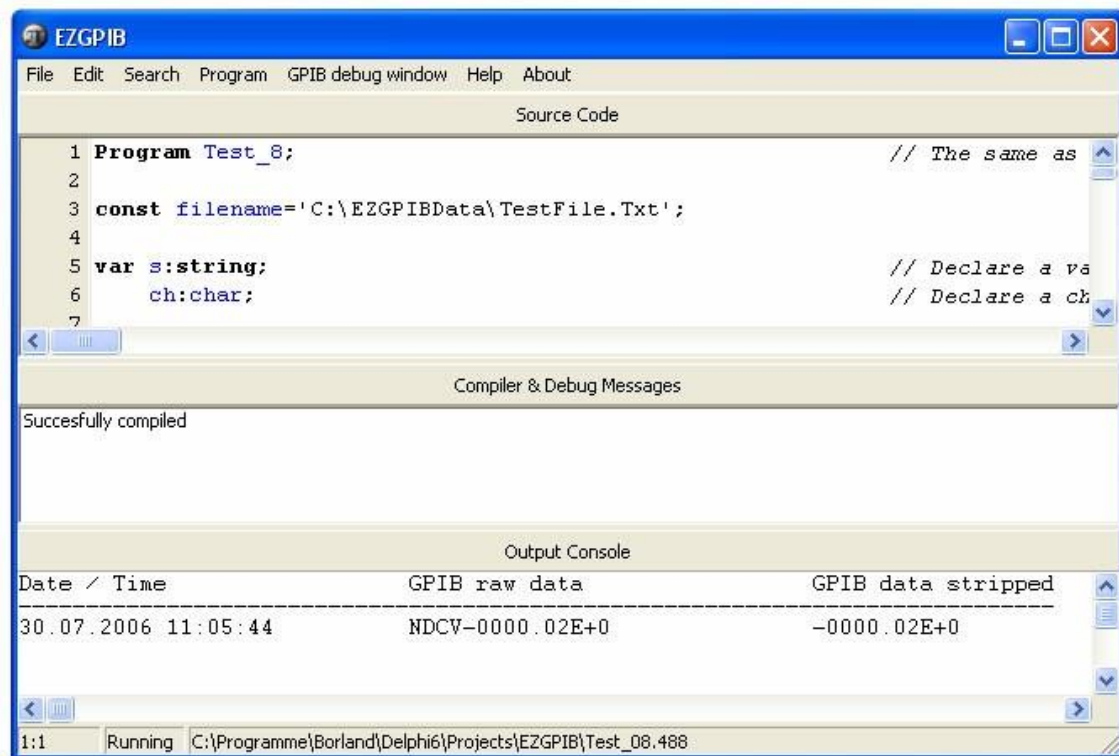


EZGPIB



A GPIB, RS232 and TCP/IP Data acquisition Tool

Brought to you by
Ulrich Bangert
df6jb@ulrich-bangert.de

this page intentionally left blank

	EZGPIB Manual	EZGPIB2_eng Rev: 2.00 Page 1/52
--	----------------------	---------------------------------------

Index

1 INTRODUCTION	3
1.1 OBITUARY FOR OM ULI BANGERT DF6JB.....	6
1.2 EZGPIB2.EXE	6
2 USER GUIDE	7
2.1 STARTING EZGPIB	7
2.1.1 What EZGPIB is.....	10
2.1.2 What EZGPIB is not.....	10
2.2 USING EZGPIB	11
2.2.1 Loading and running Scripts.....	11
2.2.2 Debugging Scripts.....	12
2.2.2.1 Console Window Debug Prints.....	12
2.2.2.2 Breakpoints (F5).....	13
2.2.2.3 Single Stepping (F7, F8 and F5+F9).....	15
2.2.3 We write a real data acquisition application.....	16
2.2.4 Data acquisition using serial ports.....	24
2.3 WHAT THE EXAMPLES DO	27
2.4 SOME NEW FEATURES	28
2.4.1 Multi Threading	28
2.4.2 GPIB32.DLL Support	29
2.4.3 VISA32.DLL Support.....	29
2.4.3.1 VISA Communication Demo.....	30
2.4.4 Prologix LAN GPIB Interface Support	31
2.4.5 EOI Detection Configuration.....	32
2.4.6 File Menu Enhancements	33
2.4.7 Implicit Variable 'UserFileName'	34
2.4.8 Single Step and Breakpoint Support	35
2.4.9 GPIB Address Parameter and Sub Address Support	35
3 EZGPIB FUNCTIONS AND PROCEDURES.....	37
3.1 BUS SETUP AND CONTROL (PROLOGIX AND DLL).....	37
3.1.1 Query Bus and Adapter Parameters	37
3.1.2 Set Bus and Adapter Parameters.....	39
3.2 PROGRAM FLOW FUNCTIONS.....	40
3.3 PROLOGIX ADAPTER SETUP	40
3.4 INSTRUMENT CONTROL	41
3.4.1 Read from Bus.....	41
3.4.2 Write to Bus	42
3.5 SERIAL PORT COMMUNICATIONS	42
3.5.1 Initial COM Port Setup	42
3.5.2 Read from Serial COM Port	42
3.5.3 Write to Serial Port.....	42
3.5.4 Get/Set Serial Port Control Pins.....	42
3.6 STRING FUNCTIONS.....	43
3.6.1 Numeric to String Conversion	43
3.6.2 String to Numeric	43
3.6.3 Date to String.....	44
3.6.4 General.....	44

	EZGPIB Manual	EZGPIB2_eng Rev: 2.00 Page 2/52
--	----------------------	---------------------------------------

3.7	FILE I/O.....	44
3.7.1	General.....	44
3.7.2	Read from File	44
3.7.3	Write to File.....	45
3.8	KEYBOARD INPUT	45
3.9	TELNET	45
3.10	DATE/TIME	46
3.11	DDE.....	46
3.12	MISCELLANEOUS	46
3.12.1	Debug Window.....	46
3.12.2	Port I/O	46
3.12.3	LED.....	46
3.13	TIME DELAY.....	47
3.14	OUTPUT TO CONSOLE SCREEN	47
3.14.1	Writing Information to Screen.....	47
3.14.2	Manipulating Screen	47
3.15	VI FUNCTIONALITY	48
3.15.1	Open.....	48
3.15.2	Close	48
3.15.3	Read.....	48
3.15.4	Write	48
4	SOFTWARE REVISION HISTORY	49
5	DOCUMENT REVISION HISTORY	52

1 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 1024x1024 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.

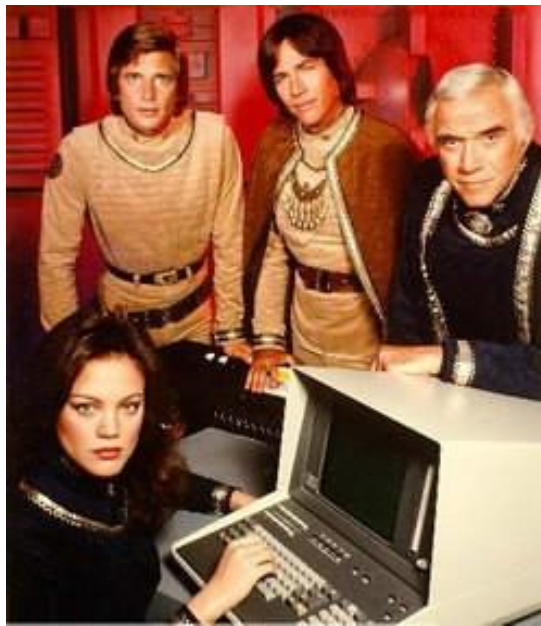
This storage screen was 'written' by a beam of electrons which affected a phosphor layer to constantly afterglow where the beam had impinged the layer. The horizontal and vertical deflection plates for that electron beam were driven by two 10 bits D/A converters which made it possible to focus the beam to any position of the screen with the mentioned resolution.

	EZGPIB Manual	EZGPIB2_eng Rev: 2.00 Page 4/52
--	----------------------	---------------------------------------

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.

	EZGPIB Manual	EZGPIB2_eng Rev: 2.00 Page 5/52
--	----------------------	---------------------------------------

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 adapter from PROLOGIX, available from <http://www.prologix.biz> which is shown in picture 3.

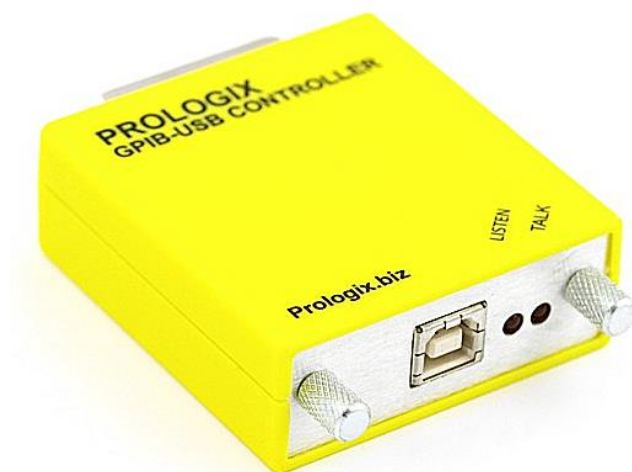
This cute little device uses a FTDI chip for USB to RS232 translation. That means that you install a simple FTDI driver available free from their web pages. This driver allocates a virtual RS232 port and you may use any software from ready to go terminal programs up to self-written stuff to communicate over this virtual RS232 with the ATMEL AVR microcontroller on the Prologix Adapter 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 adapter. 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 adapter are available for free. Google a bit for the American callsign KE5FX or search for the name "John Miles" in the web.

The actual version of Prologix GPIB-USB adapter looks like shown in Picture 4 and sells for approx. \$150,- which it is well worth due to the improvements against its predecessors. Alone the fact that it can safely be screwed to an IEEE-488 terminal and updated via the USB port is worth every cent of it.



Picture 4

1.1 Obituary for OM Uli Bangert DF6JB

Unfortunately the sad news is, that Ulrich Bangert, the creator of EZGPIB and many other extremely useful documentations and HAM-Radio gear like **EasyFax** passed away in June 2014, aged 59. His wife Ina passed away in May 2012.

Long time ago, I had several enjoyable contacts with him regarding his EasyFax converter, which I built and used extensively at that time. Finally he agreed to give me a copy of the complete source code of his project, in order to make my own software modifications and extensions for this famous converter.

I do hope that with this updated revision of the document I am able to contribute a little bit to his excellent work. I also translated this document into German language in order to make it easier for all German users which are not so fluent in English language ;-)).

Kater Karlo

1.2 EZGPIB2.EXE

My screen is a HiRes type and my old eyes are weak. So I started some tinkering inside the resource section of the executable EZGPIB.EXE by use of the famous freeware resource editor 'Resource Hacker' by Angus Johnson in order to improve the situation.

The executable EZGPIB2.EXE, which came along with this documentation, incorporates the following changes (hopefully they will be considered as improvements ;-))

- Initial window size after startup is now 1024x768
- Font size of all windows changed from 11 pix. to 16 pix. for better readability
- Font of Console Window changed to 'Courier New' for better readability
- About Box rearranged and resized as it was not readable (at least at my pc ;-))

2 User Guide

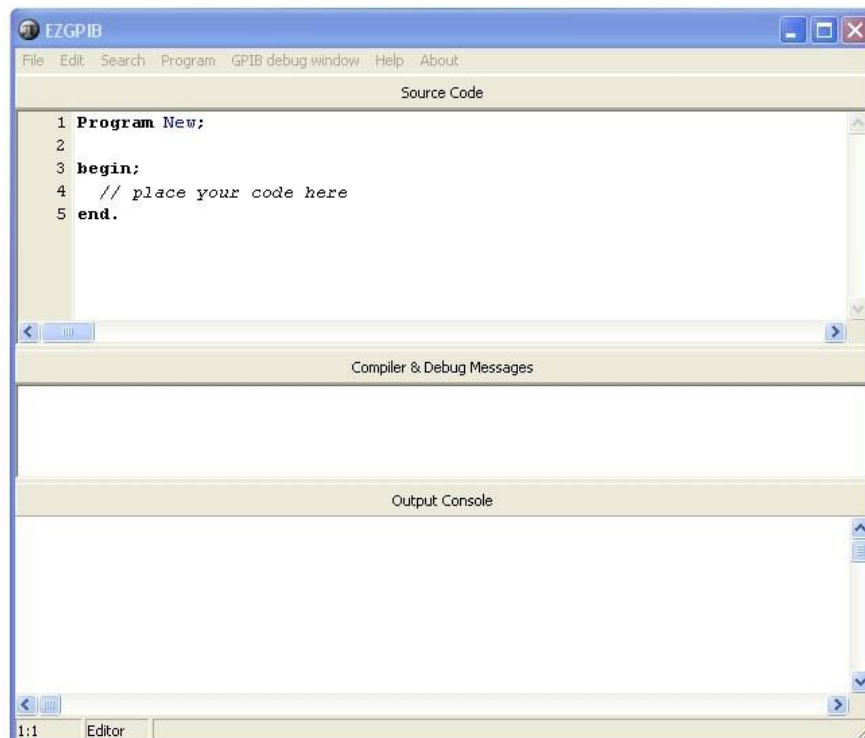
2.1 Starting EZGPIB

EZGPIB consists of a single executable file EZGPIB.EXE, the INOUT.DLL, some programming examples and this help file. EZGPIB runs under WIN 2000, XP, 7 and hopefully under newer versions too. Older versions have not been tested but are very likely to produce trouble.

On the old Prologix board, set DIP switches 1-5 to OFF and switch 6 to ON. That configures it as a GPIB bus controller, with a bus address of "0". The presence of a new (switchless) controller is detected automatically by EZGPIB and the necessary measures are taken.

When starting EZGPIB.EXE, the software will try to detect a Prologix adapter 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 or CH340 chip. If a FTDI or CH340 chip is detected, the software checks for an asserted CTS signal. If true, it sends a test string and a '++ver' cmd, to test whether a Prologix adapter answers with a version string containing the substring 'USB-GPIB' or not.

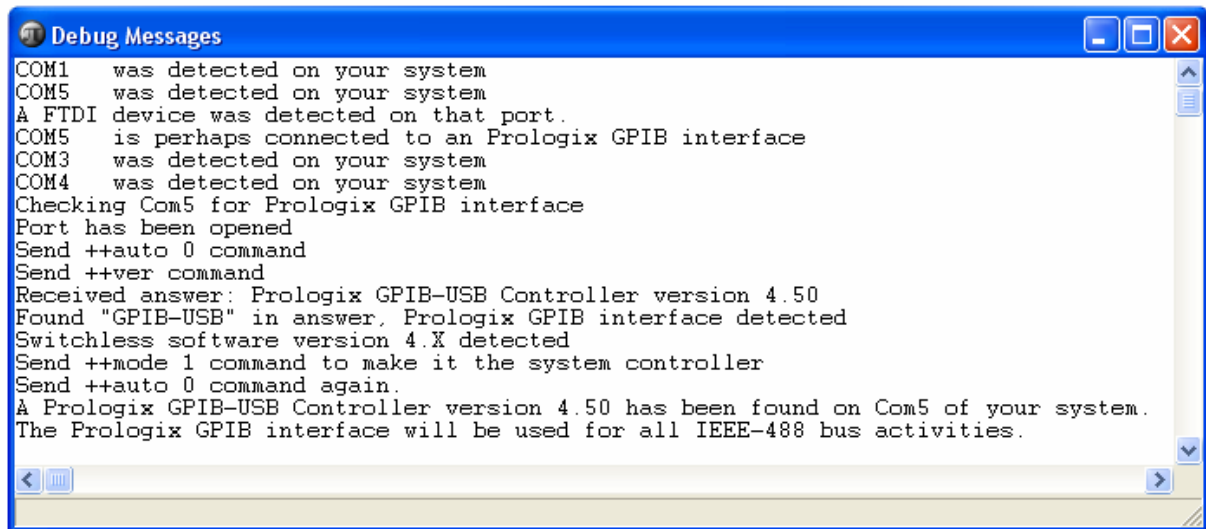
If no Prologix adapter is found, EZGPIB will try to detect whether a DLL named GPIB32.DLL is available on the PC, because that may indicate that the pc is equipped with a plugin GPIB adapter card. If the DLL is available, it is loaded and used to detect any GPIB adapter that is connected to the PC. That may be either a PCI-board or something different.



Picture 5

No matter how your PC is equipped, you are first confronted with 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 adapter or another GPIB adapter was found or why it was not found. When a Prologix adapter 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.

Note1: For a number of Prologix compatible adapters like AR488, it might happen that the CTS signal is NOT asserted by the USB interface chip. In this case, the best solution is to connect the CTS pin of the interface chip constantly to GND in order to make communication with EZGPIB possible! I strongly discourage the use of any CTS-patched versions of EZGPIB.

Note2: Even if a Prologix adapter is connected correctly to your system and a number of GPIB devices are connected correctly to the Prologix adapter, you may encounter the situation that the controller is not detected when starting EZGPIB.

In most cases this phenomenon is due to the following reason:

The Prologix adapter answers to serial commands only 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 adapter: If the **TALK** led beneath the power led stays constantly on after power up, 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.

	EZGPIB Manual	EZGPIB2_eng Rev: 2.00 Page 9/52
--	----------------------	---------------------------------------

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 text files, but gives them the extension “**.488**” to make them different from other text files.

On startup 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 ;-))

2.1.1 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 adapter 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.
- 4) If the editor part and 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!**

2.1.2 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!

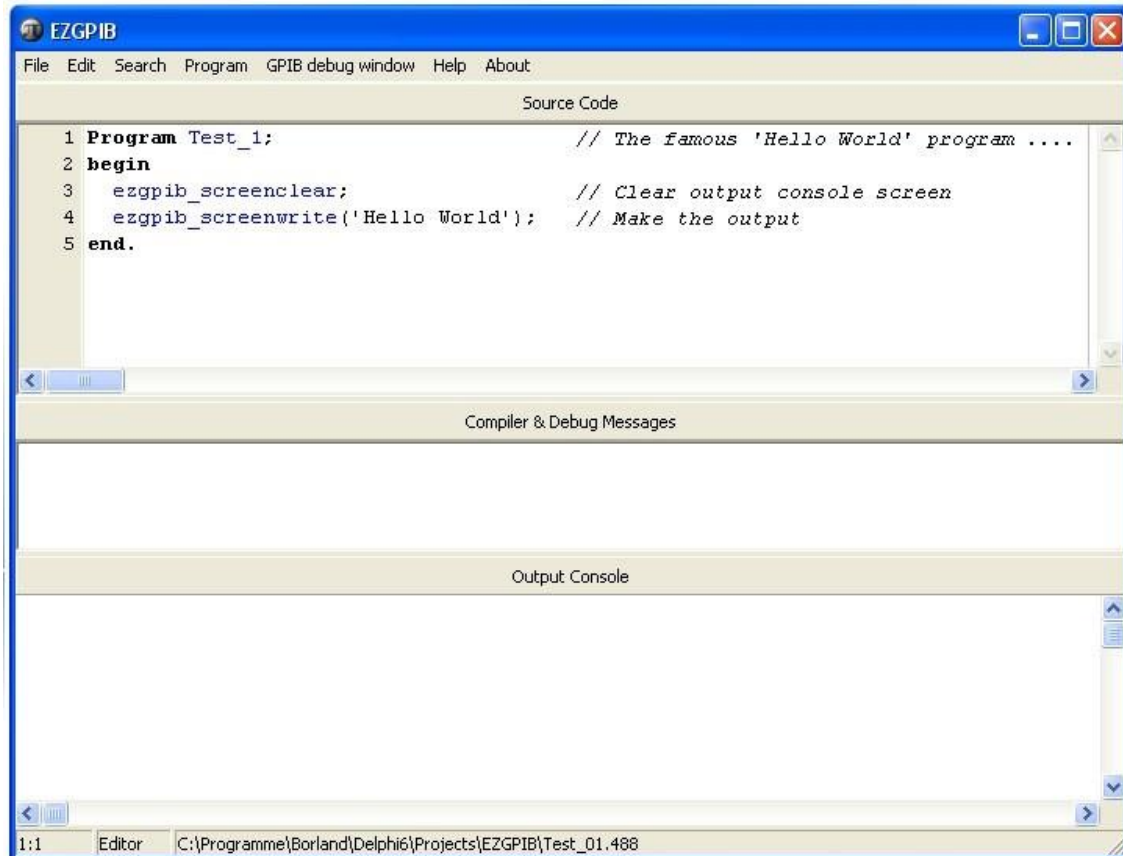
You will need a certain understanding for what your devices expect to receive over the GPIB in order perform a certain action or to send back a certain measurement result over the GPIB.

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.

2.2 Using EZGPIB

2.2.1 Loading and running Scripts

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:

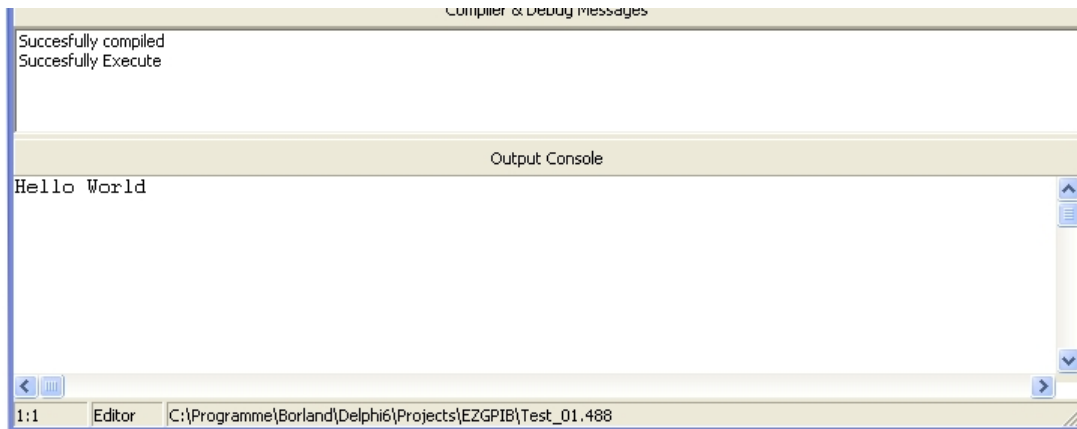


Picture 9

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

- 1) Some words are **black in a bold 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 in a standard font**. These are **constants** that cannot be assigned a specific group. In the example everything behind the *'//'* is a comment and **'Hello World'** is a **text constant**.
- 3) Some words are **blue**. These are **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 **EZGPIB 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 many available high level functions.

2.2.2 Debugging Scripts

In the very unlikely event, that you will note permanent or occasional suspicious behavior of your brand new script, EZGPIB provides some useful tools and strategies to deal with such situations. Please note that everything which is mentioned under chapters 2.2.2.n was not described in detail by the creator of the program, but is the result of 'trial and error' in the course of creating my own programs! Use this information at your own risk ;-))

2.2.2.1 Console Window Debug Prints

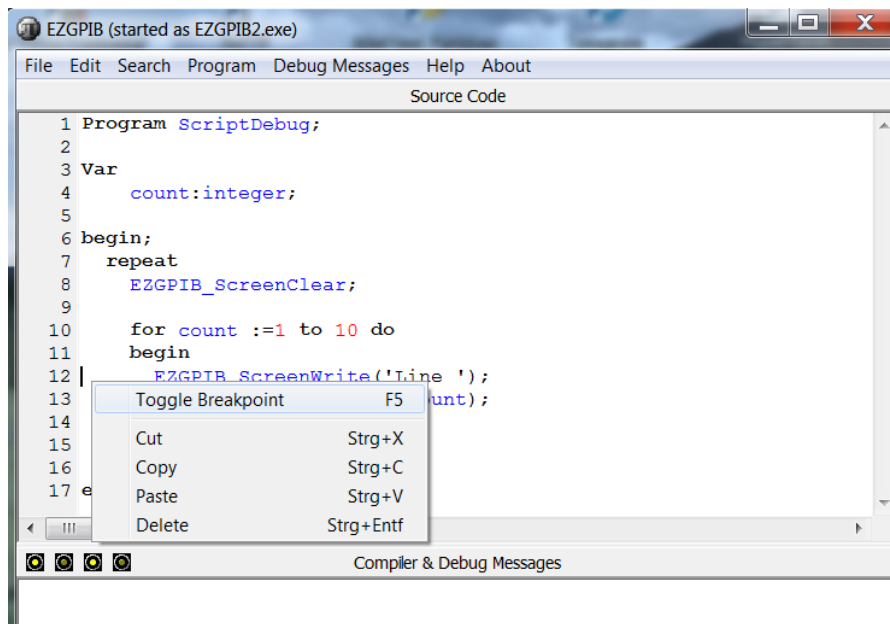
As a very straightforward method of 'hardcore' debugging, I would strongly encourage the reader to make extensive use of all routines, which are mentioned in chapter [3.14](#) of this documentation.

By use of procedure **EZGPIB_ScreenWriteLn** and various string converting routines, you will find an easy way to get information about what's going on with your program and variables while your script is running!

Unfortunately the script engine does NOT support conditional compiling, therefore you cannot use Turbo Pascal like definitions like `{ $DEFINE DEBUG }` to get rid of your debug printouts.

2.2.2.2 Breakpoints (F5)

EZGPIB provides the possibility to insert breakpoints at any executable line of your script like mentioned in chapter [2.4.8](#). Simply place the cursor the line of the 'Source Code' window, where you would like to stop the execution flow and right click with your mouse or on the mousepad. You will get a popup window showing you possible options:



By pressing F5, you will place a breakpoint at the cursor line, which is then shown in red color:

```
6 begin;  
7   repeat  
8     EZGPIB_ScreenClear;  
9  
10    for count :=1 to 10 do  
11      begin  
12        EZGPIB_ScreenWrite('Line ');  
13        EZGPIB_ScreenWriteLn(count);  
14        EZGPIB_Timesleep(0.2);  
15      end;  
16    until EZGPIB_KbdKeyPressed;  
17  end.
```

Of course you can place more than one breakpoint at a time on multiple lines of your script:

```
6 begin;  
7   repeat  
8     EZGPIB_ScreenClear;  
9  
10    for count :=1 to 10 do  
11      begin  
12        EZGPIB_ScreenWrite('Line ');  
13        EZGPIB_ScreenWriteLn(count);  
14        EZGPIB_Timesleep(0.2);  
15      end;  
16    until EZGPIB_KbdKeyPressed;  
17  end.
```

If you run the script now by pressing F9, it is automatically halted at the first breakpoint and will indicate that by inverting the line colors as shown here:

```
6 begin;
7   repeat
8     EZGPIB_ScreenClear;
9
10    for count :=1 to 10 do
11      begin
12        EZGPIB_ScreenWrite('Line ');
13        EZGPIB_ScreenWriteLn(count);
14        EZGPIB_Timesleep(0.2);
15      end;
16    until EZGPIB_KbdKeyPressed;
17 end.
```

Compiler & Debug Messages

Successfully compiled

Output Console

Note, that as shown above, nothing has been Written to the 'Output Console' window so far, which means, that **a breakpoint halts execution at the specified line, but the line itself will NOT have been EXECUTED**, when the program is halted!

If you continue the script from the breakpoint by pressing F9, the display will change as shown:

```
6 begin;
7   repeat
8     EZGPIB_ScreenClear;
9
10    for count :=1 to 10 do
11      begin
12        EZGPIB_ScreenWrite('Line ');
13        EZGPIB_ScreenWriteLn(count);
14        EZGPIB_Timesleep(0.2);
15      end;
16    until EZGPIB_KbdKeyPressed;
17 end.
```

Compiler & Debug Messages

Successfully compiled

Output Console

Line 1

You will note three things:

- 1) On the 'Output Console' the string 'Line 1' was printed, so lines 12 and 13 have been executed.
- 2) The script is halted at line 14, but note that the sleep instruction has not been executed!
- 3) The first breakpoint is still active and will halt the program again in line 12 when you continue execution with F9

Note, that key F5 has a toggle function. Breakpoints can be removed by bringing the cursor to the corresponding line and pressing F5 again. If you have multiple breakpoints, you can even do this while the running script is halted at another breakpoint.

If you toggle the breakpoint where the script is actually halted with F5, the line color will change to blue to indicate that you are changing the actual halting point!

```

6 begin;
7   repeat
8     EZGPiB_ScreenClear;
9
10    for count :=1 to 10 do
11      begin
12        EZGPiB_ScreenWrite('Line ');
13        EZGPiB_ScreenWriteLn(count);
14        EZGPiB_Timesleep(0.2);
15      end;
16    until EZGPiB_KbdKeyPressed;
17 end.
```

Finally I want to mention, that breakpoints are not only working in the main program, but can also be placed on lines inside your self-written procedures and functions.

2.2.2.3 Single Stepping (F7, F8 and F5+F9)

Instead of executing your script with F9 all at once, there is also a possibility to single step through your code line by line by use of F7 (Step Into) and F8. (Step Over).

Apparently there are two features which have not been updated / implemented by OM Uli:

- 1) While single stepping, it will NOT be indicated in the 'Source Code' window, where your script is actually halted (eg. by line color). This makes it hard for nontrivial scripts to memorize where you actually are.
- 2) Both keys F7 and F8 actually call the 'Step Into' routine. I have verified that when tinkering inside the resource section of the EZGPiB program. So it appears that 'Step Over' was never implemented.

There is a trick to overcome this, by simply using breakpoints set with F5 in EVERY LINE and 'single stepping' with the F9 key through every script line. Honestly, I don't know what the maximum number of breakpoints for a single script might be, but so far I did not run into a limit.

2.2.3 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 EZGPIB 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 EZGPIB 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 by using 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 11. 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 (despite the fact, that 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 EZGPB'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 EZGPB 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   EZGPB_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 adapter. 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 mentioned 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   EZGPiB_BusWriteData(Counter_Address, 'GA10');
7   repeat
8
9     until EZGPiB_KbdKeyPressed;
10 end.
```

Picture 16

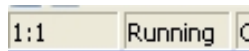
If you execute this program you will notice that the message window shows the information



Successfully compiled

Picture 17

and the status line shows



1:1 Running C

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 a value of 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:

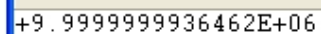
```

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:

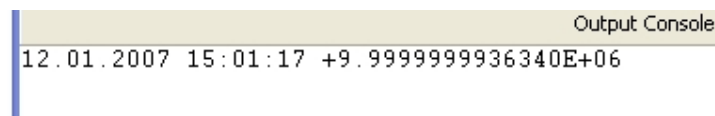
```

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

This script 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 Manual	EZGPIB2_eng Rev: 2.00 Page 23/52
--	----------------------	--

EZGPIB and the Prologix adapter 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.

2.2.4 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
```

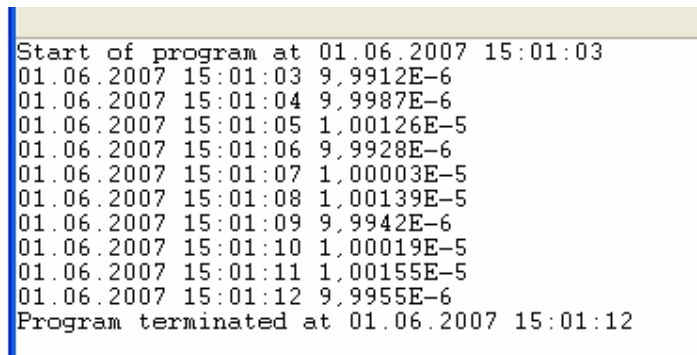
```
function Init:Boolean;
begin;
  if EZGPIB_FileExists(Filename)      //Comment this out if you
  then EZGPIB_FileDelete(Filename);   //always want to append
  EZGPIB_FileClearBuffer;             //Clear The Filebuffer
  EZGPIB_FileAddToBuffer('Time/MJD'); //Add two strings to FileBuffer
  EZGPIB_FileAddtoBuffer('TIC/s');
  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_ConvertStripToNumber(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;
```

```
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;
```

```
// main program loop
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.



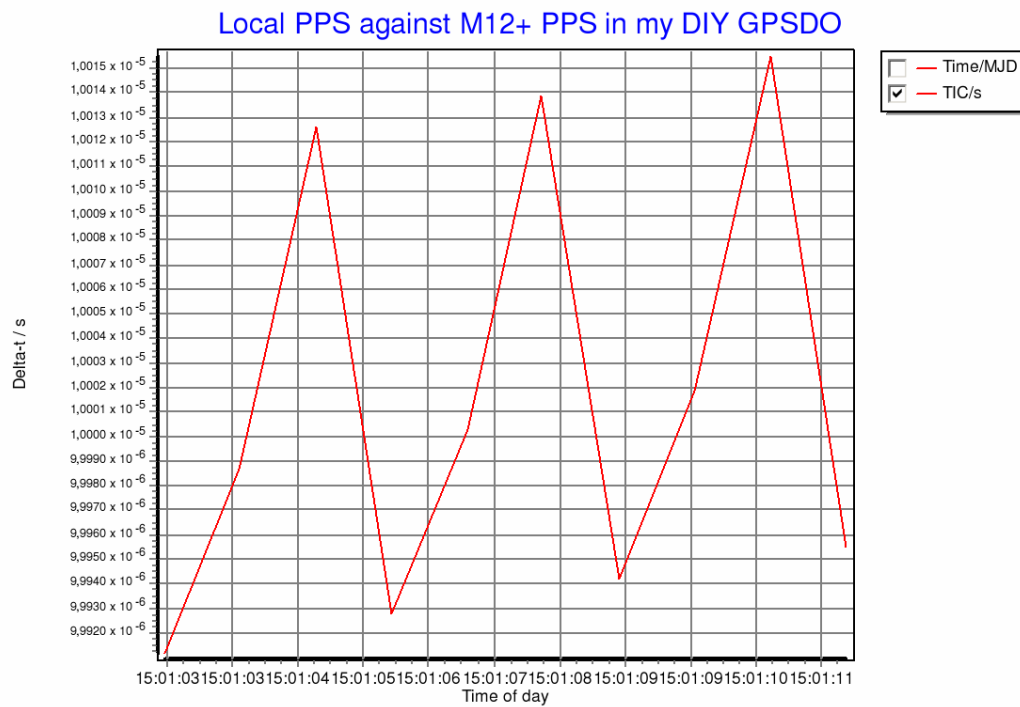
```
Start of program at 01.06.2007 15:01:03
01.06.2007 15:01:03 9,9912E-6
01.06.2007 15:01:04 9,9987E-6
01.06.2007 15:01:05 1,00126E-5
01.06.2007 15:01:06 9,9928E-6
01.06.2007 15:01:07 1,00003E-5
01.06.2007 15:01:08 1,00139E-5
01.06.2007 15:01:09 9,9942E-6
01.06.2007 15:01:10 1,00019E-5
01.06.2007 15:01:11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

	EZGPIB Manual	EZGPIB2_eng Rev: 2.00 Page 27/52
--	----------------------	--

2.3 What the Examples do

The following 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, which 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 (not possible with current version of MS-Windows!).

HP53131.488

Demonstrates serial communication with a HP53131.

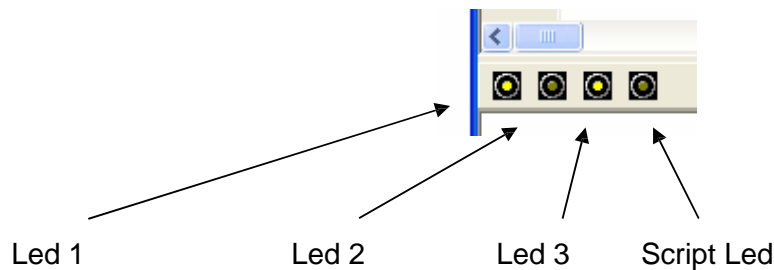
XXXX.488

I have included 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.

2.4 Some new features

2.4.1 Multi Threading

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 signals 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 adapter. This one signals its working condition by regular changing the state of **Led 2**.

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

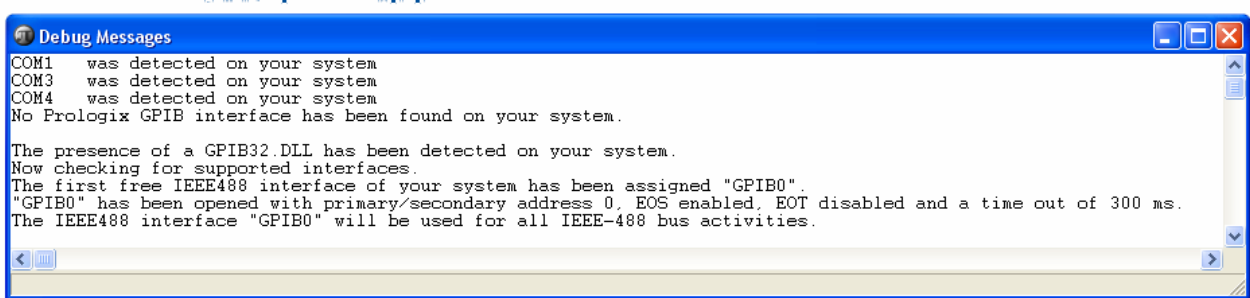
These three threads are active all of the time EZGPIB runs. So you should see all three led's blink all over the time EZGPIB is active.

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.

2.4.2 GPIB32.DLL Support

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.



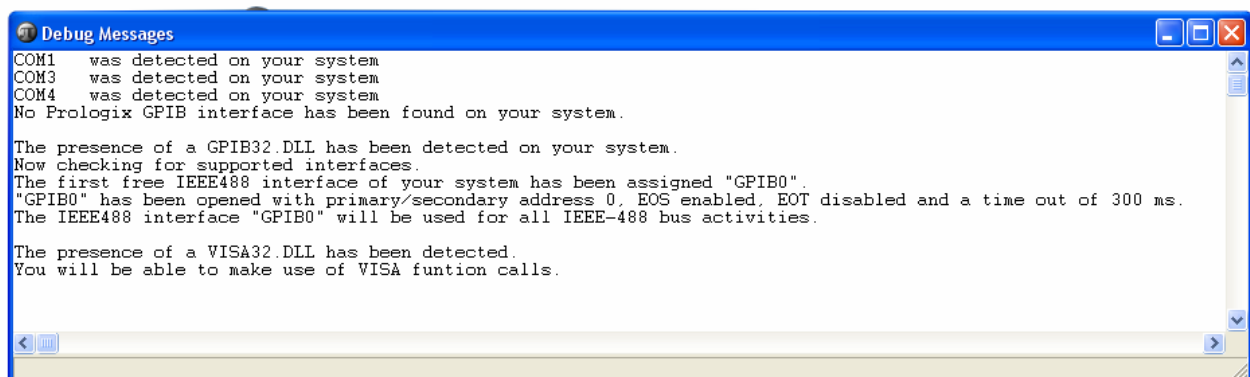
Picture 29

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 adapter as well as for the DLL based interface.

2.4.3 VISA32.DLL Support

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

2.4.3.1 VISA Communication Demo

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 power supply on address 9 of the bus and sets it to 12.5 Volt output. Then it closes the connection and connects to my HP3457 voltmeter on address 10 of the bus to read this voltage back.

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

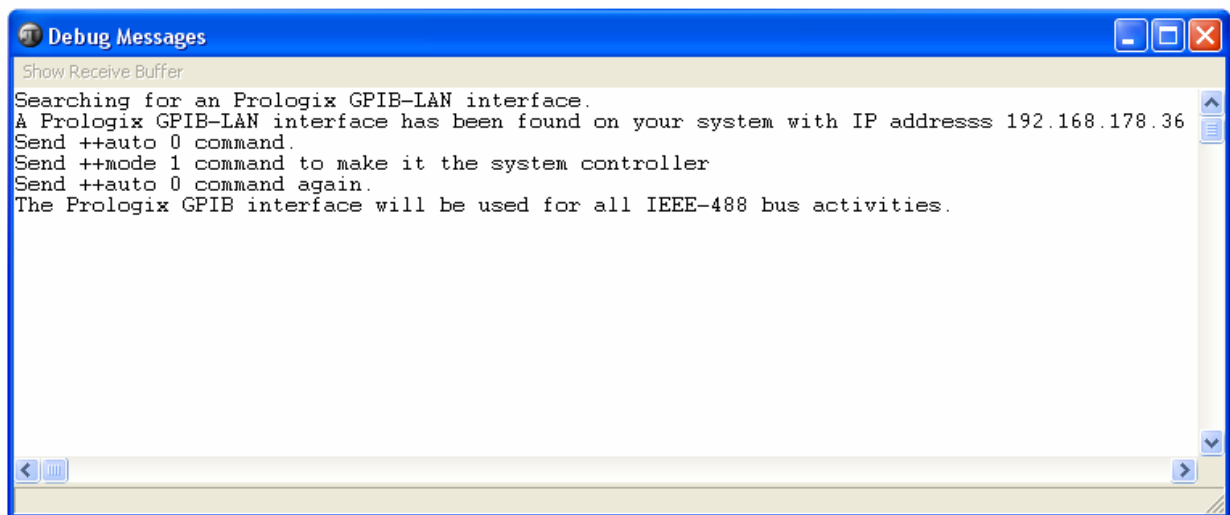
Note that EZGPIB can only detect the presence of the VISA32.DLL *but nothing else*. Use the tools that come together with the DLL to explore what interfaces and what instruments are available using the VISA driver.

2.4.4 Prologix LAN GPIB Interface Support

Starting on **version 2008-08-02**, the support for the new Prologix LAN GPIB adapter has been built in. After startup of EZGPIB, the search for a Prologix LAN GPIB adapter 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.

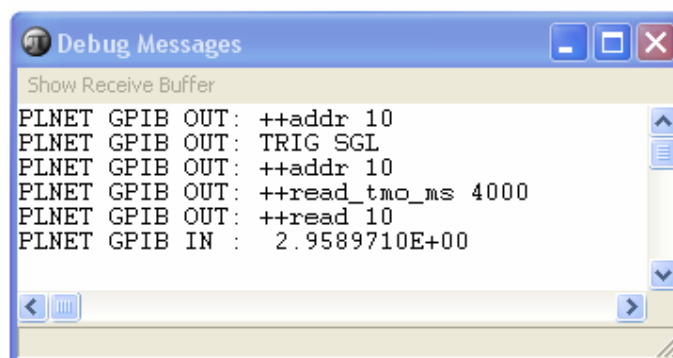
If the Prologix LAN GPIB adapter is configured to use DHCP, then this will usually happen when you connect the interface to your local network and a local DHCP server like a DSL router is available. If the interface is configured to make use of a fixed IP-address, you have to take care yourself for matching IP-address range. Consult the Prologix manual for answers to questions of network management, necessary for the LAN GPIB interface.

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



Picture 31

The IP-address displayed 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 plugin 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**'.

	EZGPIB Manual	EZGPIB2_eng Rev: 2.00 Page 32/52
--	----------------------	--

2.4.5 EOI Detection Configuration

Starting on version **2008-08-02**, the behavior 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.
- cannot 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 you are 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 one of the procedures below:

Procedure EZGPIB_BusSetEos(How:LongInt)

This procedure relates directly to the '++eos 0/1/2/3' command of the Prologix adapter.
Domain: Prologix

Procedure EZGPIB_BusSetEos(How:Boolean);

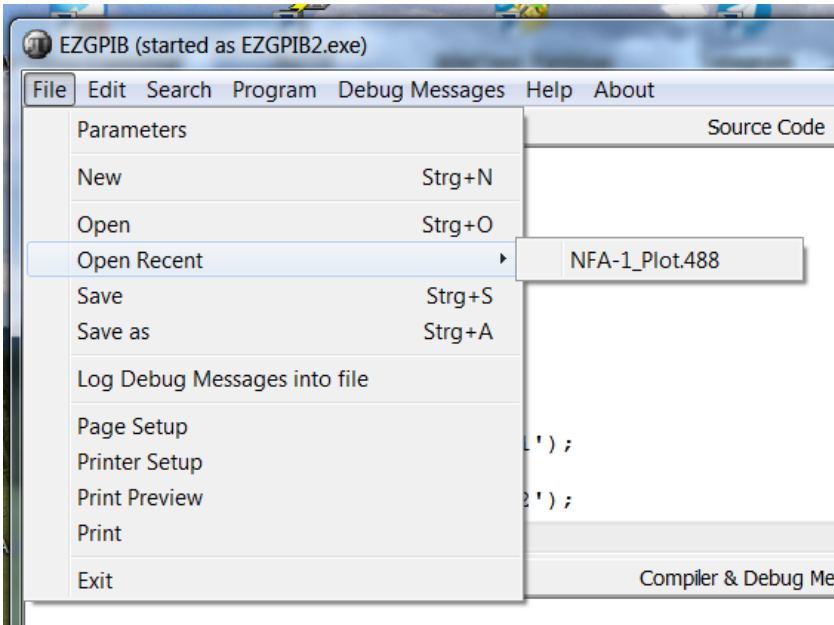
This procedure enables/disables the use of the EOS char. The default is 'TRUE';
Domain: DLL based

Procedure EZGPIB_BusSetEOSChar(How:Byte)

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

2.4.6 File Menu Enhancements

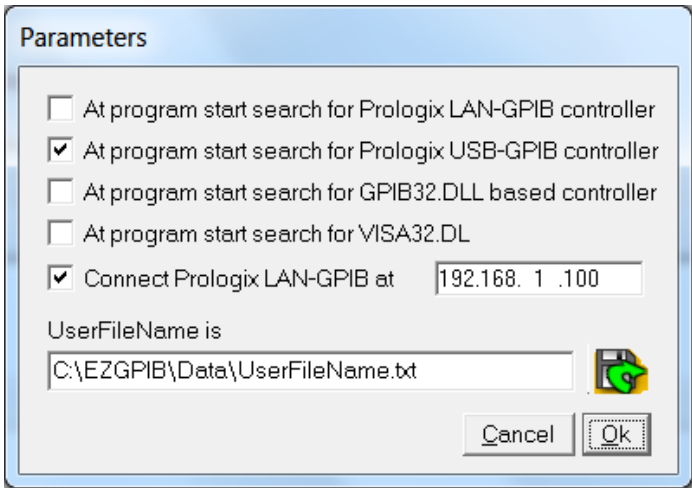
Starting with **version 2008-10-29**, the File menu has several additional entries. <Parameters> and <Log Debug Messages into file> will be explained in detail afterwards:



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 “EZGPIB2.exe_DebugLog.Txt”. Look for this file in the directory where EZGPIB2 itself is to be found.

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

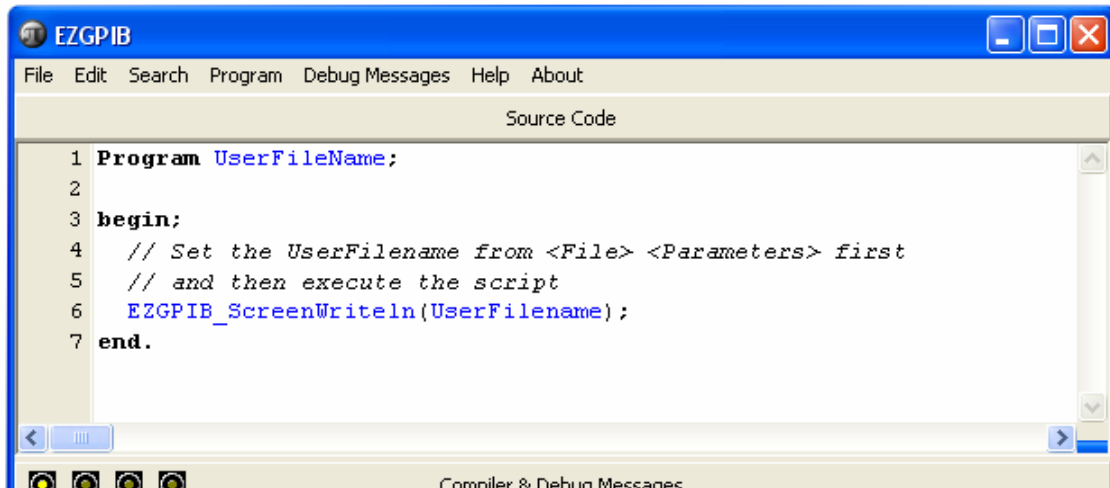


Picture 34

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

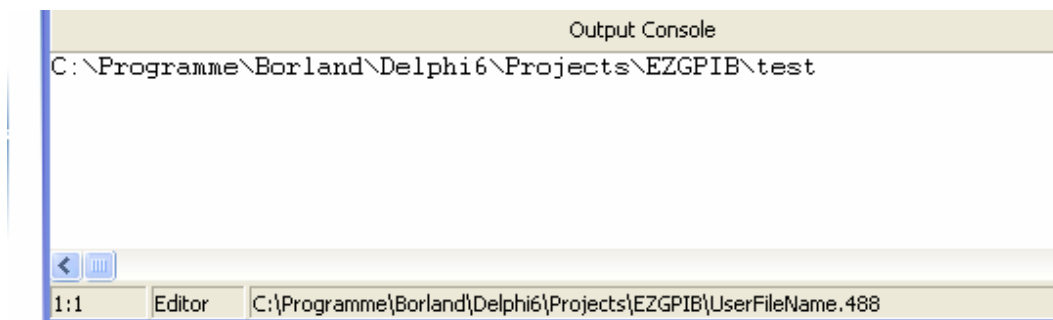
2.4.7 Implicit Variable 'UserFileName'

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



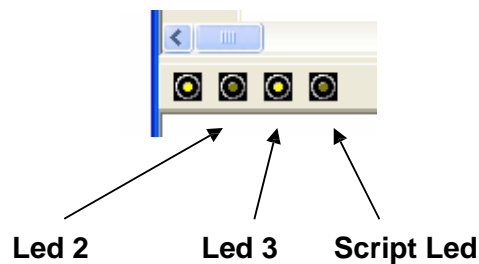
Picture 36

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

2.4.8 Single Step and Breakpoint Support

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 its own and cannot be interrupted by actions from the main thread.

Breakpoints can be set/cleared by placing the cursor in the respective line and pressing F5 several times.

2.4.9 GPIB Address Parameter and Sub Address Support

Normally the GPIB address parameter is 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 addressed with the sub address.

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

Example:

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

	EZGPIB Manual	EZGPIB2_eng Rev: 2.00 Page 36/52
--	----------------------	--

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 convention 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

	EZGPIB Manual	EZGPIB2_eng Rev: 2.00 Page 37/52
--	----------------------	--

3 EZGPIB Functions and Procedures

The following is a list of available functions and procedures that you can list with the Help menu entry. There are certain ones that have been programmed especially for EZGPIB.

The names of each dedicated function 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 and so on.

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_ entry.

3.1 Bus Setup and Control (Prologix and DLL)

3.1.1 Query Bus and Adapter Parameters

Procedure EZGPIB_BusFindAllDevices

This procedure will try to detect all active devices on the bus by reading their status byte. You can open the debug window to see what is going on. Every device in the range 0-30 is asked to report his status byte.

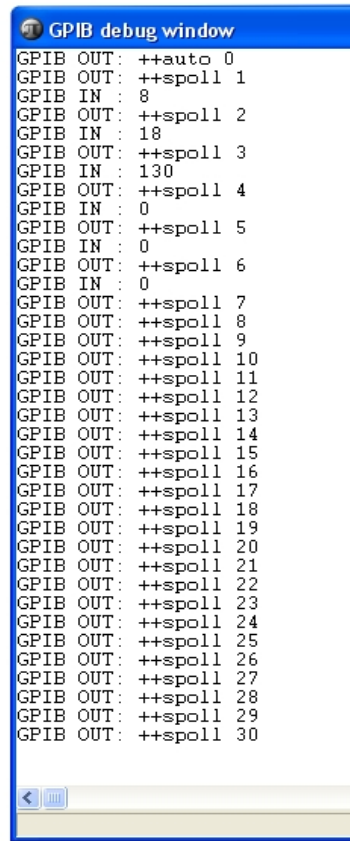
You should call this procedure at the start of every GPIB program that you write!

It sets some internal variables of EZGPIB used by other functions (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 (talk only)** that do not have a status byte and cannot be detected this way.

I found that the Wavetek 278 function generator does not have a status byte.
Domain:Prologix & DLL based

This is how the output in the debug window might look like, while executing the procedure 'EZGPIB_BusFindAllDevices', which scans the bus for available GPIB devices:



Picture 33

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 read timeout value in [ms] when reading from GPIB bus.
Range is between 0 and 32,000 milliseconds (32 seconds).
This function is directly related to the '+read_tmo_ms' command of the Prologix adapter.
Domain:Prologix

Function EZGPIB_BusGetEoi:LongInt

Queries the current setting of EOI signal assertion.
This function is directly related to the '++eoi' command of the Prologix adapter.
Domain:Prologix

	EZGPIB Manual	EZGPIB2_eng Rev: 2.00 Page 39/52
--	----------------------	--

Function EZGPIB_BusSourceOfSrq:LongInt

Returns the address of the device currently requesting service. If several devices are requesting service, the function will return the lowest device address that needs service.

It is mandatory to call '**EZGPIB_BusFindAllDevices**' once before using this function.

Domain:Prologix & DLL based

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

3.1.2 Set Bus and Adapter Parameters

Procedure EZGPIB_BusAddressDevice(Which:LongInt)

Sets the GPIB address to 'Which'. Meaning of the GPIB address depends on the operating mode of the controller. In CONTROLLER mode, it refers to the GPIB address of the instrument being controlled. In DEVICE mode, it is the address of the GPIB peripheral that the Prologix adapter is emulating. This function directly issues an '++addr Which' command to the Prologix adapter.

Note that EZGPIB does not include any special command to put the controller in device mode! In order set to device mode, you must send the command directly by calling:

EZGPIB_BusWriteData(DeviceAddr, '++mode 0');

where 'DeviceAddr' is the address of the GPIB peripheral that the controller is emulating.

Domain:Prologix

Procedure EZGPIB_BusSetEOS(How:Booelan);

This procedure enables/disables the use of an EOS GPIB termination char.

The default is 'TRUE'.

Domain: DLL based

Procedure EZGPIB_BusSetEOSChar(How:Byte)

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

Domain: DLL based

	EZGPIB Manual	EZGPIB2_eng Rev: 2.00 Page 40/52
--	----------------------	--

3.2 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_BusSrqr:Boolean

Returns 'TRUE' if a device has requested service.
Domain:Prologix & DLL based

Function EZGPIB_BusSrqrStatus:LongInt

Returns the status byte of the service requesting device.
Domain:Prologix & DLL based

Procedure EZGPIB_BusTrigger

Sends a Trigger command to the currently addressed instrument. The instrument needs to be set to single trigger mode and remotely controlled by the GPIB controller.
Domain:Prologix & DLL based

3.3 Prologix Adapter Setup

Function EZGPIB_BusGetVer:String

Returns the adapter type and current software revision returned by the '++ver' cmd of the Prologix adapter.
Domain: Prologix

Procedure EZGPIB_BusAutoOFF

This function directly relates to the '++auto 0' command of the Prologix adapter.
Domain:Prologix

Procedure EZGPIB_BusAutoON

This function directly relates to the '++auto 1' command of the Prologix adapter.
Domain:Prologix

Procedure EZGPIB_BusDisableEOI

This function directly relates to the '++eoi 0' command of the Prologix adapter.
Domain:Prologix

Procedure EZGPIB_BusEnableEOI

This function directly relates to the '++eoi 1' command of the Prologix adapter
Domain:Prologix

Procedure EZGPIB_BusGotoLocal(Device;integer)

This function directly relates to the '++loc' command of the Prologix adapter.
Domain:Prologix & DLL based

Procedure EZGPIB_BusSetEos(How:LongInt)

This function directly relates to the '++eos 0/1/2/3' command of the Prologix adapter
Domain:Prologix

	EZGPIB Manual	EZGPIB2_eng Rev: 2.00 Page 41/52
--	----------------------	--

3.4 Instrument Control

Procedure **EZGPIB_BusIFC**

Performs an Interface Clear on the GPIB bus.
Domain:Prologix & DLL based

Procedure **EZGPIB_LocalLockOut**

Performs a local lockout on the currently addressed instrument.
Domain:Prologix & DLL based

Procedure **EZGPIB_BusSetTimeout(How:Double)**

Sets the timeout value in [ms] when reading from GPIB bus.
Range is between 0 and 32,000 milliseconds (32 seconds).
Domain:Prologix & DLL based

3.4.1 Read from Bus

Function **EZGPIB_BusDataAvailable:Boolean**

Returns 'TRUE' if any device on the bus has sent data and has terminated it with the current EOI or EOS delimiter.
Domain:Prologix

Function **EZGPIB_BusGetData:string**

Returns the string that has last been sent over the bus by a device.
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 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 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 instead waits the complete 'MaxWait' time.

Use this function for longer data blocks like screen shots and so on.

Domain:Prologix & DLL based

	EZGPIB Manual	EZGPIB2_eng Rev: 2.00 Page 42/52
--	----------------------	--

3.4.2 Write to Bus

Procedure EZGPIB_BusWriteData(Device:LongInt;What:string)

Sends the string 'What' to device 'Device' over the GPIB bus.
Domain:Prologix & DLL based

3.5 Serial Port Communications

3.5.1 Initial COM 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 adapter is connected, EZGPIB can handle as many serial ports for communication to other serial devices as Windows itself can handle, 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 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.

3.5.2 Read from Serial COM Port

Function EZGPIB_ComRead(Which:Integer):string

This function reads what is available in the serial input buffer of COM port 'Which'.

3.5.3 Write to Serial Port

Procedure EZGPIB_ComWrite(Which:Integer;What:string)

This procedure outputs string 'What' to serial COM port 'Which'.

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

This procedure outputs string 'What' to serial COM port 'Which', but with 'DelayMs' milliseconds pause between the characters to deal with slow external devices.

3.5.4 Get/Set Serial Port Control Pins

Function EZGPIB_ComCTS(Which:Integer):Boolean

This function checks the condition of the CTS pin of COM port 'Which'.

Function EZGPIB_ComDSR(Which:Integer):Boolean

This function checks the condition of the DSR pin of COM port 'Which'.

Procedure EZGPIB_ComSetBreak(Which:Integer;How:Boolean)

This procedure may be used to set the transmit data line of COM port 'Which' statically to '0' or '1' in order to generate a 'break' condition on the serial line.

	EZGPiB Manual	EZGPiB2_eng Rev: 2.00 Page 43/52
--	----------------------	--

Procedure EZGPiB_ComSetDTR(Which:Integer;How:Boolean)

This procedure may be used to set the DTR line of COM port 'Which' permanently to '0' or '1' level.

Procedure EZGPiB_ComSetRTS(Which:Integer;How:Boolean)

This procedure may be used to set the RTS line of COM port 'Which' permanently to '0' or '1' level.

3.6 String Functions

3.6.1 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 decimal point to a string with a number with decimal comma.

Function EZGPiB_ConvertToDecimalPoint(Where:string):string

Converts a string containing a number with decimal comma to a string with a number with 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 the 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 with a length of 'digits' digits.

3.6.2 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 an integer number.

	EZGPIB Manual	EZGPIB2_eng Rev: 2.00 Page 44/52
--	----------------------	--

3.6.3 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).

3.6.4 General

Function EZGPIB_StringNthArgument(N:LongInt;Where: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 append 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 string 'What' in string 'Where'.

3.7 File I/O

3.7.1 General

Function EZGPIB_FileExists(Which:string):Boolean

Returns true if file 'Which' exists.

Procedure EZGPIB_FileDelete(Which:string)

Deletes file 'Which'. String 'Which' may also 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' using command line parameters 'CMDParameters'.

3.7.2 Read from File

Function EZGPIB_FileReadClose:Boolean

Closes the text file which is open for reading and reports the result.

Function EZGPIB_FileReadEOF:Boolean

Returns the EOF (End of file) condition of a text which is open for reading.

Function EZGPIB_FileReadGetBuffer:string

Reads and returns the next line of the text file which is open for reading.

Function EZGPIB_FileReadOpen(Datafilename:string):Boolean

Opens a text file for reading and returns the result.

	EZGPIB Manual	EZGPIB2_eng Rev: 2.00 Page 45/52
--	----------------------	--

3.7.3 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 append 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 appended to the file buffer. With this procedure you assemble 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 remove previous lines and produce a new output line.

Procedure EZGPIB_FileWrite(Where:string)

Writes the file buffer to 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, the file buffer is appended to the end of the file. After that, the file is closed.

Please note, that due to the operating system overhead, this is a slow acting function.

3.8 Keyboard Input

Function EZGPIB_KbdKeyPressed:Boolean

Returns 'TRUE' if a key has been pressed. Used together with the following function.

Function EZGPIB_KbdReadKey:Char

Reads a key from the keyboard. You have to test yourself before, whether a key has been pressed or not with '**EZGPIB_KbdKeyPressed**'.

Function EZGPIB_KbdReadLn:Variant

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

3.9 Telnet

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

Opens a Telnet connection to IP-address 'IPAddress' and port 'Port' and returns whether the connect run ok. This connection is given 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 active Telnet connection 'Name'.

Procedure EZGPIB_TelnetWrite(Name:string;What:string)

Writes string 'What' to active telnet connection 'Name'.

	EZGPIB Manual	EZGPIB2_eng Rev: 2.00 Page 46/52
--	----------------------	--

3.10 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.

3.11 DDE

Procedure EZGPIB_DDEServerCreateItem(Item:string)

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

Procedure EZGPIB_DDEServerAssignvalue(Item:string;Value:string)

Use this procedure to assign a DDE item (that has been created with EZGPIB_DDEServerCreateItem) a value.

Procedure EZGPIB_DDEServerClearAll

Use this procedure to clear all previously created DDE items.

3.12 Miscellaneous

3.12.1 Debug Window

Procedure EZGPIB_DebugWriteLn(V:Variant)

Outputs the variant value 'V' (may be string, integer or floating point) to the 'Debug Messages' window.

This apparently produces no output in the debug window? (at least on my pc ;-))

3.12.2 Port I/O

Direct port I/O is very likely causing troubles on actual operating system versions and will obviously not operate on all 64 bit windows versions.

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' if operational.

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' if operational.

3.12.3 LED

Procedure EZGPIB_ChangeLed

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

	EZGPIB Manual	EZGPIB2_eng Rev: 2.00 Page 47/52
--	----------------------	--

3.13 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.

3.14 Output to Console Screen

3.14.1 Writing Information to Screen

Procedure EZGPIB_ScreenWrite(V:Variant)

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

Procedure EZGPIB_ScreenWriteLn(V:Variant)

Write the variant value 'V' (may be string, integer or floating point) at the current cursor position and moves the cursor to the begin of the next line.

3.14.2 Manipulating Screen

Procedure EZGPIB_ScreenClear

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

Procedure EZGPIB_ScreenClearEol

Clears the actual cursor line from the cursor position to the end of line.

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 console screen cursor to position 'x' in line 'y'.

	EZGPIB Manual	EZGPIB2_eng Rev: 2.00 Page 48/52
--	----------------------	--

3.15 VI Functionality

3.15.1 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.

3.15.2 Close

Function **EZGPIB_viClose(VI:Integer):Integer;**

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

3.15.3 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.

3.15.4 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.

4 Software Revision 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 adapter starting with firmware 4.2.
Can make full use of the new ++read command of the Prologix adapter, but stays backward compatible to firmware 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 at 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" to the manual. This part has been written by Jack Smith, K8ZOA. Thank you Jack in the name of all EZGPIB users!

	EZGPIB Manual	EZGPIB2_eng Rev: 2.00 Page 50/52
--	----------------------	--

20090531 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 its capabilities against LABVIEW but costs less than 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 its INI-file. The ProfiLab directory also includes a ProfiLab demo project that matches the EZGPIB’s ProfiLab.488 demo.

	EZGPIB Manual	EZGPIB2_eng Rev: 2.00 Page 51/52
--	----------------------	--

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 condition of your own
```

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 AllDataAddedToBuffer // Supply 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.

04/01/2021 EZGPIB.EXE resources patched to initial windows size of 1024x768 and use of bigger fonts, to better support HiRes displays and my old eyes ;-)).
Executable file renamed to EZGPIB2.EXE.

5 Document Revision History

04/01/2021 **German document created from the original document dated 12/17/2012.**

Paragraph structure created and table of contents inserted.

Obituary for Uli Bangert DF6JB added.

Pictures in chapter [2.4.6](#) 'File Menu Enhancements' updated.

Function and procedure descriptions merged into one list.

Chapter [2.2.2](#) '**Debugging Scripts**' added.

Description of procedure **EZGPIB_BusAddressDevice** in chapter [3.1.2](#) corrected, added some hints to switch controller into 'device mode'.

Version	Date	Changes	by
1.nn	12/17/2012	Last version edited by OM Ulrich Bangert	DF6JB
2.00	04/01/2021	Miscellaneous editorial changes and updates	KaKa