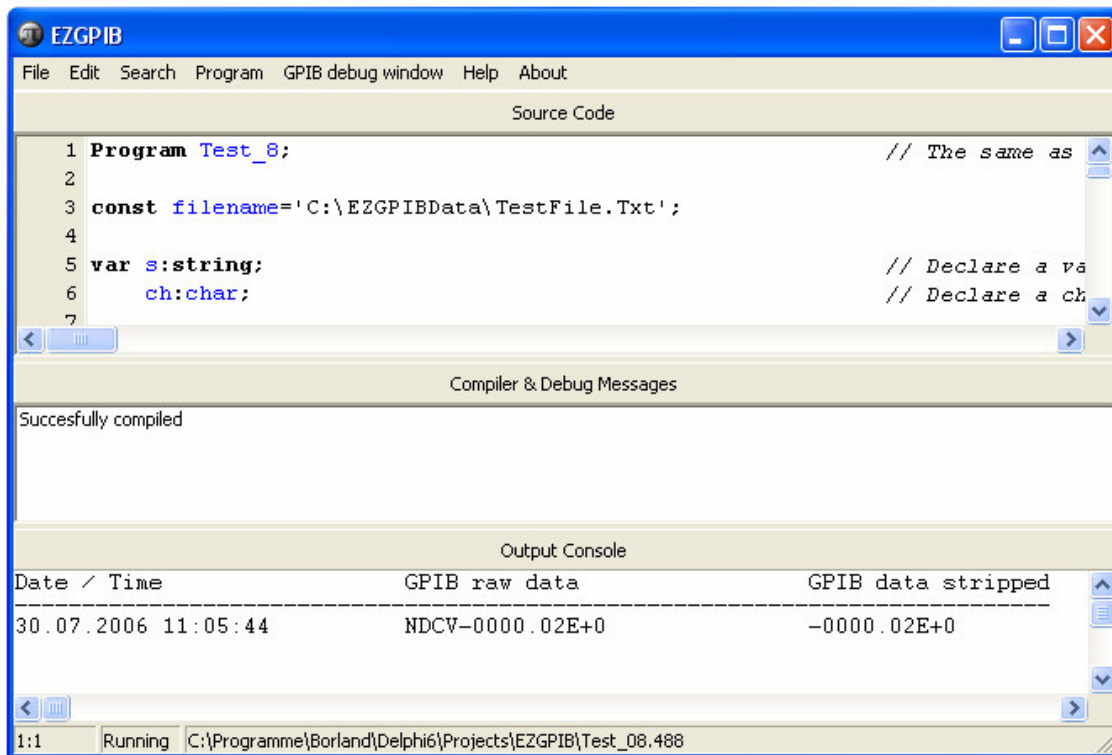


EZGPIB



A GPIB, RS232 and TCP/IP data
acquisition tool

Ulrich Bangert
df6jb@ulrich-bangert.de
Ortholzer Weg 1
D-27243 Gross Ippener
Germany

Introduction

I have been using GPIB based instruments since the time when I was a young student of physics at the RUHR-UNIVERSITAET-BOCHUM (the University at Bochum, Germany).

I had access to a TEKTRONIX 4051 which happened to be one of the first real computers that you could put on your desk (at least if your desk was large enough). The 4051 rock looked like shown in picture 1.



Picture 1

At its time the 4051 was a **tremendous** modern computer. While other computers used ASCII text based terminals to communicate to their users the 4051 had a dazzling graphics screen with a 1024 x 1024 resolution. It would be incorrect to talk about the 'pixels' of the screen because it was not a television like screen displaying pixels. Instead, what you were looking at was a really big storage tube as originally developed by TEKTRONIX for their stock of analogue storage oscilloscopes.

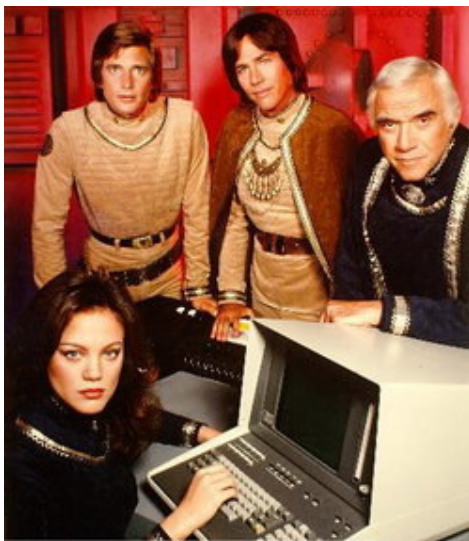
This storage screen was 'written' by a beam of electrons which affected a phosphor layer to constantly afterglow where the beam had impinged the layer. The horizontal and vertical

deflection plates for that electron beam were driven by two 10 bits D/A converters which made it possible to focus the beam to any position of the screen with the mentioned resolution.

Imagine a full blown graphics computer 10 years before IBM even planned to construct what we now call a pc and even many years before the first 'home computers' were to appear on the scene. Wow!

In fact, the 4051 was generally considered a so revolutionary concept that it was used to play the part of the 'computer' in science fiction movies!

Picture 2 is a promotional picture for one of the earlier 'Battlestar Galactica' movies. That was not what we call 'product placement' today! They really liked this thing for its futuristic design!



Picture 2

In these early days of computing BASIC was the language of choice when it came to data acquisition and companies like Hewlett & Packard (the inventors of HPIB, the 'Hewlett & Packard Interface Bus', a term that was superseded by the more neutral GPIB 'General Purpose Interface Bus' by their competitors) and TEKTRONIX had the necessary bus handling commands as integral parts of their BASIC interpreters.

GPIB handling was easy! While a

100 Print "Test"

statement would output the string "Test" to the screen, the statement

100 Print @13:"Test"

would send the string "Test" to the device with address 13 on the GPIB. Similar to that a

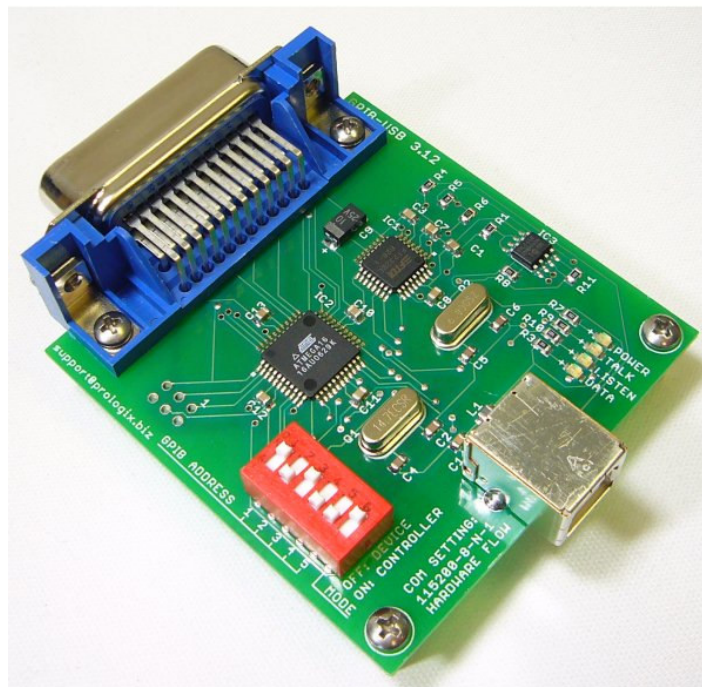
110 Input @4:A\$

statement would not read the keyboard but instead try to read an answer from GPIB device 4 into the string A\$. While the BASIC language has its limitations, writing data acquisition software has never been easier than that. The above examples apply to the 4051. HP's BASIC stuff was a bit different but on the same level of ease.

After university the first data acquisition systems for air pollution measurements that I had to deal with were based on GPIB equipped devices. Then there was a long pause in which I still had to do with data acquisition but no more GPIB based.

A lot of electronic measurement devices that I bought as part of my hobby as a radio amateur had a GPIB interface but there was no real need for me to use it. Things changed when I became interested into oscillator stability tests where data recording over hours, days or even weeks is part of the business. In this context I used ISA GPIB cards from CEC, KEITHLEY, NATIONAL INSTRUMENTS and INES (a Germany based company) and I wrote data acquisition software using LABVIEW, Borland's TURBO PASCAL and later using DELPHI.

While there were basically only 2 different GPIB controller chips on the market each manufacturer would give his cards a different address range, DMA range and interrupt range making them as incompatible among each other as possible. For that reason you have to use the drivers and DLLs specific for a certain manufacturer. And since the drivers and DLLs are not identical in terms of functionality and syntax to talk to them I had to change my sources with every change in GPIB hardware. When I had to exchange my last ISA based computer against one that featured only PCI slots I had to deal with the problem what to do now with my GPIB based measurements. Should I pay again hundreds of dollars for a PCI GPIB card and again receive new drivers and DLLs? In this situation I came over the GPIB-USB Controller from PROLOGIX, available from <http://www.prologix.biz> which is shown in picture 3.



Picture 3

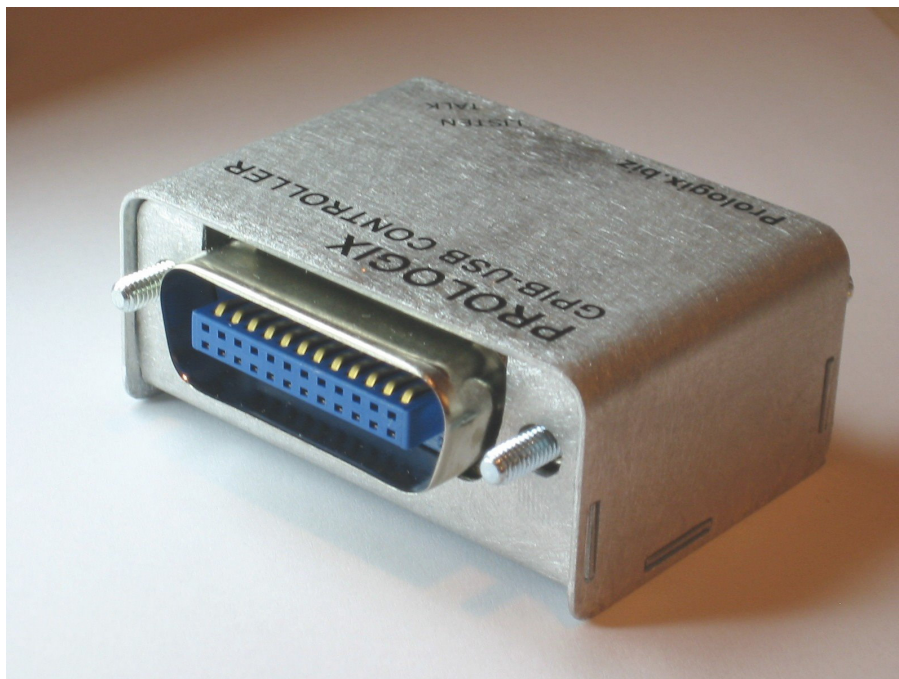
It sells for app. 125.00 US \$. This cute little device uses a FTDI chip for USB to RS232 translation. That means that you install a simple FTDI driver available free from their web pages. This driver allocates a virtual RS232 port and you may use any software from ready to go terminal programs up to self written stuff to communicate over this virtual RS232 with the ATMEL AVR microcontroller on the Prologix board.

This microcontroller in turn features a firmware that works as kind of interpreter for serial commands into GPIB actions. It has an easy to learn command syntax and the complete issue of GPIB programming is reduced to handling strings on a serial port.

While this is a worth to mention reduction in complexity concerning GPIB programming it may still leave a problem for the non-programmer to get a GPIB measurement up and running. In addition the simple serial syntax lacks some higher level action commands that I would have liked to have available for my own programming needs. That is why I started to program me my own working environment for GPIB programming. You may understand it as an IDE that is made especially for GPIB and RS232 data acquisition in conjunction with the cheap Prologix board. I called it EZGPIB. This paper is to introduce you to EZGPIB.

Note that also some tools for receiving screen plots from scopes and analyzers via the Prologix board are available for free. Google a bit for the American callsign KE5FX or search for the name “John Miles” in the web.

Update 2007-06-01: Since a few weeks a new version of the controller is available from Prologix. This one looks like shown in Picture 4 and sells for 149.95 which it is well worth due to the improvements against its predecessor. Alone the fact that it can now safely be screwed to an IEEE-488 terminal and updated via the USB port is worth every cent of it.



Picture 4

User Guide

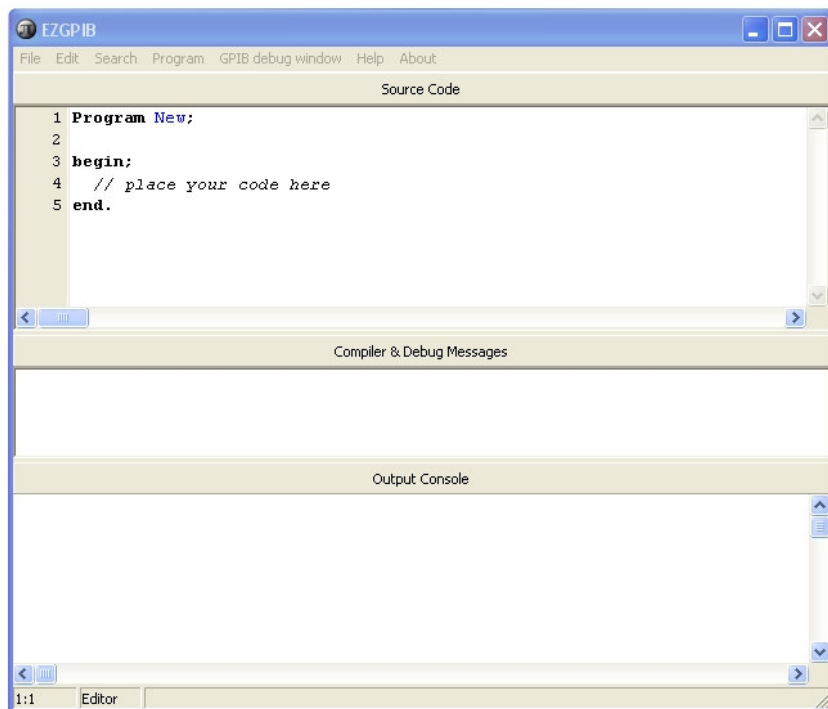
Starting EZGPIB

EZGPIB consists of a single executable file EZGPIB.EXE, the INOUT.DLL, some programming examples and this help file. EZGPIB runs under WINDOWS 2000, XP and hopefully under newer versions too. Older versions like W95 and W98 have not been tested but are very likely to produce trouble.

On the older Prologix board set the DIP switches 1-5 to the OFF position and switch 6 to the ON position. That makes the Prologix board a GPIB bus controller with the own bus address of "0". The presence of a new (switch-less) controller board is detected automatically by EZGPIB and the necessary measures are taken.

When starting EZGPIB.EXE the software will try to detect a Prologix board connected to your system. In order to do so all serial ports of your system are enumerated and checked for the presence of a FTDI chip. If a FTDI chip is detected the software sends a test string to this chip to test whether a Prologix board answers or not.

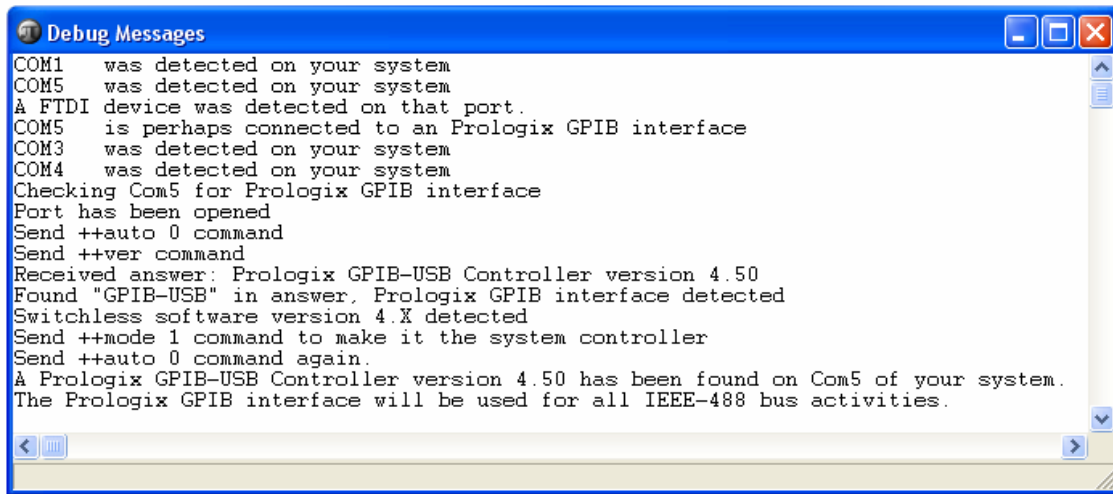
If no Prologix interface is found the next thing that EZGPIB will try to detect is whether a DLL named GPIB32.DLL is available on the pc because that may be an indicator for the fact that the pc is equipped with a plug in GPIB adapter card. If the DLL is available it is loaded and used to detect any plug GPIB adapter that is connected to the pc, may be a pci board or something different.



Picture 5

No matter how your pc is equipped you are first confronted with the EZGPIB's main window like shown above.

The next thing that I suggest to do is to open the **GPIB debug window** from the main menu. This will assist you in understanding how the Prologix board or an other GPIB adapter was found or why it was not found. When a Prologix controller is connected to my system the debug window looks like this after starting EZGPIB:



Picture 6

Again, depending on your system the information may be different from the one displayed above.

Note: Even if a Prologix board is connected correctly to your system and a number of GPIB devices is connected correctly to the Prologix board you may encounter the situation that the board is not detected when starting EZGPIB.

In most cases this phenomenon is due to the following reason: The Prologix board answers to serial commands then and only then when it is able to fully control the GPIB bus. It can only fully control the bus when enough devices on the bus are switched on!

If you have connected more than one instrument to the GPIB you should **switch on** at least 2/3 of all devices connected to the bus even if you want to talk only to a single device. This is not a specific Prologix problem but has to do with the GPIB specifications in general.

The situation that the board cannot fully control the bus is also easily identified by the leds on the Prologix board: If the **TALK** led stays constantly on beneath the power led this is an indicator for such a problem. If everything is ok then the **LISTEN** led is constantly on at this time. On <http://www.transera.com/htbasic/tutgpib.html> under 'Physical characteristics' you can read more on that.

Be prepared that the very easy to use high level commands “EZGPIB_BusWriteData” and “EZGPIB_BusWaitForData” will do the job for most of what you are going to do. Lots of other commands are to be found in EZGPIB *just because they are available* for the Prologix controller.

Two things that you need to know but that are not so evident:

- 1) The main window is divided into three sections and you may individually set the size of these sections. Move the mouse slowly from one section to the next one until the cursor changes into two parallel lines. Press the mouse button and move the section border in the direction that you like.
- 2) EZGPIB stores programs in standard test files but gives them the extension **.488** to make them different from other text files. On start up EZGPIB will look for the existence of a file named **standard.488**. If a file of that name is found in EZGPIB's directory it will be loaded automatically. That feature has been built in for the case that you work with the same setup on a regular base. If you choose **New** from the file menu you load in reality a file named **new.488**. Store everything that you want to see in the beginning of a new project into **new.488** and it will there when you need it.

What EZGPIB is

- 1) EZGPIB is an IDE with a full blown editor for writing and debugging PASCAL like source code. PASCAL is an easy to learn structured programming language. Even if you are a non-programmer you will experience that the examples that come together with EZGPIB give you an easy to understand introduction into what it is all about.
- 2) EZGPIB is a compiler for the PASCAL like program source that you have written using the editor. The most prominent difference to a standard PASCAL compiler is the fact that a lot of ready to go functions and procedures have been added to the standard language. The things that have been added are (hopefully) everything that you need for successful GPIB data acquisition in conjunction with the Prologix board and for data processing. You don't need to hassle with low level serial communication functions but can instead use high level functions as
BusWaitForData(Device:LongInt;ForWhat:string;MaxWait:Double):Boolean, which means as much as: Wait <MaxWait> seconds for a GPIB message coming from Bus device <Device>. If the answer arrives within the timeout of <MaxWait> seconds then put the message into the string <ForWhat> and return "True" as the function value. Otherwise leave <ForWhat> empty and return "False" as the function value.
- 3) EZGPIB is the runtime environment for the programs that you have written with the editor and translated with the compiler. EZGPIB does not produce standalone applications. EZGPIB applications can only be started from within EZGPIB. While this may seem as a drawback to some of you there are strong reasons for it that are beyond the scope of this discussion. There is also a big advantage in EZGPIB being its own runtime environment: The lower part of the main window is a text based output and input console and EZGPIB has built in functions and procedures that make text based i/o via this console a snap. You don't have to learn any Windows specific i/o techniques. If the editor part of the main window and the debug part of the main window bother you while executing a program you may start the program with the **Run Console** command. In this case editor and debug part are minimized during program execution. File output of measured data, which is perhaps the most important thing that you are after, is **extreme easy!**

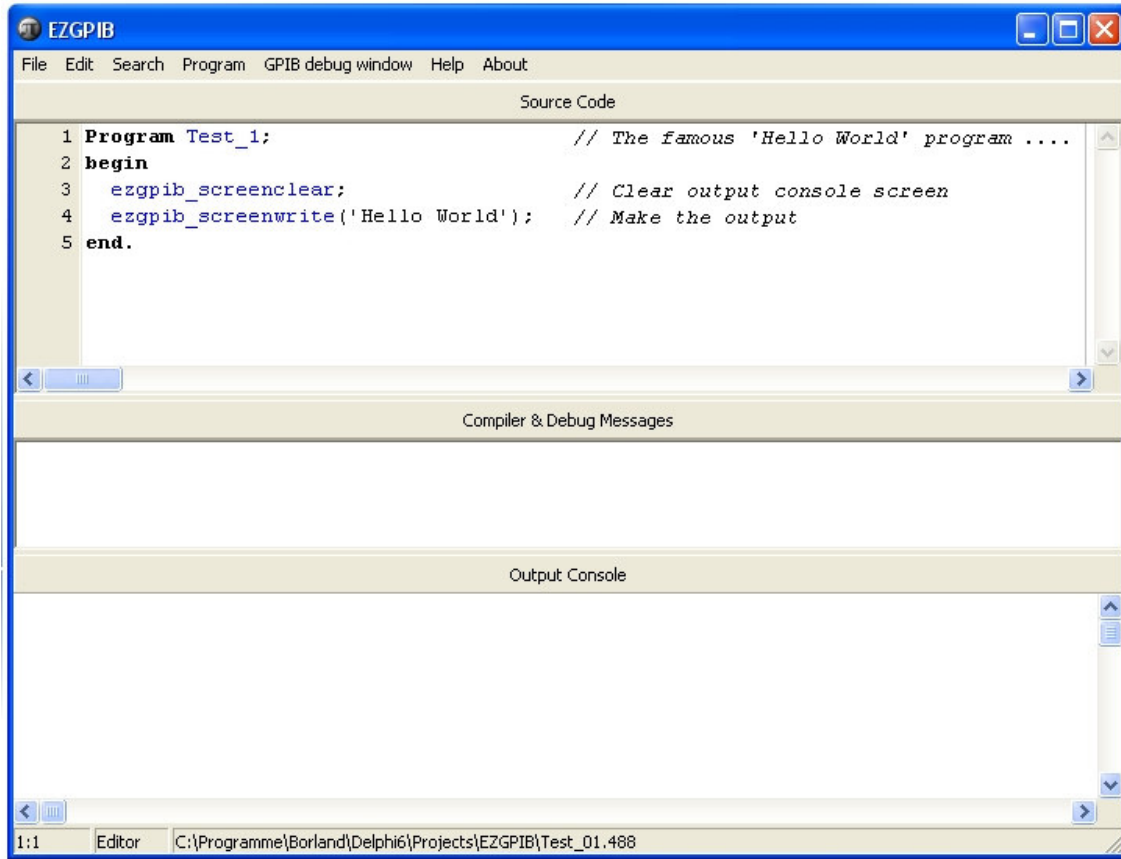
What EZGPIB is not

EZGPIB is a tool that **helps you to communicate** with your GPIB devices. However, it does not know what the **content** of the communication needs to be in order to get the expected result. **You** will have to know!

You will need a certain understanding for what your devices expect to receive over the GPIB in order perform a certain action or to send back a certain measurement result over the GPIB. Usually this means that you need the complete device manual or at least the GPIB programming part of it. Be warned that most manuals are not thought as introductory lessons into GPIB programming. On the other hand GPIB is one of the best introduced standards in the world of electronics and you should have no difficulties to find everything that can be known about GPIB in general on the web.

Using EZGPIB

EZGPIB's **File** menu is very similar to what you know from other Windows based software. Use the Open command to enter the File-Open dialogue and choose to load Test_01.488. Now your main window should look like this:

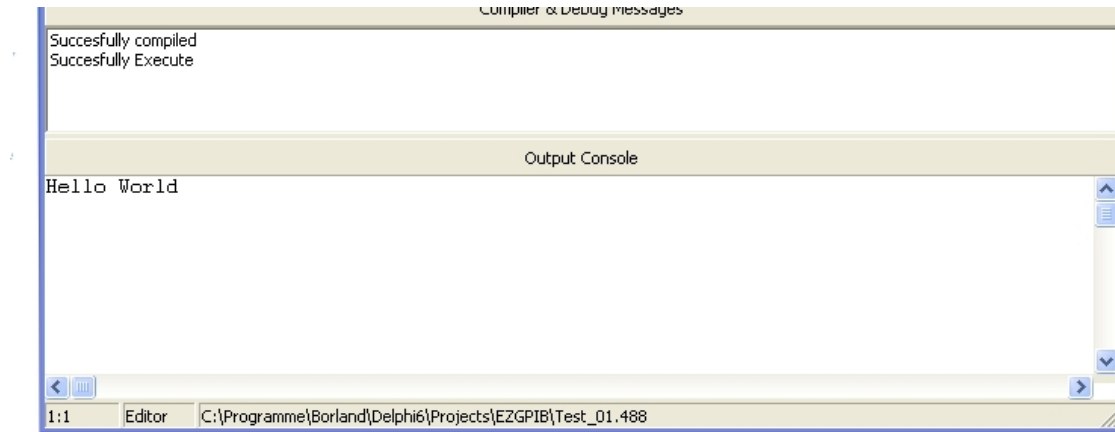


Picture 9

This is the famous 'Hello World' program that every programmer needs to do in a new programming environment. In the editor part of the window note that

- 1) Some words are black and in a fat font. These are reserved words of the PASCAL like language that may not be used for a different purpose. You are not allowed to call a program 'begin' because 'begin' is a reserved word indicating the start of a program, function or procedure.
- 2) Some words are black but in a standard font. These are constants that cannot be assigned a specific group. In the example everything behind the '*//*' is a comment and the 'Hello World' is a text constant.
- 3) Some words are blue. These are the names of programs, functions and procedures.

Now use the **Run** Command of the **Program** menu to compile and execute this program. The lower part of the main window should look like



Picture 10

Congratulations! You have compiled and executed your first EZGPIB program. It had nothing to do with GPIB and this may give you the idea that you can use EZGPIB even for other quick and dirty programming.

That does not necessarily mean that everything you do in EZGPIB must be quick and dirty. The PASCAL language that is the foundation of EZGPIB enforces you to write well structured stuff and you can handle even big problems and tasks with ease. EZGPIB is a true compiler and you will find that EZGBIP based programs are real fast.

However, you will find out that most GPIB measurement applications written in EZGPIB are quite small in terms of number of program lines. This is due to the available high level functions.

We write a real data acquisition application

Here I will describe the necessary steps to write a real data acquisition application. Of course, it will be a simple one. But every complex task can be subdivided into a group of lower complex tasks which in turn can be subdivided in even lower complex tasks up to a point where the tasks are real simple. This programming scheme is called top-down-design and is well supported by a PASCAL like programming language.

What I am going to show you is how to set a frequency counter's time base (in this case my old but spry RACAL DANA 1996) to a value of 10 seconds, then wait for the measured results and display the results in the output console from within a program loop that runs as long as no key is pressed on the keyboard. Hey you non-programmers: Sounds complicated? See and be staggered how easy it is. You guessed it: That's where the name comes from!

From the **File** menu press the **New** entry. In general the source code part of the main window will show:

```

1 Program New;
2
3 begin;
4   // place your code here
5 end.
```

Picture 11

But note that this does **not need** to be so! When using the **New** menu entry EZBPIB searches for a file **New.488** in its directory. If it is found it is loaded into the editor. What is that good for?

Imagine that from day to day you have to deal with different measurement tasks. Despite the fact that the tasks are different they may have a lot in common: The bus addresses of your GPIB devices normally stay the same and you may have made yourself a number of helping routines that you use in a more general manner. In this case you store the framework that is common for all your applications under the name **New.488**. Whenever you choose **New** from the **File** menu your complete framework will be loaded into the editor and you start to do your individual programming on the new task with everything common already there.

While there are other possibilities to do the same, for example the use of include files, I found this an elegant way for handling the framework. For the sake of simplicity let us assume that we have the situation as shown in picture 9. First we change the program's name to an appropriate one and declare a constant holding the bus address of the counter. Should the counter get a different address in the future this will be the only point where we will have to change the number!

Now the editor window may look like

```

1 Program Counter_Test;
2
3 const Counter_Address:2;
4
5 begin;
6   // place your code here
7 end.
```

Picture 12

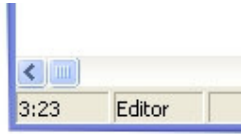
Note, that even at this early stage of development you can use the **Compile** menu entry from the **Program** menu to check the syntax of what you have written (there is nothing to really run at the moment...).

In this case you will see

```
[Error] Unnamed(3:22): is ('=') expected
```

Picture 13

because we used the wrong syntax for the constant declaration. The compiler tells us that it would like to see a '=' instead the ':' that we used. He also tells us where it has found this bug: Line 3 character 22. Note that in EZGPIOB's status line at the bottom of the main window you always see where the cursor is currently located in the editor part of the window.



Picture 14

Let us correct the mistake and include the command to set the time base to 10 seconds. Now your source should look like this. Note that EZGPIOB does not distinguish between uppercase and lowercase characters. You can use the writing that you prefer.

```

1 Program Counter_Test;
2
3 const Counter_Address=2;
4
5 begin;
6   EZGPIOB_BusWriteData(Counter_Address, 'GA10');
7 end.
```

Picture 15

The String 'GA10' (Gate Adjust to 10 seconds) is the information that you need to get from the counter's manual.

Well, this is already a program that can be executed. If you execute it with the **Run** menu entry from the **Program** menu you should see that the counter changes its time base to 10 seconds. Note, that the source contains nothing that reminds you of the Prologix board. The BusWriteData procedure handles all that stuff for you and reminds me in its simplicity to what I said about BASIC before.

I had already said that once we had switched the counter to the new time base value we would do something in a program loop until a key is pressed on the keyboard. Very similar to normal English we use a '**repeat until**' for that purpose. Even the condition when the loop shall stop sounds very familiar.

```

1 Program Counter_Test;
2
3 const Counter_Address=2;
4
5 begin;
6   EZGPIOB_BusWriteData(Counter_Address, 'GA10');
7   repeat
8
9   until EZGPIOB_KbdKeyPressed;
10 end.
```

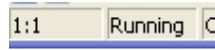
Picture 16

If you execute this program you will notice that the message window shows the information



Picture 17

and the status line says



Picture 18

which both of are indicators that your program is running in the loop. Press any key and you will see the 'Successfully Executed' message appear in the message window and the 'Running' changes back to 'Editor'.

Now let us put something useful into the loop. I did already mention before that there is an easy to use function for reading data. So let's apply it!

```

1 Program Counter_Test;
2
3 const Counter_Address=2;
4       TimeOut=15;
5
6 var   Answer:String;
7
8 begin;
9   EZGPIB_BusWriteData(Counter_Address,'G10');
10  repeat
11    if EZGPIB_BusWaitForData(Counter_Address,Answer,TimeOut) then
12      begin;
13        EZGPIB_ScreenWriteLn(Answer);
14      end;
15  until EZGPIB_KbdKeyPressed;
16 end.
```

Picture 19

Note that in order to use the elegant formulation

EZGPIB_BusWaitForData(Counter_Address,Answer,TimeOut)

we had to declare a variable 'Answer' of string type and a constant 'Timeout' with the value 15. Since the counter's time base is set to 10 he should be able to answer within 15 seconds. If we execute this program the console part of the main window will show


```

F +9.9999999940247E+06
F +9.9999999939087E+06
F +9.9999999937927E+06
F +9.9999999934082E+06
F +9.9999999930847E+06

```

Picture 20

and every ten seconds a new line will be written. You don't like the scrolling and want to get rid of that 'F' stuff in the beginning of the string so that you can convert it into a number?

Here we go:

```

1 Program Counter_Test;
2
3 const Counter_Address=2;
4     TimeOut=15;
5
6 var Answer:String;
7
8 begin;
9     EZGPiB_BusWriteData(Counter_Address,'GA10');
10    repeat
11        if EZGPiB_BusWaitForData(Counter_Address,Answer,TimeOut) then
12            begin;
13                Answer:=EZGPiB_ConvertStripToNumber(Answer);
14                EZGPiB_ScreenClear;
15                EZGPiB_ScreenWriteLn(Answer);
16            end;
17        until EZGPiB_KbdKeyPressed;
18    end.

```

Picture 21

produces

```
+9.9999999936462E+06
```

Picture 22

that is: A new result overwrites the previous result.

You may want the time information when the value was measured? Here it is:

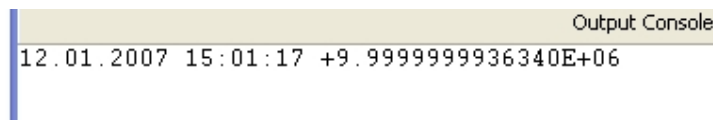
```

1 Program Counter_Test;
2
3 const Counter_Address=2;
4     TimeOut=15;
5
6 var   Answer:String;
7
8 begin;
9     EZGPIB_BusWriteData(Counter_Address,'GA10');
10    repeat
11        if EZGPIB_BusWaitForData(Counter_Address,Answer,TimeOut) then
12            begin;
13                Answer:=EZGPIB_ConvertStripToNumber(Answer);
14                EZGPIB_ScreenClear;
15                EZGPIB_ScreenWrite(EZGPIB_TimeNow);
16                EZGPIB_ScreenWrite(' ');
17                EZGPIB_ScreenWriteLn(Answer);
18            end;
19        until EZGPIB_KbdKeyPressed;
20    end.

```

Picture 23

will produce:



```

Output Console
12.01.2007 15:01:17 +9.9999999936340E+06

```

Picture 24

Let me stop here and suggest that you have a look at the examples that will demonstrate you a lot of other aspects of EZGPIB. If you find that you have difficulties with the PASCAL language itself I suggest that you get yourself a basic level introduction into PASCAL.

Please understand that you will definitely not need to become a top PASCAL programmer. You need only a very basic understanding of PASCAL and programming habits and concepts to work very successfully with EZGPIB.

To be honest: The example that I showed you above shows very bad programming habits for a number of reasons. Nevertheless it has proved to work. If we are to talk about programming habits: The worst thing that I have done in the example above is to **wait that long** for the answer of a device. Clearly, a counter with its time base set to 10 seconds can only deliver a measurement value every 10 seconds, no discussion over that. But instead of waiting for its answer we could have done other nice things in this time, for example communication with other devices. While we **wait** for one device **we cannot do something different** during this time. That leads to the question how to know when a device has data available by other than just waiting for that data.

```

1 Program Counter_Test;
2
3 const Counter_Address=2;
4     TimeOut=0.1;
5
6 var   Answer:String;
7
8 begin;
9     EZGPIB_BusFindAllDevices;
10    EZGPIB_BusWriteData(Counter_Address, 'GA10');
11    EZGPIB_ScreenClear;
12    repeat
13        EZGPIB_TimeWaitForMultipleOf(1);
14        EZGPIB_ScreenGotoXY(1,1);
15        EZGPIB_ScreenWriteLn(EZGPIB_TimeNow);
16        If EZGPIB_BusSrq then
17            begin;
18                if EZGPIB_BusSourceOfSrq=Counter_Address then
19                    begin;
20                        EZGPIB_BusWaitForData(Counter_Address, Answer, TimeOut);
21                        Answer:=EZGPIB_ConvertStripToNumber(Answer);
22                        EZGPIB_ScreenGotoXY(25,1);
23                        EZGPIB_ScreenWriteLn(Answer);
24                    end;
25                end;
26        until EZGPIB_KbdKeyPressed;
27 end.

```

Picture 25

The GPIB has a nice concept for the so called SRQ = Service Request. Service Request is a line of its own on the GPIB that can be activated by any device to indicate that it requests service. Usually devices request service if they have new measurement values available but other sources for a service request are possible. The controller needs to find out which device request service and then reads the data of this device.

EZGPIB and the Prologix board support this strategy very well. Have a look at the modified version of the counter example above on the last page. Some things have been changed against the last version of the example:

- 1) A call to EZGPIB_FindAllDevices has been added at the start of the program. This is always good habit with EZGPIB. This procedure will try to detect all active devices on the bus by reading their so called status register. Even older devices that cannot understand IEEE-488-2 syntax (*idn? and stuff like that) have in most cases a status register and can so be identified by asking for the value of the status register.
- 2) The console screen is only cleared once at the beginning of the program. After that the cursor is positioned with a EZGPIB_ScreenGotoXY command to a specific x and y position before the string is written.
- 3) Note the call to EZGPIB_WaitForMultipleOf in the main loop. This waits until an integer multiple of its call value is over. With a '1' we wait with this call until a new second has arrived. We could have leaved this out and run many times faster through the main loop but since we want to print out the time information in the main loop there is no necessity to do it faster than once per second.

- 4) Note that we print out the time information every second. In addition every second we check whether a SRQ is active or not by a call to EZGPIB_BusSrq. Only if this condition is given we ask the counter to give us his measured value. Because we can be sure that it has a value available in this situation we can use a very short timeout of 0.1 second. Note that by comparing the result of EZGPIB_BusSourceOfSrq to Counter_Address we double check that it is really the counter which requests service.

This example shows that instead of waiting for the counter we have done other things in the meantime.

Note that not **all** devices signalize a new measurement with a SRQ. Lots of devices signalize it with setting a certain bit in their status registers and you have to check their manuals to find out. In this case you will have to check their status register on a regular base. Fast devices that can deliver a new measurement value every few ms may not either use a service request or a bit in the status register. **If** they are fast, none of that is needed.

Data acquisition using serial ports

From its very beginnings up to now EZGPIB has also learned to handle serial ports well and in an easy manner. Have a look at the well commented example below which will read the regular output of an HP53131 counter used as a TIC (time interval counter). Thank you Said for inspiring me to that! You also find it under the accompanying examples.

```

Program HP53131;

Const Filename='C:\MyMeasurements\HP53131.Txt';
      HPPort=2;

Var  HP_as_String:String;           //What we get from the counter
      HP_as_Double:Double;         //What we make out of it
      Time:String;                 //Where we put the time in

function Init:Boolean;
begin;
  if EZGPIB_FileExists(Filename)   //Comment this out if you
  then EZGPIB_FileDelete(Filename); //always want to append
  EZGPIB_FileClearBuffer;          //Clear The Filebuffer
  EZGPIB_FileAddToBuffer("Time/MJD"); //Add two strings to FileBuffer
  EZGPIB_FileAddtoBuffer("TIC/s");
  EZGPIB_FileWrite(Filename);      //Append Contents of FileBuffer to FileName
  HP_as_String:="";                //Initialize some vars...
  Result:=EZGPIB_ComOpen(HPPort,9600,8,'N',1); //Open serial port and report result
end;

function DataAvailable:Boolean;
begin;
  Result:=False;
  HP_as_String:=HP_as_String+EZGPIB_ComRead(HPPort); //Add things read from port to buffer
  If Pos(#13+#10,HP_as_String)<>0 then                //If CR+LF found in buffer variable
  begin;
    EZGPIB_ConvertRemove(',',HP_as_String);           //Remove unwanted Comma
    HP_as_String:=EZGPIB_ConvertStrToNumber(HP_as_String);
    HP_as_Double:=EZGPIB_ConvertToFloatNumber(HP_as_String); //Convert string to Double
    HP_as_Double:=HP_as_Double * 1.0E-6;               //53151's numbers are in µs!
    Result:=True;
  end;
end;

```

EZGPIB Manual

```

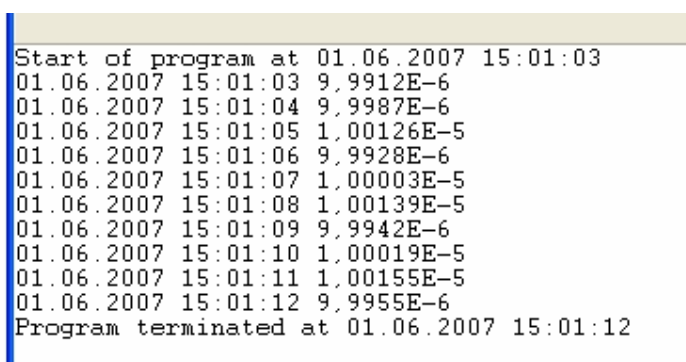
end;

procedure HandleData;
begin;
  Time:=EZGPIB_ConvertToMJD(EZGPIB_TimeNow);           //Get the time in MJD format
  Time:=EZGPIB_ConvertStripToNumber(Time);             //Make sure it uses an decimal point
  EZGPIB_FileClearBuffer;                              //Clear The Filebuffer
  EZGPIB_FileAddToBuffer(Time);                        //Add time to FileBuffer
  EZGPIB_FileAddtoBuffer(Hp_as_Double);               //Add TIC value ot FileBuffer
  EZGPIB_FileWrite(Filename);                         //Append FileBuffer to FileName
  EZGPIB_ScreenWrite(EZGPIB_TimeNow);
  EZGPIB_ScreenWrite(' ');
  EZGPIB_ScreenWriteLn(Hp_as_Double);
  Hp_as_String:=""                                   //Reset string buffer variable
end;

begin;
  EZGPIB_ScreenClear;
  EZGPIB_ScreenWrite('Start of program at ');
  EZGPIB_ScreenWriteLn(EZGPIB_TimeNow);
  if init then
  begin;
    repeat
      If DataAvailable then HandleData;
      EZGPIB_TimeSleep(0.1)
    until EZGPIB_KbdKeyPressed;
  end
  else EZGPIB_ScreenWriteLn('Error opening the com port...');
  EZGPIB_ScreenWrite('Program terminated at ');
  EZGPIB_ScreenWriteLn(EZGPIB_TimeNow);
end.

```

When looking at the output of this program the built in console screen this will look like in Picture 25.



```

Start of program at 01.06.2007 15:01:03
01.06.2007 15:01:03 9,9912E-6
01.06.2007 15:01:04 9,9987E-6
01.06.2007 15:01:05 1,00126E-5
01.06.2007 15:01:06 9,9928E-6
01.06.2007 15:01:07 1,00003E-5
01.06.2007 15:01:08 1,00139E-5
01.06.2007 15:01:09 9,9942E-6
01.06.2007 15:01:10 1,00019E-5
01.06.2007 15:01:11 1,00155E-5
01.06.2007 15:01:12 9,9955E-6
Program terminated at 01.06.2007 15:01:12

```

Picture 26

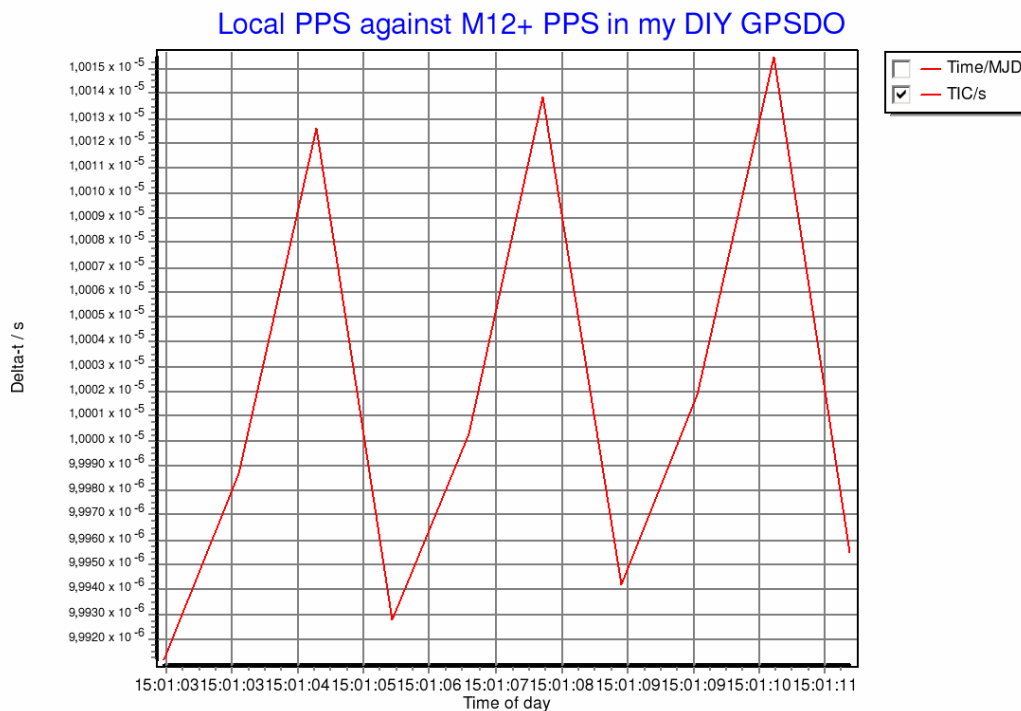
The contents of the file that is generated by the program (note that the subdirectory “MyMeasurements” has been created automatically) is

Time/MJD	TIC/s
54252.625729525462	9,9912E-6
54252.625741099473	9,9987E-6
54252.625752673484	1,00126E-5
54252.625764247496	9,9928E-6
54252.625775821973	1,00003E-5
54252.625787210651	1,00139E-5
54252.625798969995	9,9942E-6

EZGPIB Manual

54252.625810544007 1,00019E-5
 54252.625822118018 1,00155E-5
 54252.625833692029 9,9955E-6

which in turn read in with my PLOTTER utility will display as



Picture 27

What the examples do

This is a short description of what I have tried to demonstrate in the examples.

Test_01.488

This is the EZGPIB version of the famous 'Hello World' program.

Test_02.488

Demonstrates a simple **for** loop basic math and console output.

Test_03.488

Demonstrates a simple **for** loop with a timed wait condition.

Test_04.488

Demonstrates a main program loop that is terminated by the user.

Test_05.488

Demonstrates the first real GPIB data transfers

Test_06.488

Does the same as Test_05.488 but uses procedures and functions to get more structure into the program source.

Test_07.488

Does the same as Test_06.488 but adds file output capabilities.

Test_08.488

Does the same as Test_07.488 but adds DDE capabilities.

Test_09.488

Demonstrates the concept and use of service requests

Test_10.488

Demonstrates the use of include files.

Test_11.488

Demonstrates serial communication

Test_12.488

Demonstrates how to receive a screen plot from a HP5371

Test_13.488

Demonstrates direct port i/o

HP53131.488

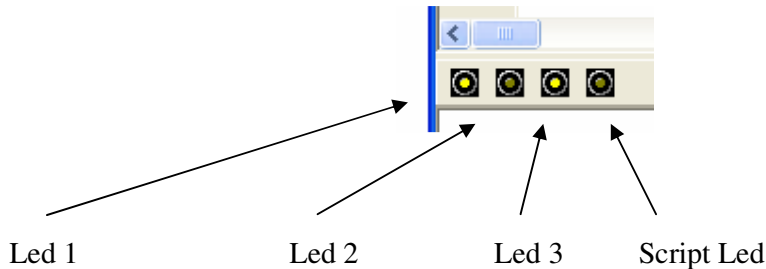
Demonstrates serial communication with a HP53131

XXXX.488

I include some scripts that I have written for my own applications. Perhaps they are helpful for you too. They are not so well documented but do their job too.

Some new features

Starting on version 2007-07-14 the internals of EZGPIB have changed a lot. Now EZGPIB is a multi-threaded program featuring four threads that indicate their condition by means of four symbolized leds on the main window.



The first thread is the program's main thread that is also associated with the main window and Windows's message queue. This thread signalizes its working condition by regular changing the state of Led 1. Note that this thread may be stopped or executed delayed due to the way that Windows manages its multitasking and message queue handling. However, the **other** threads are not subject to that and that has been the reason for making EZGPIB a multithreaded application.

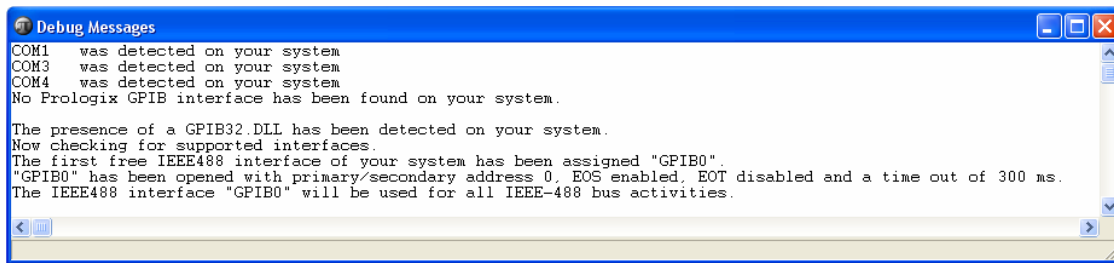
The second thread is the one that performs all the serial communication with the Prologix interface. This one signalizes its working condition by regular changing the state of Led 2.

The third thread is the one that generates and display all the messages of the GPIB debug window. This one signalizes its working condition by regular changing the state of Led 3.

These three threads are active all of the time EZGPIB runs so you should see all three leds blink all over the time EZGPIB is active. The fourth thread is the one in which your script is executed. Unlike the first three threads **I** as the author of EZGPIB do not know well what is happening inside that thread because **you** as the author of the script decide. I include a new procedure called **ChangeLed** that you can use inside your script at a position that is executed on a regular base to indicate that the script thread is active too.

Multi-threading should make everything run much smoother as before. Time-driven events in your script can no more be delayed by the GUI.

Starting on version 2007-12-24 support for GPIB32.DLL was added. That is: If no Prologix interface board is found then EZGPIB tries to detect the presence of the GPIB32.DLL which is a good indicator that a DLL based IEEE488 interface is available. Then EZGPIB tests whether the DLL reports the presence of an active IEEE488 interface.

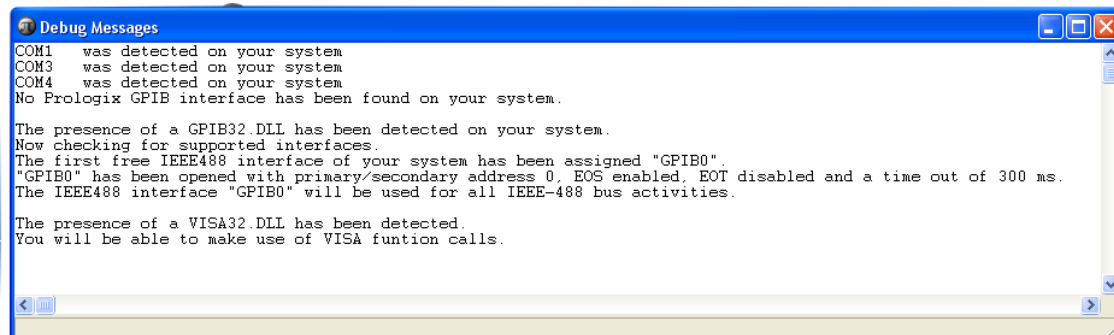


Picture 29

On my system the debug window will look like shown above in this case. Note that the support for GPIB32.DLL *has only been tested with products from National Instruments*. I can make no promise whatsoever that this support will work with boards and DLLs from other manufacturers.

The really nice thing about this DLL support is the fact that you don't have to learn anything new. Almost All commands (with some minor differences explained in the command description) work for the Prologix board as well as for the DLL based interface.

Starting on version 2008-06-08 support for VISA32.DLL was added. That is: If your system has a VISA library installed like the Agilent IO library then EZGPIB now can make use of VISA function calls. There are only 5 new functions to learn to open the world of VISA based data acquisition. If a VISA32.DLL is detected the debug window will say:



Picture 30

All the 5 new functions are used in the following demo script:

Program VISA; // Demonstrates VISA communication

```
var Status:Integer;
    CountWritten:Integer;
    CountRead:Integer;
    RM:Integer;
    VI:Integer;
    Answer:String;

begin;
    Status:=EZGPIB_viOpendefaultRM(RM);
    Status:=EZGPIB_viOpen(RM,'GPIB0::9::INSTR',0,0,VI);
    Status:=EZGPIB_viWrite(VI,'OUTPUT ON',CountWritten);
    Status:=EZGPIB_viWrite(VI,'VOLT 12.500',CountWritten);
    Status:=EZGPIB_viClose(VI);
    Status:=EZGPIB_viOpen(RM,'GPIB0::10::INSTR',0,11,VI);
    Status:=EZGPIB_viWrite(VI,'END ALWAYS',CountWritten);
    Status:=EZGPIB_viWrite(VI,'DCV',CountWritten);
    Status:=EZGPIB_viWrite(VI,'NRDGS 1,SYN',CountWritten);
    Status:=EZGPIB_viWrite(VI,'TRIG SGL',CountWritten);
    Status:=EZGPIB_viRead(VI,Answer,CountRead);
    EZGPIB_screenwriteln(Answer);
    Status:=EZGPIB_viClose(VI);
end.
```

This script opens a VISA session, then connects to my HP6632 on address 9 of the bus and sets it to 12.5 Volt output. Then it closes the connection and connects to my HP3457 on address 10 of the bus to read this voltage back.

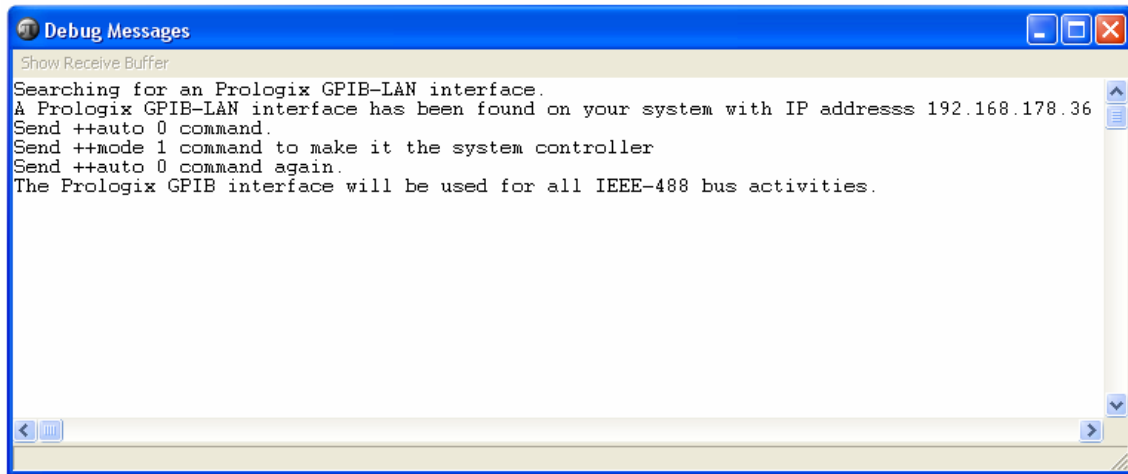
The script above together with debug outputs is to be found in VISA.488

Note that EZGPIB can only detect the presence of the VISA32.DLL *but nothing else*. Use the tools that come together with the DLL to explore what interfaces and what instruments are available using the VISA driver.

Starting on version 2008-08-02 the support for the new Prologix LAN GPIB interface has been built in. After the start of EZGPIB the search for a Prologix LAN GPIB interface is performed as the first task. Note that in order to detect the LAN GPIB interface its ip-address needs to be in the same network segment as the ip-address of your pc.

If the Prologix LAN GPIB interface is configured to make use of DHCP then this will usually be the case if you simply connect the interface to your local network and there is a DHCP server available such as a DSL router. If the interface is configured to make use of a fixed ip address you have to take care yourself for matching ip-addresses. Consult the Prologix manual for questions of network management necessary for the LAN GPIB interface.

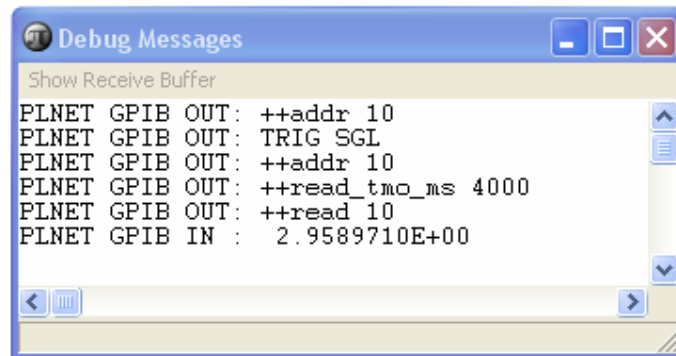
If a LAN GPIB adapter is detected this will clearly be stated in the debug window which in this case will display:



Picture 31

where the displayed ip address will of course be different on your system. No further action is necessary. Just use EZGPIB as if a USB GPIB interface or a DLL based plug in card were available.

The only difference that you may notice when working with a GPIB LAN interface is that the debug window will tell you



Picture 32

where PLNET indicates a Prologix network based device. With a USB based device every line will start with "PLUSB"

Starting on version 2008-08-02 the behaviour of EZGPIB concerning the detection of the end of device answers has changed. I had believed that the use of the bus EOI line is pretty much a standard for indicating the end of a message. In the course of time I got acquainted to more and more devices that

- would not make use of the EOI line by default but needed to be told explicitly to do so. My own devices of this type include a Rohde & Schwarz URV-5 rf voltmeter and a HP3457 multimeter
- .
- would make use of the EOI on *some* answers and *not on other* answers. Which I would not like to comment.... The HP437B power meter seems to belong to this group.
- can not be convinced to make use of the EOI line at all. The Racal Dana 1991 and 1992 counters seem to belong to this group although my own 1996 type counter uses EOI

Detection that one is confronted with a problem with a missing EOI based problem is everything else then easy business. I have for that reason decided that the standard detection method for the end of device messages will from now on be EOS based.

EOS based means that not a hardware line on the bus is obeyed but the messages itself are checked for termination characters. The most often used termination character is the <LineFeed> (ASCII char # 10_{dec} or 0A_{hex})

For all interfaces the use of EOS and the <LineFeed> as the EOS-char is now default! If you wish to change that you will have to use the two procedures

Procedure EZGPIB_BusSetEos(How:Booelan);

This procedure enables/disables the use of the EOS char. The default is "True";

Domain: Prologix & DLL based

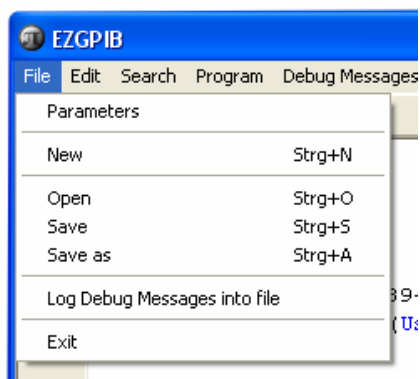
Procedure EZGPIB_BusSetEOSChar(How:Byte)

This procedure sets the EOS char. The default is "10" (Line Feed);

Domain: Prologix & DLL based

With version 2008-10-29 there are the following changes:

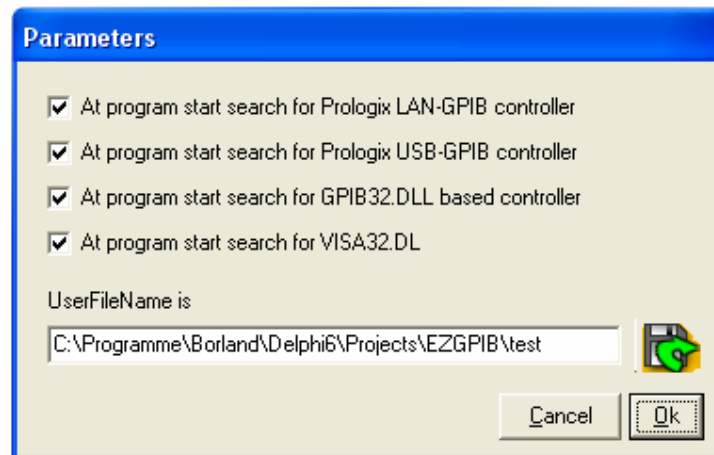
The File menu has two additional entries, <Parameters> and <Log Debug Messages into file>



Picture 33

If you check <Log Debug Messages into file> then every message from the Debug Messages Window will be additionally logged into the text file “DebogLog.Txt”. Look for this file in the directory where EZGPIB itself is to be found.

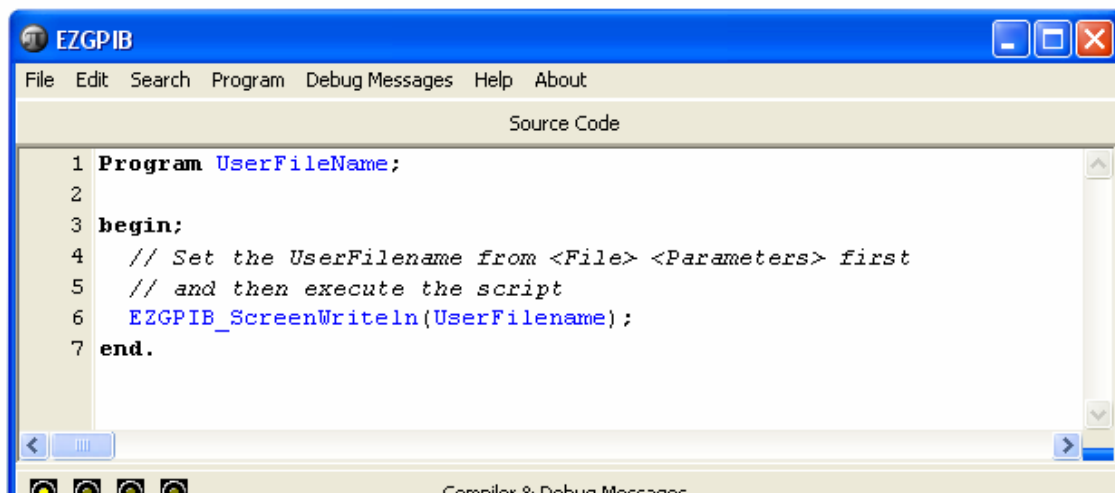
With a mouse click on <Parameters> a window opens as shown in Picture 34



Picture 34

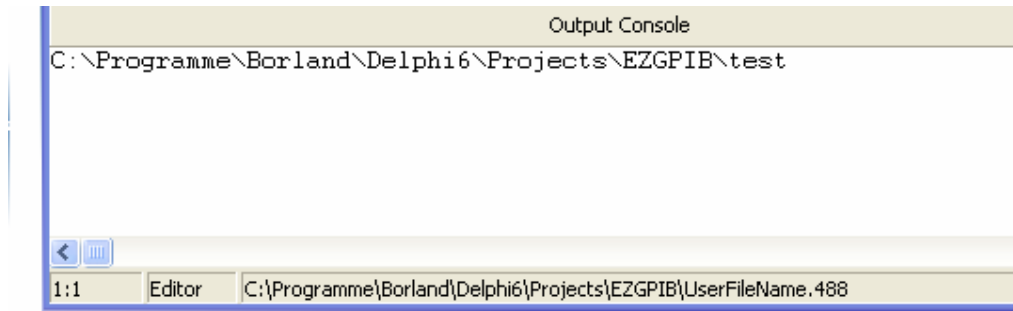
By means of this window you can set individually which types of interfaces shall be searched at program start.

In the edit field you see the name of a file. If you use the variable name “UserFileName” in your script, then the contents of this variable will hold the filename displayed here. For example the script



Picture 35

will lead to the result

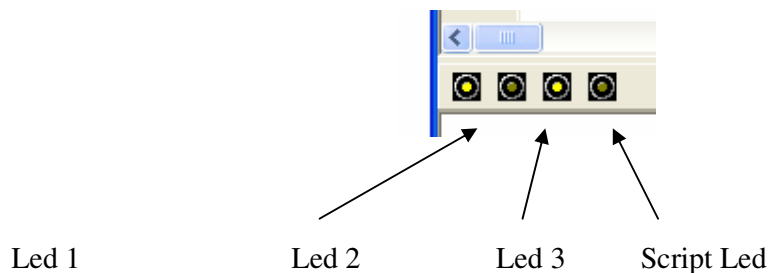


Picture 36

Note that you did not have to declare UserFilename, it is always there with the correct contents. You can edit the UserFileName directly in the edit field or use the symbol right of it to open a Windows style “File Save Dialog”.

The support for single step operation using F7 and F8 has been improved a lot.

The support for Breakpoints in the source has been improved a lot. Note: If your source contains Breakpoints it is now handled different when you execute it. With Breakpoints



Led 2 and Led 3 will flash in sync, indicating that the script is executed in the program's main thread. That makes supporting Breakpoints easier but has the disadvantage that the script execution may stop if you handle the main window.

Without Breakpoints the script is executed in a thread of it's on and cannot be interrupted by actions from the main thread.

Some minor bugs have been cured.

GPIB Address Parameter and Sub Address Support

The GPIB address parameter is normally the address of the instrument on the GPIB bus in the range 0 to 30. For older instruments, the address is often set with a DIP switch located near the GPIB connector. New instruments often define the GPIB address using the front panel keys of the instrument. For most instruments, the address parameter is simply the GPIB address.

Some systems, for example systems based on the VXI bus, require a GPIB *sub address* to be used. These systems are usually collections of instruments in a single chassis. The chassis has a GPIB address, and each individual instrument is addresses with the sub address.

EZGPIB allows the sub address to be placed in the upper 8 bits of the GPIB address parameter. The GPIB sub addresses are in the range of 0 to 30. The Prologix convention maps these addresses into the values from 96 to 126. National Instruments supports the values of 1-30. EZGPIB supports both the Prologix and National Instruments conventions for defining the subaddress within the upper eight bits.

Example:

Assume that you have an HP75000 VXI system at GPIB address 9 with a multi meter installed. The command processor of the HP75000 has sub address 0 and the multi meter sub address 3.

Using the National Instruments convention, here are the values for the sub address

Command Processor $0 + 1 = 1$
Multi Meter $3 + 1 = 4$

To move the value to the upper 8 bits, multiply the sub address value by $2^8 = 256$ and add it to the GPIB address

Command Processor address $= 1 * 256 + 9 = 265$
Multi Meter address $= 4 * 256 + 9 = 1033$

Using the Prologix convention, here are the values for the sub address

Command Processor $0 + 96 = 96$
Multi Meter $3 + 96 = 99$

To move the value to the upper 8 bits, multiply the sub address value by $2^8 = 256$ and add it to the GPIB address

Command Processor address $= 96 * 256 + 9 = 24585$
Multi Meter address $= 99 * 256 + 9 = 25353$

Of course, it is more convenient and understandable to let the EZGPIB compiler do the calculation for you. Here is the portion of code that defines the sub address from the example scripts HP75000Test.488 and HP75000TestA.488.

const

```

    HP75000=9;           { GPIB Address of VXI Mainframe }
    HP75000SystemSA = 0; { Sub address of VXI system controller }
    HP75000DVMSA = 3;    { Sub address of VXI Digital Multimeter }
    { Fully qualified address of VXI system controller (National Instruments convention) }
    HP75000System = HP75000 + 256* (1+HP75000SystemSA);
    { Fully qualified address of VXI Multimeter (National Instruments convention) }
    HP75000DVM = HP75000 + 256* (1+HP75000DVMSA);

```

const

```

    HP75000=9;           { GPIB Address of VXI Mainframe }
    HP75000SystemSA = 0; { Sub address of VXI system controller }
    HP75000DVMSA = 3;    { Sub address of VXI Digital Multimeter }
    { Fully qualified address of VXI system controller (Prologix Convention) }
    HP75000System = HP75000 + 256* (99+HP75000SystemSA);
    { Fully qualified address of VXI Multimeter(Prologix Convention) }
    HP75000DVM = HP75000 + 256* (99+HP75000DVMSA);

```

If debug messages are enabled and the Prologix interface is being used, both of these definitions will result in the follow address commands:

For the command processor:

```
++addr 9 96
```

For the multimeter:

```
++addr 9 99
```

Table of special EZGPIB functions and procedures

Beneath a big number of available functions and procedures that you can list with the **Help** menu entry there are certain ones that have been pre-programmed especially for EZGPIB. The names of each of them start with **EZGPIB_**. The next characters after the underscore try to indicate into which category the procedure or function falls. While 'Bus' indicates that it must be something GPIB specific 'Kbd' indicates that the procedure or function is keyboard relevant.

Note that instead of typing **EZGPIB_** again and again in the editor you can use the shortcut CTRL-E (or STRG-E on European keyboards) to generate an **EZGPIB_**.

Function EZGPIB_BusDataAvailable:Boolean

Returns 'True' if any device on the bus has send data and has terminated it with the current delimiter

Domain:Prologix

Function EZGPIB_BusGetAddressedDevice:LongInt

Returns the address of the currently addressed bus device

Domain:Prologix

Function EZGPIB_BusGetData:string

Returns the string that has last been sent by a device over the bus

Domain:Prologix

Function EZGPIB_BusGetEoi:LongInt

Queries the current setting of EOI. This is a low level function directly related to the ++eoi command of the Prologix board.

Domain:Prologix

Function EZGPIB_BusGetTimeOut:Double

Queries the current setting of the timeout. This is a low level function directly related to the ++read_tmo_ms command of the Prologix board

Domain:Prologix

Function EZGPIB_BusSourceOfSrq:LongInt

Returns the address of the device that is requesting service. If several devices are requesting service the function will return the lowest device address that needs service. It is necessary that you have called "EZGPIB_BusFindAllDevices" once before using this function.

Domain:Prologix & DLL based

Function EZGPIB_BusSPoll(Device:LongInt):LongInt

Performs a serial poll to return the status byte of 'Device'

Domain:Prologix & DLL based

Function EZGPIB_BusSrq:Boolean

Returns 'True' if a device has requested service

Domain:Prologix & DLL based

Function EZGPIB_BusSrqStatus:LongInt

Returns the status byte of the service requesting device

Domain:Prologix & DLL based

Function EZGPIB_BusWaitForData(Device:LongInt;ForWhat:string;MaxWait:Double):Boolean

Waits 'Maxwait' seconds or until 'Device' answers whichever is to take place earlier. If 'Device' answers within the timeout the answer is returned in 'ForWhat' and the function returns 'True'. The end of the message is detected by the delimiter sent by the device. Use this function for single measurement values.

Domain:Prologix & DLL based

Function EZGPIB_BusWaitForDataBlock(Device:LongInt;ForWhat:string;MaxWait:Double):Boolean

Waits 'Maxwait' seconds for an answer from 'Device'. If 'Device' answers within the timeout the answer is returned in 'ForWhat' and the function returns 'True'. This function does not look at delimiters that could be a part of the answer but waits the complete 'MaxWait' time. Use this function for longer data blocks as screen shots and so on.

Domain:Prologix & DLL based

EZGPIB Manual

Function EZGPIB_BusWaitForSrq(MaxWait:Double):Boolean

This function waits 'MaxWait' seconds or until a SRQ takes place whichever is earlier. It returns 'True' if a SRQ has been noticed within the timeout.

Domain:Prologix & DLL based

Procedure EZGPIB_BusSetEos(How:Boolean);

This procedure enables/disables the use of the EOS char. The default is "True";

Domain: Prologix & DLL based

Procedure EZGPIB_BusSetEOSChar(How:Byte)

This procedure sets the EOS char. The default is "10" (Line Feed);

Domain: Prologix & DLL based

Function EZGPIB_ComCts(Which:Integer):Boolean

In addition to the serial port to which the Prologix interface is connected EZGPIB can handle as many serial ports for communication to other serial devices as Windows itself can handle. This function checks the condition of the CTS pin of port "Which".

Function EZGPIB_ComDsr(Which:Integer):Boolean

In addition to the serial port to which the Prologix interface is connected EZGPIB can handle a second serial port that you can use freely for your own communication with serial devices. This function checks the condition of the DTR pin of that port.

Function EZGPIB_ComOpen(Com:LongInt;Baudrate:LongInt;DataBits:LongInt;Parity:Char;StopBits:LongInt):Boolean

In addition to the serial port to which the Prologix interface is connected EZGPIB can handle a second serial port that you can use freely for your own communication with serial devices. This function opens the port and returns 'True' if that has been possible without an error. Note that you don't have to close serial ports that your application has opened. This is automatically done for you when your application terminates

Function EZGPIB_ComRead(Which:Integer):string

In addition to the serial port to which the Prologix interface is connected EZGPIB can handle as many serial ports for communication to other serial devices as Windows itself can handle. This function reads what is available in the serial input buffer of port "Which"

Function EZGPIB_ConvertHexToInt(HexString:string):string

Converts a string containing the hexadecimal representation of a number into a string with the decimal representation of that number

Function EZGPIB_ConvertStripToNumber(V:Variant):string

Strips everything away from a string that does not belong to a number representation.

Function EZGPIB_ConvertToDecimalComma(Where:string):string

Converts a string containing a number with a decimal point to a string with a decimal komma.

Function EZGPIB_ConvertToDecimalPoint(Where:string):string

Converts a string containing a number with a decimal komma to a string with a decimal point.

Function EZGPIB_ConvertToExponential(V:Variant;TotalLength:LongInt;ExponentLength:LongInt):string

Converts the variant value 'V' (may be string, integer or floating point) into a string using exponential format of total length 'TotalLength' and an length of the exponent of 'ExponentLength'.

Function EZGPIB_ConvertToFixed(V:Variant;digits:LongInt):string

Converts the variant value 'V' (may be string, integer or floating point) into a string using fixed point format using 'digits' digits.

Function EZGPIB_ConvertToFloatNumber(Which:string):Double

Converts a string to a real number.

Function EZGPIB_ConvertToIntNumber(Which:string):LongInt

Converts a string to a integer number

Function EZGPIB_ConvertToMJD(What:Variant):string

Converts the time/date value 'What' (may be string, integer or floating point) into a string giving the Modified Julian Date representation as the result.

Function EZGPIB_FileExists(Which:string):Boolean

Returns true if the file "Which" exists.

EZGPIB Manual

Function EZGPIB_FileReadClose:Boolean

Closes the text file that has been opened for reading and reports the result

Function EZGPIB_FileReadEof:Boolean

Returns the EOF (End of file) condition of a text file that has been opened for reading

Function EZGPIB_FileReadGetBuffer:string

Reads and returns the next line of the text file opened for reading

Function EZGPIB_FileReadOpen(Datafilename:string):Boolean

Opens a text file for reading and returns the result.

Function EZGPIB_KbdKeyPressed:Boolean

Returns 'True' if a key has been pressed.

Function EZGPIB_KbdReadKey:Char

Reads a key from the keyboard. You have to test yourself before whether a key has been pressed or not.

Function EZGPIB_KbdReadLn:Variant

Waits for a keyboard input that is terminated with a carriage return

Function EZGPIB_PortIn(Port:Word):Byte

Due to the use of the INOUT.DLL EZGPIB programs can perform direct port i/o. This function returns the current value of port 'Port'

Function EZGPIB_StringNthArgument(N:LongInt;Worin:string;Delimiter:Char):string

Returns the Nth argument from a string in which the individual arguments are separated by the delimiter character

Function EZGPIB_TelnetConnect(Name:string;IPAddress:string;Port:LongInt):Boolean

Opens a Telnet connection to ip-address IP and port PORT and returns whether the connect run ok. This connection receives the name "Name". You do not need to disconnect or close Telnet connections that your application has connected. This is done automatically when your application terminates.

Function EZGPIB_TelnetRead(Name:string):string

Reads all available data from the Telnet connection "Name"

Function EZGPIB_TimeNewSecond:Boolean

Returns 'True' if a new second has arrived since the last call of the function.

Function EZGPIB_TimeNow:TDateTime

Returns the current date and time in TDateTime format.

Procedure EZGPIB_BusAddressDevice(Which:LongInt)

Returns the currently addressed device on the GPIB. This is a low level function that directly relates to the ++addr command of the Prologix board.

Domain:Prologix

Procedure EZGPIB_BusAutoOff

This is a low level function that directly relates to the ++auto 0 command of the Prologix board.

Domain:Prologix

Procedure EZGPIB_BusAutoOn

This is a low level function that directly relates to the ++auto 1 command of the Prologix board

Domain:Prologix

Procedure EZGPIB_BusDisableEoi

This is a low level function that directly relates to the ++eoi 0 command of the Prologix board

Domain:Prologix

Procedure EZGPIB_BusEnableEoi

This is a low level function that directly relates to the ++eoi 1 command of the Prologix board

Domain:Prologix

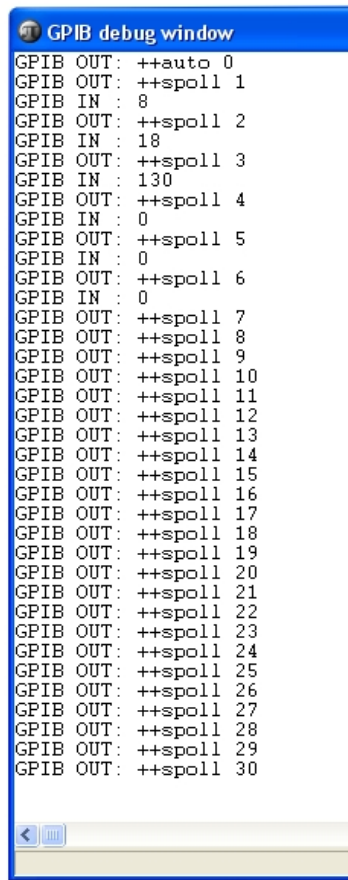
Procedure EZGPIB_BusFindAllDevices

This procedure will try to detect all active devices on the bus by reading their status byte. You can use the GPIB debug window to see what is going on: Every device in the range 0-30 is asked to report his status byte. Call this procedure at the start of every GPIB program that you write. It sets some internal variables of EZGPIB (for example the highest device address currently in use) to the correct value. **Every** device than can **deliver** a measurement value should have a status byte.

EZGPIB Manual

There may be some **pure output devices** that do not have a status byte and cannot be detected this way. I am aware of my Wavetek 278 function generator not having a status byte.

Domain:Prologix & DLL based



Picture 33

Procedure EZGPIB_BusGotoLocal(device;integer);

This is a low level function that directly relates to the ++loc command of the Prologix board

Domain:Prologix & DLL based

Procedure EZGPIB_BusIFC

Performs an Interface Clear

Domain:Prologix & DLL based

Procedure EZGPIB_LocalLockOut;

Performs an local lockout.

Domain:Prologix & DLL based

Procedure EZGPIB_BusSetEos(How:LongInt)

This is a low level function that directly relates to the ++eos 0/1/2/3 command of the Prologix board

Domain:Prologix

Procedure EZGPIB_BusSetTimeOut(How:Double)

Domain:Prologix & DLL based

Procedure EZGPIB_BusTrigger

Domain:Prologix & DLL based

Procedure EZGPIB_BusWriteData(Device:LongInt;What:string)

Sends the string 'What' to device 'Device' over the GPIB

Domain:Prologix & DLL based

EZGPIB Manual

Procedure EZGPIB_ChangeLed

Changes the state of the rightmost status leds. Use this to indicate that your script performs a certain line of code in a regular manner.

Procedure EZGPIB_ComSetBreak(Which:Integer,How:Boolean)

In addition to the serial port to which the Prologix interface is connected EZGPIB can handle as many serial ports for communication to other serial devices as Windows itself can handle. This procedure may be used to set the transmit data line of port "Which" statically to '0' or '1'

Procedure EZGPIB_ComSetDtr(Which:Integer, How:Boolean)

In addition to the serial port to which the Prologix interface is connected EZGPIB can handle as many serial ports for communication to other serial devices as Windows itself can handle. This procedure may be used to set the DTR line of port "Which" statically to '0' or '1'

Procedure EZGPIB_ComSetRts(Which:Integer,How:Boolean)

In addition to the serial port to which the Prologix interface is connected EZGPIB can handle as many serial ports for communication to other serial devices as Windows itself can handle. This procedure may be used to set the RTS line of port "Which" statically to '0' or '1'

Procedure EZGPIB_ComWrite(Which:Integer,What:string)

In addition to the serial port to which the Prologix interface is connected EZGPIB can handle as many serial ports for communication to other serial devices as Windows itself can handle. Outputs the string 'What' serially over the port "Which".

Procedure EZGPIB_ComWriteWithDelay(Which:Integer, What:string;DelayMS:LongInt)

In addition to the serial port to which the Prologix interface is connected EZGPIB can handle as many serial ports for communication to other serial devices as Windows itself can handle. Outputs the string 'What' serially over the port "Which" but with 'DelayMs' milliseconds pause between the characters.

Procedure EZGPIB_ConvertAddToString(Where:string;What:Variant)

Use this procedure to attach the variant value 'What' (may be string, integer or floating point) to the end of the string 'Where'.

Procedure EZGPIB_ConvertRemove(What:string;FromWhere:string)

Removes all instances of "What" in "Where"

Procedure EZGPIB_DDEServerAssignvalue(Item:string;Value:string)

EZGPIB can be a DDE server for other programs. Use this procedure to assign a DDE item (that has been created with EZGPIB_DDECreateItem) a value.

Procedure EZGPIB_DDEServerClearAll

EZGPIB can be a DDE server for other programs. Use this procedure to clear all previously created DDE items.

Procedure EZGPIB_DDEServerCreateItem(Which:string)

EZGPIB can be a DDE server for other programs. Use this procedure to create a new DDE item by the name of 'Which'.

Procedure EZGPIB_DebugWriteLn(s:Variant)

Outputs the variant value 's' (may be string, integer or floating point) to the debug & message window

Procedure EZGPIB_FileAddToBuffer(What:Variant)

EZGPIB features an easy mechanism for file output. Basically there is an internal file buffer variable which happens to be a string and represents one line of the data file. With this procedure you attach a new item that you want to write to the file at the end of the file buffer. A tab character will be automatically inserted into the file buffer before "What" is attached to the file buffer. With this procedure you construct yourself the next line to be written to the file. If the line is complete use EZGPIB_FileWrite to output the file buffer to the physical file.

Procedure EZGPIB_FileClearBuffer

This procedure clears the internal file buffer variable in order to construct a new output line.

Procedure EZGPIB_FileDelete(Which:string)

Deletes the file 'Which'. 'Which' may contain the path of the file

Procedure EZGPIB_FileExecute(WhichProgram:string;CMDParameters:string)

From within a EZGPIB program you can execute another program with the filename 'WhichProgram' and use the command line parameters 'CMDParameters'.

EZGPIB Manual

Procedure EZGPIB_FileWrite(Where:string)

Writes the file buffer to the file 'Where'. 'Where' may contain the path of the file. If the file does not exist it is automatically created (including the necessary path!). If the file exists then the file buffer is appended to the end of the file. After that the file is closed.

Procedure EZGPIB_PortOut(Port:Word;What:Byte)

Due to the use of the INOUT.DLL EZGPIB programs can perform direct port i/o. This procedure writes the value of byte 'What' to port 'Port'

Procedure EZGPIB_ScreenClear

Clears the console output screen and positions the cursor to line 1 position 1.

Procedure EZGPIB_ScreenClearEol

Clears the line where the cursor is currently positioned from the cursor position to the end

Procedure EZGPIB_ScreenCursorOff

Switches the console screen cursor off.

Procedure EZGPIB_ScreenCursorOn

Switches the console screen cursor on.

Procedure EZGPIB_ScreenGotoXY(x:LongInt;y:LongInt)

Positions the cursor to position x in line y

Procedure EZGPIB_ScreenWrite(s:Variant)

Write the variant value s ('What' (may be string, integer or floating point) at the current cursor position

Procedure EZGPIB_ScreenWriteLn(s:Variant)

Write the variant value s (may be string, integer or floating point) at the current cursor position and position the cursor at the start of the next line.

Procedure EZGPIB_TelnetWrite(Name:string;What:string)

Writes "What" to telnet connection "Name"

Procedure EZGPIB_TimeSleep(HowLong:Double)

Do nothing for 'HowLong' seconds.

Procedure EZGPIB_TimeWaitForMultipleOf(Seconds:LongWord)

Wait until a integer number of seconds has been reached. Example: EZGPIB_TimeWaitForMultipleOf(60) will wait until the start of the next minute, EZGPIB_TimeWaitForMultipleOf(1800) will wait until the start of the next half hour.

function EZGPIB_viOpenDefaultRM(var rm:Integer):Integer;

Opens a VISA session. Returns a handle in rm for that session that needs to be used in subsequent VISA function calls.

Function result is 0 when the call succeeds.

function EZGPIB_viOpen(RM:Integer;ResourceName:String;AccessMode:Integer;TimeOut:Integer;
var vi:Integer):Integer;

Opens a VISA connection to a single instrument. Returns a handle to this connection in vi that needs to be used in subsequent read/write and close calls.

Function result is 0 when the call succeeds.

function EZGPIB_viClose(VI:Integer):Integer;

Closes the connection to handle vi.

Function result is 0 when the call succeeds.

function EZGPIB_viRead(VI:Integer;var Buffer:String; var RetCount:integer):Integer;

Read the string buffer from VISA connection vi. Returns the number of chars read in retcount.

Function result is 0 when the call succeeds.

function EZGPIB_viWrite(VI:Integer;Buffer:String; var RetCount:integer):Integer;

Write the string buffer to VISA connection vi. Returns the number of chars written in retcount.

Function result is 0 when the call succeeds.

EZGPIB Functions and Procedures – Functional Grouping

Bus (Prologix and DLL)

Control and Setup

Query Parameters

Function EZGPIB_BusGetAddressedDevice:LongInt

Returns the address of the currently addressed bus device
Domain:Prologix

Function EZGPIB_BusGetTimeOut:Double

Queries the current setting of the timeout. This is a low level function directly related to the ++read_tmo_ms command of the Prologix board
Domain:Prologix

Function EZGPIB_BusSourceOfSrq:LongInt

Returns the address of the device that is requesting service. If several devices are requesting service the function will return the lowest device address that needs service. It is necessary that you have called “EZGPIB_BusFindAllDevices” once before using this function.
Domain:Prologix & DLL based

Function EZGPIB_BusGetEoi:LongInt

Queries the current setting of EOI. This is a low level function directly related to the ++eoi command of the Prologix board.
Domain:Prologix

Procedure EZGPIB_BusAddressDevice(Which:LongInt)

Returns the currently addressed device on the GPIB. This is a low level function that directly relates to the ++addr command of the Prologix board.
Domain:Prologix

Function EZGPIB_BusWaitForSrq(MaxWait:Double):Boolean

This functions waits ‘MaxWait’ seconds or until a SRQ takes place whichever is earlier. It returns ‘True’ if a SRQ has been noticed within the timeout.
Domain:Prologix & DLL based

Procedure EZGPIB_BusFindAllDevices

This procedure will try to detect all active devices on the bus by reading their status byte. You can use the GPIB debug window to see what is going on: Every device in the range 0-30 is asked to report his status byte. Call this procedure at the start of every GPIB program that you write. It sets some internal variables of EZGPIB (for example the highest device address currently in use) to the correct value.

Every device than can **deliver** a measurement value should have a status byte. There may be some **pure output devices** that do not have a status byte and cannot be detected this way. I am aware of my

Wavetek 278 function generator not having a status byte.

Domain:Prologix & DLL based

Set Adapter Behavior

Procedure EZGPIB_BusSetEos(How:Boolean);

This procedure enables/disables the use of the EOS char. The default is “True”;
Domain: Prologix & DLL based

Procedure EZGPIB_BusSetEOSChar(How:Byte)

This procedure sets the EOS char. The default is “10” (Line Feed);
Domain: Prologix & DLL based

Program Flow Functions

Function EZGPIB_BusSPoll(Device:LongInt):LongInt

Performs a serial poll to return the status byte of 'Device'

Domain:Prologix & DLL based

Function EZGPIB_BusSrq:Boolean

Returns 'True' if a device has requested service

Domain:Prologix & DLL based

Function EZGPIB_BusSrqStatus:LongInt

Returns the status byte of the service requesting device

Domain:Prologix & DLL based

Procedure EZGPIB_BusTrigger

Domain:Prologix & DLL based

Prologix Adapter Setup

Function EZGPIB_BusGetVer : String

Returns the adapter type and current software revision-related to ++ver command of the Prologix board.

Domain: Prologix

Procedure EZGPIB_BusAutoOff

This is a low level function that directly relates to the ++auto 0 command of the Prologix board.

Domain:Prologix

Procedure EZGPIB_BusAutoOn

This is a low level function that directly relates to the ++auto 1 command of the Prologix board

Domain:Prologix

Procedure EZGPIB_BusDisableEoi

This is a low level function that directly relates to the ++eoi 0 command of the Prologix board

Domain:Prologix

Procedure EZGPIB_BusEnableEoi

This is a low level function that directly relates to the ++eoi 1 command of the Prologix board

Domain:Prologix

Procedure EZGPIB_BusGotoLocal(device;integer);

This is a low level function that directly relates to the ++loc command of the Prologix board

Domain:Prologix & DLL based

Procedure EZGPIB_BusSetEos(How:LongInt)

This is a low level function that directly relates to the ++eos 0/1/2/3 command of the Prologix board

Domain:Prologix

Instrument Control

Procedure EZGPIB_BusIFC

Performs an Interface Clear

Domain:Prologix & DLL based

Procedure EZGPIB_LocalLockOut;

Performs an local lockout.

Domain:Prologix & DLL based

Procedure EZGPIB_BusSetTimeOut(How:Double)

Domain:Prologix & DLL based

Read from Bus

Function EZGPiB_BusDataAvailable: Boolean

Returns 'True' if any device on the bus has send data and has terminated it with the current delimiter
Domain:Prologix

Function EZGPiB_BusGetData:string

Returns the string that has last been sent by a device over the bus
Domain:Prologix

Function EZGPiB_BusWaitForData(Device:LongInt;ForWhat:string;MaxWait:Double):Boolean

Waits 'Maxwait' seconds or until 'Device' answers whichever is to take place earlier. If 'Device' answers within the timeout the answer is returned in 'ForWhat' and the function returns 'True'.The end of the message is detected by the delimiter sent by the device. Use this function for single measurement values.

Domain:Prologix & DLL based

Function EZGPiB_BusWaitForDataBlock(Device:LongInt;ForWhat:string;MaxWait:Double):Boolean

Waits 'Maxwait' seconds for an answer from 'Device'. If 'Device' answers within the timeout the answer is returned in 'ForWhat' and the function returns 'True'.This function does not look at delimiters that could be a part of the answer but waits the complete 'MaxWait' time. Use this function for longer data blocks as screen shots and so on.

Domain:Prologix & DLL based

Write to Bus

Procedure EZGPiB_BusWriteData(Device:LongInt;What:string)

Sends the string 'What' to device 'Device' over the GPIB
Domain:Prologix & DLL base

Serial Port Communications

Initial Port Setup

Function EZGPiB_ComOpen(Com:LongInt;Baudrate:LongInt;DataBits:LongInt;Parity:Char;StopBits:LongInt):Boolean

In addition to the serial port to which the Prologix interface is connected EZGPiB can handle a second serial port that you can use freely for your own communication with serial devices. This function opens the port and returns 'True' if that has been possible without an error. Note that you don't have to close serial ports that your application has opened. This is automatically done for you when your application terminates.

Read from Serial Port

Function EZGPiB_ComRead(Which:Integer):string

In addition to the serial port to which the Prologix interface is connected EZGPiB can handle as many serial ports for communication to other serial devices as Windows itself can handle. This function reads what is available in the serial input buffer of port "Which"

Write to Serial Port

Procedure EZGPiB_ComWrite(Which:Integer,What:string)

In addition to the serial port to which the Prologix interface is connected EZGPiB can handle as many serial ports for communication to other serial devices as Windows itself can handle. Outputs the string 'What' serially over the port "Which".

Procedure EZGPiB_ComWriteWithDelay(Which:Integer, What:string;DelayMS:LongInt)

In addition to the serial port to which the Prologix interface is connected EZGPiB can handle as many serial ports for communication to other serial devices as Windows itself can handle. Outputs the string 'What' serially over the port "Which" but with 'DelayMs' milliseconds pause between the characters.

Serial Port Control Pins

Function EZGPIB_ComCts(Which:Integer):Boolean

In addition to the serial port to which the Prologix interface is connected EZGPIB can handle as many serial ports for communication to other serial devices as Windows itself can handle. This function checks the condition of the CTS pin of port "Which".

Function EZGPIB_ComDsr(Which:Integer):Boolean

In addition to the serial port to which the Prologix interface is connected EZGPIB can handle a second serial port that you can use freely for your own communication with serial devices. This function checks the condition of the DTR pin of that port.

Procedure EZGPIB_ComSetBreak(Which:Integer,How:Boolean)

In addition to the serial port to which the Prologix interface is connected EZGPIB can handle as many serial ports for communication to other serial devices as Windows itself can handle. This procedure may be used to set the transmit data line of port "Which" statically to '0' or '1'

Procedure EZGPIB_ComSetDtr(Which:Integer, How:Boolean)

In addition to the serial port to which the Prologix interface is connected EZGPIB can handle as many serial ports for communication to other serial devices as Windows itself can handle. This procedure may be used to set the DTR line of port "Which" statically to '0' or '1'

Procedure EZGPIB_ComSetRts(Which:Integer,How:Boolean)

In addition to the serial port to which the Prologix interface is connected EZGPIB can handle as many serial ports for communication to other serial devices as Windows itself can handle. This procedure may be used to set the RTS line of port "Which" statically to '0' or '1'

String Functions

Numeric to String Conversion

Function EZGPIB_ConvertHextoInt(HexString:string):string

Converts a string containing the hexadecimal representation of a number into a string with the decimal representation of that number

Function EZGPIB_ConvertStripToNumber(V:Variant):string

Strips everything away from a string that does not belong to a number representation.

Function EZGPIB_ConvertToDecimalComma(Where:string):string

Converts a string containing a number with a decimal point to a string with a decimal comma.

Function EZGPIB_ConvertToDecimalPoint(Where:string):string

Converts a string containing a number with a decimal comma to a string with a decimal point.

Function

EZGPIB_ConvertToExponential(V:Variant;TotalLength:LongInt;ExponentLength:LongInt):string

Converts the variant value 'V' (may be string, integer or floating point) into a string using exponential format of total length 'TotalLength' and an length of the exponent of 'ExponentLength'.

Function EZGPIB_ConvertToFixed(V:Variant;digits:LongInt):string

Converts the variant value 'V' (may be string, integer or floating point) into a string using fixed point format using 'digits' digits.

String to Numeric

Function EZGPIB_ConvertToFloatNumber(Which:string):Double

Converts a string to a real number.

Function EZGPIB_ConvertToIntNumber(Which:string):LongInt

Converts a string to a integer number

Date to String

Function EZGPIB_ConvertToMJD(What:Variant):string

Converts the time/date value 'What' (may be string, integer or floating point) into a string giving the Modified Julian Date representation as the result. (Use EZGPIB_TimeNow to get the current date and time in TDateTime format.)

General

Function EZGPIB_StringNthArgument(N:LongInt;Worin:string;Delimiter:Char):string

Returns the Nth argument from a string in which the individual arguments are separated by the delimiter character

Procedure EZGPIB_ConvertAddToString(Where:string;What:Variant)

Use this procedure to attach the variant value 'What' (may be string, integer or floating point) to the end of the string 'Where'.

Procedure EZGPIB_ConvertRemove(What:string;FromWhere:string)

Removes all instances of "What" in "Where"

File I/O

General

Function EZGPIB_FileExists(Which:string):Boolean

Returns true if the file "Which" exists.

Procedure EZGPIB_FileDelete(Which:string)

Deletes the file 'Which'. 'Which' may contain the path of the file

Procedure EZGPIB_FileExecute(WhichProgram:string;CMDParameters:string)

From within a EZGPIB program you can execute another program with the filename 'WhichProgram' and use the command line parameters 'CMDParameters'.

Read from File

Function EZGPIB_FileReadClose:Boolean

Closes the text file that has been opened for reading and reports the result

Function EZGPIB_FileReadEOF:Boolean

Returns the EOF (End of file) condition of a text file that has been opened for reading

Function EZGPIB_FileReadGetBuffer:string

Reads and returns the next line of the text file opened for reading

Function EZGPIB_FileReadOpen(Datafilename:string):Boolean

Opens a text file for reading and returns the result.

Write to File

Procedure EZGPIB_FileAddToBuffer(What:Variant)

EZGPIB features an easy mechanism for file output. Basically there is an internal file buffer variable which happens to be a string and represents one line of the data file. With this procedure you attach a new item that you want to write to the file at the end of the file buffer. A tab character will be automatically inserted into the file buffer before "What" is attached to the file buffer. With this procedure you construct yourself the next line to be written to the file. If the line is complete use EZGPIB_FileWrite to output the file buffer to the physical file.

Procedure EZGPIB_FileClearBuffer

This procedure clears the internal file buffer variable in order to construct a new output line.

Procedure EZGPIB_FileWrite(Where:string)

Writes the file buffer to the file 'Where'. 'Where' may contain the path of the file. If the file does not exist it is automatically created (including the necessary path!). If the file exists then the file buffer is appended to the end of the file. After that the file is closed.

Keyboard Input

Function EZGPIB_KbdKeyPressed:Boolean

Returns 'True' if a key has been pressed.

Function EZGPIB_KbdReadKey:Char

Reads a key from the keyboard. You have to test yourself before whether a key has been pressed or not.

Function EZGPIB_KbdReadLn:Variant

Waits for a keyboard input that is terminated with a carriage return

Telnet

Function EZGPIB_TelnetConnect(Name:string;IPAddress:string;Port:LongInt):Boolean

Opens a Telnet connection to ip-address IP and port PORT and returns whether the connect run ok. This connection receives the name "Name". You do not need to disconnect ore close Telnet connections that your application has connected. This is done automatically when your application terminates.

Function EZGPIB_TelnetRead(Name:string):string

Reads all available data from the Telnet connection "Name"

Procedure EZGPIB_TelnetWrite(Name:string;What:string)

Writes "What" to telnet connection "Name"

Date/Time

Function EZGPIB_TimeNewSecond:Boolean

Returns 'True' if a new second has arrived since the last call of the function.

Function EZGPIB_TimeNow:TDateTime

Returns the current date and time in TDateTime format.

DDE

Procedure EZGPIB_DDEServerAssignvalue(Item:string;Value:string)

EZGPIB can be a DDE server for other programs. Use this procedure to assign a DDE item (that has been created with EZGPIB_DDECreateItem) a value.

Procedure EZGPIB_DDEServerClearAll

EZGPIB can be a DDE server for other programs. Use this procedure to clear all previously created DDE items.

Procedure EZGPIB_DDEServerCreateItem(Which:string)

EZGPIB can be a DDE server for other programs. Use this procedure to create a new DDE item by the name of 'Which'.

Miscellaneous

Debug Window

Procedure EZGPIB_DebugWriteLn(s:Variant)

Outputs the variant value 's' (may be string, integer or floating point) to the debug & message window

Port I/O

Procedure EZGPIB_PortOut(Port:Word;What:Byte)

Due to the use of the INOUT.DLL EZGPIB programs can perform direct port i/o. This procedure writes the value of byte 'What' to port 'Port'

Function EZGPIB_PortIn(Port:Word):Byte

Due to the use of the INOUT.DLL EZGPIB programs can perform direct port i/o. This function returns the current value of port 'Port'

LED

Procedure EZGPIB_ChangeLed

Changes the state of the rightmost status leds. Use this to indicate that your script performs a certain line of code in a regular manner.

Time Delay

Procedure EZGPIB_TimeSleep(HowLong:Double)

Do nothing for 'HowLong' seconds.

Procedure EZGPIB_TimeWaitForMultipleOf(Seconds:LongWord)

Wait until a integer number of seconds has been reached. Example:
EZGPIB_TimeWaitForMultipleOf(60) will wait until the start of the next minute,
EZGPIB_TimeWaitForMultipleOf(1800) will wait until the start of the next half hour.

Output to Console Screen

Writing Information to Screen

Procedure EZGPIB_ScreenWrite(s:Variant)

Write the variant value s (What' (may be string, integer or floating point) at the current cursor position

Procedure EZGPIB_ScreenWriteLn(s:Variant)

Write the variant value s (may be string, integer or floating point) at the current cursor position and position the cursor at the start of the next line.

Manipulating Screen

Procedure EZGPIB_ScreenClear

Clears the console output screen and positions the cursor to line 1 position 1.

Procedure EZGPIB_ScreenClearEol

Clears the line where the cursor is currently positioned from the cursor position to the end

Procedure EZGPIB_ScreenCursorOff

Switches the console screen cursor off.

Procedure EZGPIB_ScreenCursorOn

Switches the console screen cursor on.

Procedure EZGPIB_ScreenGotoXY(x:LongInt;y:LongInt)

Positions the cursor to position x in line y

VI Functionality

Open

Function EZGPIB_viOpenDefaultRM(var rm:Integer):Integer;

Opens a VISA session. Returns a handle in rm for that session that needs to be used in subsequent VISA function calls. Function result is 0 when the call succeeds.

Function

**EZGPIB_viOpen(RM:Integer;ResourceName:String;AccessMode:Integer;TimeOut:Integer;
var vi:Integer):Integer;**

Opens a VISA connection to a single instrument. Returns a handle to this connection in vi that needs to be used in subsequent read/write and close calls. Function result is 0 when the call succeeds.

Close

Function EZGPIB_viClose(VI:Integer):Integer;

Closes the connection to handle vi. Function result is 0 when the call succeeds.

Read

Function EZGPIB_viRead(VI:Integer;var Buffer:String; var RetCount:integer):Integer;

Read the string buffer from VISA connection vi. Returns the number of chars read in retcount. Function result is 0 when the call succeeds.

Write

Function EZGPIB_viWrite(VI:Integer;Buffer:String; var RetCount:integer):Integer;

Write the string buffer to VISA connection vi. Returns the number of chars written in retcount. Function result is 0 when the call succeeds.

History

- 20070330 Reception of large data blocks (screen plots) with the `BusWaitForDataBlock` routine improved a lot
- 20070531 Now handles an unlimited number of serial ports
 Now handles an unlimited number of telnet connections
 Can update the firmware of the Prologix controller starting with firmware 4.2
 Can make full use of the new `++read` command of the Prologix controller but stays backward compatible to version 3.12c
 Added `EZGPIB_ConvertRemove` procedure
 Added `EZGPIB_FileExists` function
 Added `EZGPIB_LocalLockout` procedure
 Renamed `EZGPIB_GTL` procedure to `EZGPIB_GotoLocal`
 Debug window now completely thread-driven.
- 20070821 Included functions and procedures for file **input** handling. Have a look to functions starting with **fileopen** and also note the **NthArgument** function. Completely rewritten to multi-threading
- 20071224 Support for DLL based interfaces added.
- 20080126 Some bugs removed
- 20080608 VISA compatibility added
- 20080619 Some EOS related bugs removed
- 20080802 Support for Prologix LAN GPIB interface built in. Device messages end detection is set to EOS as default with `<LF>` as the default EOS char.
- 20080809 Corrected some bugs that had come in with the introduction of the LAN GPIB interface.
- 20081129 Some bugs removed
- 20090213 Some bugs removed. Added the part “EZGPIB Functions and Procedures – Functional Grouping” to the manual. This part has been written by Jack Smith, K8ZOA. Thank you Jack in the name of all EZGPIB users.
- 20090531 A support for GPIB secondary addresses has been included. Read “GPIB Address Parameter and Sub Address Support” which has been written by David J. Holigan, daveh@essnh.com. Thank you Dave in the name of all EZGPIB users.

EZGPIB can now be started several times, if the .Exe files are located in different subdirectories. So one version can deal with a USB device, a second with a LAN based device and a third with a DLL-based device.

20091107 One of the biggest improvements of EZGPIB is that I have found a flawless way for communication between EZGPIB and ProfiLab. ProfiLab is a German (may be installed in English and French language too) based construction kit system for data acquisition systems that is in many aspects comparable to LABVIEW. However it has a much smaller footprint than LABVIEW, is a bit limited in it's capabilities against LABVIEW but costs less then 100 €!! Learn more about ProfiLab at

<http://www.abacom-online.de/uk/html/profilab-expert.html>

One of the few flaws of ProfiLab is that it does not provide GPIB communications. That makes it the ideal partner for EZGPIB which makes GPIB communication a snap. EZGPIB on the other hand lacks the rich set of graphic controls and displays that ProfiLab provides. They team up! Among other ways ProfiLab can communicate with hardware or other software by means of an "imported DLL". These imported DLLS appear on the working screen as a building block having a number of inputs and a number of outputs. I have written such a DLL for ProfiLab which can be configured to have 1 to 1000 inputs and 1 to 1000 outputs. Once the DLL is configured it can be connected to other ProfiLab components in the usual way. Nothing more needs to be obeyed in the ProfiLab project. The counter piece to that in EZGPIB is a set of two functions named

EZGPIB_ProflabOut(Output:Integer; Value:Double)

and

EZGPIB_ProflabIn(Input:Integer): Double

which directly operate on the ProfiLab building block by means of shared memory. The ProfiLab directory contains the necessary DLL that should be copied to ProfiLab's root directory together with it's INI-file. The Profilab directory also includes a ProfiLab demo project that matches the EZGPIB's ProfiLab.488 demo.

20121204 It has been reported by some users of EZGPIB that writing data to a file can be very time consuming with large data files, for example 20 minutes for a 23 MB file. As it turned out the EZGPIB_FileWrite(Filename) procedure was the bottleneck of its all since it opens the file then appends the data to the end of the file and hereafter closes it which is not economical with large files to say the least. Normally a snip of code that writes a data file would look something like this:

```
Repeat
  EZGPIB_FileClearBuffer;
  EZGPIB_FileAddtoBuffer(Var1);
  EZGPIB_FileAddtoBuffer(Var2);
  .
  .
  EZGPIB_FileAddtobuffer(VarN);
  EZGPIB_FileWrite(Filename);
Until AllDataWritten // Supply a condition of your own
```

EZGPiB Manual

In the new version you can use a EZGPiB_FileAddtoBuffer(#13+#10) to generate kind of a "new line" in the buffer where you now can add the next set of vars without being necessary to write out every line on its own. Instead you generate a big to very big buffer and write it out completely with EZGPiB_FileWrite(Filename) once.

So the new thing looks like:

```
EZGPiB_FileClearBuffer;
Repeat
  EZGPiB_FileAddtoBuffer(Var1);
  EZGPiB_FileAddtoBuffer(Var2);
  .
  .
  EZGPiB_FileAddtobuffer(VarN);
  EZGPiB_FileAddtoBuffer(#13+#10);
Until AllDataIsAddedToBuffer // Supply a condition of your own
EZGPiB_FileWrite(Filename);
```

and is ***much*** faster. Note that EZGPiB_FileClearBuffer and EZGPiB_FileWrite(Filename) are called only once per buffer.

20121217 A small bug which affected the first call of EZGPiB_FileAddtoBuffer after an EZGPiB_FileClearBuffer has been fixed