WEBINGENIA

# TC65 Development

## Building programs for the TC65/TC65i

**Florent Clairambault**

8 October 2011

Development concepts and hints around the Cinterion TC65/TC65i.

# Table of Contents

1 Introduction ..... 5
1.1 About this document ..... 5
1.2 Waiting for your comments ..... 5
1.3 Who am I ..... 5
2 Why is this chip great ..... 5
2.1 Java ..... 5
2.2 OTAP ..... 5
2.3 The big picture ..... 6
2.4 Nobody's perfect ..... 6
2.4.1 JVM ..... 6
2.4.2 Storage ..... 6
2.4.3 Potential interfaces issues ..... 6
3 TC65 development pre-requisites ..... 6
3.1 Hardware ..... 6
3.2 Java ..... 6
3.3 Installing the IDE ..... 7
3.3.1 Which version? ..... 7
3.3.2 Installation of Netbeans + Cinterion WTK ..... 7
3.3.3 Installation with Netbeans 7.0 ..... 9
4 Your first program ..... 11
4.1 Create the project ..... 11
4.2 Type some code ..... 11
4.3 Check the connection ..... 13
4.4 Run it on the chip ..... 14
4.5 Set the autostart ..... 15
4.6 Only way to do things? ..... 15
5 TC65 Programming: Keys points ..... 15
5.1 AT Commands ..... 15
5.2 Settings ..... 16
5.3 URC ..... 16
5.4 SMS Management ..... 16
5.5 Power management ..... 17
5.5.1 Why ..... 17

5.5.2 How ... 18
5.6 Serial ... 18
5.7 GPIO ... 18
5.7.1 Pin by pin (AT) ... 18
5.7.2 Port configuration (AT) ... 19
5.7.3 Port configuration (Java API) ... 19
5.8 I2C ... 20
5.9 GPS ... 20
5.9.1 CommConnection with NMEA parsing ... 20
5.9.2 URC with Position parsing ... 21
5.10 Call features ... 22
5.11 Watchdog ... 22
5.12 Date ... 23
6 Upstream communication ... 24
6.1 Introduction ... 24
6.2 SMS ... 24
6.3 HTTP ... 25
6.4 TCP connection ... 25
6.4.1 Text protocol ... 25
6.4.2 Proprietary binary ... 25
6.4.3 M2MP ... 25
7 Libraries ... 25
7.1 Why ... 25
7.2 Creating a library ... 25
7.3 Using a library ... 27
8 Remotely upgrading your programs with OTAP ... 28
8.1 Introduction ... 28
8.2 Setting up the server ... 29
8.3 Setting up the project ... 29
8.4 Deploying ... 31
8.5 Launching OTAP with an AT Command ... 31
8.6 Debugging OTAP ... 32
8.7 Launching OTAP with an SMS ... 32
8.8 Launching OTAP from your programs ... 34

8.9 Use the midlet version ..... 34
9 Going a little further ..... 34
9.1 Compiler optimization ..... 34
9.2 Using the pre-processor ..... 34
9.3 Working with multiple chip versions ..... 36
9.3.1 General consideration ..... 36
9.3.2 Best effort ..... 36
9.3.3 Java API or not? ..... 36
9.3.4 EGS5 ..... 37
9.4 Signed / unsigned ..... 37
9.5 Obfuscating ..... 37
9.5.1 Why ..... 37
9.5.2 Without obfuscation ..... 38
9.5.3 Make your own kind of obfuscation ..... 40
9.6 Improving performances ..... 40
9.6.1 Introduction ..... 40
9.6.2 Reducing allocations ..... 40
9.6.3 Limiting AT calls ..... 40
9.6.4 Multithreading ..... 40
9.6.5 Network stack ..... 41
9.6.6 I/O Blocking or not? ..... 41
9.7 The NVRAM deadly mistake ..... 41
10 Some personal advices ..... 42
10.1 Project management: Start simple ..... 42
10.2 Product: Limit human actions ..... 42
10.3 Product: Find errors on the field ..... 42
10.4 Software is important ..... 42
10.5 Real time is very fine ..... 42
11 BONUS ..... 43
11.1 Sample code ..... 43
11.1.1 Logging class ..... 43
11.1.2 Asynchronous HTTP requests ..... 43
11.1.3 String splitting ..... 46
11.1.4 BufferedReader ..... 47

11.1.5 Launching an update from your program ..... 47
11.1.6 Watchdog on a TC65 v2 ..... 48
11.2 M2MP specifications ..... 49
11.2.1 Introduction ..... 49
11.2.2 Why is it working like that? ..... 49
11.2.3 Specifications ..... 50
12 FAQ ..... 52

# 1 Introduction

## 1.1 About this document

The purpose of this document is to help new or long-time TC65/TC65i developers to build programs for the TC65/TC65i chip but also leaders of M2M projects to understand how this chip could help them build better, faster and safer M2M projects.

This document doesn't intend to replace by any mean the Cinterion documentation. You should see it as a complementary document for all the wonders you might have.

## 1.2 Waiting for your comments

I'm waiting for your comments to fix / improve / complete this document. This should always be considered as a draft.

## 1.3 Who am I

I'm a software engineer, and the co-owner of a small company named WebIngenia where we build complete M2M solutions: embedded software, server, web/mobile interfaces. I've been working with the TC65 chips since the very first version.

We offer TC65i / TC65 / XT65 development services. You can contact us to talk about your projects: http://webingenia.com/contact

# 2 Why is this chip great

Why did you start with this chip, why is it great?

## 2.1 Java

Java allows you to make fast and reliable programs.

The Garbage Collector manages the memory of your programs. You cannot have memory leaks or bad pointer algorithmic unless you store unused objects in a container (vector, array, hastable, etc.)/

The exception catching mechanism allows you to build programs with some code parts that "should work". That meaning roughly that if a component fails, your program could still perform well.

## 2.2 OTAP

OTAP stands for "Over The Air Provisioning". You can update a program completely remotely without the need of implementing any sort of auto-update mechanism. This could be performed with a "Hello World" program updated to the next version "Hello World v2"

And what is even better is that obfuscated and compressed, TC65 jar files (containing your programs) are really small. You can have a complete, quite complex, multithreaded program weight less than 25 KB. It won't consume much of your data plan.

You can add to your programs some auto-updating features. At startup, it could compare its midlet version (we'll talk about it later) to the official version required by the servers. You should also add some security mechanisms to prevent auto-updating loops.

## 2.3 The big picture

The big picture is that you can considerably reduce the time-to-market. You can build your TC65 program really quickly (a week to a month) and start selling your products without spending months to test the product. And you can still add the features the users want after your product has been released.

## 2.4 Nobody's perfect

### 2.4.1 JVM

Because this chip uses a JVM it cannot be considered as a real-time performance system. But because it does perform very well on many situations, you could consider as a perfect real-time communication chip.

You shouldn't compare the Java SE performances with the J2ME performances. J2ME doesn't optimize its code on runtime; it behaves similarly to script language.

### 2.4.2 Storage

The chip also has a limited amount of storage memory and this memory is limited to 100 000 R/W operations. Some wear leveling mechanisms are applied but we don't know the quality of them. You should not write too much data on it.

I have never seen any project overload or corrupts its memory but still, this is something you should keep in mind. For long-term data logging applications, you could add an external Flash memory using the SPI port (see the Cinterion's FOTAP guide).

### 2.4.3 Potential interfaces issues

They are reports of potential interfaces issues:

- ADC might reports some values that are totally unrelated to the values.
- The GPIO10's pulse might count too much pulses.

These issues are undocumented but I personally highly suspect they exist.

# 3 TC65 development pre-requisites

## 3.1 Hardware

You must have a TC65 v2, TC65 v3, TC65i v1, XT65 v1, XT65 v2, EGS5 chip. There is no simulator. You need a computer with Windows XP, Windows Vista or Windows 7. You should take the latest version of the Module Exchange Suite for it to work on Windows 7.

## 3.2 Java

You should have done some java before. That's pretty much the complete knowledge you should have.

Whether you have done some J2ME or J2SE before doesn't really matter. What matters is that you are able to write some clean code.

## 3.3 Installing the IDE

### 3.3.1 Which version?

I recommend you take the latest version of the IDE chipped with your TC65/TC65i SDK. Because any version will work with any chip (see 9.3 - Working with multiple chip versions).

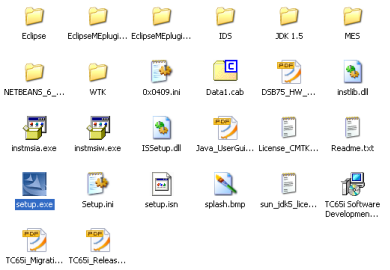
If you want to do some debugging, it might be a better idea to stick to the SDK version of your chip. If you want to debug a TC65v2 chip (wich SDK is chipped with Netbeans 5.5) on Netbeans 6.0 you have to setup Netbeans 6.0 just like Netbeans 5.5 was.

In fact, you could even install the latest version of Netbeans (Netbeans 7.0 at that time), install the WTK as a mobile platform and use it as your main TC65i development environment. That's why I like to do, but to be as close as possible as the standard Cinterion SDK, we won't discuss this here.

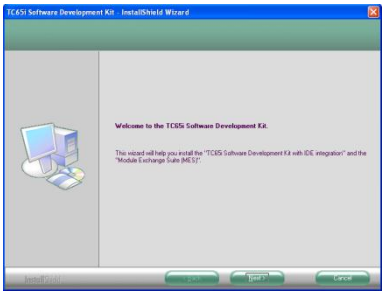
### 3.3.2 Installation of Netbeans + Cinterion WTK

The installation is REALLY easy. Just to show you the way, here are ALL the steps of the installation process. You basically just have to press "Next" all the time.

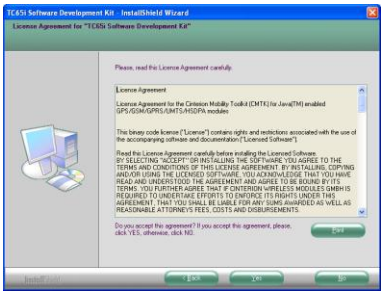
We will be using NetBeans because this is the default IDE.



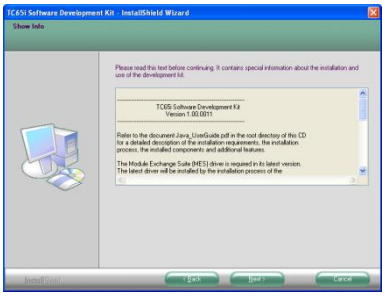

Open the "setup" application



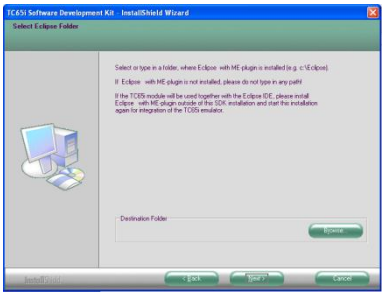

Press "Next"



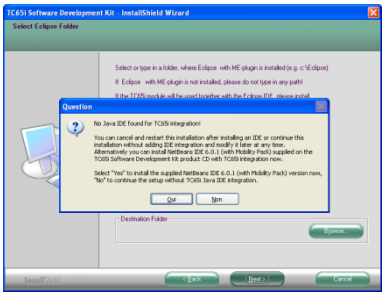

Press "Next"




Press "Next"



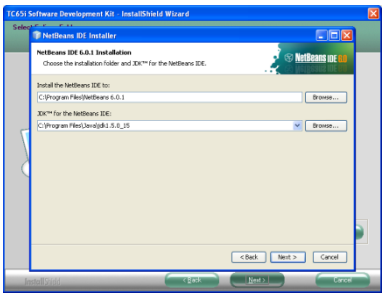

Press "Next"



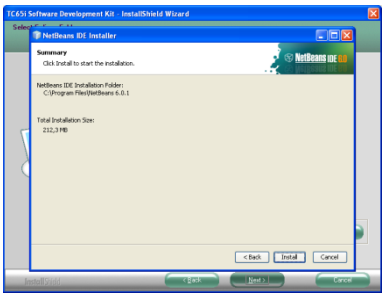

Press "Yes"


Press "Next"

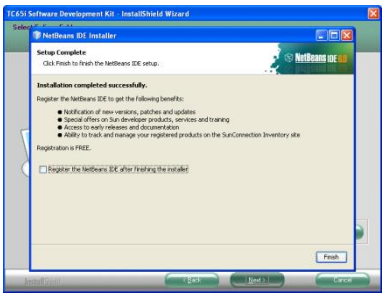

Press "Next"



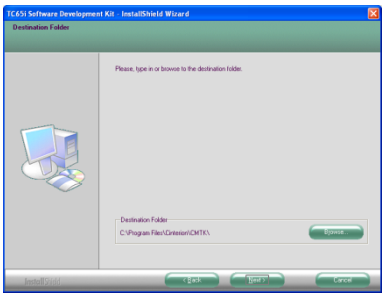

Press "Next"



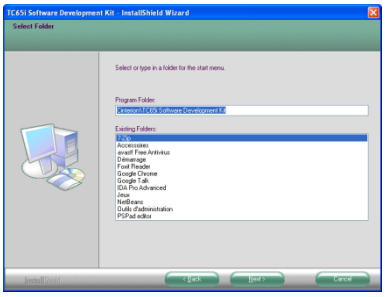
Press “Install”



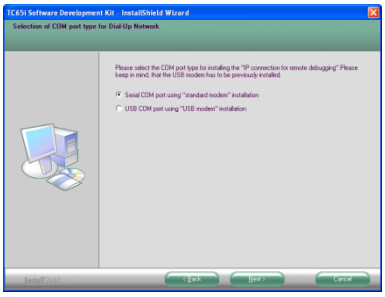
Press “Finish”



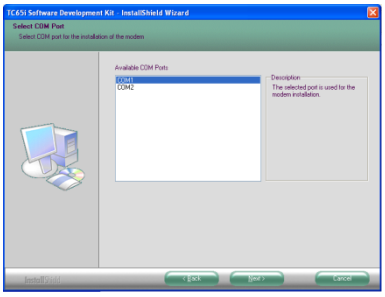
Press “Next”



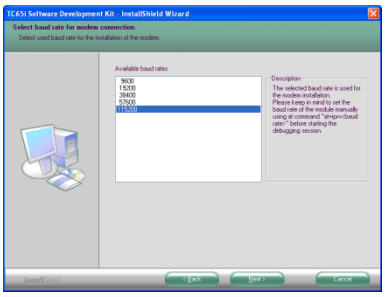
Press “Next”



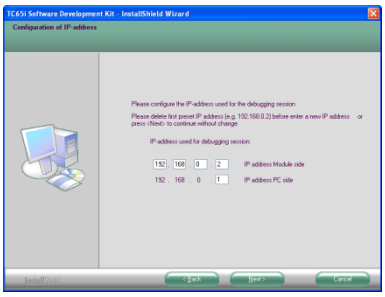
Select Serial or USB depending on what connections you have on the chip.



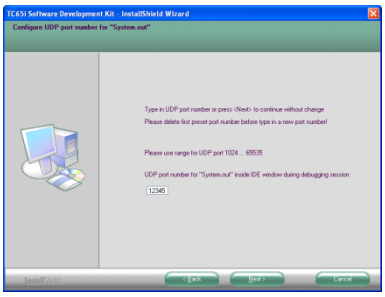
If you choose serial, you will have to choose the serial port.



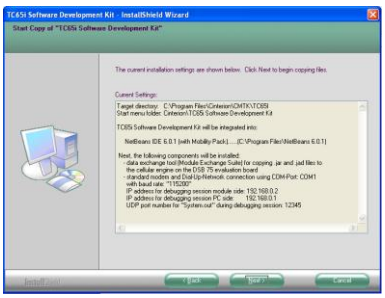
And the speed. You should select 115 200 bps.



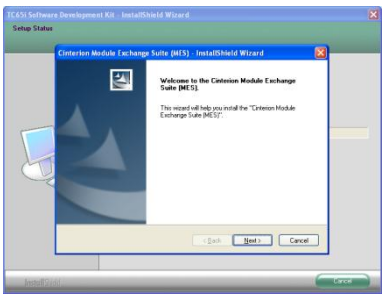
Let the IP address as is unless you network already has this numbering scheme.



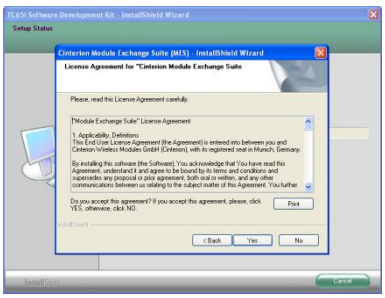
No need to change the debugging port



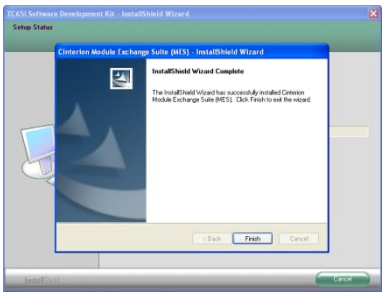
Press “Next” to install MES



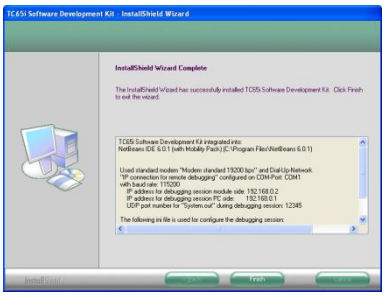
Press “Next”



Press “Yes”



Press “Finish”



Press “Finish”

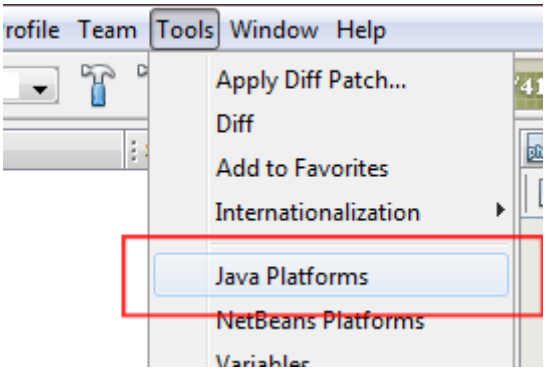
And the installation is over. You are ready to build your first program.

### 3.3.3 Installation with Netbeans 7.0

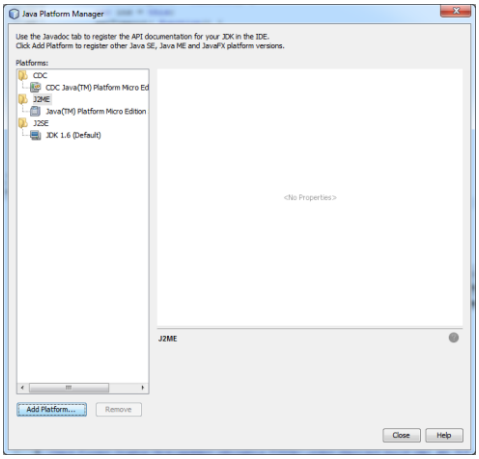
This still require a complete Netbeans 6.0 installation.

I’ll let you install Netbeans 7.0. You should choose the biggest version as this is the only one that supports the Mobile Environment.

In Netbeans, you will need to:

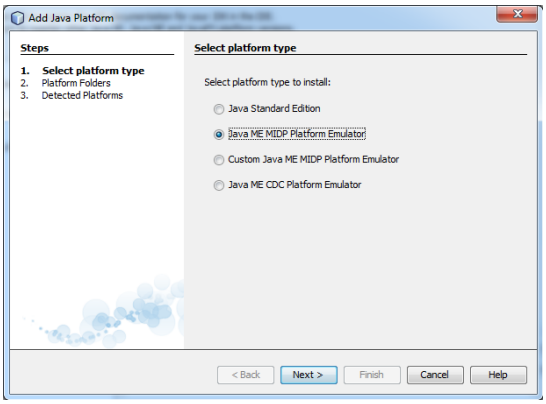



Open the “Tools” / “Java Platforms” menu

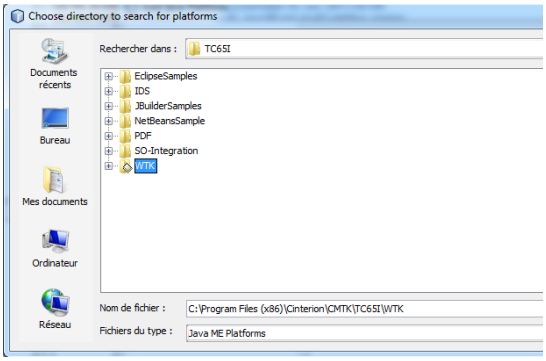



You should see a window close to this one. You have no reference no any kind of TC65i SDK. That’s your issue.

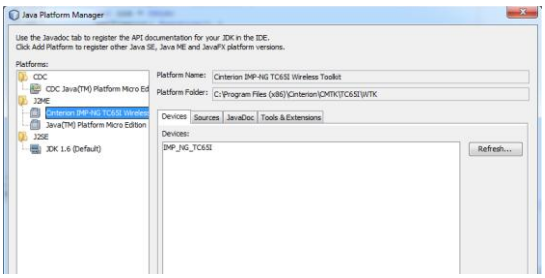
To fix this, you will click on the “Add Platform” button



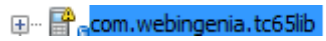

You can select the ME MIDP platform emulator.



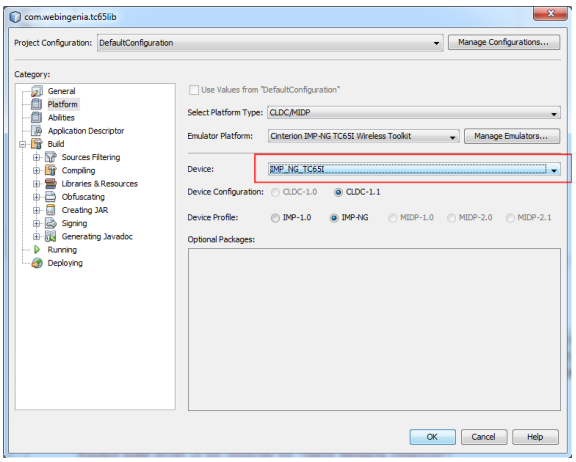

Then you will have to select the Cinterion SDK. It's in C:\Program files\Cinterion\CMTK\TC65i\WTK. If you have a 64bits system, "Program files" will be replaced by "Program files (x86)".



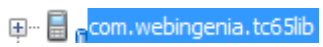

You can click "next", "next", "finish" and you will have a new java platform.




Then you can open any previous project you have. But you might have a warning on it.




If this happen, you can do a right click on the project, select "properties" and let the IDE select the good "Device". You press OK and…

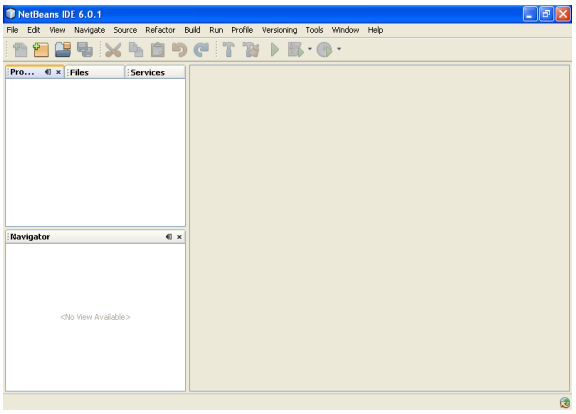



You project is loaded successfully.

# 4 Your first program

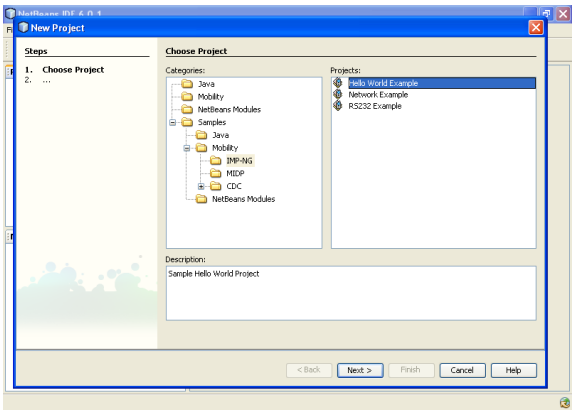
## 4.1 Create the project

Open NetBeans by clicking on the NetBeans icon on the desktop.

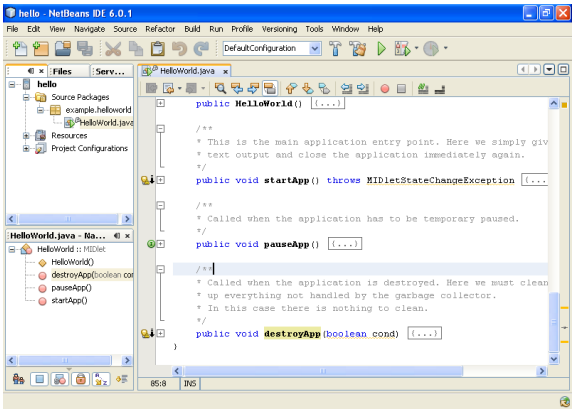




This is the window you should have once NetBeans is loaded.

Go to “File” / “New Project”




Select the “Samples” / “Mobility” / “Hello World Example”.




You created the structure of a TC65 Midlet.

## 4.2 Type some code

The program starts in the startApp method. But you shouldn’t type all your code within this method. Because according to the “Java UserGuide” document:

Note: Take care that the `startApp()` method is always properly terminated before calling the `destroyApp()` method. For example, avoid that threads launched by `startApp()` enter a closed loop, and be sure that all code was entirely executed. This is especially important for OTAP, which needs to call `destroyApp()`.

So, if your program is doing something sequentially, you should launch the sequential code in another thread. Here is what your main code could look like:

```
public class Core implements Runnable {

    private Thread _thread = new Thread( this, "core" );

    private boolean _loop;

    public void start() {
        _loop = true;
        _thread.start();
    }

    public void stop() {
        _loop = false;
        _thread.join();
    }

    public void run() {
        while (_loop) {
            try {
                Thread.sleep(1000);
                System.out.println("Main code running...");

            } catch (Exception ex) {
                System.err.println("Exception " + ex.getClass() + " : " +
ex.getMessage());
            }
        }
    }
}
```

In your main “HelloWorld.java” file, you just add to add this variable to keep a reference to the “Core” instance, start it in “startApp” and stop it (if you want) in “destroyApp”. You have the added code written in bold:

```
    private Core _core;

    public void startApp() throws MIDletStateChangeException {
        System.out.println("startApp");

        System.out.println("Loading core...");
        _core = new Core();
        System.out.println("Launching core...");
        _core.start();
    }

    public void destroyApp(boolean cond) {
        System.out.println("destroyApp(" + cond + ")");

        _core.stop();
```

```
        notifyDestroyed();
    }
```

## 4.3 Check the connection

You can open a serial or serial/USB connection using a terminal client. I always use putty because it has some logging features.

If you're using a serial port, you should always open the port at a speed of 115200 bps and then send the "AT+IPR=115200" so that the chip doesn't have to auto-detect the speed. This is recommended for program uploading.

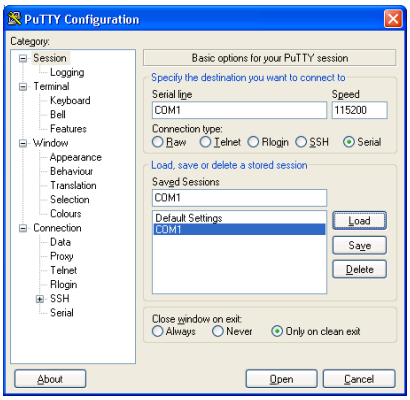


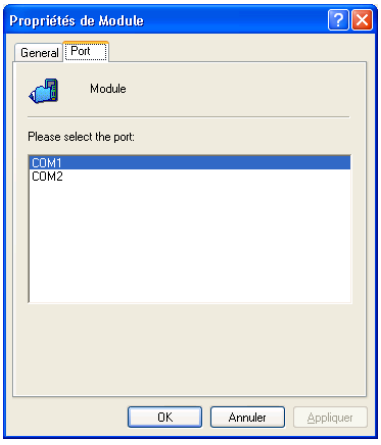

Figure 1 : Serial port openin with putty

```
AT+GMM
TC65

OK
AT+GMR
REVISION 03.000

OK
AT+IPR=115200
OK
```

Figure 2 : Commands sent to the module

Once you're connected to the chip, you can send some AT Commands to check that the chip responds:

Then in your workstation view, open the properties of the "Module" and set its port to the right one (COM1 in my case).



## 4.4 Run it on the chip

You can only run your program on the chip. There isn't any emulator available.

To run a program, you just have to press the green arrow to upload it to the chip and launch it.


To run the program, the SDK will in fact deploy it locally. You should see something like that:

```
Jad URL for OTA execution:
http://localhost:8082/servlet/org.netbeans.modules.mobility.project.jam.JAMServlet/
C%3A/TC65projs/hello/dist//HelloSample.jad
Starting emulator in execution mode
        ========================================================================
          Starting TC65I emulator for running application.   Please wait ...
        ========================================================================


        Using following ini file for debugging session:
        "C:\Documents and Settings\Administrateur\Application
Data\Cinterion\CMTK\TC65I\WM_Debug_config.ini"

        COM port used for "emulator session": COM1

        Used baud rate for the module "115200 baud"...

         >>> Starting Download of Jar and Jad file... <<<


        Downloading "C:\TC65projs\hello\dist\nbrun48365\HelloSample.jad"...
                ...finished

        Downloading "C:\TC65projs\hello\dist\nbrun48365\HelloSample.jar"...
                ...finished

        Start Java program inside the module...


        Java program is started inside the module without debugging.
        Emulator is closed, while the java program is running inside the module.
```

You can get the output on the same port you used to deploy ("`ASC0`" here), on another port ("`ASC1`" or "`USB`" for instance), on a file ("`FILE`") or remotely ("`UDP`") by debugging it or not. The options are described on the "AT Command set" document, page 79.

```
Constructor
startApp
Loading core...
Launching core...
Main code running...
Main code running...
Main code running...
```

## 4.5 Set the autostart

In production, you need your programs to start with the chip. This can be setup easily:

- You set the autostart delay to 10s (it can be set from 0 to 100s) by typing this :

```
AT^SCFG=”Userware/Autostart/Delay”,”100”
```

*Note: On TC65 (before the TC65i), the maximum delay was defined to 25s.*

- You set the autostart program to “hellosample.jad” :

```
AT^SCFG=”Userware/Autostart/AppName”,””,”a:/hellosample.jad”
```

- You enable the autostart :

```
AT^SCFG="Userware/Autostart","","1"
```

The empty parameter (“”) stand for an empty password setup. Unless you are really sure about what you are doing, don’t set a password. If you forget it, you will have to burn a new firmware to reset it.

You can check all these settings by typing “AT^SCFG?”. The interesting part will look like that:

```
^SCFG: "Userware/Autostart","1"
^SCFG: "Userware/Autostart/AppName","a:/hellosample.jad"
^SCFG: "Userware/Autostart/Delay","100"
```

This is it! You have your first program, ready to be used in a production environment.

## 4.6 Only way to do things?

As you might discover in your next days, the local deployment and debugging process can become quite instable.

That’s why some people did some programs that don’t use the MES deployment method but a completely rewritten (and open-source) deployment code. The most interesting one is JObexFTP from Ricardo Schmidt.

I prefer to automatically deploy the program from the IDE on an FTP server to be able to send the OTAP command either by an OTAP SMS, a SMS interpreted by my program, a network command, a “console” command (I’ll talk about this later) or anything else that is supposed to easily launch an OTAP process.

Not being able to debug isn’t really an issue (also I usually love debuggers) because I personally try to add as much logging as possible to be able to diagnose issues when the program runs in the field.

# 5 TC65 Programming: Keys points

## 5.1 AT Commands

ATCommand object allow you interact with the AT command layer of the chip. The main rules, about how to instantiate and use this class, are:

- You can only instantiate a limited number of them
- If you share `ATCommand` instances between threads, you have to synchronize them yourself
- When you stop using an `ATCommand` instance, you need to release it by calling the "`release`" method.

You can also register a class to some event and then connect one or more receiver to an ATCommand to receive event from this ATCommand through the ATCommandListener interface.

These event messages are the URC. They can be used to detect a change of GPIO state of the reception of an SMS.

## 5.2 Settings

You can use the embedded RecordStore object to store data into the TC65. Thought, I don't recommend using this. It doesn't make things simpler, it has been reported to get corrupted and it will be harder to analyze afterwards.

You should try as much as possible to keep a simple text file so that your technicians and end-users can modify the settings easily.

## 5.3 URC

The Unsolicited Result Codes are codes sent by the TC65 core to the program using the AT Command format. They can give you a lot of data, like changes in the quality of the GSM signal, in the state of the battery, in the state of the GPIO.

You can see an example of URC management for detecting SMS arrival in "*5.4 SMS Management*".

One of the most useful URC is this one:

```
AT+CMER=2,0,0,2,0
```

It gives you these events:

```
+CIEV: battchg,5        -- Battery charge level (0-5)
+CIEV: signal,7         -- Signal level (0-7)
+CIEV: service,1        -- Service is available (true/false)
+CIEV: sounder,0        -- When the phone is ringing (true/false)
+CIEV: message,0        -- If unread message (true/false)
+CIEV: call,0           -- If call is in progress (true/false)
+CIEV: roam,0           -- If we in roaming (true/false)
+CIEV: smsfull,0        -- If the memory is full of SMS (true/false)
+CIEV: rssi,3           -- Signal strengh (0-5)
```

## 5.4 SMS Management

```
/**
 * SMS Management class
 * @author Florent Clairambault / www.webingenia.com
 */
public class SMSManagement implements ATCommandListener {

	ATCommand _atc;

	/**
	 * Default constructor
```

```
	 * @param atc ATCommand
	 */
	public SMSManagement( ATCommand atc ) {
		if ( Logger.E_VERBOSE )
			Logger.Log( "SMSManagement.SMSManagement();" );
		_atc = atc;
		synchronized ( _atc ) { // ATC could be shared, we need to sync it
			_atc.addListener( this );
			try {
				_atc.send( "at+cmgf=1\r" );
				_atc.send( "at+cnmi=1,1\r" );
			} catch ( Exception ex ) {
				if ( Logger.E_CRITICAL )
					Logger.Log( "SMSManagement", ex );
			}
		}
		ReceiveAndTreat();
	}

	/**
	 * Receives AT Events
	 * @param ev Event received
	 */
	public void ATEvent( String ev ) {
		if ( Logger.E_VERBOSE )
			Logger.Log( "SMSManagement.ATEvent( \"" + ev + "\");" );

		try {
			if ( ev.indexOf( "+CMTI" ) >= 0 ) {
				if ( Logger.E_NOTICE )
					Logger.Log( "Received a message !" );
			}
		} catch ( Exception ex ) {
			if ( Logger.E_CRITICAL )
				Logger.Log( "SMSManagement.ATEvent", ex );
		}
	}
}
```

## 5.5 Power management

### 5.5.1 Why

Power management isn't usually an issue, but some project need to take advantage of this feature. If you use a TC65i with a battery and/or a solar power, you will need to take care of it.

Because power saving brings some constraints, it should only be chosen when necessary.

The different functionality levels can be seen in "TC65i AT Command Set", page 44. To make it short, you have 4 states:

| 0 – Non-cyclic sleep | 1 – Full functionnality | 7 – Cyclic sleep | 9 – Cyclic sleep |
|---|---|---|---|
| The chip is totally sleeping in this mode. | The chip is working. | The chip wakes up when data is received on the serial port. | The chip wakes up when data is received on the serial port. |

### 5.5.2 How

When you know you won't require doing anything on your chip, you can put it in complete sleep mode; you must put all your threads in sleep or waiting mode (see 0 for sample waiting code) and then set the sleep mode by sending on an ATCommand:

```
AT+CFUN=0
```

If you want to go a little bit further, you should also disable the GSM to avoid any GSM related consumption. This can be easily done by putting the chip in "plane mode":

```
AT^SCFG="MEopMode/Airplane","1"
```

## 5.6 Serial

A Serial port is handled like any other pair of InputStream and OutputStream connection :

```
CommConnection comm = (CommConnection)
Connector.open("comm:com0;baudrate=115200;blocking=on");
InputStream is = comm.getInputStream();
OutputStream os = comm.getOutputStream();
```

Then, you can do anything you want with it. Well anything ok, but you might not know where to start from. In most the case, you only need some basic string parsing.

## 5.7 GPIO

GPIO is one of the most commonly used communication part of the TC65. And it is really simple to handle.

You should keep in mind that the GPIO can't detect electrical changes occurring for less than 50ms.

You can control it using one AT Command or through the API (available only since the TC65i) through InPort and OutPort classes. If you only have some TC65i chips, you should stick to the java API.

You can use the GPIO pin by pin or setup a port where you combine the pins. The value you can send to the created port depends on the number of pins. For 2 pins, you can set/read values from 0 to 2. For 10 pins, you can set/read values from 0 to 1024.

To use it, you must activate the GPIO bus:

```
AT^SPIO=1
```

### 5.7.1 Pin by pin (AT)

The setup some Input and Output pin, let's say GPIO 3 in input and GPIO 4 in output

```
AT^SCPIN=1,2,0
AT^SCPIN=1,3,1
```

Then you can get the state of the GPIO 3:

```
AT^SGIO=2
```

This will return you this in case of high state:

```
^SGIO: 1

OK
```

Set high state on GPIO 4:

```
AT^SSIO=3,1
```

### 5.7.2 Port configuration (AT)

Let's say we now want GPIO 5,6,7,8 to make a 4 bits port. It means we will be able to send values between 0 and 16 ($2^4$).

We will send this:

```
AT^SCPIN=1,4,0
AT^SCPIN=1,5,0
AT^SCPIN=1,6,0
AT^SCPIN=1,7,0
AT^SCPORT=4,5,6,7
```

And receive the number of the created port:

```
^SCPORT: 112

OK
```

The created port can be considered like any other GPIO except that you can receive (and send) more values than just 0 and 1.

Here reading values from this port can be done by sending:

```
AT^SGIO=112
```

You will get:

```
^SGIO: 0
```

### 5.7.3 Port configuration (Java API)

Doing in the java API is really easy:

```
Vector pins = new Vector();

pins.addElement("GPIO5");
pins.addElement("GPIO6");
pins.addElement("GPIO7");
pins.addElement("GPIO8");

InPort inport = new InPort(pins);
System.out.println( "Value of the port : " + inport.getValue() );
```

If you want to manage it as an event, you can create a listener by implementing the InPortListener interface that basically just make you add the "`void portValueChanged( int value )`" method.

*Note that once you use the GPIOs from the java interface, you cannot use them anymore through the AT Command interface.*

## 5.8 I2C

It's clearly explained in the I2cBusConnection javadoc. The most important part for me is the example given:

```
// Open I2C connection and data streams:
i2cConnection cc = (I2cBusConnection) Connector.open("i2c:0;baudrate=100");
int baudrate = cc.getBaudRate();
InputStream inStream  = cc.openInputStream();
OutputStream outStream = cc.openOutputStream();

// Write Transfer Frame, where a = message ID, AE = Slave Address and write
// request:
String data = "<aAE000102030405060708090A0B0C0E0F>";
outStream.write(data.getBytes(), 0, data.length());
outStream.flush();
```

What isn't really explained is how you are supposed to parse messages. You basically need to get the part between the '<' char and the '>' char by reading each char one by one.

## 5.9 GPS

GPS can be handled by an external chip connected on a serial port or by the chip itself (if you have an XT65 chip).

There's no "XT65i" chip. In terms of evolution, the XT65 v2 has the same features as the TC65 v3. So, it's quite close to the TC65i.

As written in the XT65 rel2 java documentation:

> The on-board GPS functionality can be accessed in 4 different ways from a Java application.
> - AT commands via ATCommand
> - Java API JSR179
> - Transparent via CommConnection
> - Transparent via ATCommand

I personally used two modes:

- `ATCommand` with positions sent by URC - Really easy to setup but the URC can sometimes be missed by the chip. You might miss some position if you choose short period URC (every 2 seconds or less). This will be ok if you don't need real-time tracking.
- `CommConnection` - This is a little bit more complex because it requires parsing NMEA-0183. But with it you can have some real-time tracking (meaning you won't lose any data coming from the chip) and extract some other data.

### 5.9.1 CommConnection with NMEA parsing

I won't get into the details because parsing NMEA needs a complete parsing library (I built my own), you need to:

- Get each line from the CommConnection with your own BufferedReader (see 11.1.4).

- Parse each line using the NMEA-0183 protocol
- Create a position object from the parsed NMEA sentences

### 5.9.2 URC with Position parsing

You need to create a class implementing the ATCommandListener interface and send to the ATCommand class that you will connect to the following commands :

```
AT^SGPSS=1,0
AT^SGPSP=5
```

Then you will receive in your ATCommandListener class some events like this :

```
2008/09/28,13:14:27,48.8438389,N,002.2863598,E,498,000.00,000.00,0
```

This can be almost parsed « as is » by splitting the comma separated values. Here is the "Position" object you could have with the correct ParseXTText that can be used to parse URC string (like the one above):

```
public class Position {
	public  double
	 lat,		/** Latitude */
	 lon,		/** Longitude */
	 speed,		/** Speed */
	 alt;		/** Altitude */

	public long date;  /** Date (in a timestamp format) */

	public static Position ParseXTText( String str ) {
		Position pos = new Position();

		String[] spl = Common.strSplit( ',', str );

		pos.date = XTDateToTimestamp( spl[ 0 ], spl[ 1 ] );

		pos.lat = Double.parseDouble( spl[ 2 ] );
		if ( spl[ 3 ].compareTo( "S" ) == 0 )
			pos.lat *= -1;

		pos.lon = Double.parseDouble( spl[ 4 ] );
		if ( spl[ 5 ].compareTo( "W" ) == 0 )
			pos.lon *= -1;

		pos.alt = Double.parseDouble( spl[ 6 ] );
		pos.speed = Double.parseDouble( spl[ 7 ] );
		return pos;
	}
	public static long XTDateToTimestamp( String sDate, String sTime ) {
		int YY = Integer.parseInt( sDate.substring( 0, 4 ) ),
		MM = Integer.parseInt( sDate.substring( 5, 7 ) ),
		DD = Integer.parseInt( sDate.substring( 8, 10 ) ),
		hh = Integer.parseInt( sTime.substring( 0, 2 ) ),
		mm = Integer.parseInt( sTime.substring( 3, 5 ) ),
		ss = Integer.parseInt( sTime.substring( 6, 8 ) );

		java.util.Calendar cal = new CalendarImpl();
		cal.set( java.util.Calendar.DAY_OF_MONTH, DD );
		cal.set( java.util.Calendar.MONTH, MM - 1 );
```

```
            cal.set( java.util.Calendar.YEAR, YY );
            cal.set( java.util.Calendar.HOUR_OF_DAY, hh );
            cal.set( java.util.Calendar.MINUTE, mm );
            cal.set( java.util.Calendar.SECOND, ss );
            Date date = cal.getTime();
            long time = date.getTime();

            time /= 1000;

            return time;
        }
}
```

## 5.10 Call features

The TC65 chip can actually be embedded into a phone. That wouldn't make a great or even cost-effective product, but still, that's possible.

In that way you can handle calls like you would with any modem, using "ATD<number>;" to call a voice number (";" stands for voice), ATA to answer a call, ATH to hang-up calls. But wait, how do you know you're receiving a call?

This is also quite easy, this is a URC that you don't have to register, it will send a "`RING`" event on the `ATCommandListener` class that listen on an `ATCommand` instance.

But the way to really know, who is calling and what is the status of the calls you are currently making is to register to the "`^SLCC`" URCs, using the "`AT^SLCC=1`" command.

## 5.11 Watchdog

From Wikipedia:

> A watchdog timer is a computer hardware or software timer that triggers a system reset if the main program, due to some fault condition, such as a hang, neglects to regularly service the watchdog (writing a "service pulse" to it, also referred to as "kicking the dog"). The intention is to bring the system back from the nonresponsive state into normal operation.

The watchdog was brought with the TC65 v3 firmware (available on the TC65 v2 hardware). Before, it was possible to use it through an unofficial piece of code described in "11.1.6 Watchdog on a TC65 v2".

It must be configured by an AT command. It is recommended to set it to restart mode:

```
AT^SCFG="Userware/Watchdog","1"
```

And then, within your java code, you will use the **com.siemens.icm.misc.Watchdog** class to use it.

Let's say we want to do a watchdog code. We will start from the "Core" thread class of "4.2 Type some code":

```
public class Watchdog implements Runnable {

    private Thread _thread = new Thread( this, "watchdog" );

    private boolean _loop;
```

```
    public void start() {
        Watchdog.start( 60 );
        _loop = true;
        _thread.start();
    }

    public void stop() {
        _loop = false;
        _thread.join();
    }

    public boolean checkProgramState() {
        return true; // This is where you should check that your program
                     // is working fine
    }

    public void run() {
        while (_loop) {
            try {
                System.out.println("Watchdog still running...");

                if ( checkProgramState )
                    Watchdog.kick();

                Thread.sleep(30000);
            } catch (Exception ex) {
                System.err.println("Exception " + ex.getClass() + " : " +
ex.getMessage());
            }
        }
    }
}
```

In every program where I implement the M2MP protocol, in the “checkProgramState” method, one of the things my programs usually check is if the last data received from the server was less than 30 minutes ago.

## 5.12 Date

What is this to write about the date management on the TC65/TC65i? Well there’s something you should know and that might surprise you. The TC66 internal clock isn’t synchronized with the JVM clock. The java clock is synchronized with the internal clock at startup and then they go separate ways. You can’t change the JVM clock and the TC65/TC65i clock (`AT+CCLK` command) doesn’t have any effect on the JVM clock.

So what can you do about that?

- You can restart the chip everytime you need to change your java clock
- You can use the internal clock everytime you need it. That way you only use one clock. This requires you to parse the result of the “`AT+CCLK`” command. It will be quite slow due to the ATCommand usage and the string parsing.
- You can define an offset between the “real” time and the JVM clock. This is by far the best solution it is fast and it allows to resync your clock anytime you want.

This is how you can set the offset from a simple time server:

```
long timeOffset_;

public static long chipTime() {
	return ( new Date() ).getTime() / 1000;
}

public static void setCurrentTime( long time ) {
	timeOffset_ = time - chipTime();
}

public static boolean setTimeOffsetFromServer( String url ) {
	try {
		String strTime = httRequest( url, null );
		long time = Long.parseLong( strTime );
		setCurrentTime( time + 1 );
		return true;
	}
	catch ( Exception ex ) {
		if ( Logger.BUILD_DEBUG )
			Logger.log( "Common.setTimeOffsetFromServer", ex, true );
		return false;
	}
}
```

This is the corrected time:

```
public static long time() {
	return chipTime() + timeOffset_;
}
```

This is the PHP server code you need to have:

```
<?php
header('Content-Type: text);
date_default_timezone_set("UTC");
echo time();
?>
```

I'll let you translate it to other languages.

# 6 Upstream communication

## 6.1 Introduction

This is the key point of this chip. It's a machine to machine chip, it's supposed to exchange with the world through a server or other chips. But on most of the M2M projects, this aspect is looked over and they use an inefficient way of reporting data to their organization.

## 6.2 SMS

SMS is simple, quite reliable, two-way but inefficient.

## 6.3 HTTP

The HTTP protocol is the most used because it's easy to setup both on the client and the server side. But this protocol consumes a lot of bandwidth and doesn't allow two way communication unless you implement some http comet (this is what is used by long-polling ajax website like gmail chat or facebook chat).

## 6.4 TCP connection

### 6.4.1 Text protocol

Like the SMTP protocol. It's easier to debug. But it also consumes a lot of bandwidth.

### 6.4.2 Proprietary binary

You can do anything with your own binary protocol. But you should take care of making it maintainable. If the next version of your protocol breaks the previous one you will have to build a server that handles both of them. It will get trickier at every evolution.

### 6.4.3 M2MP

This is a protocol we've created for real-time M2M projects. It's really simple. The goal of the protocol is to transmit as few data as possible in a very simple way.

On these messages you can send anything you want. It can be XML content or binary file content. You can encapsulate any other protocol (or the M2MP protocol itself) into the M2MP protocol.

From a developer point of view, you only have to send messages like this:

```
m2mpConnection.send( String channel, byte[] data );
m2mpConnection.send( String channel, byte[][] data );
```

# 7 Libraries

## 7.1 Why

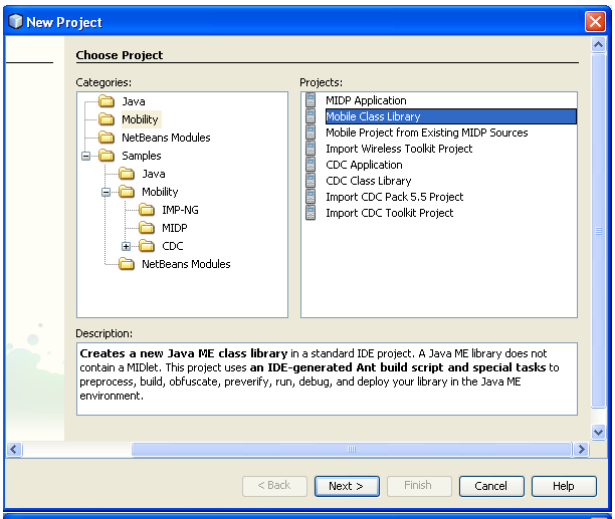
Libraries can be used in some situations:

- You want to have a set of shared classes among some projects
- You can sell classes without having to give their sources
- Since the TC65i, you download separately the libraries (liblet) and the main midlet to avoid downloading too much data at each program update

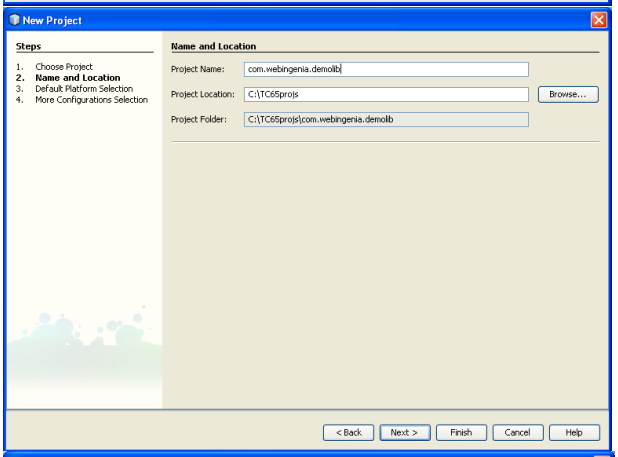
## 7.2 Creating a library

Creating a library on the TC65 is pretty much like creating a midlet program. The main difference is like any other library; you don't have a starting point. It's just a set of classes. But nothing prevents you from creating threads within your library.

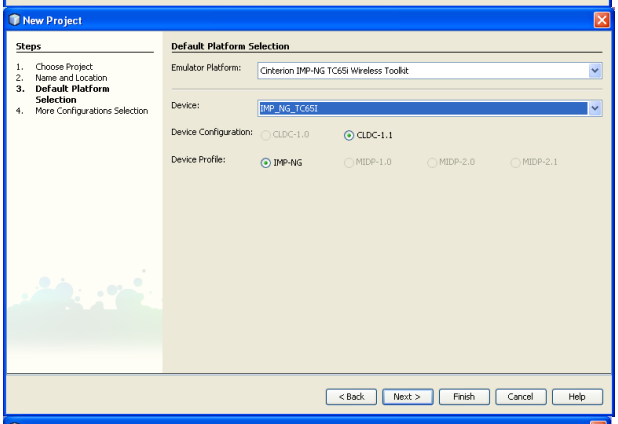
Let's do it step by step:



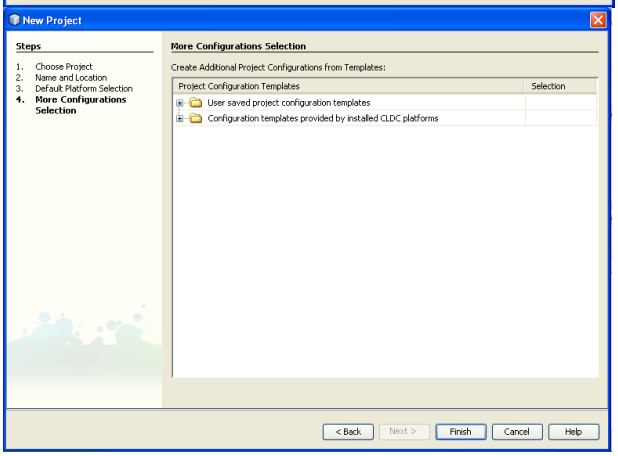

You create a new project. This one has to be a “mobile class library”




You choose a name for your library. In our case we choose “com.webingenia.demolib”.




Your platform has to be IMG-NG / CDLC-1.1




Don’t select any of these configurations.

Then you can create a package within your project. In our case we just created "com.webingenia.demolib". And you should add at least one class. We called ours "DemoClass".

Our DemoClass object won't be too complex:

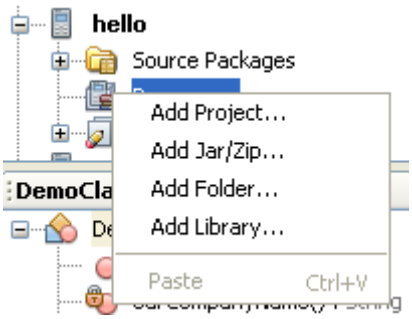
```
package com.webingenia.demolib;

public class DemoClass {

    private final String _companyName = "Webingenia";

    private String ourCompanyName() {
        return _companyName;
    }

    /**
     * Get the company name
     * @return Name of the company
     */
    public String getCompany() {
        return ourCompanyName();
    }
}
```

You should know that the library can be:

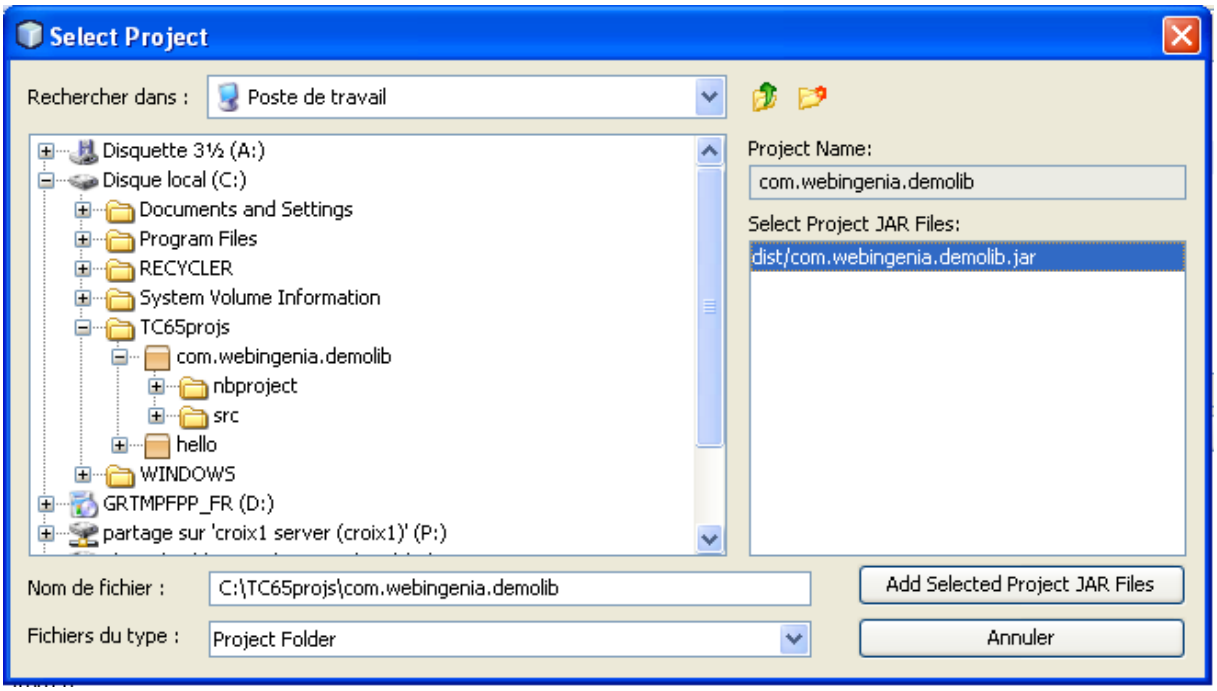
- Compressed
- Obfuscated up to level 8
- Optimized

## 7.3 Using a library

So we now have a library and we want to use it. We go back to our main project "hello". We will select the "resources" directory, right-click and select the "Add project" menu.




Then we can load our previous project "com.webingenia.com" and its jar file:




If you now want to use the “DemoClass” of the library, you just need to type in the file where you will use it the following code:

```
import com.webingenia.demolib.DemoClass;
```

And then using it within your code is as simple as that:

```
DemoClass demo = new DemoClass();
System.out.println("Company : "+demo.getCompany());
```

This is the basics but you can do anything you want with your library. Nothing prevents you from creating threads, allocating ATCommand or opening sockets within the scope of your library.

# 8 Remotely upgrading your programs with OTAP

## 8.1 Introduction

Over The Air Provisionning allows you to remotely upgrade any TC65 chip. This feature is embedded into the system of the TC65. You don’t have to do anything to add it to your program, but you can also launch the update from your program.

There are three scenarios where you need to an update:

- First install, you just need a program to install your program through 2/3 AT Commands (occurs one, easy to build a software to do it, see my TC65FM)
- Normal updates, you send "update" commands to your program through the communication channel you chose and your program launches the update. This is something you have to integrate in your software.
- The OTAP SMS / safe / "oops, I messed up the software update system" (occurs rarely, you can use my SMSOTAP software)

**WARNING:** As described in the FAQ, if you restart the chip when the destroyApp is called, the OTAP SMS method won't work because your chip will restart when the update process will begin. If you are in this situation, you can still save the day: You launch two updates, one to restart the chip (and leave your programs) and one when your chip is restarting so that the SMS OTAP is executed before your program runs. If you made the (wrong) choice of defining the setting "Userware/Autostart/Delay" to "0", you might be totally stuck.

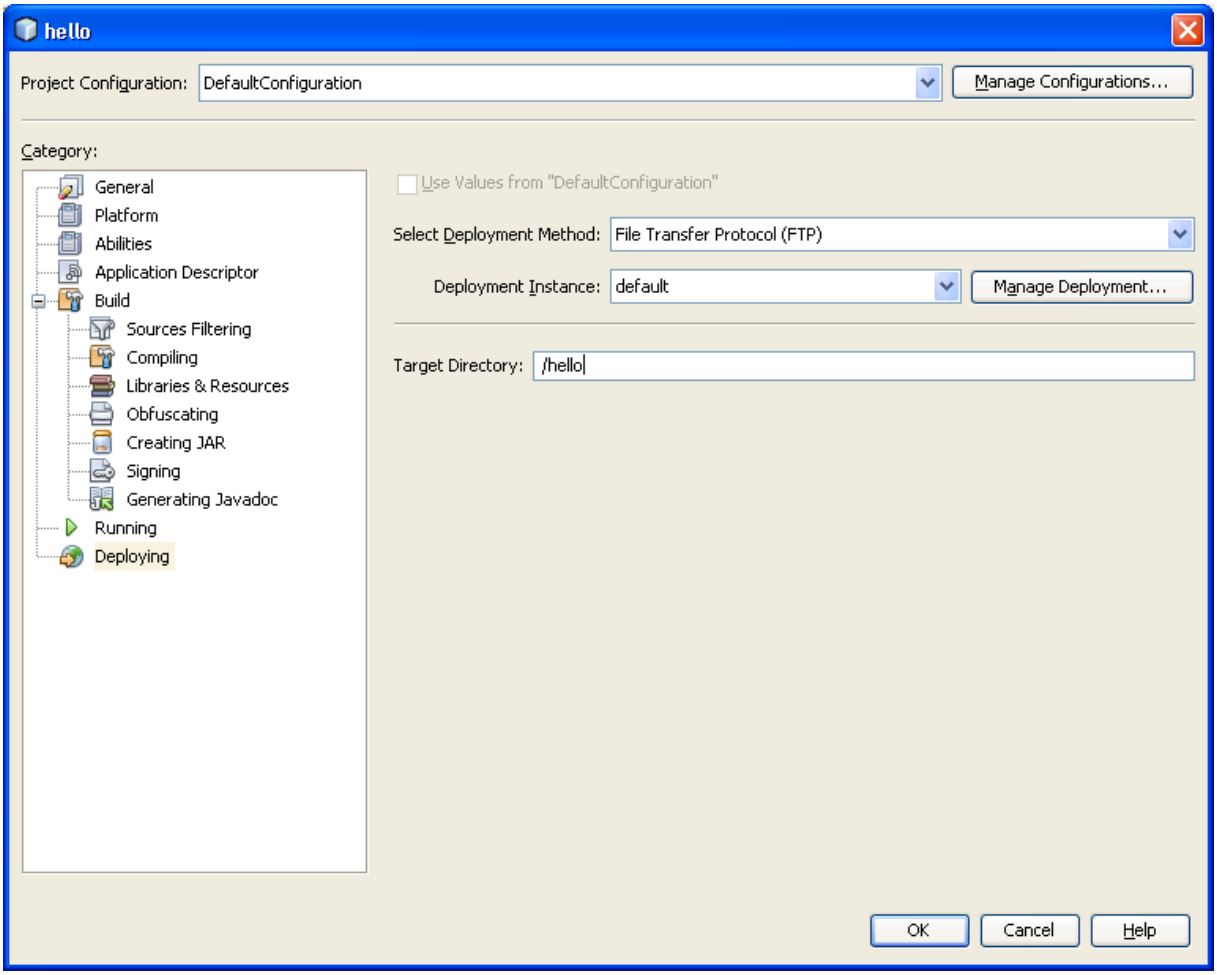
## 8.2 Setting up the server

For the purpose of the demo, we will use http://webingenia.com:8080/test/ as the hosting place for our programs.

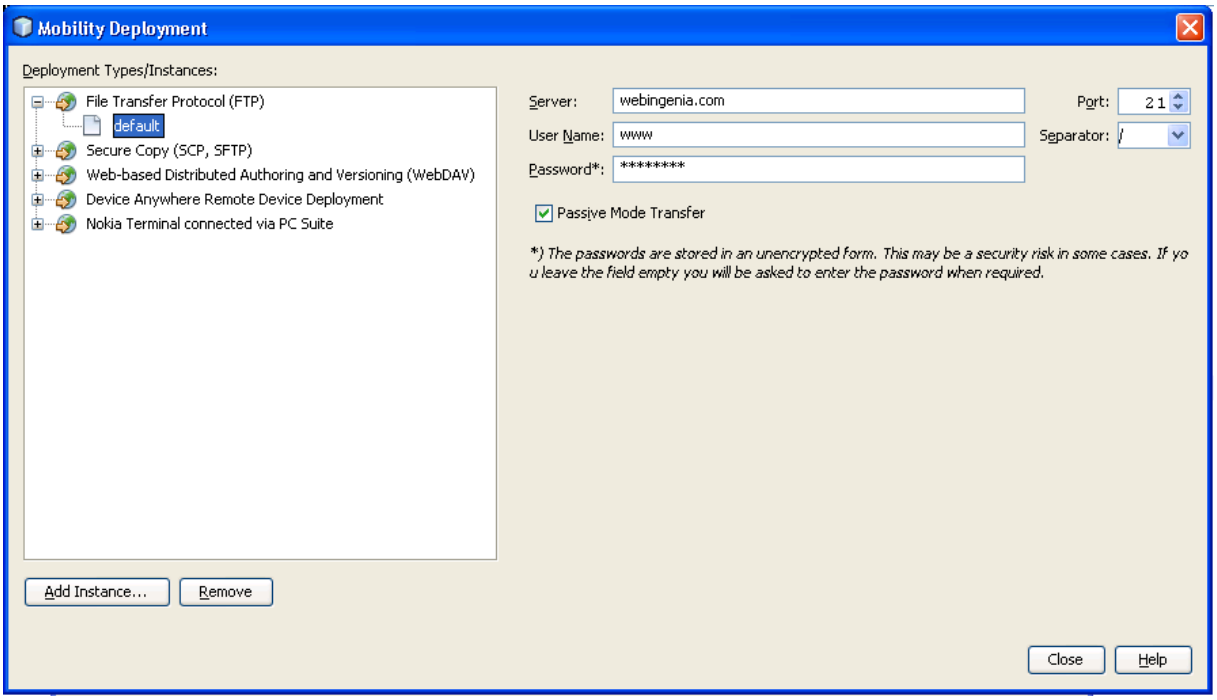
Well, this is your problem. The easiest way is to have a FTP account that points to an HTTP directory. But you can do anything else you want to.

## 8.3 Setting up the project

In your project properties, you need to choose where you will upload your files to. You can select it in the "Deploying" category:

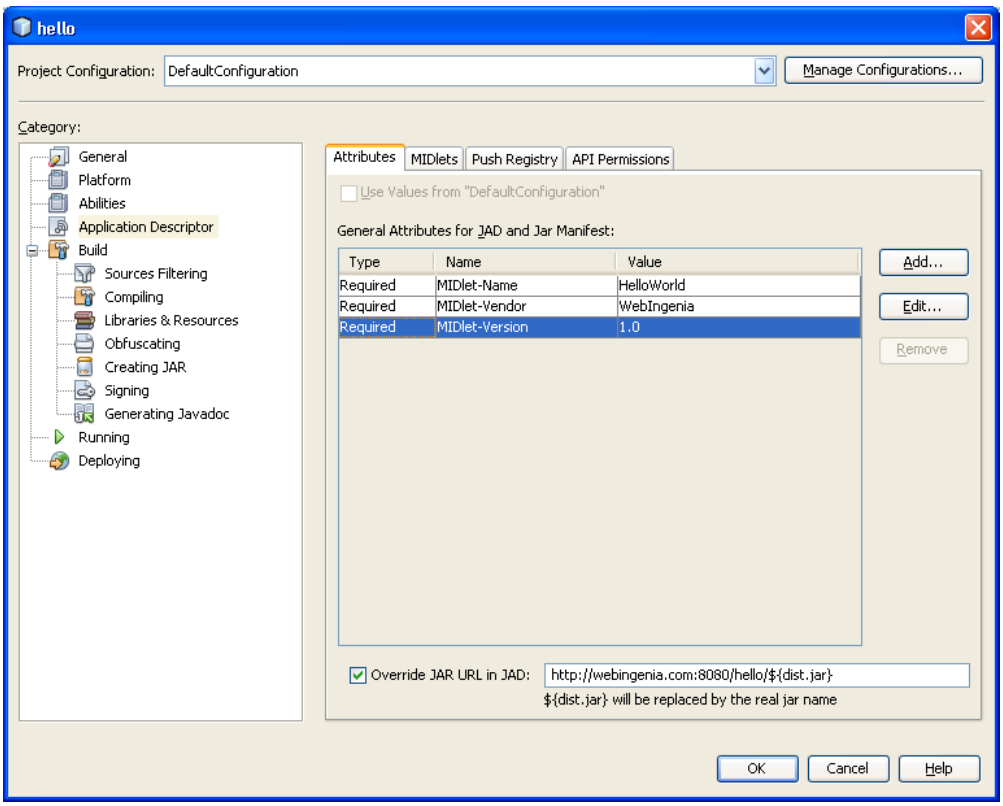



You need to change the deployment instance or create a new one so that it actually uploads to your server:




So here I selected the "webingenia.com" FTP server with user "www" and choose to upload in the "/hello" directory.

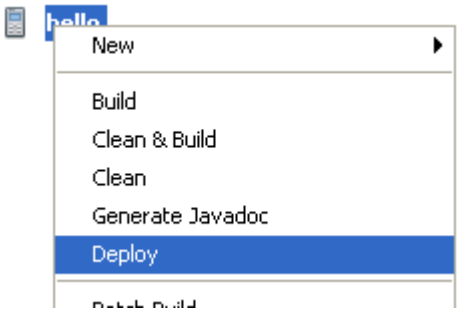
Then you need to go into the "Application Descriptor" category and select the address of your server:




So here the target server address is "http://webingenia.com:8080/hello/", the ${dist.jar} parameter will be replaced by the name of your jar file.

## 8.4 Deploying

This is the simple part, you right-click on your project and select “deploy”.




You should get something like this :

```
build:
pre-deploy:
do-deploy:
init:
set-password:
deploy-ftp:
Creating directory: /hello
Directory created OK
sending files
transferring C:\TC65projs\hello\dist\HelloSample.jad
transferring C:\TC65projs\hello\dist\HelloSample.jar
2 files sent
post-deploy:
post-deploy:
deploy:
```

On the server side, HelloWorld.jad shoud look like that:

```
MIDlet-Jar-Size: 2051
MIDlet-Jar-URL: http://webingenia.com:8080/hello/HelloSample.jar
MIDlet-Name: HelloWorld
MIDlet-Vendor: WebIngenia
MIDlet-Version: 1.0
MicroEdition-Configuration: CLDC-1.1
MicroEdition-Profile: IMP-NG
```

And the jar file has to have the same size as dscribed in the jad file:

```
-rw-r--r-- 1 www www-data  285 jun  5 17:26 HelloSample.jad
-rw-r--r-- 1 www www-data 2051 jun  5 17:26 HelloSample.jar
```

## 8.5 Launching OTAP with an AT Command

You begin to setup the OTAP parameters:

```
AT^SJOTAP=,http://webingenia.com:8080/hello/HelloSample.jad,a:,,,gprs,internet,gues
t,guest,,,,
```

And you launch the OTAP process:

```
AT^SJOTAP
```

Please note that the OTAP parameters are stored in memory. They can't be overridden by an OTAP SMS.

## 8.6 Debugging OTAP

Sometimes, like everything is the computer world, the OTAP operation fails. You can know why by typing, just after the "`AT^SJOTAP`" command, the following command:

```
AT^SCFG="Trace/Syslog/OTAP","1"
```

This is what you will get:
*(The lines I sent are in bold)*

```
AT^SJOTAP=,http://webingenia.com:8080/hello/HelloSample.jad,a:,,,gprs,m2minternet,,
,,,
OK
AT^SJOTAP
OK
AT^SCFG="Trace/Syslog/OTAP","1"
SYSLOG ENABLED

[OTAP] GPRS connection established.
[OTAP] Try to get http://webingenia.com:8080/hello/HelloSample.jad ...
[OTAP] Connected.
[OTAP] Transfer finished.
[OTAP] Try to get http://webingenia.com:8080/hello/HelloSample.jar ...
[OTAP] Connected.
[OTAP] Transfer finished.
[OTAP] JAM status: 900 Success.
[OTAP] Reboot now.
^SYSSTART
```

When you reach "`^SYSSTART`", your chip has restarted.

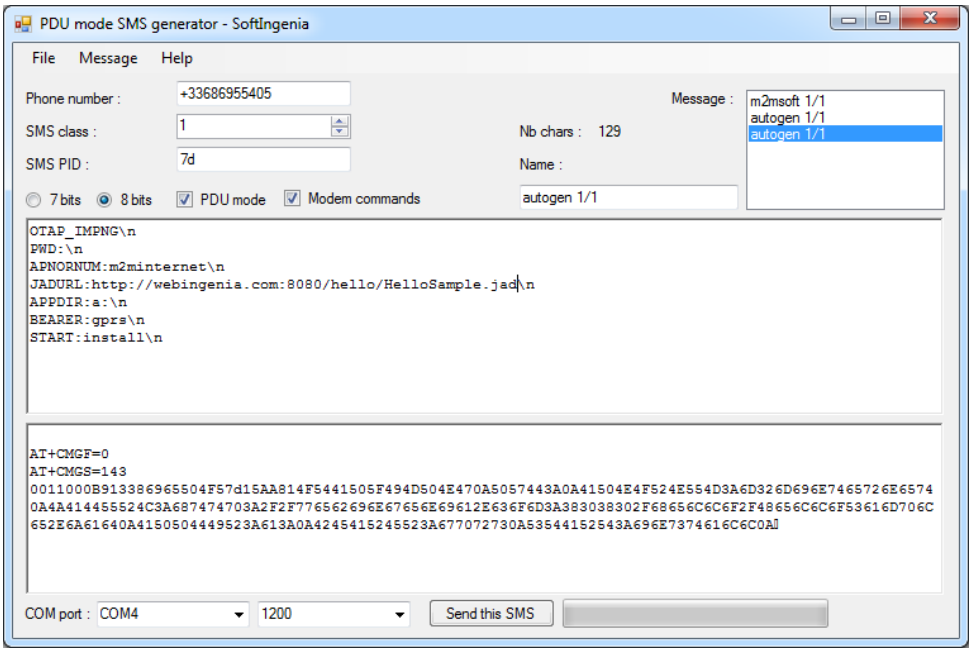
## 8.7 Launching OTAP with an SMS

You can upgrade the program of any TC65/TC65i chip with a simple SMS. This SMS will be directly handled by the chip. Even if your program is totally locked, you won't have a problem upgrading it. The only exception to that rule is if your TC65 program is using a lot of bandwidth, the SMS is considered as low priority and will only reach the chip when the chip stops transmitting some data. This is a rare situation, but it can also be a very scary one.

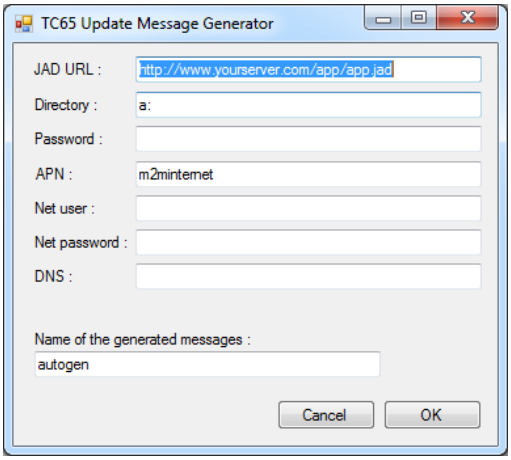
You can read the exact specifications of the SMS to send, in the "Java UserGuide" document. The format of the SMS is pretty simple, but it must be exactly as described in the documentation.

I made a tool to generate these special SMS. You must use a TC65 as a modem to send these SMS messages.You can download the tool here: http://florent.clairambault.fr/smsotap.

This is the tool:




And you even have an integrated update message generator in “Message” / “Build TC65 Update Message”:




You should take care of the security of your update process. I don’t recommend you setup a specific URL address as it could change (can you really predict how will be your IT infrastructure in 5 years?). But you should definitely setup a password.

- A password should be used to update with OTA (SMS Authentication)
- Parameters should be set to fixed values (AT^SJOTAP) whenever possible so that they cannot be changed over the air.
- The HTTP server should be secure. (e.g. Access control via basic authentication).

You should note that you can debug SMS launched OTAP the same way you can debug AT Command launched OTAP. You just need to activate the OTAP logging on the chip because you send it some SMS.

## 8.8 Launching OTAP from your programs

This is the method you should use for almost all your scenarios. With the sample update method, this is how I do the update

```
if ( updateRequired ) {
	Settings set = Settings.getInstance();
	Common.update(
		Global.atc,
		set.getSetting( "apn" ),
		set.getSetting( "jadurl" )
	);
}
```

## 8.9 Use the midlet version

It is very important to know what device runs what version of the program.

In your project options, in the "general" menu you can click on the checkbox "Automated Application Version Incrementation". To get it in your program, your startup midlet should retrieve the value of the `getAppProperty( "MIDlet-Version" )` command.

This information will make your programs management a lot easier. You can include them in your bug reports.

You can create an automatic versions management system that will update some chips when their program's version is bellow a defined version number.

# 9 Going a little further

## 9.1 Compiler optimization

Have a look at the logging class I wrote on 11.1.1.

You can use this logging class by typing:

```
if ( Logger.E_VERBOSE )
	Logger.Log( "I'm logging now !");
```

The state of the variable like `Logger.E_VERBOSE` is defined by the value of `Logger.loggingLevel`, it will be set to true if `Logger.loggingLevel` is superior or equal to 4.

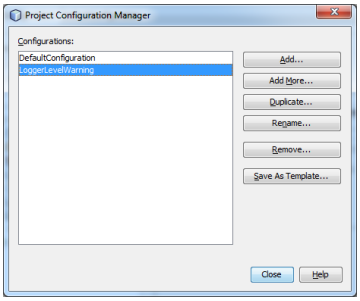
These variables have the "final" keyword; it means they cannot be changed at runtime. So when you change the `loggingLevel`, you change the state of `Logger.E_VERBOSE`. The java compiler knows if the code that follows the condition on "`if( Logger.E_VERBOSE)`" will ever be executed or not.

If you have activated the optimizations, the compiler will NOT add the code that will never be executed. It means that you program will weigh less, contain less debugging information and run faster.

## 9.2 Using the pre-processor

Yes, you might not know that, but the J2ME has a pre-processor. And that's a pretty amazing feature.

Let's say I want to create a version of my library with a reduced logging level. I will create a new Configuration for the project:




So, I know will override some (but only SOME) of the settings of my "DefaultConfiguration" project. I will leave most of the options to the "DefaultConfiguration" ones by keeping the "Use Values from "DefaultConfiguration"" option checked. Most of them except the "Compiling tab" where I will change the debug block level from "debug" to "warn" and the "creating jar" where I will change the JAD and JAR file name by appending a "_warn" at the end of their name.


You could for example make an automated build of your code in 4 different versions: one for each of your project configuration, which actually are each of your chips and deploy them at the same time.

I can now make some specific code, like:

```
/** Build logging level */
public static final int buildLoggingLevel =
//#if DefaultConfiguration
E_DEBUG;
//#elif LoggerLevelWarning
//# E_NOTICE;
//#endif
```

If I change the configuraton to "LoggerLevelWarning", this code will automatically switch to:

```
/** Build logging level */
public static final int buildLoggingLevel =
//#if DefaultConfiguration
//# E_DEBUG;
//#elif LoggerLevelWarning
E_NOTICE;
//#endif
```

You could also have written it like that:

```
/** Build logging level */
public static final int buildLoggingLevel =
//#if DebugLevel=="debug"
E_DEBUG;
//#elif      DebugLevel=="warn"
//# E_NOTICE;
//#endif
```

One very interesting pre-processing command is the “#debug warn”, “#mdebug warn”, “#enddebug warn”. But it doesn’t support else conditions. So it can’t be used everywhere.

For more details on how to use the pre-processor on J2ME, you should read this page: http://antenna.sourceforge.net/wtkpreprocess.php

And the best part is that you can automatically deploy your two configurations using the “batch deploy” right-click menu. If you chose to deploy it using FTP, you will actually deploy the two jars with their jad files automatically.

## 9.3 Working with multiple chip versions

### 9.3.1 General consideration

You can make a program for multiple versions of a chip. The easiest way is to level down the functionalities of the programs by the oldest chip. That means not using some brand new AT Command or java API.

Sometimes, some AT commands might not behave the same way between two versions of the chip but generally the AT commands are thought to be backward compatible (only some optional arguments are added in the more recent versions).

### 9.3.2 Best effort

You might be interested in using the news API on news chip and stick with the old ones or not use any on old chips. Calling a class that doesn’t exist throws a `NoClassDefFoundError`. If you want to catch it, you need to catch it through a “`catch( Throwable th)`” code. `Throwable` is the base class of the `Exception`.

The best example of this need is the software watchdog; you might definitely need to try it.

To make a call to a java API that you are not sure that exists on the current chip, you just need to check the current version of the chip (by fetching the version of the chip). Or you can try and catch `NoClassDefFoundError` but it’s going to cost you compared to analyzing the chip’s version once.

### 9.3.3 Java API or not?

Honestly, once you have built your ATCommand wrappers, you won’t really care about using the JAVA API or not. I even ended-up building some wrappers that offer the same API as the java ones.

The Java API is faster, easier to use and doesn’t have some constraints of the ATCommand based API. It also improves reliability because events are dropped if the event receivers come. With the ATCommand API, you can queue up to 5 URC events, after that URC won’t work anymore.

### 9.3.4 EGS5

With the EGS5, Cinterion decided to switch the namespaces of all the classes to "`com.siemens`" to "`com.cinterion`". This mean you have almost no effort to port it but you can't "EASILY" make the same program work on both platforms.

If you really wanted to do that, I guess you could create a base class with two implementations, one that use some "com.siemens" namespace classes and one that use the "com.cinterion" namespaces classes. But you would have to rewrite one base class and two proxy classes for each class of the TC65i's SDK. That's a LOT of stupid work to do.

If you are faced with this situation, I would recommend using the preprocessor to create a chip specific import code. You could add the "sdkns" (like "SDK Namespace") capacity, and create a project configuration with "siemens" value and the other with the "cinterion" value.

Then it would look like that:

```
//#if sdkns=="siemens"
import com.siemens.icm.io.ATCommandListener;
//#elif sdkns=="cinterion"
//# import com.cinterion.io.ATCommandListener;
//#endif
```

## 9.4 Signed / unsigned

Java doesn't care about unsigned number, unfortunately, the rest of the computer world does. So In a lof of situations, you will have to convert data back and forth between unsigned and signed types.

If you need to respect a protocol where you send some UInt32 (C# notation) / __uint32 (C Notation) data, you will need in fact to work with a long number type and then extract the lower 32 bits to a byte array :

```
public static void longToUInt32Bytes(long l, byte[] out, int i) {
    out[i++] = (byte) ((l & 0xFF000000L) >> 24);
    out[i++] = (byte) ((l & 0x00FF0000L) >> 16);
    out[i++] = (byte) ((l & 0x0000FF00L) >> 8);
    out[i++] = (byte) ((l & 0x000000FFL));
}
```

## 9.5 Obfuscating

### 9.5.1 Why

When you upload a program into a TC65 chip, it won't be possible to download the .jar file. In fact any directory of file ending with ".jar" won't be accessible from java code or external interface. In the "Java User Guide", you can read this:

> The copy protection rules for Java applications prevent opening, reading, copying, moving or renaming of JAR files. It is not recommended that the name of a Java application (for example <name>.jar) be used for a directory, since the copy protection will refuse access to open, copy or rename such directories.

So you don't need to obfuscate programs that will be deployed only once.

When you want to upgrade the program of your chips, you need to make it available. Somehow, someone could have access to your (compiled) program. That means that they could easily decompile and reuse your code as their own.

I wanted to add support for a tracking program on the Globalsat TR-102. I used an update the company gave and their site and it took me something like two days to fully understand and actually "port" my program to their hardware.

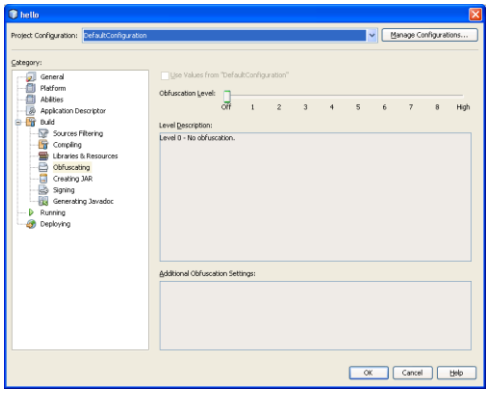



**Figure 3 : Choosing obfuscation level**

You can choose the obfuscation level in your project properties in the "Build" / "Obfuscating" category. The obfuscation level can be between 0 (no obfuscation) and 9 (full obfuscation). I recommend you set 0 for debugging & tests and 9 for production. If you decide to build libraries for the TC65i, you should not set it to more than 8, because beyond that point every public classes except Midlets are renamed.

### 9.5.2 Without obfuscation

The code below is too simple to make a good example of code obfuscation. But still, it will show you how easily code can be extracted from your .jar files.

So here is the main code of the first Midlet we built:

```
public class Core implements Runnable {

    private Thread _thread = new Thread( this, "core" );

    private boolean _loop;

    public void start() {
        _loop = true;
        _thread.start();
    }

    public void stop() {
        _loop = false;
    }

    public void run() {
        while (_loop) {
            try {
                Thread.sleep(1000);
                System.out.println("Main code running...");

            } catch (Exception ex) {
                System.err.println("Exception " + ex.getClass() + " : " +
ex.getMessage());
            }
        }
    }
}
```

We take the very good JD (Java Decompiler) available here: http://java.decompiler.free.fr/ to decompile the code and we get:

```
public class Core implements Runnable {

    private Thread _thread;
    private boolean _loop;

    public Core() {
        this._thread = new Thread(this, "core");
    }

    public void start() {
        this._loop = true;
        this._thread.start();
    }

    public void stop() {
        this._loop = false;
    }

    public void run() {
        while (this._loop) {
            try {
                Thread.sleep(1000L);
                System.out.println("Main code running...");
            } catch (Exception ex) {
                System.err.println("Exception " + ex.getClass() + " : " +
ex.getMessage());
            }
        }
    }
}
```

Well, what do you think about that? Private variable are initialized in the constructor, they are all appended by “this.” But other than that, your code is exactly the same.

Now, if we obfuscate, we get that:

```
public final class a implements Runnable
{
  private Thread jdField_a_of_type_JavaLangThread = new Thread(this, "core");
  private boolean jdField_a_of_type_Boolean;

  public final void a()
  {
    this.jdField_a_of_type_Boolean = true;
    this.jdField_a_of_type_JavaLangThread.start();
  }

  public final void b()
  {
    this.jdField_a_of_type_Boolean = false;
  }

  public final void run()
  {
    while (this.jdField_a_of_type_Boolean)
```

```
      try
      {
        Thread.sleep(1000L);
        System.out.println("Main code running...");
      }
      catch (Exception localException)
      {
        System.err.println("Exception " + localException.getClass() + " : " +
localException.getMessage());
      }
  }
}
```

Here we have a very simple 30 lines code and it's getting quite hard to read. Imagine what it would be with any project containing 400 lines of code. And most of the projects contain a lot more than that.

### 9.5.3 Make your own kind of obfuscation

Still, if you want to protect how you will handle GPIO, pulse counter or serial port, this won't be enough. You need to add some lines of dumb code that look like the real code. User searching for GPIO AT commands won't be able to find the right one.

## 9.6 Improving performances

### 9.6.1 Introduction

If you really need to improve performance, the TC65 chip might not be the best product for you. As the TC65 is NOT a real-time programming chip, I would highly recommend you identify your performances goals and requirements before using the chip.

### 9.6.2 Reducing allocations

If you need high performance, you will need to reduce the number of allocation and reuse as much as possible the same objects.

If you work with a buffer of data (a `byte[]`), you should allocate it once and then gave its content to child methods using the offset and then length.

For your business logic objects, you should get the used objects and only change the member variables.

### 9.6.3 Limiting AT calls

Any AT Command sent (call of the "send" method on an ATCommand object if you prefer) will roughly take 50ms to execute. It can considerably slow your programs. It means that the execution of 20 AT commands could take one second to execute.;

So you shouldn't use them too much and you should never use them to retrieve regularly some data.

If you need to retrieve some data that doesn't change, just cache it (in a variable). If you need to retrieve some data that changes, you should consider using the right URC.

### 9.6.4 Multithreading

Multithreading itself slows down the raw speed of your programs because you make the TC65 processor change its context to execute other threads.

But most of the times you won't in fact execute more than one a thread in the same time. Most of the time, you need to create thread to handle slow or even blocking I/O operations.

Multithreading, and particularly in java, isn't complex. You have to think how your programs will behave when doing two (or more) things in the same time and how you want these threads to communicate.

You can see an example of basic asynchronous request sending in 0. To understand the code you basically need to see where the communication on the two threads is done. You should try to lock as few objects as possible.

### 9.6.5 Network stack

The network stack of the TC65 is quite basic. In a TCP connection, the TC65 waits for the acknowledge packet of the last data packet sent before sending the next one. This simplifies the network implementation and reduces the stack memory footprint.

This mean that the minimum time between two packets of information is the time it takes to send a data packet and receive it's acknowledge packet.

### 9.6.6 I/O Blocking or not?

For all the sources you can open an input stream from, you can choose if they are blocking or not.

If you open your source (say a CommConnection as seen in 5.6) in "blocking" mode, each time you request a character that is not yet available, your thread will be "blocked". You can avoid being blocked by using the InputStream::available method to check if there are some bytes to read.

## 9.7 The NVRAM deadly mistake

**WARNING:** This is not factual or precise but still VERY important.

You might never have thought about it, but the settings you define using the AT commands are set into a separate memory of the chip, the NVRAM.

If I haven't any real proof supporting that some flash memory was corrupted but I have some proof of the NVRAM getting corrupted. Here is how it looks like:

```
0000fd09,05e2686973745f7461736b2e632c31303030302c6162706d6d6d2e632c3132373
22c4d6f6465204d616e616765723a426c6f636b2032303030204e5652414d207265616420666616c75
7265206f72204461746120636f7272757074
```

This is hex-translated to:

```
hist_task.c,10000,abpmmm.c,1272,Mode Manager:Block 2000 NVRAM read failure or Data corrupt
```

If that happens, your device is bricked. My guess is the perfect way to destroy your NVRAM is to change your settings often and to restart often (like more than once a day). So don't do it.

# 10 Some personal advices

## 10.1 Project management: Start simple

You should make your specifications (whether it's your own project or you want someone else to write it) as simple as possible, specifying the final goal and defining some stages of features that the program should have. That way, you can start the tests early, improve what's going wrong and redefine your priority with a clear view of where you are going.

What? That's not new, that's agile methodology? You can call it whatever you want, it's what you should do if you want your project to be delivered in time.

You should also avoid changing built and tested components, or plan some new testing phase. Once components are built and connected to each other, it's usually a testing mess when we rewrite one of them.

## 10.2 Product: Limit human actions

Avoid as much as possible human actions. Anything that can be done automatically should be done that way. Human make errors, and these errors can make you lose hours. The best way is to avoid any local action and limit any remote action. Even if it's more complex, it's still less complex than having to understand what your technicians or your users have done on the chip that made it behave so oddly.

## 10.3 Product: Find errors on the field

You need to build your program by acknowledging that it will eventually fail. Any main loop (of each thread) must contain some code to report errors at least on a console, but it's best to send it on a server or by SMS.

You might have to update your program remotely to add some logs in order to have a better understanding of the current issues. Updating your program is critical.

## 10.4 Software is important

OK, I'm biased; I want to take your money to build software. BUT, seriously, Steve Jobs makes a lot of money selling hardware because he has the best software out there.

But for sure: The best thing is to have your software on your hardware. This gives you a lock on everything.

Software can even fix some hardware issues. I did it many times.

## 10.5 Real time is very fine

Defining a way to establish a bidirectional real-time communication is not just a "nifty feature". It's giving the human the possibility to interact with devices in a confortable way. Whether they are maintenance technicians or users, they will like it.

That's why I defined the M2MP protocol, to give a simple and extensible way to do some real-time communications. If you intend to build your own real-time protocol, you should have a look at this one and MQTT's one.

# 11 BONUS

## 11.1 Sample code

### 11.1.1 Logging class

```
/**
 * Logging class
 * @author Florent Clairambault / www.webingenia.com
 */
public class Logger {

	// This dirty code has a purpose, it allows compile time optimization
	// jar file size depends on these options
	/** Logging level */
	public static final int loggingLevel = 3;
	/** Shows everything */
	public static final boolean E_DEBUG = (loggingLevel >= 5);
	/** Shows verbose logging */
	public static final boolean E_VERBOSE = (loggingLevel >= 4);
	/** Shows notice logging */
	public static final boolean E_NOTICE = (loggingLevel >= 3);
	/** Shows warning logging */
	public static final boolean E_WARNING = (loggingLevel >= 2);
	/** Shows critical logging */
	public static final boolean E_CRITICAL = (loggingLevel >= 1);

	/**
	 * Logs anything
	 * @param str String sent to logging
	 */
	public static void log(  String str  ) {
		synchronized ( System.out ) {
			System.out.println( Thread.currentThread().getName() + " : " +
str );
		}
	}

	/**
	 * Logs exception
	 * @param str String sent to logging
	 * @param ex Exception caught
	 */
	public static void log(  String str, Exception ex  ) {
		log(  str + " ex : " + ex.getClass() + " : " + ex.getMessage()  );

		// If we have an exception thrown in loop, we prefer that it doesn't
slows down
		// the other threads
		Thread.yield();
	}
}
```

### 11.1.2 Asynchronous HTTP requests

```
/**
 * Async http POST request sender
 * @author Administrateur
 */
public class AsyncHttpRequests implements Runnable {
```

```
    private final String _webUrl = "http://test.webingenia.com/asyncHttpRequest";
    private Vector _queue = new Vector();
    private Thread _thread = new Thread(this, "AsyncHttpThread");

    public AsyncHttpRequests() {
        // We want this thread to run in the lowest priority possible
        _thread.setPriority(Thread.MIN_PRIORITY);
        _thread.start();
    }

    public void run() {
        String data = null; // This is the data we will get from the queue

        try {

            synchronized (_queue) { // We need to make sure we won't
                try {
                    // While there's nothing to send
                    while (_queue != null && _queue.size() == 0) {
                        // We put the thread on hold
                        _queue.wait();
                    }
                    data = (String) _queue.firstElement();

                } catch (InterruptedException ex) {
                    ex.printStackTrace();
                }
            }

            // We're sending the main data
            if (sendData(data)) {
                synchronized (_queue) { // We lock the queue to remove one of its
elements
                    _queue.removeElement(data);
                }
            } else {
                Thread.sleep(2000);
            }

        } catch (Exception ex) {
            ex.printStackTrace();
        }
    }

    /**
     * Adds some data
     * @param data Data to add
     */
    public void addData(String data) {
        synchronized (_queue) {
            _queue.addElement(data);
            _queue.notify();
        }
    }

    private boolean sendData(String data) {
        return (httpPost(_webUrl, data.getBytes()) != null);
    }
```

```
}
```

If you want your code to actually work, you need to add these methods:

```
    /**
     * Send some data by posting them
     * @param url Url to send the data
     * @param data Content of the POST data
     * @return data returned
     */
    private static String httpPost(String url, byte[] data) {
        if (data == null) {
            data = new byte[0];
        }
        HttpConnection conn = null;
        InputStream is = null;
        OutputStream os = null;
        try { // Test HTTP connection

            // We prepare the POST request
            conn = (HttpConnection) Connector.open(url);
            conn.setRequestMethod(HttpConnection.POST);
            conn.setRequestProperty("Content-Type", "application/x-www-form-
urlencoded");
            os = conn.openOutputStream();
            os.write(data);

            // We display the generated content
            return inputStreamReadToEnd(conn.openInputStream());

        } catch (Exception ex) {
            ex.printStackTrace();
            return null;
        } // Whatever happens, we close everything
        finally {
            try {
                if (is != null) {
                    is.close();
                }
                if (os != null) {
                    os.close();
                }
                if (conn != null) {
                    conn.close();
                }
            } catch (Exception ex) {
                if (Logger.E_CRITICAL) {
                    Logger.Log(36830, "Common.httpPost.2", ex);
                }
            }
        }
    }

    private static String inputStreamReadToEnd(InputStream is) throws IOException {
        StringBuffer sb = new StringBuffer();
        int ch;
        while ((ch = is.read()) != -1) {
            sb.append((char) ch);
        }
```

```
        return sb.toString();
    }
```

In case you haven't understood how you are supposed to use this class, well here is who we do it :

```
AsyncHttpRequests sender = new AsyncHttpRequests() ;
sender.addData("toto"); // We're queuing data, and wakinup the thread
                        // but nothing is blocking us
sender.addData("tata"); // We're adding some data to the queue
sender.addData("titi"); // We're still adding some data to the queue
                        // The sender's thread will continue its work
```

### 11.1.3 String splitting

Source of this code comes from:
http://www.particle.kth.se/~lindsey/JavaCourse/Book/Code/P3/Chapter24/SNAP/Worker.java

```
/**
 * Splits a string into multiple strings
 * @param separator Separator char
 * @param source_string Source string
 * @return Array of strings
 */
public static String[] strSplit(char separator, String source_string) {

    // First get rid of whitespace at start and end of the string
    String string = source_string.trim();
    // If string contains no tokens, return a zero length array.
    if (string.length() == 0) {
        return (new String[0]);
    }

    // Use a Vector to collect the unknown number of tokens.
    Vector token_vector = new Vector();
    String token;
    int index_a = 0;
    int index_b = 0;

    // Then scan through the string for the tokens.
    while (true) {
        index_b = string.indexOf(separator, index_a);
        if (index_b == -1) {
            token = string.substring(index_a);
            token_vector.addElement(token);
            break;
        }
        token = string.substring(index_a, index_b);
        token.trim();
        token_vector.addElement(token);
        index_a = index_b + 1;
    }

    // Copy elements into a string array.
    String[] str_array = new String[token_vector.size()];
    for (int i = 0; i < str_array.length; i++) {
        str_array[i] = (String) (token_vector.elementAt(i));
    }
    return str_array;
```

```
} // split
```

### 11.1.4 BufferedReader

```
/**
 * Cheap implementation of an ASCII stream reader
 * @author Florent Clairambault / www.webingenia.com
 */
public class BufferedReader {

    InputStream _is;

    public BufferedReader(InputStream is) {
        _is = is;
    }

    private StringBuffer _buffer = new StringBuffer();
    public String readLine() {
        int i;
        char c;
        try {
            while ( (i = _is.read()) != -1 ) {
                c = (char) i;
                if ( c == '\n' || c == '\r' ) {
                    String str = _buffer.toString();
                    _buffer.setLength(0);
                    return str;
                }
                else
                    _buffer.append( c );
            }
        } catch (IOException ex) {
            ex.printStackTrace();
        }

        return null;
    }
```

### 11.1.5 Launching an update from your program

```
/**
 * Update the program
 * @param atc ATCommand to use
 * @param apn Connection string to use
 * @param target Target program to update from
 */
public static void update( ATCommand atc, String apn, String target ) {
       try {
              String ret1 = null, ret2 = null;
              synchronized ( atc ) {
                     ret1 = atc.send( "AT^SJOTAP=," + target + ",a:,,," + apn + "\r"
);
                     ret2 = atc.send( "AT^SJOTAP\r" );
              }

              if ( Logger.BUILD_DEBUG ) {
                     Logger.log( "Common.update: ret1=\"" + ret1.replace( '\r', '.'
).replace( '\n', '.' ) + "\", ret2=\"" + ret2.replace( '\r', '.' ).replace( '\n',
'.' ) + "\"" );
              }
```

```
        } catch ( Exception ex ) {
                Logger.log( "update", ex, true );
        }
}
```

### 11.1.6 Watchdog on a TC65 v2

In TC65 v2, there was a way to activate the watchdog and it is explained here, but it is officially supported by Cinterion. Here is the code:

```
private void configureHardwareWatchDog( final boolean active ) {
        ATCommand cmd = null;
        OutputStream os = null;
        InputStream is = null;
        try {
                cmd = new ATCommand( false );
                Logger.log( "configureHardwareWatchDog", cmd.send(
"AT^SSPI=0000,A000,A000\r" ) );
                os = cmd.getDataOutputStream();
                is = cmd.getDataInputStream();
                if ( active ) {
                        os.write( "<a200102050078001E>".getBytes() );
                } else {
                        os.write( "<a200102000078001E>".getBytes() );
                }
                os.flush();
                Thread.sleep( 50 );
                os.write( "<a200202>".getBytes() );
                os.flush();
                Thread.sleep( 50 );
                os.write( "#".getBytes() );
// Read the response.
                final byte[] buff = new byte[100];
                byte length = 0;
                int ch = is.read();
                while ( ch > 0 ) {
                        buff[length] = (byte) ch;
                        length++;
                        ch = is.read();
                }
                Logger.log( "configureHardwareWatchDog", new String( buff, 0, length )
);
        } catch ( final Throwable t ) {
                Logger.log( "configureHardwareWatchDog", "Unexpected error: " + t );
        } finally {
                if ( os != null ) {
                        try {
                                // According to the documentation, closing the output
                                // stream doesn't have any real effect.
                                // But this might change in the future, so closing it
                                // doesn't hurt.
                                os.close();
                                os = null;
                        } catch ( final IOException ignored ) {
                                // Ignore this exception.
                        }
                }
                if ( is != null ) {
                        try {
```

```
                        is.close();
                        is = null;
                    } catch ( final IOException ignored ) {
                        // Ignore this exception.
                    }
                }
                if ( cmd != null ) {
                    try {
                        // A data stream is closed only when the AT command is
                        // released.
                        cmd.release();
                        cmd = null;
                    } catch ( final ATCommandFailedException ignored ) {
                        // Ignore this exception.
                    }
                }
            }
}
```

## 11.2 M2MP specifications

### 11.2.1 Introduction

The M2MP is very simple client/server protocol; it solves some problems I had during some real-time projects I worked on.

It's a low-level protocol to encapsulate byte arrays value. Its main goal is to enable to send byte arrays and arrays of byte arrays on named channels. Once you have that basic layer you can build your own upper-level protocol very easily.

You have 4 types of messages:

- Identification : So that the server knows what chip he is talking too
- Acknowledge request and response : To check that the connection is still alive and that last data was received
- Byte array message on named channel
- Array of byte arrays message on named channel

### 11.2.2 Why is it working like that?

Google managed to create incredibly great applications with a new kind of database: Every data is stored through one key and unlimited number of columns. I think it's one of the challenges of all software architectures, keeping a simple basis that will allow you to evolve and scale with the growing number of users, projects (and devices).

The M2MP protocol is somehow quite similar to the bigdata paradigm, it's quite basic but allows you to do pretty much anything.

`String channelName, Byte[] value` or `String channelName, Byte[][] value`

With this basis, you can send any message.

Here are some samples of GPIO transmission.

#### 11.2.2.1 Low level (best one)

This is the best way to handle things; you can save bandwidth and easily manage your data.

To send that the GPIO 5 changed to 1, you could either send:

```
send( "gpio:10", new byte[] { 1 } ); //A
```

or

```
send( "gpio", new byte[] { 10, 1 } ); //B
```

or

```
send( "gpio", new byte[] { 5 + (1 << 7) } ); //C
```

#### 11.2.2.2 Simple text

```
send( "gpio:10", new byte[] { '1' } ); //D
```

or

```
send("gpio", new byte[][] { "10".getBytes(), "1".getBytes() }); //E
```

#### 11.2.2.3 Encapsulation

You can encapsulate any other format, like JSON. Instead of making JSON request and getting JSON response. You could have a bidirectional JSON protocol

```
void sendJson( JSONObject json ) {
	StringWriter sw = new StringWriter();
	JSONWriter writer = new JSONWriter(sw);
	this.toJSONWriter(writer);
	jsonStr = sw.toString();
	send("json", jsonStr.getBytes() );
}
```

You can do exactly the same thing with XML as you can guess.

#### 11.2.2.4 Relaying messages

Nothing prevents you from relaying all the data to another input stream, like a serial connection.

### 11.2.3 Specifications

These are the specifications:

**Identification**

The client sends its name:

```
[ 1 ] { 0x01 } // Identification request header
[ 1 ] { 0x05 } // Size of the identification
[ 5 ] { 't', 'r', '1', '2', '3' } // Identifier
```

The server replies "ok" or "not ok"

```
[ 1 ] { 0x01 } // Identification response
[ 1 ] { 0x01 } // 0x01 for OK, 0x00 for not ok
```

**Ping / Keep alive**

Client sends a ping request:

```
[ 1 ] { 0x02 } // client ping request header
[ 1 ] { 0x15 } // the number (here 0x15) can be incremented or random
```

Server answers the ping request:

```
[ 1 ] { 0x02 } // client ping server response header
[ 1 ] { 0x15 } // where 0x15 is the number of the ping request
```

Server sends a ping request:

```
[ 1 ] { 0x03 } // server ping request header
[ 1 ] { 0x16 } // the number (here 0x16) can be incremented or random
```

Client answers the ping request:

```
[ 1 ] { 0x03 } // server ping client response header
[ 1 ] { 0x16 } // where 0x16 is the 0x16 number of the ping request
```

**Defining a named channel**

*This applies to both client to server and server to client communication.*

```
[ 1 ] { 0x20 } // channel definition frame header
[ 1 ] { 0x0D } // where 13 is the size of the following message
[ 1 ] { 0x03 } // where 3 is the id of the channel
[ 12 ] { 'c', 'h', 'a', 'n', 'n', 'e', 'l', ' ', 'n', 'a', 'm', 'e' } // the name of the channel : "channel name"
```

**Sending some data**

*This applies to both client to server and server to client communication.*

For 0 to 254 sized messages:

```
[ 1 ] { 0x21 } // one byte sized data transmission frame header
[ 1 ] { 0x06 } // size of the data
[ 1 ] { 0x03 } // where 3 is the id of the channel
[ 5 ] { 0x01, 0x02, 0x03, 0x04, 0x05 } // data transmitted on the channel
```

For 255 to 65534 sized data messages:

```
[ 1 ] { 0x41 } // two bytes sized data transmission frame header
[ 2 ] { 0x00, 0x06 } // size of the data
[ 1 ] { 0x03 } // where 3 is the id of the channel
[ 5 ] { 0x01, 0x02, 0x03, 0x04, 0x05 } // data transmitted on the channel
```

For 65 535 octets to 4 294 967 294 sized data messages:

```
[ 1 ] { 0x61 } // four bytes sized data transmission frame header
[ 4 ] { 0x00, 0x00, 0x00, 0x06 } // size of the data
[ 1 ] { 0x03 } // where 3 is the id of the channel
```

```
4 [ 5 ] { 0x01, 0x02, 0x03, 0x04, 0x05 } // data transmitted on the channel
```

For data array transmission, it's the same except frame header are `0×22, 0×42, 0×62` instead of 0×21, 0×41, 0×61. If you transmit data array like this one `Byte[][] data = { { 0×01, 0×02 }, { 0×03, 0×04, 0×05 }, { 0×06, 0×07, 0×08, 0×09 } }`:

```
1 [ 1 ] { 0x22 } // one byte sized data transmission frame header
2 [ 1 ] { 0x0D } // size of the data
3 [ 1 ] { 0x03 } // where 3 is the id of the channel
4 [ 1 ] { 0x02 } // size of the first array
5 [ 2 ] { 0x01, 0x02 }
6 [ 1 ] { 0x03 } // size of the second array
7 [ 3 ] { 0x03, 0x04, 0x05 }
8 [ 1 ] { 0x04 } // size of the third array
9 [ 4 ] { 0x06, 0x07, 0x08, 0x09 }
```

For bigger arrays, you need to define:

```
1 [ 1 ] { 0x42 } // two bytes sized data transmission frame header
2 [ 1 ] { 0x10 } // size of the data (16)
3 [ 1 ] { 0x03 } // where 3 is the id of the channel
4 [ 2 ] { 0x00, 0x02 } // size of the first array
5 [ 2 ] { 0x01, 0x02 }
6 [ 2 ] { 0x00, 0x03 } // size of the second array
7 [ 3 ] { 0x03, 0x04, 0x05 }
8 [ 2 ] { 0x00, 0x04 } // size of the third array
9 [ 4 ] { 0x06, 0x07, 0x08, 0x09 }
```

And so on:

```
1 [ 1 ] { 0x62 } // four bytes sized data transmission frame header
2 [ 1 ] { 0x16 } // size of the data (22)
3 [ 1 ] { 0x03 } // where 3 is the id of the channel
4 [ 4 ] { 0x00, 0x00, 0x00, 0x02 } // size of the first array
5 [ 2 ] { 0x01, 0x02 }
6 [ 4 ] { 0x00, 0x00, 0x00, 0x03 } // size of the second array
7 [ 3 ] { 0x03, 0x04, 0x05 }
8 [ 4 ] { 0x00, 0x00, 0x00, 0x04 } // size of the third array
9 [ 4 ] { 0x06, 0x07, 0x08, 0x09 }
```

# 12 FAQ

I removed the FAQ from here. It's not really useful in a document anyway; it is mostly used by people searching on search engines.