Pira.cz

FMIO.DLL – Programming Guide

What's the FMIO.DLL?

The FMIO.DLL is a dynamic-link library that enables easy access to the P75/P175/P275 FM analyzer from user application. The user does not need to solve how to access the communication port or how to handle I/O buffering in his application.

The DLL is originally created in Borland Delphi but should work with any programming language due to use of *stdcall* convention and simple format of data exchange. Please follow the documentation of your programming language for more details about how to use external functions and procedures from DLL.

FMIO.DLL features:

- Basic and system (Connect, Disconnect, Version, …)
- Write control and query commands to the port
- Read the values measured
- Scope and FFT data
- Support for multiple devices

List of functions and procedures provided by the DLL

```
function Version: PChar; stdcall;
procedure ConnectionSetup(ConType: boolean; PortNum: integer; Host: PChar;
  Port: integer); stdcall;
function Connect: boolean; stdcall;
procedure Disconnect; stdcall;
procedure WriteCommand(Command: PChar); stdcall;
function WriteBufferLength: integer; stdcall;
function ReadBufferLength: integer; stdcall;
function ReadResponse: Pchar; stdcall;
function ReadValue(Key: PChar): PChar; stdcall;
procedure ClearValues; stdcall;
function GetArray(Key: PChar): TGetArray; stdcall;
function Connected: boolean; stdcall;
function PCharToFloat(Source: PChar): single; stdcall;
procedure SetTCPReconnectTimeout(Timeout: integer); stdcall;
```

Function or procedure	**Description**
Version	Returns actual DLL version.
ConnectionSetup	Specifies connection parameters. ConType: false (RS232) or true (TCP/IP) PortNum: RS232 (COM) port number. Used if ConType=false. Host, Port: TCP/IP connection parameters. Used if ConType=true.
Connect	Connects to the device using the connection parameters above. Returns true if connected successfully.
Disconnect	Disconnects from the device. Call also before repeated connects.
WriteCommand	Sends a command to the device. If already sending previous command, new commands are temporarily placed in write buffer and sent automatically after processing previous commands. For complete list of commands see the P75/P175 or P275 technical manual.
WriteBufferLength	Returns number of commands waiting in the write buffer.
ReadBufferLength	Returns number of responses waiting in the read buffer.
ReadResponse	Returns the responses from the read buffer in the same order as received from the device. Call repeatedly until ReadBufferLength is zero. Data format: Key+'='+Value
ReadValue	Returns last known value for specified key. If no value is assigned, it returns empty string. For complete list of keys see the P75/P175 or P275 technical manual.
ClearValues	Clears all values read from the device.
GetArray	Returns an array of scope or FFT data depending on the key specified. The commands ?I and 1?W must be written before. List of keys: MPXScope – MPX scope data (vertical: kHz of deviation, horizontal: 0.002 ms step) MPXFFT – MPX FFT data (vertical: dB, horizontal: 0.732422 kHz step) IFFFT – Carrier FFT data (vertical: dB, horizontal: 1.465 kHz step, 0 kHz on element index 118) LeftScope – Left audio scope data (vertical: kHz of deviation, horizontal: 0.006667 ms step) RightScope – Right audio scope data (vertical: kHz of deviation, horizontal: 0.006667 ms step) Data format:

	type TGetArray=record Length: Word; Values: array[0..1023] of single; end;
Connected	Returns actual connection state.
PCharToFloat	Converts a null-terminated string into a floating point data type. Can be used for direct conversion of a value returned by the device. Leading and trailing blanks as well as the units are ignored.
SetTCPReconnectTimeout	Specifies a delay in seconds between two attempts to connect via TCP/IP in case that the connection has not been successful. Default timeout is 32 seconds.

Notes:

PChar – pointer to a null-terminated string.
Word – unsigned 16-bit.
Integer – signed 32-bit.
Single – 4 bytes long real type (floating point).
Boolean – true/false variable that occupies one byte of memory.
The Keys are case-sensitive!

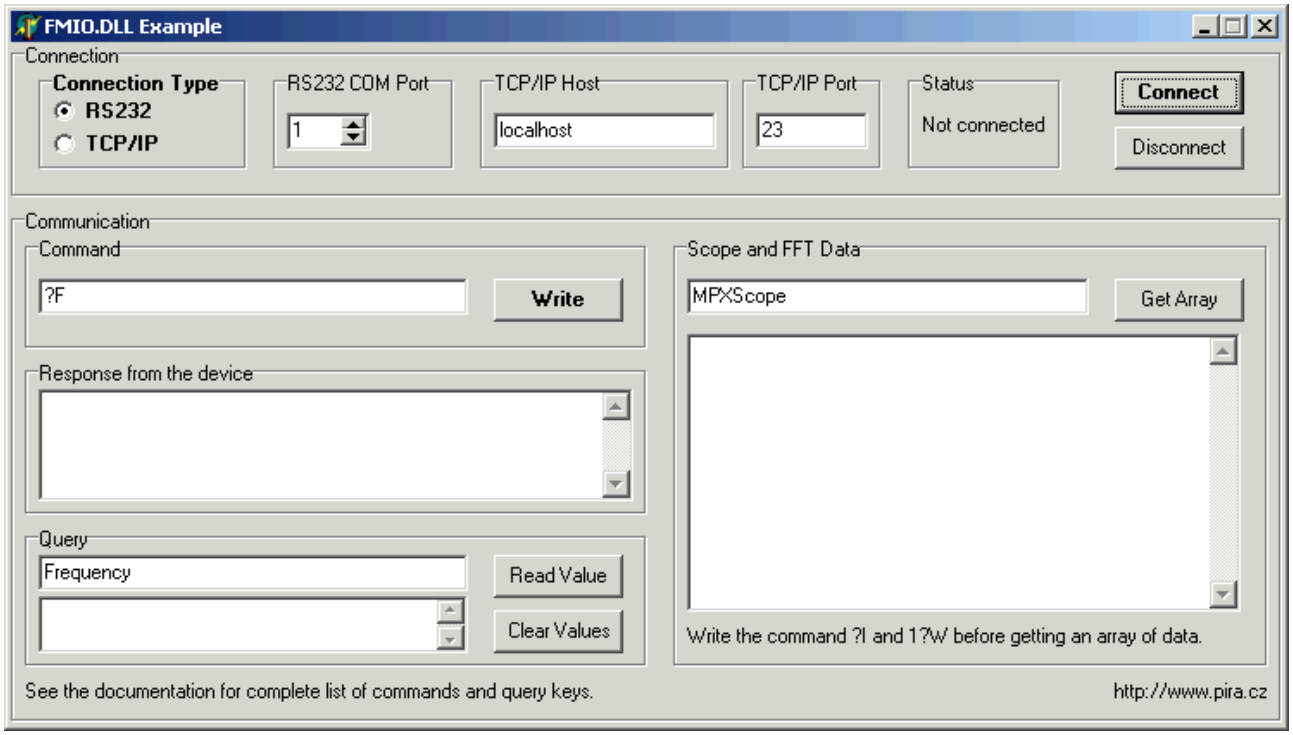
Example of use

Download the example zip file from the website (section FM Analyzer – Software). The example illustrates how to use the DLL in Delphi. Implementation in other programming languages is very similar. Please follow the documentation of your programming language for more details about how to use external functions and procedures from DLL.

Before running the example make sure the device communicates properly using the original control software.

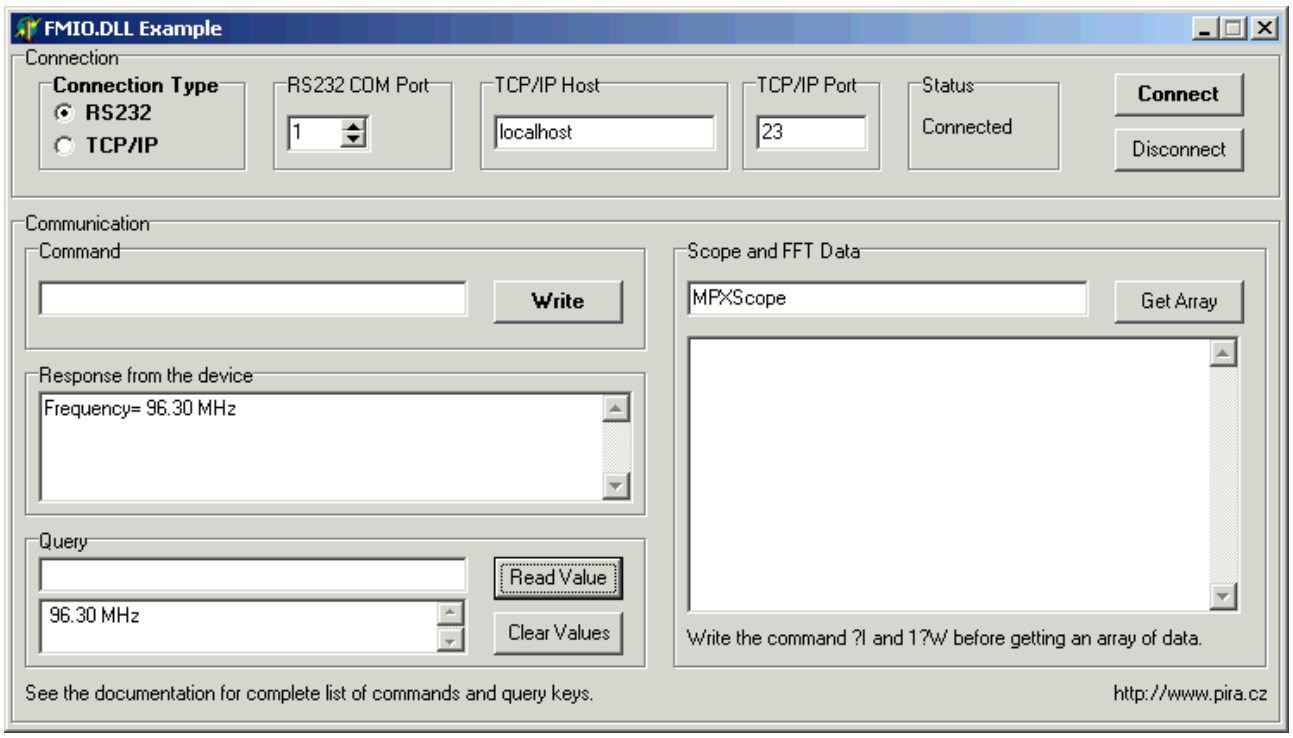
The example zip file contains:

fmio.dll	The DLL file. The DLL must be placed in the application folder and must be distributed with it.
example.exe	Compiled example. Just double-click to run it.
example.dpr	The example Delphi project file. Open this file in Delphi for edit the example.
unit1.pas	The example main unit file.
unit1.dfm	The example main form definition.
fmio.pdf	This file.



First steps

1. Fill the Connection fields and click Connect. The Status must indicate ‘Connected’.
2. Click Write button. The command ‘?F’ is sent to the device. In the response field actual device frequency will be shown.
3. Click Read Value button. A query with the key ‘Frequency’ will give the value.
4. Try other commands and keys to control the device and read the values. Complete list of the commands and keys is a part of the original P75/P175 or P275 technical manual.

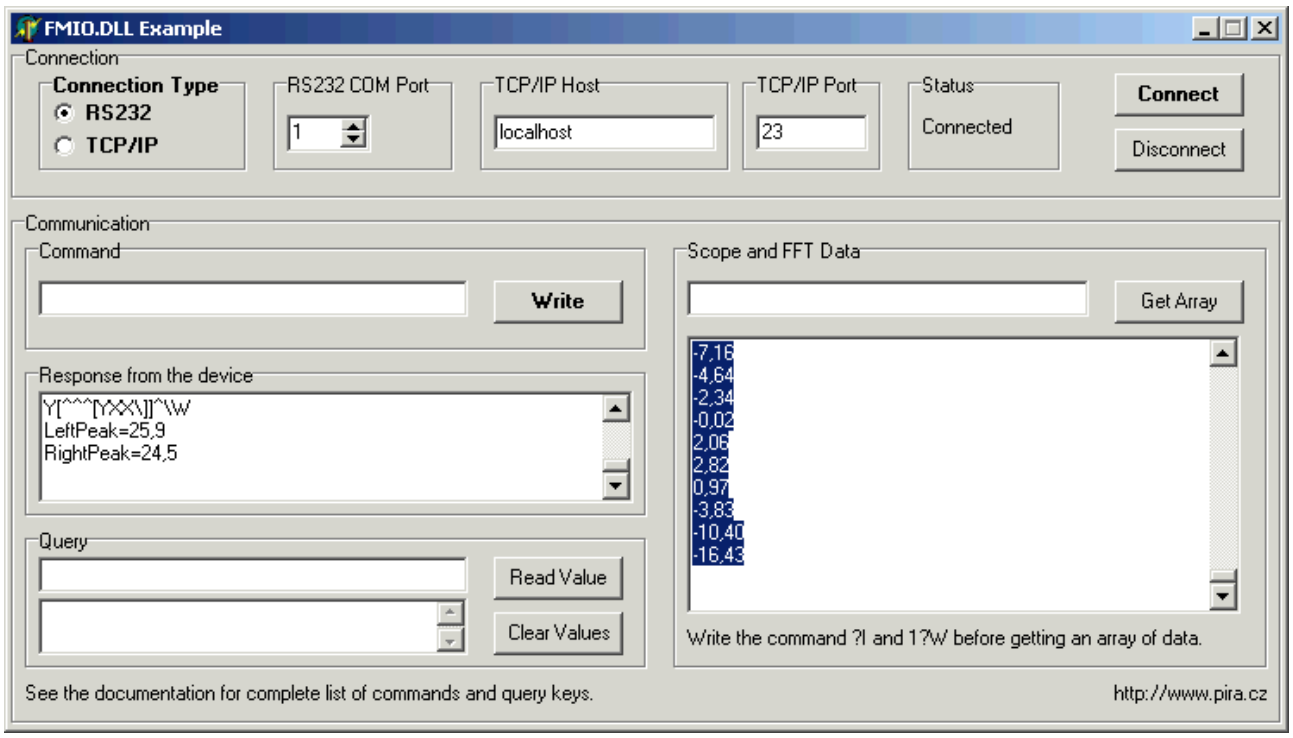


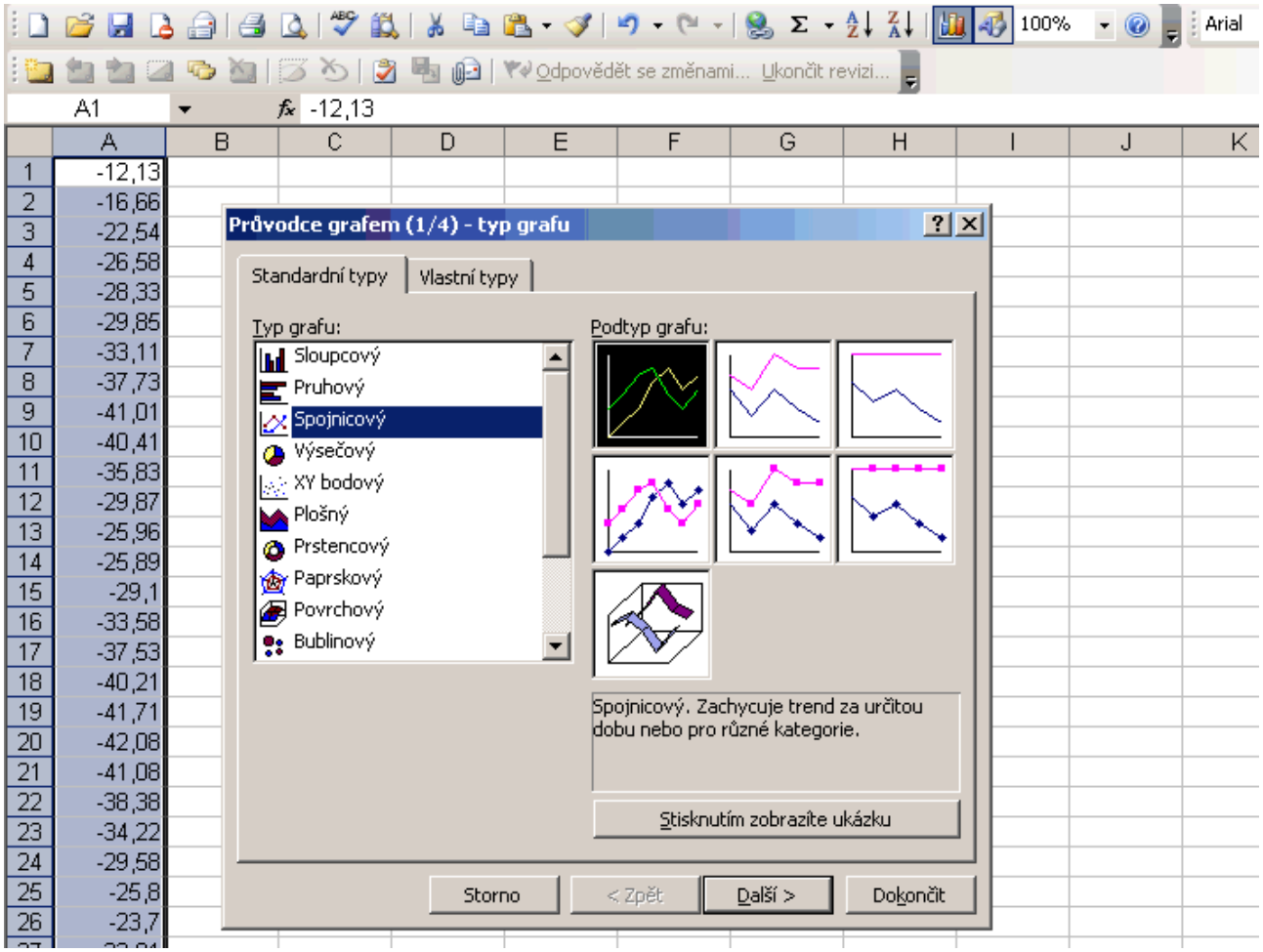
Scope and FFT functions

These functions are not provided directly by the device but are computed inside the DLL using built-in digital signal processing (DSP) module. Source data must be obtained from the device by commands ?I and 1?W in each process.

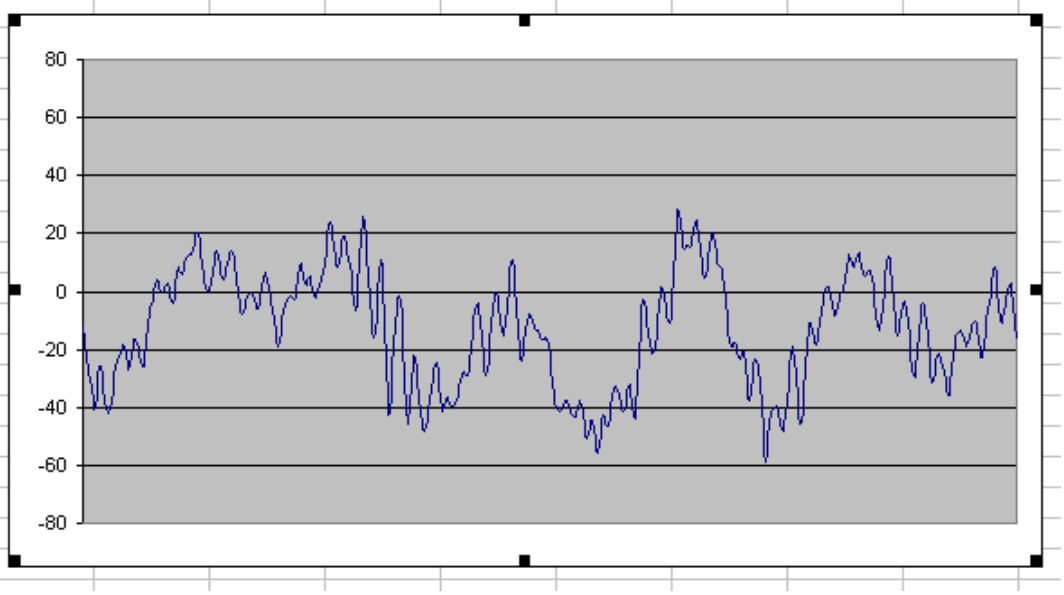
1. Tune the device to the frequency of interest.
2. Write command ?I
3. Write command 1?W to get a sample of the signal. The DSP processing will follow automatically.
4. Click on Get Array to get MPX scope data.
5. Select all the data and copy them into clipboard (for example using right mouse button).
6. Run MS Excel, select first cell and paste the data from clipboard.
7. Make a graph from the data.
8. Try to get and visualize other data arrays using other keys.

Note: The DSP module provides two additional dB values accessible using the query keys 'LeftPeak' and 'RightPeak'.


FMIO.DLL Example
Connection
Connection Type
RS232
TCP/IP
RS232 COM Port
1
TCP/IP Host
localhost
TCP/IP Port
23
Status
Connected
Connect
Disconnect
Communication
Command
Write
Response from the device
Y[^^^[YXX\]]^\W
LeftPeak=25,9
RightPeak=24,5
Query
Read Value
Clear Values
Scope and FFT Data
Get Array
-7,16
-4,64
-2,34
-0,02
2,06
2,82
0,97
-3,83
-10,40
-16,43
Write the command ?I and 1?W before getting an array of data.
See the documentation for complete list of commands and query keys.
http://www.pira.cz


A1
-12,13
Průvodce grafem (1/4) - typ grafu
Standardní typy
Vlastní typy
Typ grafu:
Sloupcový
Pruhový
Spojnicový
Výsečový
XY bodový
Plošný
Prstencový
Paprskový
Povrchový
Bublinový
Podtyp grafu:
Spojnicový. Zachycuje trend za určitou dobu nebo pro různé kategorie.
Stisknutím zobrazíte ukázku
Storno
< Zpět
Další >
Dokončit

	A
1	-12,13
2	-16,66
3	-22,54
4	-26,58
5	-28,33
6	-29,85
7	-33,11
8	-37,73
9	-41,01
10	-40,41
11	-35,83
12	-29,87
13	-25,96
14	-25,89
15	-29,1
16	-33,58
17	-37,53
18	-40,21
19	-41,71



The example source code listing (unit1.pas)

```
unit Unit1;

interface

uses
  Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls, Forms,
  Dialogs, StdCtrls, Spin, ExtCtrls;

type
  TForm1 = class(TForm)
    GroupBox1: TGroupBox;
    GroupBox2: TGroupBox;
    Button1: TButton;
    Button2: TButton;
    RadioGroup1: TRadioGroup;
    GroupBox3: TGroupBox;
    SpinEdit1: TSpinEdit;
    GroupBox4: TGroupBox;
    GroupBox5: TGroupBox;
    Edit1: TEdit;
    Edit2: TEdit;
    GroupBox6: TGroupBox;
    Edit3: TEdit;
    Button3: TButton;
    GroupBox7: TGroupBox;
    Memo1: TMemo;
    Timer1: TTimer;
    GroupBox8: TGroupBox;
    Button6: TButton;
    Button7: TButton;
    Edit4: TEdit;
    Memo2: TMemo;
    GroupBox9: TGroupBox;
    Button8: TButton;
    Edit5: TEdit;
    Memo3: TMemo;
    GroupBox10: TGroupBox;
    Label1: TLabel;
    Label2: TLabel;
    Label3: TLabel;
    Label4: TLabel;
    procedure Button1Click(Sender: TObject);
    procedure Button2Click(Sender: TObject);
    procedure Button3Click(Sender: TObject);
    procedure Timer1Timer(Sender: TObject);
    procedure Button6Click(Sender: TObject);
    procedure Button7Click(Sender: TObject);
    procedure Button8Click(Sender: TObject);
    procedure FormClose(Sender: TObject; var Action: TCloseAction);
  private
    { Private declarations }
  public
    { Public declarations }
  end;

type
    TGetArray=record
      Length: Word;
      Values: array[0..1023] of Single;
      end;

var
```

```
  Form1: TForm1;
  LastConState: boolean;

implementation

{$R *.dfm}

function Connect: boolean; stdcall; external 'fmio.dll';
function Connected: boolean; stdcall; external 'fmio.dll';
procedure Disconnect; stdcall; external 'fmio.dll';
procedure ConnectionSetup(ConType: boolean; PortNum: integer; Host: PChar; Port:
integer); stdcall; external 'fmio.dll';
function ReadBufferLength: integer; stdcall; external 'fmio.dll';
procedure WriteCommand(Command: PChar); stdcall; external 'fmio.dll';
function ReadResponse: Pchar; stdcall; external 'fmio.dll';
function ReadValue(Key: Pchar): Pchar; stdcall; external 'fmio.dll';
function GetArray(Key: Pchar): TGetArray; stdcall; external 'fmio.dll';
procedure ClearValues; stdcall; external 'fmio.dll';

procedure TForm1.Button1Click(Sender: TObject);     //CONNECT
var ConType: boolean;
    Port: integer;
begin
if (RadioGroup1.ItemIndex=0) then ConType:=False else ConType:=True;
//ConType false = Serial port, ConType true = TCP connection
try Port:=StrToInt(Edit2.Text); except Port:=23; end;
ConnectionSetup(ConType, SpinEdit1.Value, PChar(Edit1.Text), Port);
LastConState:=false;
if (Connect) then Label1.Caption:='Connecting' else ShowMessage('Unable to
connect');
end;

procedure TForm1.Button2Click(Sender: TObject);    //DISCONNECT
begin
if (Connected) then Label1.Caption:='Disconnecting' else Label1.Caption:='Not
connected';
Disconnect;
end;

procedure TForm1.Button3Click(Sender: TObject);    //WRITE
begin
WriteCommand(PChar(Edit3.Text));
Edit3.Text:='';
end;

procedure TForm1.Timer1Timer(Sender: TObject);
var ConState: boolean;
begin
if (ReadBufferLength>0) then Memo1.Lines.Add(String(ReadResponse));
ConState:=Connected;
if (ConState<>LastConState) then
  begin
  if (ConState) then Label1.Caption:='Connected' else Label1.Caption:='Not
connected';
  LastConState:=ConState;
  end;
end;

procedure TForm1.Button6Click(Sender: TObject);    //READ VALUE
begin
Memo2.Text:=ReadValue(PChar(Edit4.Text));
Edit4.Text:='';
end;
```

```
procedure TForm1.Button7Click(Sender: TObject);    //CLEAR VALUES
begin
ClearValues;
Memo1.Lines.Clear;
end;

procedure TForm1.Button8Click(Sender: TObject);    //GET ARRAY
var GetArrayBuf: TGetArray;
    i, length: integer;
begin
Memo3.Clear;
GetArrayBuf:=GetArray(PChar(Edit5.Text));
Edit5.Text:='';
length:=GetArrayBuf.Length;
for i:=1 to length do
  Memo3.Lines.Add(FloatToStrF(GetArrayBuf.Values[i-1],ffFixed,7,2));
end;

procedure TForm1.FormClose(Sender: TObject; var Action: TCloseAction);
begin
Disconnect;
end;

end.
```

FMIO.DLL source code listing

```
library fmio;

uses
  Forms,
  SysUtils,
  Classes,
  StrUtils,
  dm1 in 'dm1.pas' {DM: TDataModule};

{$R *.res}

type
    TGetArray=record
      Length: Word;
      Values: array[0..1023] of single;
      end;

var StrBuf: array[0..32767] of char;         //null-terminated string buffer
    StrBufP: Pchar;                          //pointer to the buffer
    GetArrayBuf: TGetArray;                  //buffer for Scope and FFT data

function ExportToPchar(Str: string): PChar;  //copies the string to the buffer
begin                                        //and returns a pointer to the
                                             //first character
if (length(Str)>32767) then Str:=LeftStr(Str,32767);
StrBufP:=@StrBuf;
StrPCopy(StrBufP, Str);
Result:=StrBufP;
end;

function Version: PChar; stdcall;
begin
Result:='1.5';
end;

function Connect: boolean; stdcall;
begin
Result:=DM.Connect;
end;

procedure SetTCPReconnectTimeout(Timeout: integer); stdcall;
begin
DM.SetTCPReconnectTimeout(Timeout);
end;

procedure Disconnect; stdcall;
begin
DM.Disconnect;
end;

procedure ConnectionSetup(ConType: boolean; PortNum: integer; Host: PChar; Port:
integer); stdcall;
begin
DM.ConnectionSetup(ConType, PortNum, string(Host), Port);
end;

function WriteBufferLength: integer; stdcall;
begin
Result:=DM.WriteBufferLength;
end;

function ReadBufferLength: integer; stdcall;
```

```
begin
Result:=DM.ReadBufferLength(DM.ADev);
end;

procedure WriteCommand(Command: PChar); stdcall;
begin
DM.WriteCommand(string(Command));
end;

function ReadResponse: Pchar; stdcall;
begin
Result:=ExportToPchar(DM.ReadResponse)
end;

function ReadValue(Key: PChar): PChar; stdcall;
begin
Result:=ExportToPchar(DM.ReadValue(string(Key)));
end;

procedure ClearValues; stdcall;
begin
DM.ClearValues;
end;

function Connected: boolean; stdcall;
begin
if (DM.DevOutOfRange(DM.ADev)) then begin Result:=false; exit; end;
Result:=DM.DeviceList[DM.ADev].Connected;
end;

function GetArray(Key: PChar): TGetArray; stdcall;
var i: integer;
begin
GetArrayBuf.Length:=0;
if (DM.DevOutOfRange(DM.ADev)) then exit;
with DM.DeviceList[DM.ADev] do
begin
if (Key='MPXScope') then
  begin
  GetArrayBuf.Length:=701;
  for i:=1 to GetArrayBuf.Length do GetArrayBuf.Values[i-1]:=scope[i-1];
  end;
if (Key='MPXFFT') then
  begin
  GetArrayBuf.Length:=165;
  for i:=1 to GetArrayBuf.Length do GetArrayBuf.Values[i-1]:=fft[i];
  end;
if (Key='IFFFT') then
  begin
  GetArrayBuf.Length:=235;
  for i:=1 to GetArrayBuf.Length do GetArrayBuf.Values[i-1]:=maxhold[i-1];
  end;
if (Key='LeftScope') then
  begin
  GetArrayBuf.Length:=182;
  for i:=1 to GetArrayBuf.Length do GetArrayBuf.Values[i-1]:=stlout[i];
  end;
if (Key='RightScope') then
  begin
  GetArrayBuf.Length:=182;
  for i:=1 to GetArrayBuf.Length do GetArrayBuf.Values[i-1]:=stpout[i];
  end;
end;
Result:=GetArrayBuf;
```

```
end;

function PCharToFloat(Source: PChar): single; stdcall;
begin
Result:=DM.PCharToFloat(string(Source));
end;

function AddDevice: integer; stdcall;
begin
Result:=DM.AddDevice;
end;

function CheckDevices: integer; stdcall;
begin
Result:=DM.CheckDevices;
end;

function SetDevice(DHandle: integer): boolean; stdcall;
begin
Result:=DM.SetDevice(DHandle);
end;

function ActiveDevice: integer; stdcall;
begin
Result:=DM.ADev;
end;

function DeviceCount: integer; stdcall;
begin
Result:=Length(DM.DeviceList);
end;

Exports
  Version, Connect, Disconnect, ConnectionSetup, WriteBufferLength,
  ReadBufferLength, WriteCommand, ReadResponse, Connected,
  ReadValue, ClearValues, GetArray, PCharToFloat,
  AddDevice, CheckDevices, SetDevice, DeviceCount, ActiveDevice,
  SetTCPReconnectTimeout;

begin
DM:=TDM.Create(Application);
end.
```

Note: Sources from the DM data module are not available for public use.

Support for Multiple Devices

Until now we've considered a control of a single FM analyzer device only. The DLL, however, supports multiple access to almost unlimited number of FM analyzers. Communication with each device works in complete independence.

For accessing of particular device, the functions are the same as previously described. Additional functions are implemented, especially for creating a new instance of the device and for selecting a device you want to speak with.

```
function AddDevice: integer; stdcall;
function CheckDevices: integer; stdcall;
function SetDevice(DHandle: integer): boolean; stdcall;
function ActiveDevice: integer; stdcall;
function DeviceCount: integer; stdcall;
```

Function or procedure	Description
AddDevice	Creates a new device. Returns an index of the new device. The new device is automatically set as active. Usually, the ConnectionSetup function call should follow. The AddDevice function is called one time on the DLL loading so there's no need to take it into consideration if only one device has to be controlled.
CheckDevices	Returns an index of the first device (checking is always made from device index 0) which contains new data in receiving queue.
SetDevice	Sets active device. That is a device you want to control using the set of standard functions described in previous sections. Returns true if the device index (handle) is valid, returns false otherwise.
ActiveDevice	Returns an index of the actual device. That is a device selected by the SetDevice function.
DeviceCount	Returns total number of devices. The value returned is equal to the total number of calling of the AddDevice function.

Conclusion

DLL version: 1.7
Document revision: 11.10.2016

Your feedback is important! Please send us any bug reports or modification request.

mail@pira.cz http://pira.cz