maXbox Starter 19

Start with WinCOM / Arduino Programming

1.1 Embedded Computing and a Spreadsheet

This tutorial illustrates what the WinCOM (Component Object Model) interface level does and what you can do to make sure your code works with an Excel spreadsheet.
In the first part of this tutorial we export data to a spreadsheet by using WinCOM (not the same as COM for the RS232 Serial interface, so I stress the name WinCOM).
In the second part we control a DC-motor with Arduino and deliver the data to the excel file mentioned above. Hope you did already work with the Starter 1 till 18 (especially the 18 with Arduino) at:

http://sourceforge.net/apps/mediawiki/maxbox/

Arduino hardware is programmed using a Wiring-based language (syntax and libraries), similar to C++ and Object Pascal with some slight simplifications and modifications. The Arduino is what is known as a Physical or Embedded Computing platform, which means that it is an interactive system that through the use of hardware and software can interact with its environment.
In VCL applications as maXbox use, interfaces are a fundamental element in the COM, SOAP, and CORBA distributed object models. Delphi provides base classes for these technologies that extend the basic interface functionality in *TInterfacedObject*, which simply implements the *IInterface* interface methods.
COM classes add functionality for using class factories and class identifiers (CLSIDs).
You simply can create such an ID with `CreateClassID`:

```
    writeln(CreateClassID)
>> {59D2BE9B-5C47-42AC-8265-C9C5FA6B1830}
```

The big advantage is that we can call the global `CreateOleObject` function, passing in the GUID for the `CoClass` (there is a constant for this GUID[1] defined at the top of the _TLB unit).
`CreateOleObject` returns an `IDispatch` pointer (var) for the default interface to an `OLEVariant`.

---
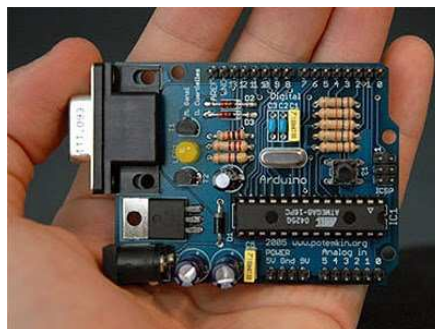
[1] A CLSID is simply a GUID that identifies a COM object.

Let's begin with WinCOM (or DCOM for distributed) that stands for Component Object Model (COM) from Microsoft. We also can read Common Object Model.
COM is a language-independent (but NOT platform independent) software component model that enables interaction between software components and applications running on a Win platform. The key aspect of COM is that it enables communication between components, between applications, and between clients and servers through clearly defined interfaces. Interfaces provide a way for clients to ask a COM component which features it supports at runtime.

When COM or an interface application is not available, like Excel, you get an Exception: invalid Class String or an `Interface not supported` error.

1.2 Out of the Box Export

As you already know the tool is split up into the toolbar across the top, the editor or code part in the centre and the output window at the bottom or the interface part on the right. Change that in the menu `/View` at our own style.

In maXbox you will control the Arduino motor as a script, so the Arduino program IS the script that starts the motor, configuration and the electronics too.

Before this starter code will work you will need to download maXbox from the website. It can be down-loaded from http://www.softwareschule.ch/maxbox.htm (you'll find the download maxbox3.zip on the top left of the page). Once the download has finished, unzip the file, making sure that you preserve the folder structure as it is. If you double-click `maxbox3.exe` the box opens a default demo program. Test it with F9 / F2 or press **Compile** and you should hear a sound. So far so good now we'll open the examples:

```
318_excel_export3.TXT
299_animationmotor_arduino.txt //if you have an arduino
```

If you can't find the two files try also the zip-file loaded from:
http://www.softwareschule.ch/examples/318_excel_export3.txt

Now let's take a look at the code of this fist part project. Our first line is

```
01 program Excel_maXcel_Export3;
```

We have to name the same, means the program's name is above.

This example requires two objects from the classes: `TStringGrid` and `OleObject` of `CoClass` so the second one is to return an `OleVariant`.
Therefore another way to use dispatch interfaces is to assign them to a Variant. By assigning the interface returned by `CreateOleObject` to a Variant, you can take advantage of the Variant type's built-in support for interfaces. Simply call the methods of the interface, and the Variant automatically handles all `IDispatch` calls, fetching the dispatch ID and invoking the appropriate method.

```
39  XLApp:= CreateOLEObject('Excel.Application');
```

First we start with the dimension of the `StringGrid`.
After creating the object in line 107 we use properties to set the dimension to 11 * 11. All the strings and their associated objects for a particular column can be accessed using the cols property. The rows property gives you access to all the strings and their associated objects for a particular row.

```
107 mySt:= TStringGrid.Create(self)
108  with mySt do begin
109    rowcount:= 11;
110    colcount:= 11;
```

So the class `TStringGrid` has some methods and properties like `Cells` you can find also in an Ole Excel object. All the strings within a string grid are contained in the Cells property, which you can use to access a particular string within the grid, like a spreadsheet.

The Cells property is an array of strings, one string for each cell in the grid. Use the Cells property to access a string within a particular cell. `ACol` is the column coordinates of the cell, and `ARow` is the row coordinates of the cell. The first row is row zero, and the first column is column zero

```
for k:= 1 to ColCount - 1 do
  for t:= 1 to RowCount - 1 do
    Cells[k,t]:= intToStr(t*k);    //one to one
```

Next we call our function `SaveAsExcelFile2()` or `SaveAsExcelFileDirect` which has no temporary data structure. We pass the string grid to the function as an object, a map name of the worksheet and the name of the excel file too. The worksheet name belongs to Excel and will deliver through its interface as a COM-object. The same goes for other methods. But how does the COM-object knows which methods are available?
In COM, this mechanism is the type library. Type libraries are a binary, programming language-neutral way for a COM object to expose type metadata at runtime. Because type libraries are opened and parsed by system APIs, languages such as Delphi can import them and gain the advantages of `vtable` binding, even if the component was written in a different programming language.
Sounds difficult - so it is, but the use is easy enough.

☞Because you can access a COM component through the so called "late binding", creating a type library for the component is not strictly required.
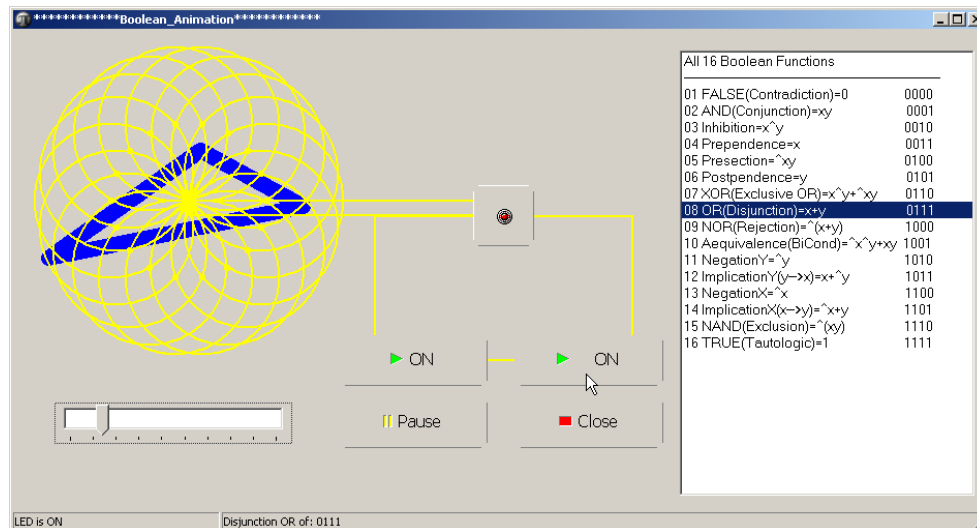
1: The GUI of the Motor App

Clients are restricted to late binding when accessing such a class. The `AutoDual` value causes all type information (including `dispids`) to be included for a class so marked.
So again we create in line 39 our object to get access to an Excel sheet.
A typical WinCOM client session looks like this:

```
39  XLApp:= CreateOleObject('Excel.Application');
40  try
41    XLApp.Visible:= true;
42    // Add Workbook
43  XLApp.Workbooks.Add(xlWBatWorkSheet);
44    Sheet:= XLApp.Workbooks[1].WorkSheets[1];
45    Sheet.Name:= aSheetName;
46 //sheet is another OLE object
    end;
```

As I said, by assigning the interface returned by this `CreateOleObject` to a Variant, we can take advantage of the Variant type's built-in support for interfaces and their methods too.

One of those methods is the `Add` or the `SaveAs()` method:

```
43   XLApp.Workbooks.Add(xlWBatWorkSheet);
```

☞OLE means object linking and embedding and is used for example to select a location in a worksheet and choose Paste Format from an Edit menu of Excel to embed the OLE object in the worksheet (aka `OleContainer`).
Let's have a look at the interface and the magic behind:

```
function CreateOleObject(const ClassName: String): IDispatch;
```

What's about this `ClassName`, it must be a global identifier.
The compiler attaches a GUID to a class or structure declared or defined (full COM object definitions only) with the `uuid` attribute. The `uuid` attribute takes a string as its argument. This string names a GUID in normal registry format with or without the { } delimiters. For example the following test case:

```
Var
  aG: TGUID;
  aG:= ProgIDtoClassID('Excel.Application');

//function GUIDToString(const GUID: TGUID): string)
```

```
  Writeln(GUIDToString(aG));

  Writeln(ClassIDToProgID(aG));
```

The output is indeed the GUID of our Excel Application!:
```
{00024500-0000-0000-C000-000000000046}
Excel.Application.11
```

The string names an existing GUID in normal registry format so you can find them in the registry. This attribute can be applied in a redeclaration. This allows the system headers to supply the definitions of interfaces such as `IUnknown`, and the redeclaration in some other header (such as COMDEF.H) to supply the GUID (is a 128-bit randomly generated number).

☝In facts there are 2 programming aspects used in WinCOM applications. COM is both a specification and an implementation. The COM specification defines how objects are created and how they communicate with each other. According to this specification, COM objects can be written in different languages, run in different process spaces and on different versions.
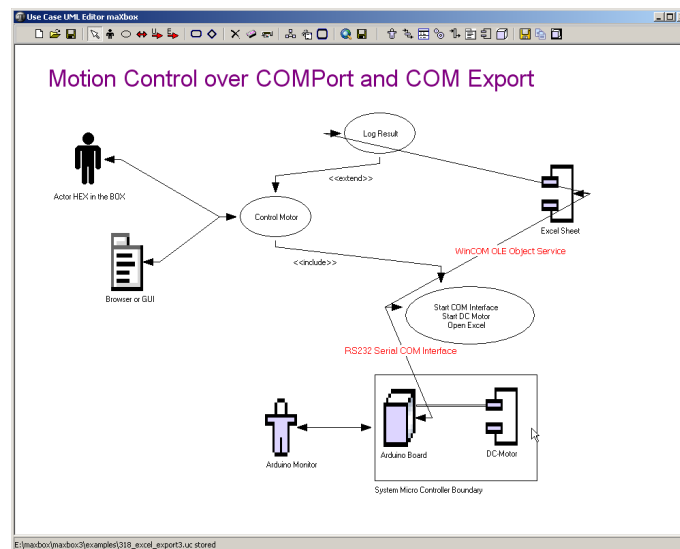
2: The Use Case of the App

So let's get back to our sheet create and iterate in line 48. In line 48 till 57 you see the assignment of to Array data to the cells of the sheet: the worksheet itself is a global `const` in line 11, instead of this you can also set another sheet type or name as a parameter.

```
10 Const
11   xlWBATWorksheet = -4167;
```

In line 53 we save the sheet in a first workbook:

```
48 for i:= 0 to AGrid.ColCount do
49   for j:= 0 to AGrid.RowCount do
50     XLApp.Cells[i+1,j+1]:= aData2[i][j];
51 // Save Excel Worksheet
52 try
53   XLApp.Workbooks[1].SaveAs(AFileName);
54   Result:= True;
55 except
56   Msg('maXcel export error'); // Error ?
57 end;
```

☞ **COM names** implementation is built into the Win32 subsystem, which provides a number of core services that support the written specification. The COM library contains a set of standard interfaces that define the core functionality of a COM object, and a small set of API functions designed for the purpose of creating and managing COM objects.
A COM object provides an interface for each set of related methods and properties.

☝ Note that COM properties are not identical to properties on VCL objects. COM properties always use read and write access methods with different names.
As COM has evolved, it has been extended beyond the basic COM services. COM serves as the basis for other technologies such as Automation, ActiveX controls, and Active Directories.

📖 So far we have learned little about WinCOM and GUID or class names. Now it's time to run your program at first with F9 (if you haven't done yet) and learn something about the Excel Export.

📖 Applications can access the interfaces of COM components that exist on the same computer as the application or that exist on another computer on the network using a mechanism called Distributed COM (DCOM).

There is a second version of the function called `SaveAsExcelFileDirect()` calls the cells without a temporary structure of an dynamic array direct from string grid to cells:

```
82    for i:= 0 to AGrid.ColCount -1 do
83      for j:= 0 to AGrid.RowCount -1 do
84        XLApp.Cells[i+2,j+1]:= AGrid.Cells[i,j]; //direct fill
```

One word concerning fixed rows: Each grid must have a least one row that isn't fixed. In other words, the value of the `FixedRows` property must always be at least one less than the value of the `RowCount` property, which contains the number of rows in the grid.
The `FixedRows` property determines the number of nonscrolling rows within a grid. The default value is `1`. Nonscrolling rows remain fixed at top of the grid, even when you scroll other rows. The same goes for a spread sheet!
If you want to change the width of a single column within a grid without changing other columns, use the `ColWidths` property during run time. If you change the `DefaultColWidth` property value after changing the width of specified columns (ColWidths[2]:= 10;), all the columns become the height specified in the `DefaultColWidth` property once again.
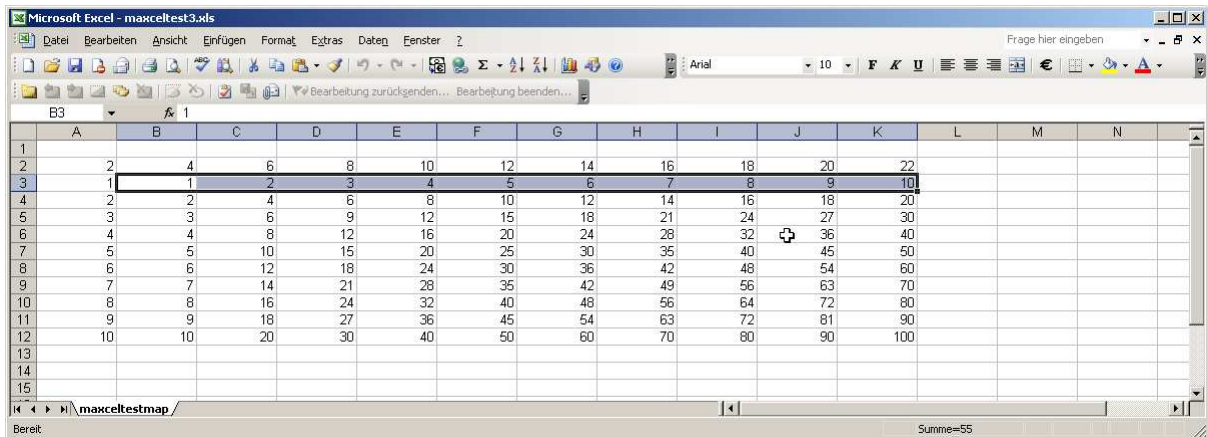
3: The Output Export

Next we enter part two of the app, the motion control with Arduino or maXboxMotor.

```
299_animationmotor_arduino.txt //if you have an Arduino
```

When the COM port finishes with initializations, it starts the DC-motor and sends some velocity data back which we export to Excel. That's the Use Case.

```
184 cPort.WriteStr(inttostr(tb.Position));
185  printF('motor out %d', [tb.Position]);
186  //cport.readStr(aout,length(aout));
187  cPort.readStr(aout,3);
188  writeln('ardu back to maxcel '+aout);
```

Have you tried the program, it's also possible to test the app without Arduino or Excel. The **Compile** button is also used to check that your code is correct, by verifying the syntax before the program starts. Another way to check the syntax before run is F2 or the **Syntax Check** in the menu Program.

Imagine we want to upload that produced excel sheet like control data to enable download from another machine or client.
Of course a lot of lines to get a file from the web try it shorter with the top function `wGet()`:

```
wGet('http://www.softwareschule.ch/download/maxboxmotor.xls','mytestmotor.xls');
```

It downloads the entire file into memory if the data is compressed (Indy does not support streaming decompression for HTTP yet). Next we come closer to the COM configuration.
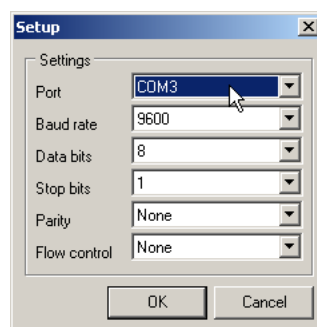
4: COM Port Settings

1.3 Serial Line Motor Drive

Please read more about serial coding in Tutorial 15 and Arduino in Tutorial 17! The serial communications line is simply a way for the Arduino to communicate with the outside world, in this

6

case to and from the PC (via USB) and the Arduino IDE's Serial Monitor or from the uploaded code to I/O Board back. We just create and configure our COM Settings (depends in which COM Port the USB Hub works) or try it with the app in picture 4 on the button "Settings":

```
procedure TForm1_FormCreateCom(Sender: TObject);
begin
  cPort:= TComPort.Create(self);
  with cPort do begin
    BaudRate:= br9600;
    Port:= COMPORT; //'COM5';
    Parity.Bits:= prNone;
    StopBits:= sbOneStopBit;
    DataBits:= dbEight;
  end;
```

The Arduino can be used to develop stand-alone interactive objects or it can be connected to a computer to retrieve or send data to the Arduino and then act on that data (e.g. Send sensor data out to the web or write data on a control motor or LED).
Now we change the code site to the Arduino Editor (see picture 6 below) to explain how he handles our commands (chars).
`Serial.begin` tells the Arduino to start serial communications and the number within the parenthesis, in this case 9600, sets the baud rate (characters per second) that the serial line will communicate or serve at.

```
int motorPin = 9; // Pin of the Motor
int val = 0;      //init the motor

char incoming[4] = {}; //include end sign

void setup() {

    Serial.begin(9600);        // connect to the serial port

    pinMode(motorPin, OUTPUT); //declare the motor's pin as output
}
```

In the main loop we have an "if and while statement". The condition it is checking the value in (`Serial.read`). The `Serial.available` command checks to see if any characters have been sent down the serial line and if the value is more than zero.
With `memset` you fill a block of memory, in our case we must reset with zeroes to prevent this fault:

```
60      //set 60
70      //set 70
120     //set 120
600     //set 60 …we get 600 oops, too fast with 600
```

```
memset( addressToSet, valueToSet, sizeof(howManyBytes));
```

The `sizeof`(array) returns the size of your array in bytes – since it automatically recalculates when compiled, you never have to change it if you change your array size.
If any characters have been received then the condition is met and the code within the "if statements" code block is now executed in a while loop, you see the motor running and with the track bar in maXbox (see picture 1) you can influence the speed of motor (sound)!
The condition it is checking is simply a Char [4] in an array including the null terminated sign☺.

```
void loop () {
  int i = 0;
  if (Serial.available() > 0) {
```

```
    //memset, otherwise zeros as 0 stand by: → 60 will be 600
    memset(incoming, 0, sizeof(incoming));
    while (Serial.available() > 0 && i < sizeof(incoming) - 1) {
      incoming[i] = Serial.read();
      i++;
      delay(3);
    }


  analogWrite(motorPin, val);

//in comparison with a LED that is a digitalWrite()

val = Serial.read();       // read the serial port
  if (val !=-1){
    if (val=='1'){
      digitalWrite(ledPin1,HIGH);
    }
    else if (val=='A'){
      digitalWrite(ledPin1,LOW);
      }
```

`Serial.print`("Data entered: "); and this is our way of sending data back from the Arduino to the PC. In this case the print command sends whatever is within the parenthesis to the PC, via the USB cable, where we can read it in the Serial Monitor window or in maXbox.

☞ With `digitalWrite`() a LED or motor is driven from GPIO Output scheme, but with `analogWrite`() it is from PWM Output scheme:

analogWrite*(pinNumber, value);*
digitalWrite*(pinNumber, value);*

They produce different current on `hte` output pin.
All of the electrical signals that the Arduino works with are either analog or Digital. PWM stands for Pulse-Width Modulation, a method of emulating an analog signal through a digital pin. It is a value between or including 0 and 255. So the Arduino has the capability to output a digital signal that acts as an analog signal, this signal is called Pulse Width Modulation (PWM).
On the Arduino UNO PWM pins are signified by a ~ sign.

digitalWrite: Assign a HIGH or LOW value to a pin already declared as an output.
analogWrite: Assign a value **between** or including 0 (LOW) and 255 (HIGH). This allows you to set output to a PWM value instead of just HIGH or LOW.

In Line 245 we find a last function of the RTL (Runtime Library) of Indy:

```
83  Writeln(DateTimeToInternetStr(Now, true))
```

We get the real time zone based time back! This information of RTL functions is contained in various unit files that are a standard part of Delphi or Indy. This collection of units is referred to as the RTL (run time library). The RTL contains a very large number of functions and procedures for you to use.

By the way: In my research and debugging, I found that the function `GetTimeZoneInformation` was returning a value oriented towards converting a Time from GMT to Local Time. We wanted to do the reverse for getting the difference.

The issue with `TIdMessage.UseNowForTime` = False bug was that the `TIdMessage` was calling Borland's date function instead of using the Date property, see Appendix.

1.4 FrameWorkFlow

At last but not least some words about sockets and streams in order to prepare to control the motor from a browser over the internet. A listening server socket component automatically accepts client connection requests when they are received. You receive notification every time this occurs in an `OnCommandGet` event.

Server connections are formed by server sockets when a listening socket accepts a client request. A description of the server socket that completes the connection to the client is sent to the client when the server accepts the connection. The connection is then established when the client socket receives this description and completes the connection.

Socket connections can be divided into three basic types, which reflect how the connection was initiated and what the local socket is connected to. These are

- Client connections.
- Listening connections.
- Server connections.

Once the connection to a client socket is completed, the server connection is indistinguishable from a client connection. Both end points have the same capabilities and receive the same types of events. Only the listening connection is fundamentally different, as it has only a single endpoint.

Sockets provide the interface between your network server or client application and the networking software. You must provide the interface between your application and the clients that use it. You can copy the API of a standard third party server (such as Apache), or you can design and publish your own API.
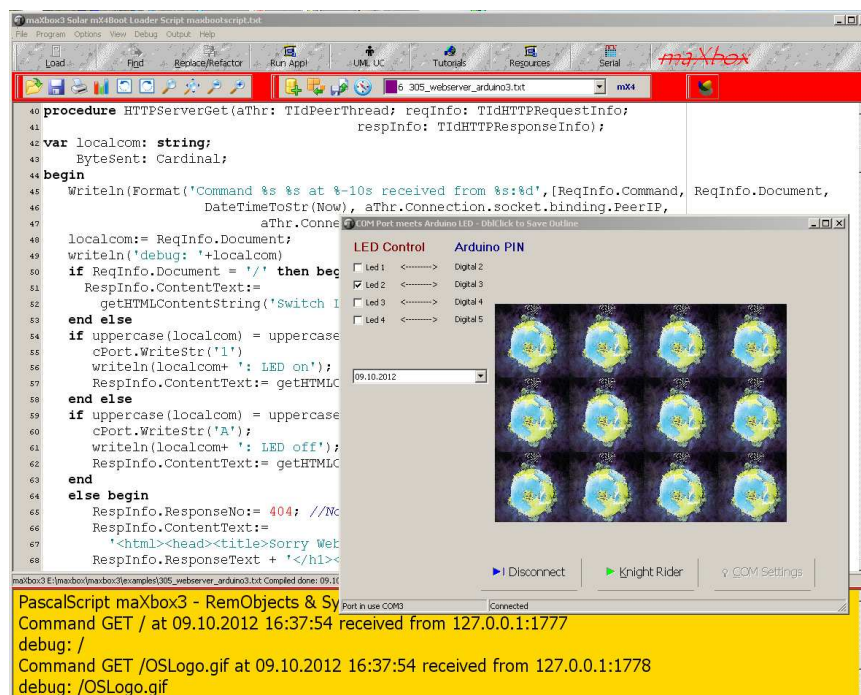
5: the BOX and the GUI

Sockets let your network application communicate with other systems over the network. Each socket can be viewed as an endpoint in a network connection. It has an address that specifies:

- The system on which it is running.
- The types of interfaces it understands.
- The port it is using for the connection.

A full description of a socket connection includes the addresses of the sockets on both ends of the connection. You can describe the address of each socket endpoint by supplying both the IP address or host and the port number, not the same as the serial port.

6: Arduino Editor with Serial Monitor

Many of the protocols that control activity on the Internet are defined in Request for Comment (RFC) documents that are created, updated, and maintained by the Internet Engineering Task Force (IETF), the protocol engineering and development arm of the Internet. There are several important RFCs that you will find useful when writing Internet applications:

- RFC822, "Standard for the format of ARPA Internet text messages," describes the structure and content of message headers.
- RFC1521, "MIME (Multipurpose Internet Mail Extensions) Part One: Mechanisms for Specifying and Describing the Format of Internet Message Bodies," describes the method used to encapsulate and transport multipart and multiformat messages.
- RFC1945, "Hypertext Transfer Protocol—HTTP/1.0," describes a transfer mechanism used to distribute collaborative hypermedia documents.

In this line we just start a browser to test our connection in a so called frame work flow ☺

```
31 procedure letOpenBrowser;
32 // TS_ShellExecuteCmd = (seCmdOpen,seCmdPrint,seCmdExplore);
33 begin
34 //ShellAPI.ShellExecute(Handle,PChar('open'),'http://127.0.0.1:80/',Nil,Nil,0);
35   S_ShellExecute('http:'+IPADDR+':'+IntToStr(APORT)+'/','',seCmdOpen)
36 end;
```
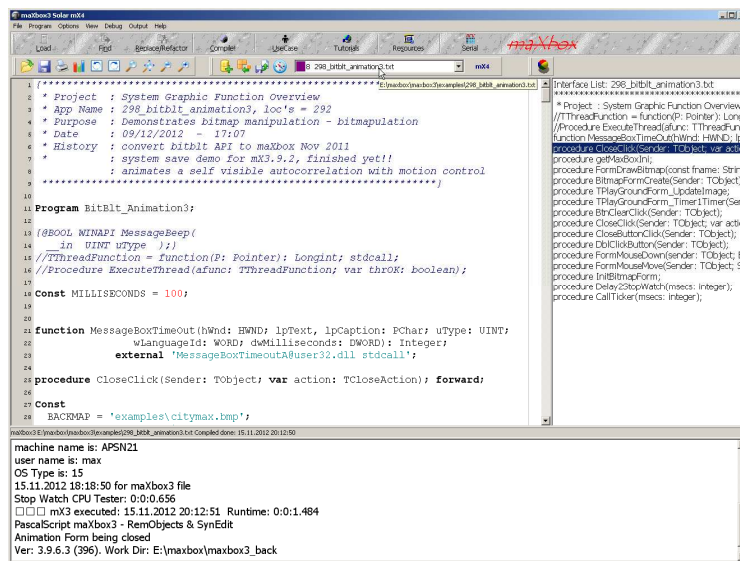
⌨ ♏ Try to get data back from Arduino as a test case to store in an excel sheet:

```
Serial.print() in Arduino and cPort.ReadStr() in maXbox

Function ReadStr( var Str: string; Count: Integer): Integer');  //CPort Lib

//S_ShellExecute('http:'+IPADDR+':'+IntToStr(APORT)+'/soa_delphi.pdf','',seCmdOpen)
```

Next *Tutorial Nr. 20* shows the topic "Coding Regular Expressions" in many cases.
The Arduino board is made of an Atmel AVR Microprocessor, a crystal or oscillator (basically a crude clock that sends time pulses to the microcontroller to enable it to operate at the correct what type of Arduino you have, you may also have a USB connector to enable it to be connected to a PC or Linux to upload or retrieve data. The board exposes the microcontrollers I/O (Input/Output) pins to enable you to connect those pins to other circuits, buses or to sensors, etc.

Feedback @
max@kleiner.com

Literature:
Kleiner et al., Patterns konkret, 2003, Software & Support

Links of maXbox and Arduino:

http://www.softwareschule.ch/maxbox.htm

http://www.ecotronics.ch/

http://en.wikipedia.org/wiki/Arduino

http://sourceforge.net/projects/maxbox

http://sourceforge.net/apps/mediawiki/maxbox/

http://sourceforge.net/projects/delphiwebstart

http://ecotronics.ch.honorius.sui-inter.net/wordpress/2012/motor-uber-serielle-schnittstelle-steuern/#

1.5 Appendix

```
Function DateTimeToInternetStr(const Value: TDateTime): String;
var
  strOldFormat, strOldTFormat,
  strDate: String;
  wDay,
  wMonth,
  wYear:WORD;
begin
  {needed to prevent wild results}
  Result := '';
  strOldFormat := ShortDateFormat ;

  ShortDateFormat := 'DD.MM.YYYY';

  // Date
  case DayOfWeek(Value) of
    1: strDate := 'Sun, ';
    2: strDate := 'Mon, ';
    3: strDate := 'Tue, ';
    4: strDate := 'Wed, ';
    5: strDate := 'Thu, ';
    6: strDate := 'Fri, ';
    7: strDate := 'Sat, ';
  end;
  DecodeDate(Value, wYear, wMonth, wDay);
  strDate := strDate + IntToStr(wDay) + #32;
  case wMonth of
     1: strDate := strDate + 'Jan ';
     2: strDate := strDate + 'Feb ';
     3: strDate := strDate + 'Mar ';
     4: strDate := strDate + 'Apr ';
     5: strDate := strDate + 'May ';
     6: strDate := strDate + 'Jun ';
     7: strDate := strDate + 'Jul ';
     8: strDate := strDate + 'Aug ';
     9: strDate := strDate + 'Sep ';
    10: strDate := strDate + 'Oct ';
    11: strDate := strDate + 'Nov ';
    12: strDate := strDate + 'Dec ';
  end;
  //Correction
  strOldTFormat := LongTimeFormat;
  LongTimeFormat := 'HH:NN:SS';
  strDate := strDate + IntToStr(wYear) + #32 + TimeToStr(Value);
  Result := strDate + #32 + DateTimeToGmtOffSetStr(OffsetFromUTC,False);
  LongTimeFormat := strOldTFormat;
{
  strOldTFormat := LongDateFormat;
  LongDateFormat := 'HH:NN:SS';
  strDate := strDate + IntToStr(wYear) + #32 + TimeToStr(Value);
  LongDateFormat := strOldTFormat;
  Result := strDate + #32 + DateTimeToGmtOffSetStr(OffsetFromUTC,False);
  ShortDateFormat := strOldFormat ;
}
end;
```

1.6 Appendix Arduino Code Motor Drive

Hier nun also der vollständige Code:

```
1
2
3
4  /*
5   * Motorengeschwindigkeit über serielle Schnittstelle regeln
6   * Author: Silvia Rothen, rothen ecotronics, Bern, Switzerland
7   */
8  int motorPin = 9;  // an diesem Pin hängt der Motor
9  int val = 0; //zuerst steht der Motor
10 char incoming[4] = {}; //wegen Endzeichen
11
12 void setup() {
13     Serial.begin(9600);          // connect to the serial port
14     pinMode(motorPin, OUTPUT);
15 }
16 void loop() {
17
18   int i = 0;
19
20   if (Serial.available() > 0) {
21     //sonst bleiben die 0 erhalten 60 -> 600
22     memset(incoming, 0, sizeof(incoming));
23     while (Serial.available() > 0 && i < sizeof(incoming) - 1) {
24       incoming[i] = Serial.read();
25       i++;
26       delay(3);
27     }
28
29     //array of char in int wandeln
30     val = atoi(incoming);
31     //Geschwindigkeit limitieren
32     if (val < 0) {
33       val = 0;
34     } else if (val > 255) {
35       val = 255;
36     }
37
38     Serial.print("Umlaufgeschwindigkeit ");
39     Serial.println(val);
40     //Motorgeschwindigkeit setzen
41     analogWrite(motorPin, val);
42   }
   }
```

//***************************************Arduino Code Finished*********************

1.7 Appendix Arduino App

1.8 Appendix Arduino Motor Layout