2011

Roadstar

Programmer To Programmer instruction

v.3.2.

TABLE OF CONTENTS

CONNECTOR (LOOK FROM OUTSIDE)

UP

```
-----------------------
| [A] [9] [8] [7] [6] |
| [5] [4] [3] [2] [1] |
-----------------------
```

BOTTOM

PINOUT DESCRIPTION

Description	Direction	TC65i	Out Connector
VDC: +9V -- +32	POWER		1
GND	POWER		5
Digital Input 1	INPUT	GPIO_1	3
Digital Input 2	INPUT	GPIO_2	2
Digital Input 3	INPUT	GPIO_3	6
Digital Input 4/10(pulse)	INPUT	GPIO_4/GPIO_10	4
Digital Output 1	OUTPUT	GPIO_5	7
Digital Output 2	OUTPUT	GPIO_8	8
Analog Input 1	INPUT	ADC_1	9
Analog Input 2	INPUT	ADC_2	A

Internal Connections:

- Watch Dog Reset – TC65i GPIO_7
- GPS Reset – TC65i GPIO_6
- Led Diode output – TC65i GPIO_9

Device connecting instructions:

- Output Pins will give GROUND for outside relay when pins are active
- Digital Sensors are activated when positive voltage is connected
- Analog sensors are measuring positive voltage

Device programming instructions:

- Output is active when state is HIGH (1)
- Input is in state LOW (0) when positive voltage is connected (reversed)
- If watch dog counter is not reseted for 240 seconds it will reset TC65i

Programming MIDlet for Roadstar

```
      Make a sync AT Command class (good practice for synchronization)
public class SyncAtCommand extends ATCommand
{

      SyncAtCommand( boolean csdSupport ) throws ATCommandFailedException
      {
            super( csdSupport );
      }

      public    synchronized    String    send    (    String    ATCmd    )    throws
ATCommandFailedException
      {
            return super.send( ATCmd );
      }

      public    synchronized    String    sendRval    (String    ATCmd)    throws
ATCommandFailedException
      {
            String s = super.send(ATCmd);
            s = s.substring(2);
            s = s.substring(s.indexOf(':')+2,s.indexOf('\r'));
            return s;
      }
      }

      Have an instance of class in your "main"
      SyncAtCommand ATComm2;
      ATComm2 = new SyncAtCommand(false);
      ATComm2.send("ATE0\r");
```

Configure Pins (example):

```
      ATComm2.send("AT^SPIO=1\r");
      ATComm2.send("AT^SCPIN=1,0,0\r");//for input GPIO_1
      ATComm2.send("AT^SCPIN=1,1,0\r");//for input GPIO_2
      ATComm2.send("AT^SCPIN=1,2,0\r");//for input GPIO_3

      ATComm2.send("AT^SCPIN=1,4,1\r");//for output GPIO_5
      ATComm2.send("AT^SCPIN=1,7,1\r");//for output GPIO_8
```

Join in port if needed:

```
      INPORT = ATComm2.sendRval("at^scport=0,1,2\r");
                     InPortCheckCommand = "AT^SGIO=" + INPORT + '\r';
      Since hardware input is reversed, if needed ,use bitmask to get "normal"
      values:
      PinHardwareMask = 7; (111₂)

      PortValue = PortValue ^ PinHardwareMask;…
```

OutputPin Function(Example)

```
private void SetPinVal(int PinID, int val)
{

	try
	{
		String strRcv;
		strRcv = ATComm2.send("AT^SSIO=" + PinID + "," + val + "\r");
		if (strRcv.indexOf(OK) < 0) throw new ATCommandFailedException("Value
change failed");

	}
	catch(Exception ee)
	{

	}
}
```

Input PortReading

```
InPortCheckCommand = "AT^SGIO=" + INPORT + '\r';
private void CheckInPort()
	{
		try
		{
		INPORTVALUE=Integer.parseInt(ATComm2.sendRval(InPortCheckCommand));
		}
		catch(ATCommandFailedException ee)
		{

		}

	}
```

ADC Reading

Input is scaled to TC65i range so calculation factor is needed:

```
public static final double ADCFaktor = 11.44;
private static final String ADC1READCOMMAND = "AT^SRADC=0\r";
private void ObradiADC(int Koji)
	{
		try
		{
			String Response = ATComm2.send(ADC1READCOMMAND);
			Response = Response.substring(Response.indexOf(":"));
			Response = Response.substring(6);
			Response = Response.substring(0, Response.indexOf("\r"));
			ADC1 = (int)(Integer.parseInt(Response)*ADCFaktor);
		}
		catch (Exception ee)
		{
		}
	}
```

PulseCount Reading:

```
private void ActivatePulseCounter()
	{
		try
		{
			String Resp = ATComm2.send("AT^SCCNT=1,0\r");
			if (Resp.indexOf("OK") >= 0)
			{
				//////System.out.println("Pulse Count OK");
			}
			else
			{
			}

		}
		catch (Exception ee)
		{
		}

	}
	private void ResetPulse()
	{
		try
		{
			String Resp = ATComm2.send("AT^SSCNT=0\r");
			if (Resp.indexOf("OK") >= 0)
			{
				//////System.out.println("Pulse reset OK");
			}
			else
			{
			}

		}
		catch (Exception ee)
		{
		}

	}
private static final String PULSECHECKCMD="AT^SSCNT=2\r";
ATComm2.send(PULSECHECKCMD);//Send pulse read command and receive answer thru
ATCommand Listener


		ATList Listener = new ATList();
		ATComm2.addListener(Listener);

class ATList implements ATCommandListener
	{
		public void ATEvent(String Event)
		{
			try
			{
```

```
                    if (Event.indexOf("SSCNT") >= 0)
                    {

Event = Event.substring(Event.indexOf(":") + 2, Event.indexOf(":") + 12);
PulseCount = Integer.parseInt(Event); }



                }
                catch (Exception e)
                {
                                    }

        }
        public void RINGChanged(boolean SignalState) { }
        public void DCDChanged(boolean SignalState) { }
        public void DSRChanged(boolean SignalState) { }
        public void CONNChanged(boolean signalState) { }


                                                }
```

GPS Reading:

On beginning reset GPS:

```
            SetPinVal(5, 1);
            Thread.sleep(100);
            SetPinVal(5, 0);
```

Open Serial port:

```
CommConnection  commConn;
InputStream     inStream;
OutputStream    outStream;
String strCOM = "comm:com1;blocking=off;baudrate=9600";
                  commConn = (CommConnection)Connector.open(strCOM);
                  inStream  = commConn.openInputStream();
                  outStream = commConn.openOutputStream();
```

Write uBlox start command

```
byte[]  GPSCommandBtCommandHOT  =  new  byte[]{(byte)0xB5,  (byte)0x62,  (byte)0x06,
(byte)0x04,  (byte)0x04,  (byte)0x00,  (byte)0x00,  (byte)0x00,  (byte)0x02,  (byte)0x00,
(byte)0x10,  (byte)0x68};
outStream.write(GPSCommandBtCommandHOT, 0, GPSCommandBtCommandHOT.length);
```

Read NMEA data:

```
byte nmea[] = new byte[1000];

int read = inStream.read(nmea);
… Parse data…
```

Reset WatchDog:

```
public void RestartWatchDog()
	{
		try
		{
			for(int i =0;i<2; i++)
			{
				SetPinVal (6,1);
				Thread.sleep(1);
				SetPinVal(6,0);
				Thread.sleep(1);
			}
		}
		catch (Exception ee)
		{
		}
	}
```

GPRS Sending:

```
SocketConnection commConn;
InputStream inStream;
OutputStream outStream;
String connProfile = "bearer_type=gprs;access_point="
                              + APNString + ";username="
                              + GPRSUserString + ";password="
                              + GPRSPasswordString + ";timeout=60";


String serv = "socket://" + ServerIPString + ":"
                              + ServerPortString + ";" + connProfile;

commConn = (SocketConnection) Connector.open(serv,
                              Connector.READ_WRITE, true);

outStream = commConn.openOutputStream();
inStream = commConn.openInputStream();

Send and Receive data using streams…
```

Important tips:

TC65i Restart:

```
public void restartMod() {
            try {

                  String strRcv;
                  strRcv = ATComm1.send("AT+CFUN=1,1\r");
                  if (strRcv.indexOf("OK") < 0)
                        throw new ATCommandFailedException("Couldn't restart");
                  else
                  Thread.sleep(10000);

            } catch (Exception e) {

                                    }
      }
```

Network Params:

If have problems connecting make a routine to check network params
CREG
If CREG==0 you should probably restart TC65i using AT+CFUN=1,1\r command

ATComandListener:

In time to time check is you ATCommandListener working properly by sending PulseRead command and verify receiving answer. It is a rare problem and in most cases caused by implementation of lot of code inside listener. If listener stop working, restart TC65i.

GarbageColector:

From time to time activate garbage collector to free up memory:

```
Runtime.getRuntime().gc();
```

-