JAR file: when the JVM uses the JAR, you can normally not replace it, this function provides a workaround.

Test class

The Test class for the RuntimeControl class is:

com.ewon.ewontest.TestRuntimeControl

Example

All the RuntimeControl function are described in the TestClass and in the javadoc, please refer to this material.

EventManager: log events in events.txt and in “real time log”.

Class: com.ewon.ewonitf.EventManager

This class provides the following features:

- Log an event in events.txt
- Log an event in the “real time log”

Test class

The Test class for the EventManager class is:

com.ewon.ewontest.TestEventAndSystem

Example

All the EventManager function are described in the Test class and in the javadoc, please refer to this material.

EwonSystem: Get “system up time” and set the clock

Class: com.ewon.ewonitf.EwonSystem

Functions related to the eWON and not entering any other category.

This class provides the following features:

- Get the time in second since the eWON booted.
- Change the eWON real time clock: set the time.

Test class

The Test class for the EwonSystem class is:

com.ewon.ewontest.TestEventAndSystem

Example

All the EwonSystem function are described in the Test class and in the javadoc, please refer to this material.

StorageControl: Erase system data, config and files, or save config

Class: com.ewon.ewonitf.StorageControl

This class provides the following features:

- Save any change to the configuration (com or config)
- Reset COM config to default
- Erase historical recordings or alarms or events files
- Format partitions

- Clear pending actions in the scheduled action manager queue.

Test class

The Test class for the StorageControl class is:

com.ewon.ewontest.TestStorageControl

Example

All the StorageControl function are described in the Test class and in the javadoc, please refer to this material.

NetManager: Close PPP connection, get interface information

Class: com.ewon.ewonitf.NetManager

This class provides the following features:

- Close the PPP connection
- Manager transparent forwarding
- Get PPP, WAN, VPN IP address
-

Test class

The Test class for the NetManager class is:

com.ewon.ewontest. TestNetManager

Example

All the NetManager function are described in the Test class and in the javadoc, please refer to this material.

The example bellow show a few calls to the class methods.

```
try
{
   NetManager.closePppConnection();
   System.out.println("PPPIP: "+NetManager.getPppIp());
   NetManager.setTransparentForwardingDest("10.0.0.1");
}
catch (Exception e)
{
   System.out.println("Error pppAndTfTest: "+e.toString());
}
```

CommConnection: Working with serial port.

Class: javax.microedition.io.CommConnection

This is a standard java J2ME class and the mechanism to access the serial is standard to J2ME also.

Opening the port looks like this:

```
CommConnection cc =
(CommConnection)Connector.open("comm:com0;baudrate=57600;bitsperchar=8;stopbits=1;pari
ty=none;autorts=on");
```

You should check the comm: syntax in the javax.microedition.io.CommConnection javadoc for all the options available when opening the port.

Test Class

The Test class for the CommConnection class is:

com.ewon.ewontest. TestCommConnection

Example

The comm ports access is using the Connector class to create the connection object that will provide an input and output stream to the serial port.

The example bellow show how to create simple echo.

```
try
{
   String CommProperty = System.getProperty("microedition.commport");
   System.out.println(CommProperty);

   CommConnection cc =
            (CommConnection)Connector.open("comm:com0;baudrate=57600;bi
   int baudrate = cc.getBaudRate();
   InputStream is  = cc.openInputStream();
   OutputStream os = cc.openOutputStream();
   int ch = 0;
   while(ch != 'Z')
   {
      ch = is.read();
      os.write(ch);
   }
   is.close();
   os.close();
   cc.close();
```

Flash File System

Class: javax.microedition.io.FileConnection

The functions library to access the eWON filesystem is based on JSR-075 and is thus standard.

We provide a few examples, but the complete documentation is available in the javadoc.

Example

This example creates a file and puts a few line in it:

```
try
{
    FileConnection fconn =
            (FileConnection) Connector.open("file:////usr/MyTest0.txt");
    fconn.create();
    OutputStream os = fconn.openOutputStream();
    PrintStream ps = new PrintStream(os);

    ps.println("0123457-A");
    ps.println("0123457-B");
    ps.println("0123457-C");
    ps.println("0123457-D");

    fconn.close();
}
catch (IOException ioe)
{
    System.out.println("Error createFileTest: "+ioe.toString());
}
```

Using IP sockets

The socket interface is also J2ME compliant.

An example class exists also in the test for reference. This calls is in:

TestSocket.java.

Asynchronous events

Some Java functions would block the whole JVM if they where executed in the JVM main thread, in order to avoid this situation, the eWON JVM implements an asynchronous mechanism where a different thread will execute the function.

For example when you read on a socket in the java program, the actual read operation is not executed in the main JVM thread but in another thread.

All the threads required are created at eWON boot time and there is a maxiumu of 6 thread for asynchronous operations. So as long as you don't have 6 Java thread doing an asynchronous operation at the same time, you won't have problems. If you do, one of the operation will fail.