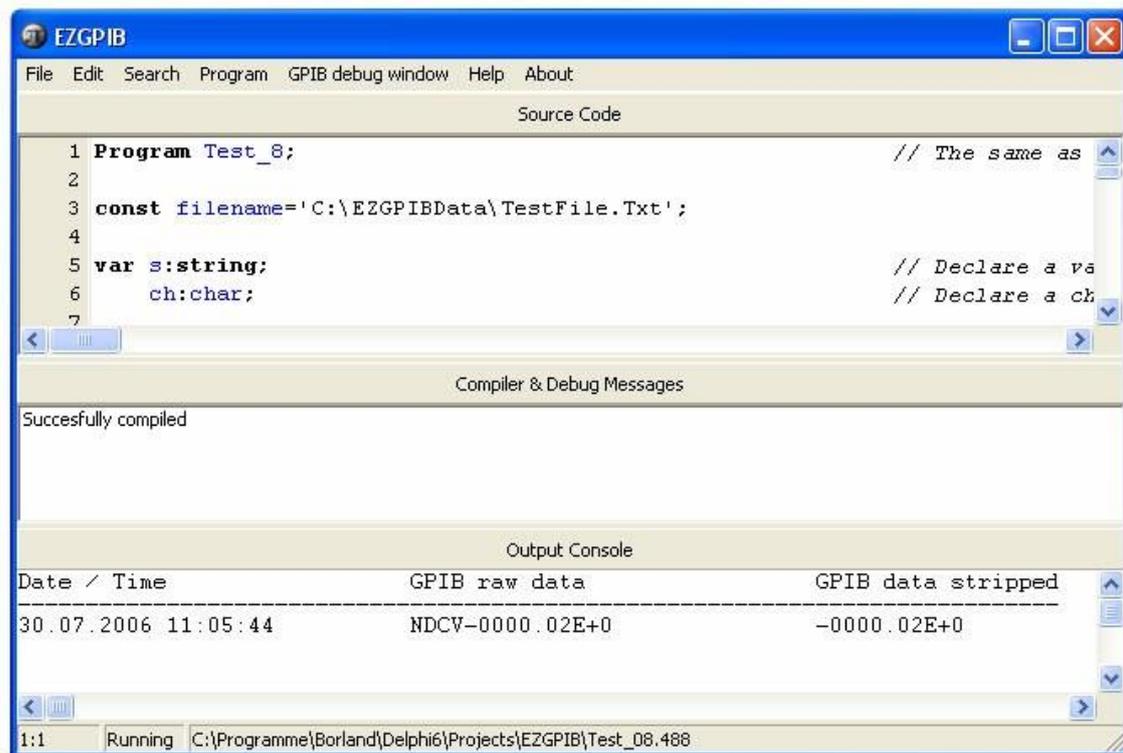


EZGPIB



The screenshot shows the EZGPIB application window with a menu bar (File, Edit, Search, Program, GPIB debug window, Help, About) and three panes:

- Source Code:** Contains Delphi code for a program named Test_8. The code includes a constant filename, variable declarations for a string and a character, and comments.
- Compiler & Debug Messages:** Displays the message "Successfully compiled".
- Output Console:** Shows a table of GPIB data.

```
1 Program Test_8; // The same as
2
3 const filename='C:\EZGPIBData\TestFile.Txt';
4
5 var s:string; // Declare a va
6     ch:char; // Declare a ch
7
```

Date / Time	GPIB raw data	GPIB data stripped
30.07.2006 11:05:44	NDCV-0000.02E+0	-0000.02E+0

1:1 Running C:\Programme\Borland\Delphi6\Projects\EZGPIB\Test_08.488

Ein GPIB, RS232 und TCP/IP Datenerfassungstool

Zur Verfügung gestellt von
Ulrich Bangert
df6jb@ulrich-bangert.de

Leere Seite

Inhaltsverzeichnis

1 EINFÜHRUNG	3
1.1 NACHRUF FÜR OM ULI BANGERT DF6JB.....	7
1.2 EZGPIB2.EXE.....	7
2 BENUTZERHANDBUCH	8
2.1 EZGPIB STARTEN	8
2.1.1 Was EZGPIB ist.....	11
2.1.2 Was EZGPIB nicht ist.....	11
2.2 EZGPIB BENUTZEN	12
2.2.1 Skripte Laden und Ausführen.....	12
2.2.2 Skript Debugging	13
2.2.2.1 Debug Ausgaben im Konsolenfenster.....	13
2.2.2.2 Haltepunkte (F5).....	14
2.2.2.3 Einzelschritte (F7, F8 und F5+F9).....	17
2.2.3 Wir schreiben eine echte Datenerfassungsanwendung.....	17
2.2.4 Datenerfassung über serielle Schnittstellen	26
2.3 WAS DIE BEISPIELE ZEIGEN	29
2.4 EINIGE NEUE FUNKTIONEN	30
2.4.1 Multi Threading	30
2.4.2 GPIB32.DLL Unterstützung.....	31
2.4.3 VISA32.DLL Support.....	31
2.4.3.1 VISA Kommunikationsbeispiel	32
2.4.4 Unterstützung der Prologix LAN-GPIB Schnittstelle	33
2.4.5 Konfiguration der 'EOI' Erkennung.....	34
2.4.6 Verbesserungen im Menü 'File'	35
2.4.7 Implizite Variable 'UserFileName'.....	36
2.4.8 Einzelschritt- und Haltepunktunterstützung	37
2.4.9 Unterstützung von GPIB Haupt- und Subadressen	37
3 EZGPIB FUNKTIONEN UND ROUTINEN.....	39
3.1 BUSKONFIGURATION UND -STEUERUNG (PROLOGIX UND DLL).....	39
3.1.1 Abfrage von Bus- und Adapterparametern	39
3.1.2 Setzen von Bus- und Adapterparametern	41
3.2 PROGRAM FLOW FUNCTIONS.....	42
3.3 PROLOGIX ADAPTER EINSTELLUNGEN.....	42
3.4 GERÄTESTEUERUNG	43
3.4.1 Lesen vom Bus	43
3.4.2 Schreiben am Bus.....	44
3.5 KOMMUNIKATION ÜBER SERIELLE SCHNITTSTELLEN.....	44
3.5.1 Einrichten des seriellen Anschluss.....	44
3.5.2 Lesen vom seriellen Anschluss	44
3.5.3 Schreiben zum seriellen Anschluss.....	44
3.5.4 Serielle Schnittstelle Steuerpins abfragen/setzen.....	44
3.6 STRING FUNKTIONEN.....	45
3.6.1 Numerisch zu String Konvertierung.....	45
3.6.2 String zu Numerisch.....	45
3.6.3 Datum zu String	46
3.6.4 Allgemeines Stringfunktionen.....	46

- 3.7 DATEI E/A 46**
- 3.7.1 Allgemein 46
- 3.7.2 Lesen aus Dateien 46
- 3.7.3 Schreiben in Dateien 47
- 3.8 TASTATUREINGABEN 47**
- 3.9 TELNET 47**
- 3.10 DATUM/ZEIT 48**
- 3.11 DDE 48**
- 3.12 VERSCHIEDENES 48**
- 3.12.1 Debug Meldungen ausgeben 48
- 3.12.2 Direkter Port E/A 48
- 3.12.3 Skript LED 49
- 3.13 ZEITVERZÖGERUNG 49**
- 3.14 AUSGABE AM KONSOLENFENSTER 49**
- 3.14.1 Information am Schirm schreiben 49
- 3.14.2 Bildschirm bearbeiten 49
- 3.15 VI FUNKTIONALITÄT 50**
- 3.15.1 Öffnen 50
- 3.15.2 Schliessen 50
- 3.15.3 Lesen 50
- 3.15.4 Schreiben 50
- 4 SOFTWARE ÄNDERUNGSÜBERSICHT 51**
- 5 BENUTZERHANDBUCH ÄNDERUNGSÜBERSICHT 54**

1 Einführung

Ich benutze GPIB-basierte Instrumente seit meiner Jugend als Physikstudent an der RUHR-UNIVERSITÄT-BOCHUM (Universität Bochum)

Ich hatte Zugang zu einem TEKTRONIX 4051, der zufällig einer der ersten echten Computer war, die Sie auf Ihren Schreibtisch stellen konnten (zumindest wenn Ihr Schreibtisch groß genug war). Der 4051 "Felsbrocken" sah aus, wie in Abbildung 1 dargestellt.



Abbildung 1

Zu seiner Zeit war der 4051 ein super moderner Computer. Während andere Computer textbasierte ASCII-Terminals verwendeten, um mit ihren Benutzern zu kommunizieren, verfügte der 4051 über einen großen Grafikbildschirm mit einer Auflösung von 1024x1024. Es wäre falsch, über die „Pixel“ des Bildschirms zu sprechen, da es sich nicht um einen Fernsehbildschirm handelte, auf dem Pixel angezeigt werden. Stattdessen hat man in eine wirklich große Speicherröhre gesehen, wie sie ursprünglich von TEKTRONIX für ihre analogen Speicheroszilloskope entwickelt wurde.

Dieser Speicherschirm wurde von einem Elektronenstrahl „beschrieben“, der die Leuchtstoffschicht so anregte, dass sie ständig nachglühte, wo der Strahl auf die Schicht aufprallte. Die horizontalen und vertikalen Ablenkplatten für diesen Elektronenstrahl wurden von zwei 10-Bit-D/A-Wandlern gespeist, die es ermöglichten, den Strahl mit der genannten Auflösung auf eine beliebige Position des Bildschirms zu lenken.

Stellen Sie sich einen vollwertigen Grafikcomputer vor, 10 Jahre bevor IBM überhaupt vorhatte, einen so genannten PC zu bauen, und sogar viele Jahre bevor die ersten „Heimcomputer“ auf den Markt kamen. Wow!

Tatsächlich wurde der 4051 allgemein als ein so revolutionäres Konzept angesehen, dass er auch dazu verwendet wurde, um die Rolle des „Computers“ in Science-Fiction-Filmen zu spielen!

Abbildung 2 ist ein Werbebild für einen der früheren „Battlestar Galactica“ Filme. Das war nicht das, was wir heute "Produktplatzierung" nennen! Sie mochten dieses Ding wirklich wegen seines futuristischen Designs!

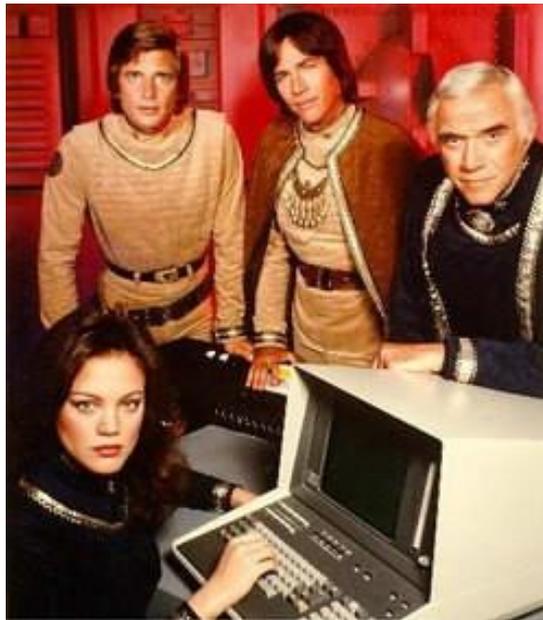


Abbildung 2

In diesen frühen Tagen des Rechnens war BASIC die Sprache der Wahl, wenn es um Datenerfassung ging und Unternehmen wie Hewlett & Packard (die Erfinder von HPIB, dem 'Hewlett & Packard Interface Bus', ein Begriff, der vom neutraleren GPIB 'General Purpose Interface Bus' ihrer Konkurrenten abgelöst wurde) und TEKTRONIX verfügten über die erforderlichen Busbefehle als integrierter Bestandteil ihrer BASIC-Interpreter.

Die GPIB-Handhabung war einfach! Während die Anweisung

100 Print "Test"

die Zeichenfolge "Test" auf dem Bildschirm ausgibt, würde die Anweisung

100 Print @13:"Test"

die Zeichenfolge "Test" an das Gerät mit der Adresse 13 am GPIB Bus senden.

Ähnlich dazu würde die Anweisung

110 Input @4:A\$

nicht von der Tastatur lesen, sondern versuchen stattdessen eine Antwort von GPIB-Gerät 4 in die Zeichenfolge A\$ einzulesen. Während die BASIC-Sprache ihre Grenzen hat, war das Schreiben von Datenerfassungssoftware nie einfacher. Die obigen Beispiele gelten für den 4051. Der BASIC-Dialekt von HP war etwas anders, aber auf dem gleichen einfachen Niveau.

Nach dem Studium basierten die ersten Datenerfassungssysteme für die Messung von Luftverschmutzung, mit denen ich mich befassen musste, auf Geräten mit GPIB-Ausstattung. Dann gab es eine lange Pause, in der ich immer noch mit Datenerfassung zu tun hatte, aber nicht mehr GPIB basiert.

Viele elektronische Messgeräte, die ich als Teil meines Hobbys als Funkamateurler gekauft hatte, hatten eine GPIB-Schnittstelle, aber ich musste sie nicht wirklich verwenden. Die Dinge änderten sich, als ich mich für Oszillator Stabilitätstests interessierte, bei denen die Datenaufzeichnung über Stunden, Tage oder sogar Wochen Teil des Geschäfts ist. In diesem Zusammenhang habe ich ISA GPIB-Karten von CEC, KEITHLEY, NATIONAL INSTRUMENTS und INES verwendet und Datenerfassungssoftware mit LABVIEW, Borlands TURBO PASCAL und später mit DELPHI geschrieben.

Während es im Grunde nur 2 verschiedene GPIB-Controller Chips auf dem Markt gab, gab jeder Hersteller seinen Karten einen anderen Adressbereich, DMA-Bereich und Interrupt-Bereich, wodurch sie untereinander so inkompatibel wie nur möglich wurden. Aus diesem Grund müssen Sie die für einen bestimmten Hersteller spezifischen Treiber und DLLs verwenden. Da die Treiber und DLLs in Bezug auf Funktionalität und Syntax nicht identisch sind, musste ich meine Quellen bei jeder Änderung der GPIB-Hardware ändern. Als ich meinen letzten ISA basierten Computer gegen einen Computer mit nur PCI-Steckplätzen austauschen musste, musste ich mich mit dem Problem befassen, was ich jetzt mit meinen GPIB-basierten Messungen tun sollte. Sollte ich erneut Hunderte von Dollar für eine PCI-GPIB Karte bezahlen und erneut neue Treiber und DLLs erhalten? In dieser Situation bin ich auf den GPIB-USB-Controller von PROLOGIX gestoßen, der unter <http://www.prologix.biz> verfügbar ist (siehe Abbildung 4).

Dieses niedliche kleine Gerät verwendet einen FTDI Chip für das USB zu RS232 Interface. Das bedeutet, dass Sie einen einfachen FTDI-Treiber installieren, der kostenlos auf deren Webseite verfügbar ist. Dieser Treiber weist einen virtuellen RS232-Port zu und Sie können jede Software verwenden, von einsatzbereiten Terminalprogrammen, bis hin zu selbstgeschriebenen Inhalten, um über diesen virtuellen RS232 Anschluss mit dem ATMEL AVR-Mikrocontroller auf dem Prologix Adapter zu kommunizieren.

Dieser Mikrocontroller verfügt wiederum über eine Firmware, die als eine Art Interpreter für serielle Befehle in GPIB Aktionen fungiert. Die Befehlssyntax ist leicht zu erlernen, und das gesamte Problem der GPIB-Programmierung beschränkt sich auf die Verarbeitung von Zeichenfolgen an einer seriellen Schnittstelle.

Dies ist zwar eine erwähnenswerte Verringerung der Komplexität bei der GPIB Programmierung, es kann sich jedoch dem Nicht-Programmierer das Problem stellen, eine GPIB Messung zum Laufen zu bringen. Außerdem fehlen der einfachen seriellen Syntax einige übergeordnete Aktionsbefehle, die ich gerne für meine eigenen Programmieranforderungen zur Verfügung gehabt hätte. Deshalb habe ich angefangen, mir meine eigene Arbeitsumgebung für die GPIB Programmierung zu programmieren. Sie können es als eine IDE verstehen, die speziell für die GPIB und RS232 Datenerfassung in Verbindung mit dem billigen Prologix Adapter entwickelt wurde. Ich habe es EZGPIB genannt. Dieses Dokument soll Ihnen EZGPIB vorstellen und näherbringen.

Beachten Sie, dass einige Tools zum Empfangen von Bildschirmplots von Oszilloskopen und Analysatoren über den Prologix Adapter kostenlos verfügbar sind. Googeln sie nach dem amerikanischen Rufzeichen KE5FX oder suchen sie im Web nach dem Namen "John Miles".

Die aktuelle Version des Prologix GPIB-USB Adapters sieht aus wie in Abbildung 4 und kostet ca. \$150,- was er aufgrund der Verbesserungen gegenüber seinen Vorgängern durchaus wert ist. Allein die Tatsache, dass er sicher an eine IEEE-488 Buchse geschraubt und über den USB Anschluss aktualisiert werden kann, ist jeden Cent wert

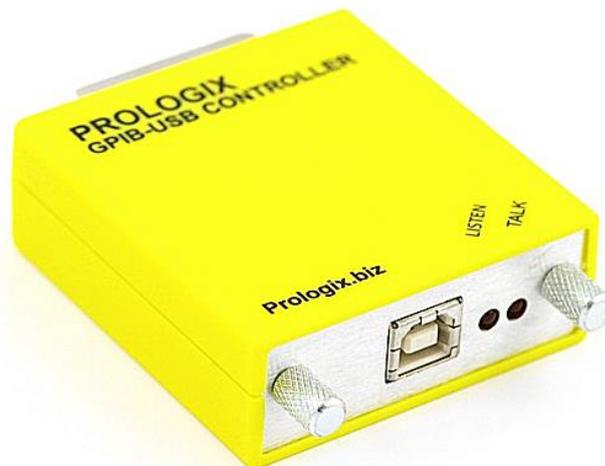


Abbildung 4

	EZGPIB Benutzerhandbuch	EZGPIB2_ger Ausgabe: 2.00 Seite 7/54
--	--------------------------------	--

1.1 Nachruf für OM Uli Bangert DF6JB

Die traurige Nachricht ist leider, dass Ulrich Bangert, der Schöpfer von EZGPIB und vielen anderen äußerst nützlichen Dokumentationen und HAM-Radio Geräten wie EasyFax, im Juni 2014 im Alter von 59 Jahren verstorben ist. Seine Frau Ina ist im Mai 2012 verstorben.

Vor langer Zeit hatte ich einige erfreuliche Kontakte mit ihm wegen seines EasyFax Konverters, den ich zu dieser Zeit gebaut und ausgiebig verwendet habe. Schließlich erklärte er sich bereit, mir eine Kopie des vollständigen Quellcodes seines Projekts zu überlassen, um meine eigenen Softwaremodifikationen und Erweiterungen für diesen extrem guten Konverter vorzunehmen.

Ich hoffe, dass ich mit dieser aktualisierten Überarbeitung des Dokuments und Übersetzung in Deutsche Sprache im Nachhinein noch ein wenig zu seiner hervorragenden Arbeit beitragen kann.

Kater Karlo

1.2 EZGPIB2.EXE

Mein Bildschirm ist eine hochauflösende Type und meine alten Augen sind schwach ;-)). Deshalb habe ich angefangen, den Ressourcenbereich der ausführbaren Datei EZGPIB.EXE mit dem bekannten Freeware Ressourceneditor „Resource Hacker“ von Angus Johnson zu patchen, um die Situation mittels eines 'hires' Patches für mich zu verbessern.

Die ausführbare Datei EZGPIB2.EXE, die mit dieser Dokumentation geliefert wurde, enthält die folgenden Änderungen (hoffentlich werden sie als Verbesserungen gesehen ;-))

- Die anfängliche Fenstergröße nach dem Start beträgt jetzt 1024x768 Pixel
- Die Schriftgröße aller Fenster wegen besserer Lesbarkeit von 11 auf 16 Pixel geändert.
- Schrift im 'Console Window' wegen besserer Lesbarkeit auf 'Courier New' geändert
- Die About Box wurde in der Größe geändert und die Texte neu angeordnet, da sie nicht lesbar war (zumindest auf meinem PC ;-))

2 Benutzerhandbuch

2.1 EZGPIB Starten

EZGPIB besteht aus der ausführbaren Datei EZGPIB.EXE (Original) oder EZGPIB2.EXE (mit 'hires' Patch), der INOUT.DLL, einigen Programmierbeispielen und diesem Handbuch. EZGPIB läuft unter WIN 2000, XP, 7 und hoffentlich auch unter neueren Versionen. Ältere Versionen wurden nicht getestet, verursachen jedoch sehr wahrscheinlich Probleme.

Stellen Sie auf der alten Prologix Karte die DIP-Schalter 1-5 auf OFF und 6 auf ON. Damit wird sie als GPIB-Buscontroller mit einer Busadresse von „0“ konfiguriert. Das Vorhandensein eines neuen (schalterlosen) Controllers wird von EZGPIB automatisch erkannt und die erforderlichen Einstellungen vorgenommen.

Beim Starten von EZGPIB(2).EXE versucht die Software, einen an Ihr System angeschlossenen Prologix Adapter zu erkennen. Zu diesem Zweck werden alle seriellen Schnittstellen Ihres Systems aufgelistet und auf das Vorhandensein eines FTDI oder CH340 Chips überprüft. Wenn ein FTDI oder CH340 Chip erkannt wird, sucht die Software nach einem aktivierten CTS Signal. Wird ein solches erkannt, so wird eine Testzeichenfolge und das Kommando "++ ver" gesendet, um zu testen, ob ein Prologix Adapter mit einem Versionsstring antwortet, der den Teilstring "USB-GPIB" enthält oder nicht.

Wenn keine Prologix Adapter gefunden wird, versucht EZGPIB zu erkennen, ob eine DLL mit dem Namen GPIB32.DLL auf dem PC verfügbar ist, da dies möglicherweise darauf hinweist, dass der PC mit einer internen GPIB-Adapterkarte ausgestattet ist. Wenn die DLL verfügbar ist, wird sie geladen und zum Erkennen eines an den PC angeschlossenen GPIB-Adapters verwendet. Das kann entweder eine PCI-Karte oder etwas anderes sein.

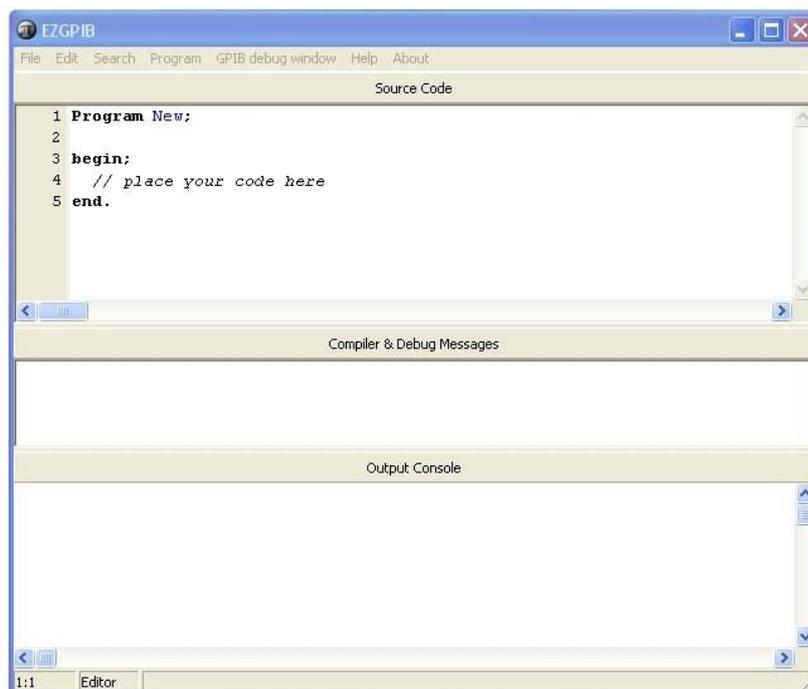
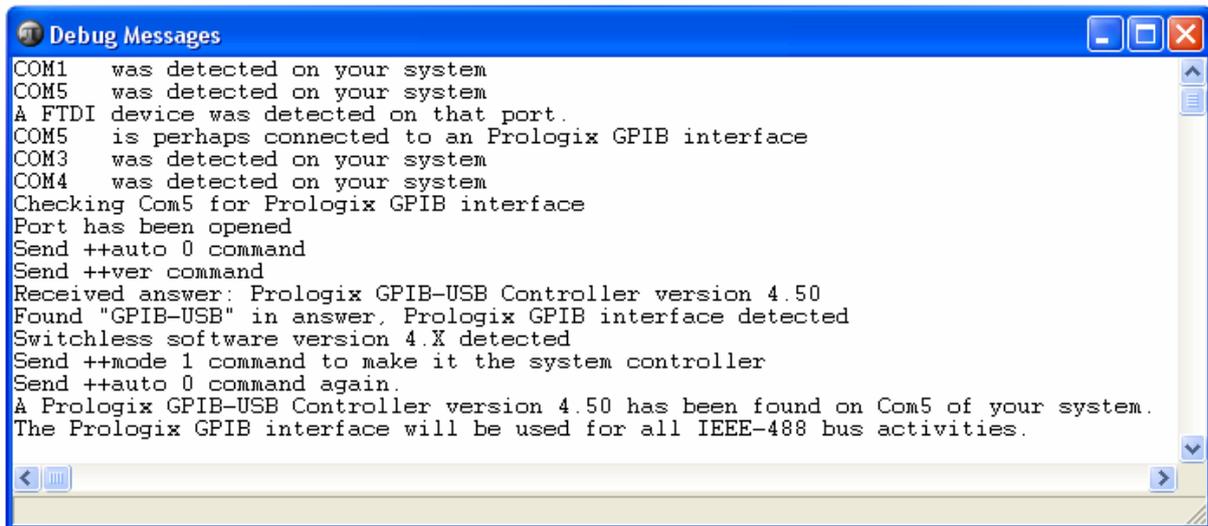


Abbildung 5

Unabhängig davon, wie Ihr PC ausgestattet ist, werden Sie zunächst mit dem Hauptfenster von EZGPIB wie oben gezeigt konfrontiert.

Als nächstes schlage ich vor, das **GPIB Debug Fenster** über das Hauptmenü zu öffnen. Dies hilft Ihnen zu verstehen, wie der Prologix Adapter oder ein anderer GPIB Adapter gefunden wurde oder warum er nicht gefunden wurde. Wenn ein Prologix Adapter an mein System angeschlossen ist, sieht das Debug Fenster nach dem Start von EZGPIB folgendermaßen aus:



```
COM1 was detected on your system
COM5 was detected on your system
A FTDI device was detected on that port.
COM5 is perhaps connected to an Prologix GPIB interface
COM3 was detected on your system
COM4 was detected on your system
Checking Com5 for Prologix GPIB interface
Port has been opened
Send ++auto 0 command
Send ++ver command
Received answer: Prologix GPIB-USB Controller version 4.50
Found "GPIB-USB" in answer, Prologix GPIB interface detected
Switchless software version 4.X detected
Send ++mode 1 command to make it the system controller
Send ++auto 0 command again.
A Prologix GPIB-USB Controller version 4.50 has been found on Com5 of your system.
The Prologix GPIB interface will be used for all IEEE-488 bus activities.
```

Abbildung 6

Abhängig von Ihrem System können die Informationen von den oben angezeigten abweichen.

Hinweis1: Bei einer Reihe von Prologix kompatiblen Adaptern wie AR488 kann es vorkommen, dass das CTS-Signal NICHT vom USB-Schnittstellenchip aktiviert wird. In diesem Fall besteht die beste Lösung darin, den CTS Pin des Schnittstellenchips ständig mit GND zu verbinden, um die Kommunikation mit EZGPIB zu ermöglichen! Ich rate aus verschiedenen Gründen dringend von der Verwendung von CTS-gepatchten Versionen von EZGPIB ab.

Hinweis2: Auch wenn der Prologix Adapter korrekt an Ihr System angeschlossen ist und eine Reihe von mehreren GPIB-Geräten korrekt an den Prologix Adapter angeschlossen sind, kann es vorkommen, dass dieser beim Starten von EZGPIB nicht erkannt wird.

In den meisten Fällen ist dieses Phänomen auf folgenden Grund zurückzuführen:

Der Prologix Adapter antwortet immer nur dann auf serielle Befehle, wenn er den GPIB-Bus vollständig steuern kann. **Der Bus kann aber nur dann vollständig gesteuert werden, wenn genügend Geräte am Bus eingeschaltet sind!**

Wenn Sie mehr als ein Instrument am GPIB BUS angeschlossen haben, sollten Sie mindestens **2/3 aller angeschlossenen Geräte einschalten**, auch wenn Sie nur mit einem einzelnen Gerät sprechen möchten. Dies ist kein spezifisches Prologix Problem, sondern hat allgemein mit den GPIB Spezifikationen zu tun.

	EZGPIB Benutzerhandbuch	EZGPIB2_ger Ausgabe: 2.00 Seite 10/54
--	--------------------------------	---

Die Situation, dass der Adapter den Bus nicht vollständig steuern kann, lässt sich auch leicht an den LEDs auf dem Prologix Adapter erkennen: Wenn die **TALK** LED nach dem Einschalten konstant leuchtet, ist dies ein Indikator für ein solches Problem. Wenn alles in Ordnung ist, ist die **LISTEN** LED zu diesem Zeitpunkt ständig eingeschaltet.

Auf der Seite <http://www.transera.com/htbasic/tutgpib.html> können Sie unter "Physikalische Eigenschaften" mehr darüber lesen.

Seien Sie sich dessen bewusst, dass Sie mit den sehr einfach zu verwendenden Befehlen "**EZGPIB_BusWriteData**" und "**EZGPIB_BusWaitForData**" die Arbeit für die meisten Ihrer Aufgaben erledigen können.

Viele andere Befehle sind in EZGPIB nur deshalb zu finden, ***weil sie für den Prologix Adapter verfügbar sind*** ;-)).

Zwei Dinge, die Sie wissen müssen, die aber nicht so offensichtlich sind:

- 1) Das Hauptfenster ist in drei Abschnitte unterteilt, deren Größe Sie individuell festlegen können. Bewegen Sie die Maus langsam von einem Abschnitt zum nächsten, bis sich der Cursor in zwei parallele Linien verwandelt. Drücken Sie die Maustaste und verschieben Sie den Abschnittsrand in die gewünschte Richtung.
- 2) EZGPIB speichert Programme in Standard Textdateien, gibt ihnen jedoch die Erweiterung "**.488**", um sie von anderen Textdateien zu unterscheiden.

Beim Start sucht EZGPIB nach einer Datei mit dem Namen "**standard.488**".

Wenn eine Datei mit diesem Namen im Verzeichnis von EZGPIB gefunden wird, wird sie automatisch geladen. Diese Funktion wurde für den Fall integriert, dass Sie regelmäßig mit demselben Setup arbeiten.

Wenn Sie im Dateimenü die Option Neu auswählen, laden Sie in Wirklichkeit eine Datei mit dem Namen "**new.488**". Speichern Sie alles, was Sie zu Beginn eines neuen Projekts sehen möchten, in "**new.488**" und es wird dann angezeigt, wenn Sie es brauchen ;-))

2.1.1 Was EZGPIB ist

- 1) EZGPIB ist eine IDE mit einem vollständigen Editor zum Schreiben und Debuggen von PASCAL ähnlichem Quellcode. PASCAL ist eine leicht zu erlernende strukturierte Programmiersprache. Selbst wenn Sie kein Programmierer sind, werden Sie feststellen, dass die Beispiele, die mit EZGPIB zusammenkommen, Ihnen eine leicht verständliche Einführung in das geben, worum es geht.
- 2) EZGPIB ist ein Compiler für die PASCAL ähnliche Skriptsprache, die Sie mit dem Editor geschrieben haben. Der auffälligste Unterschied zu einem Standard PASCAL Compiler ist die Tatsache, dass der Standardsprache viele einsatzbereite Funktionen und Prozeduren hinzugefügt wurden. Die Dinge, die hinzugefügt wurden, sind (hoffentlich) alles, was Sie für eine erfolgreiche GPIB-Datenerfassung in Verbindung mit dem Prologix Adapter und für die Datenverarbeitung benötigen. Sie müssen sich nicht mit seriellen Kommunikationsfunktionen auf niedriger Ebene herumschlagen, sondern können stattdessen Funktionen auf hoher Ebene verwenden:

BusWaitForData(Device:LongInt; ForWhat:string; MaxWait:Double):Boolean;

Dies bedeutet Folgendes: Es wird <MaxWait> Sekunden auf eine GPIB-Nachricht, die vom Busgerät <Gerät> kommt, gewartet. Wenn die Antwort innerhalb des Zeitlimits von <MaxWait> Sekunden eintrifft, wird die Nachricht in die Zeichenfolge <ForWhat> eingefügt und als Funktionswert "TRUE" zurückgegeben. Andernfalls bleibt <ForWhat> leer und als Funktionswert wird "FALSE" zurückgegeben.

- 3) EZGPIB ist die Laufzeitumgebung für die Programme, die Sie mit dem Editor geschrieben und mit dem Compiler übersetzt haben. EZGPIB erstellt keine eigenständigen Anwendungen. EZGPIB Skript Anwendungen können nur innerhalb von EZGPIB gestartet werden. Während dies für Einige als Nachteil erscheinen mag, gibt es starke Gründe dafür, die den Rahmen dieser Diskussion sprengen. EZGPIB hat auch einen großen Vorteil: Es ist eine eigene Laufzeitumgebung: Der untere Teil des Hauptfensters ist eine textbasierte Ausgabe- und Eingabekonzole, und EZGPIB verfügt über integrierte Funktionen und Prozeduren, die textbasierte E/A's über diese Konsole zum Kinderspiel machen. Sie müssen keine Windows-spezifischen E/A Techniken erlernen.
- 4) Wenn Sie der Editor und Debug Abschnitt des Hauptfensters beim Ausführen eines Programms stört, können Sie das Programm mit dem Befehl **Run Console** starten. In diesem Fall werden Editor und Debug Teil während der Programmausführung minimiert. Die Dateiausgabe von Messdaten, die vielleicht das Wichtigste ist, nach dem Sie suchen, ist **extrem einfach!**

2.1.2 Was EZGPIB nicht ist

EZGPIB ist ein Werkzeug, mit dem Sie **mit Ihren GPIB-Geräten kommunizieren** können. Es ist jedoch nicht bekannt, wie **der Inhalt der Kommunikation** aussehen muss, um das erwartete Ergebnis zu erzielen. Das **müssen Sie wissen!**

Sie benötigen ein bestimmtes Verständnis dafür, was Ihre Geräte über die GPIB Schnittstelle erwarten, um eine bestimmte Aktion auszuführen oder ein bestimmtes Messergebnis über die GPIB Schnittstelle zurück zu senden.

Normalerweise bedeutet dies, dass Sie das vollständige Gerätehandbuch oder zumindest den GPIB-Programmierteil davon benötigen. Seien Sie gewarnt, dass die meisten Handbücher nicht als Einführung in die GPIB-Programmierung gedacht sind. Auf der anderen Seite ist GPIB einer der am besten eingeführten Standards in der Welt der Elektronik, und Sie sollten keine Schwierigkeiten haben, alles was man über GPIB im Allgemeinen wissen muss, im Internet zu finden

2.2 EZGPIB Benutzen

2.2.1 Skripte Laden und Ausführen

Das Menü "File" (Datei) von EZGPIB ist dem sehr ähnlich, was Sie von anderen Windows Programmen kennen. Verwenden Sie darin den Befehl "Open", um den Datei Dialog zu öffnen und **Test_01.488** zu laden. Jetzt sollte Ihr Hauptfenster folgendermaßen aussehen:

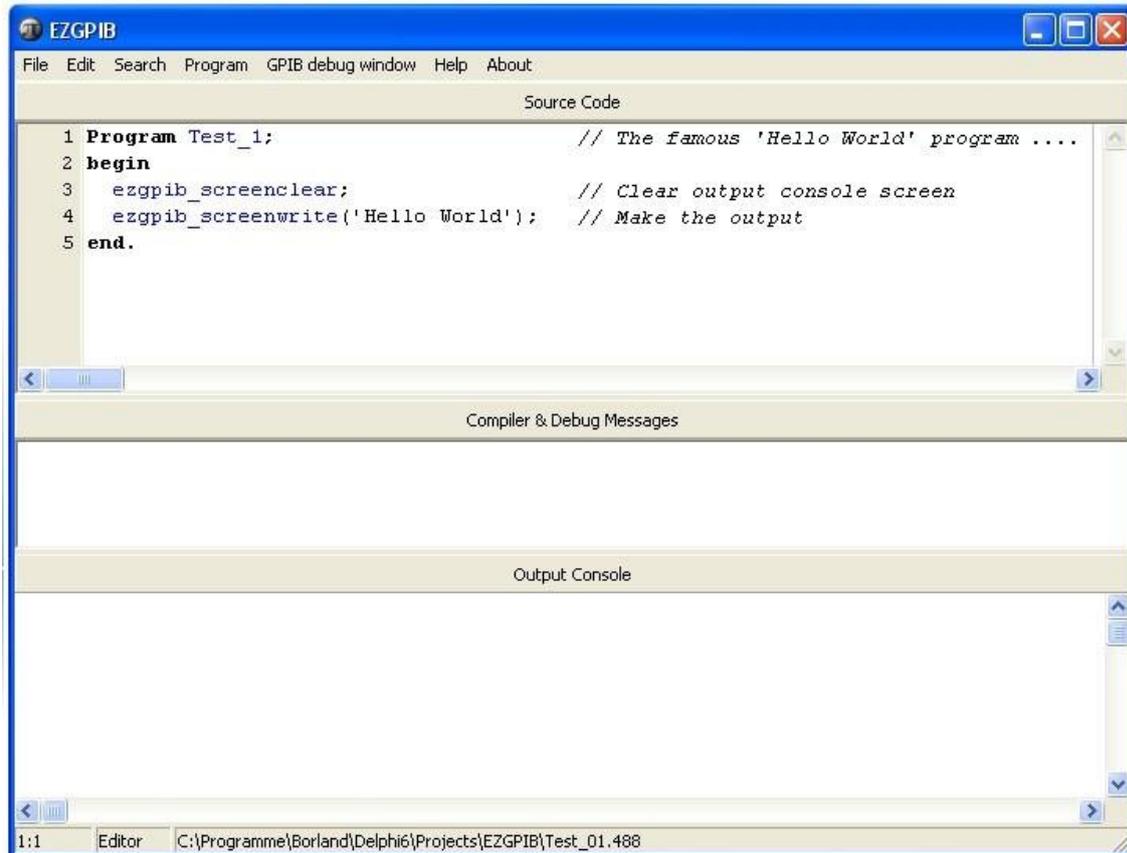


Abbildung 9

Dies ist das berühmte "Hello World" -Programm, das jeder Programmierer einmal in einer neuen Programmierumgebung ausführen muss. Beachten Sie im Editorteil "Source Code" des Fensters Folgendes:

- 1) Einige Wörter sind **schwarz in Fettdruck**. Dies sind **reservierte Wörter** der PASCAL-ähnlichen Sprache, die nicht für einen anderen Zweck verwendet werden dürfen. Sie dürfen ein Programm nicht "begin" nennen, da "begin" ein reserviertes Wort ist, das den Start eines Programms, einer Funktion oder einer Prozedur angibt.
- 2) Einige Wörter sind in einer **Standardschrift schwarz**. Dies sind **Konstanten**, denen keine bestimmte Gruppe zugewiesen werden kann. Im Beispiel ist alles hinter dem "/" ein Kommentar und "Hello World" eine **Text Konstante**.
- 3) Einige Wörter sind **blau**. Es sind **Namen von Programmen, Funktionen und Prozeduren**.

Verwenden Sie nun den Befehl **“Run“** im Menü Program, um dieses Programm zu kompilieren und auszuführen.

Der untere Teil des Hauptfensters sollte nun so aussehen



Abbildung 10

Herzlichen Glückwunsch! Sie haben Ihr erstes EZGPIB Programm kompiliert und ausgeführt. Es hatte nichts mit GPIB zu tun und dies sollte Ihnen eine Idee davon geben, dass Sie EZGPIB auch anderweitig für **“quick and dirty“** Programmierung verwenden können.

Das bedeutet nicht, dass Alles, was Sie in EZGPIB tun, **“quick and dirty“ sein muss!**

Die Sprache PASCAL, die die Grundlage von EZGPIB bildet, zwingt Sie dazu, gut strukturierte Inhalte zu schreiben. Sie können damit selbst große Probleme und Aufgaben problemlos bewältigen. EZGPIB ist ein echter Compiler und Sie werden feststellen, dass **EZGPIB basierte Programme sehr schnell sind!**

Sie werden jedoch feststellen, dass die meisten in EZGPIB geschriebenen GPIB Messanwendungen in Bezug auf die Anzahl der Programmzeilen recht klein sind. Dies ist auf die vielen verfügbaren Funktionen auf hoher Ebene zurückzuführen.

2.2.2 Skript Debugging

Für den sehr unwahrscheinlichen Fall, dass Sie ein dauerhaftes oder gelegentliches verdächtiges Verhalten Ihres brandneuen Skripts feststellen, bietet EZGPIB einige nützliche Tools und Strategien, um mit solchen Situationen umzugehen. Bitte beachten Sie, dass Alles, was in den Kapiteln 2.2.2.n erwähnt wird, vom Ersteller des Programms nicht im Detail so beschrieben wurde, sondern das Ergebnis von „Versuch und Irrtum“ im Zuge der Erstellung eigener Programme ist! Verwenden Sie diese Informationen daher auf eigenes Risiko ;-))

2.2.2.1 Debug Ausgaben im Konsolenfenster

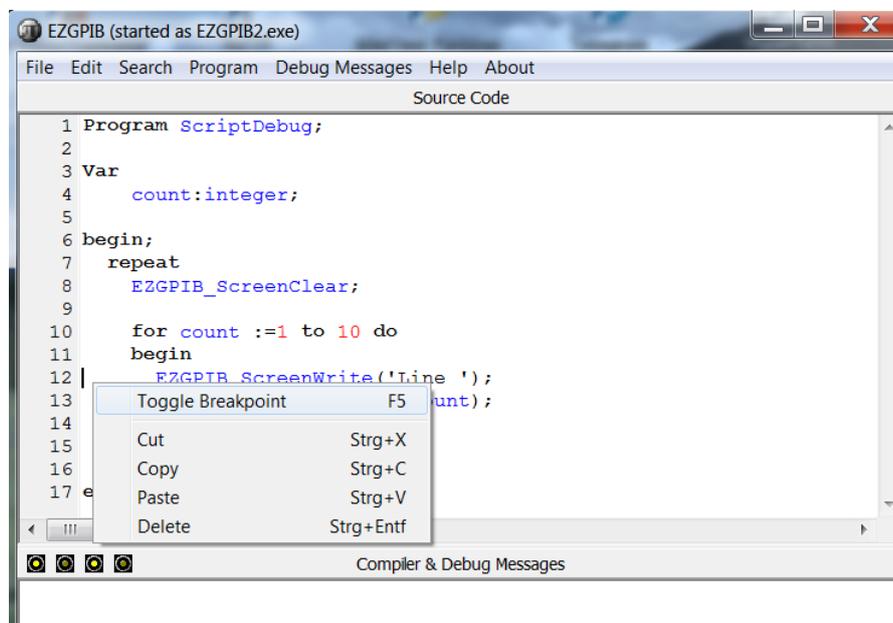
Als sehr einfache Methode zum „Hardcore“ -Debugging möchte ich den Leser nachdrücklich dazu ermutigen, alle in Kapitel [3.14](#) dieser Dokumentation erwähnten Routinen umfassend zu nutzen.

Mithilfe der Prozedur **EZGPIB_ScreenWriteLn** und der diversen Konvertierungsroutinen für Zeichenfolgen können Sie auf einfache Weise Informationen darüber abrufen, was mit Ihrem Programm und Ihren Variablen geschieht, während Ihr Skript ausgeführt wird.

Leider unterstützt die eingesetzte Skript Engine KEIN bedingtes Kompilieren, daher können Sie keine Turbo Pascal ähnlichen Definitionen wie `{ $ DEFINE DEBUG }` verwenden, um Ihre Debug Ausdrücke zu entfernen.

2.2.2.2 Haltepunkte (F5)

EZGPIB bietet die Möglichkeit, Haltepunkte in jede ausführbare Zeile Ihres Skripts einzufügen, wie im Kapitel [2.4.8](#) beschrieben. Platzieren Sie den Cursor einfach in jene Zeile des Fensters "Source Code", in der Sie den Ausführungsfluss stoppen möchten und klicken Sie mit der rechten Maustaste oder mit der rechten Maustaste auf das Mousepad. Sie erhalten ein Popup Fenster mit den möglichen Optionen:



Durch Drücken von F5 setzen Sie einen Haltepunkt an der Cursorzeile, der dann in roter Farbe angezeigt wird:

```
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.
```

Natürlich können Sie mehrere Haltepunkte gleichzeitig in mehreren Zeilen Ihres Skripts setzen:

```
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.
```

Wenn Sie das Skript jetzt ausführen indem Sie F9 drücken, wird es automatisch am ersten Haltepunkt angehalten und zeigt dies durch Invertierung der Zeilenfarbe, wie nachfolgend zu sehen, an:

```
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

Beachten Sie, dass, wie oben zu sehen, noch nichts in das Fenster "Output Console" geschrieben wurde! Dies bedeutet, dass **ein Haltepunkt die Ausführung in der angegebenen Zeile anhält, die Zeile selbst jedoch noch NICHT AUSGEFÜHRT wurde, wenn das Programm angehalten wird!**

Wenn Sie das Skript vom Haltepunkt aus fortsetzen indem Sie F9 drücken, ändert sich die Anzeige wie folgt:

```
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

Sie werden drei Dinge bemerken:

- 1) In der "Output Console" wurde die Zeichenfolge "Zeile 1" ausgegeben. Dies bedeutet, dass die Zeilen 12 und 13 ausgeführt wurden.
- 2) Das Skript wird in Zeile 14 angehalten, aber beachten Sie, dass die Anweisung "EZGPIB_Timesleep(0.2)" noch nicht ausgeführt wurde!
- 3) Der erste Haltepunkt ist noch aktiv und stoppt das Programm in Zeile 12 erneut, wenn Sie die Ausführung mit F9 fortsetzen.

Beachten Sie, dass die Taste F5 eine Umschaltfunktion hat. Haltepunkte können entfernt werden, indem Sie den Cursor auf die entsprechende Zeile bewegen und erneut F5 drücken. Wenn Sie mehrere Haltepunkte haben, können Sie dies sogar tun, während das laufende Skript an einem anderen Haltepunkt angehalten wird.

Wenn Sie den Haltepunkt, an dem das Skript tatsächlich angehalten wird, mit F5 umschalten, ändert sich die Linienfarbe in Blau, um anzuzeigen, dass Sie den tatsächlichen Haltepunkt ändern!

```
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.
```

	EZGPIB Benutzerhandbuch	EZGPIB2_ger Ausgabe: 2.00 Seite 17/54
--	--------------------------------	---

Abschließend möchte ich noch erwähnen, dass Haltepunkte nicht nur im Hauptprogramm funktionieren, sondern auch in Zeilen innerhalb Ihrer selbstgeschriebenen Prozeduren und Funktionen platziert werden können.

2.2.2.3 Einzelschritte (F7, F8 und F5+F9)

Anstatt Ihr Skript mit F9 komplett auszuführen, besteht auch die Möglichkeit, die Ausführung zeilenweise mit F7 (Step Into) und F8 (Step Over) durchzuführen.

Anscheinend gibt es zwei Funktionen, die von OM Uli nicht aktualisiert / implementiert wurden:

- 1) Die Zeile, in der Ihr Skript nach einem Einzelschritt tatsächlich angehalten wird, wird im Fenster "Source Code" NICHT (z. B. durch geänderte Schriftfarbe) angezeigt! Dies macht es für nicht triviale Skripte schwierig, sich zu merken, wo das Skript aktuell tatsächlich angehalten wurde.
- 2) Die Tasten F7 und F8 rufen beide dieselbe Routine "Step Into" auf. Ich habe das beim Basteln im Ressourcenbereich des EZGPIB-Programms überprüft. Es scheint also, dass die Routine "Step Over" nie implementiert wurde oder fehlerhaft war und daher in dieser Version nicht verwendet wird.

Es gibt einen Trick, um das zu umgehen, indem Sie einfach Haltepunkte verwenden, die zuvor in JEDER LINIE mit F5 gesetzt wurden. Sie können dann mit der Taste F9 jede Skriptzeile im „Einzelschritt“ abarbeiten. Ehrlich gesagt weiß ich nicht, wie viele Haltepunkte ein einzelnes Skript maximal haben kann, aber bisher bin ich nicht auf ein Limit gestoßen.

2.2.3 Wir schreiben eine echte Datenerfassungsanwendung

Hier werde ich die notwendigen Schritte beschreiben, um eine echte Datenerfassungsanwendung zu schreiben. Natürlich wird es eine Einfache sein! Aber jede komplexe Aufgabe kann in eine Gruppe von weniger komplexen Aufgaben unterteilt werden, die wiederum in noch niedrigere komplexe Aufgaben unterteilt werden können, bis zu einem Punkt, an dem die Aufgaben wirklich einfach sind. Dieses Programmierschema wird als Top-Down Design bezeichnet und wird von der PASCAL ähnlichen Programmiersprache gut unterstützt.

Ich werde Ihnen zeigen, wie Sie die Zeitbasis eines Frequenzzählers (in diesem Fall mein alter, aber guter RACAL DANA 1996) auf einen Wert von 10 Sekunden einstellen, dann auf die gemessenen Ergebnisse warten und die Ergebnisse aus einer Programmschleife heraus, die ausgeführt wird, solange keine Taste auf der Tastatur gedrückt wird, in der Ausgabekonzole anzuzeigen.

Hey ihr Nicht-Programmierer: Klingt kompliziert? Sehen Sie es sich an und lassen Sie sich überraschen, wie einfach es ist! Sie haben es erraten: Daher kommt der Name EZGPIB!

Drücken Sie im Menü **“File“** auf den Eintrag **“New“**. Im Allgemeinen wird im Quellcodeteil des Hauptfensters danach Folgendes angezeigt:

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

Abbildung 11

Beachten Sie jedoch, dass dies **nicht so sein muss!** Bei Verwendung des Menüeintrags "New" sucht EZGPIB in seinem Verzeichnis nach einer Datei **New.488**. Wenn diese gefunden wird, wird sie in den Editor geladen. Wofür ist das gut?

Stellen Sie sich vor, Sie müssen sich von Tag zu Tag mit verschiedenen Messaufgaben befassen. Trotz der Tatsache, dass die Aufgaben unterschiedlich sind, haben sie möglicherweise viele Gemeinsamkeiten: Die Busadressen Ihrer GPIB Geräte bleiben normalerweise gleich, und Sie haben möglicherweise eine Reihe von Hilfsroutinen erstellt, die Sie allgemein verwenden. In diesem Fall speichern Sie das für alle Ihre Anwendungen gemeinsame Framework unter dem Namen **New.488**

Immer wenn Sie im Menü "**File**" die Option "**New**" auswählen, wird Ihr komplettes Framework in den Editor geladen und Sie beginnen mit der individuellen Programmierung der neuen Aufgabe mit allen bereits vorhandenen Informationen

Obwohl es andere Möglichkeiten gibt, dasselbe zu tun, beispielsweise mithilfe von "Include" Dateien, fand ich dies eine elegante Möglichkeit mit dem Framework umzugehen. Nehmen wir der Einfachheit halber an, dass wir die in Abbildung 11 gezeigte Situation haben. Zuerst ändern wir den Namen des Programms in einen geeigneten und deklarieren eine Konstante, die die Busadresse des Zählers enthält. Sollte der Zähler in Zukunft eine andere Adresse erhalten, ist dies der einzige Punkt, an dem wir die Nummer ändern müssen!

Jetzt könnte das Editorfenster so aussehen

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

Abbildung 12

Beachten Sie, dass Sie bereits in diesem frühen Entwicklungsstadium den Menüeintrag "**Compile**" aus dem Menü "**Program**" verwenden können, um die Syntax Ihrer Texte zu überprüfen (obwohl im Moment nichts wirklich ausgeführt werden kann...)

In diesem Fall wird folgendes angezeigt



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

Abbildung 13

Da wir für die Konstantendeklaration eine falsche Syntax verwendet haben, teilt uns der Compiler mit, dass anstelle des von uns verwendeten ":" ein "=" erwartet wird. Er teilt uns auch mit, wo dieser Fehler gefunden wurde: Zeile 3, Zeichen 22. Beachten Sie, dass Sie in der Statuszeile von EZGPIB am unteren Rand des Hauptfensters immer sehen, wo sich der Cursor derzeit im Editorteil des Fensters befindet.

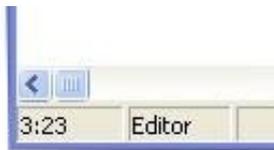


Abbildung 14

Korrigieren wir den Fehler und einen Befehl einfügen, um die Zeitbasis auf 10 Sekunden festzulegen. Jetzt sollte Ihre Quelle so aussehen. Beachten Sie, dass EZGPIB nicht zwischen Groß- und Kleinbuchstaben unterscheidet. Sie können die Schriftart verwenden, die Sie bevorzugen.

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

Picture 15

Die erforderliche Zeichenfolge "GA10" (**G**ate **A**djust auf **10** Sekunden) ist jene Information, die Sie aus dem Handbuch des Zählers erhalten müssen!

Nun, dies ist bereits ein Programm, das ausgeführt werden kann. Wenn Sie es mit dem Menüeintrag "Run" aus dem Menü "Program" ausführen, sollten Sie sehen, dass der Zähler seine Zeitbasis auf 10 Sekunden ändert. Beachten Sie, dass die Quelle nichts enthält, was Sie an den Prologix Adapter erinnert. Die **BusWriteData** Prozedur erledigt all diese Dinge für Sie und erinnert mich in ihrer Einfachheit an das, was ich zuvor über BASIC gesagt habe.

Ich hatte bereits erwähnt dass wir, sobald wir den Zähler auf den neuen Zeitbasiswert umgestellt hatten, etwas in einer Programmschleife tun würden, bis eine Taste auf der Tastatur gedrückt wird. Sehr ähnlich wie beim normalen Englisch verwenden wir zu diesem Zweck ein "repeat until" (wiederholen bis) Konstrukt. Sogar der Zustand, in dem die Schleife anhalten soll, kommt mir sehr bekannt vor.

```
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.
```

Abbildung 16

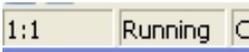
Wenn Sie das Programm starten, werden Sie sehen, dass im Meldungsfenster die Information



Sucessfully compiled

Picture 17

und in der Statuszeile



1:1 Running C

Picture 18

angezeigt wird.

Beides sind Indikatoren dafür, dass Ihr Programm in einer Schleife ausgeführt wird. Wenn Sie eine beliebige Taste drücken, wird im Meldungsfenster die Meldung **'Successfully Executed'** angezeigt und die Meldung "Running" wechselt wieder zurück zu "Editor".

Lassen Sie uns nun etwas Nützliches in die Schleife einfügen. Ich habe bereits zuvor erwähnt, dass es eine benutzerfreundliche Funktion zum Lesen von Daten gibt. Also lassen Sie uns diese anwenden!

```
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.
```

Abbildung 19

Beachten Sie, dass wir zur Verwendung der eleganten Formulierung

EZGPIB_BusWaitForData(Counter_Address,Answer,TimeOut)

eine Variable 'Answer' vom Typ String und ein konstantes 'Timeout' mit dem Wert 15 deklarieren mussten. Da die Zeitbasis des Zählers auf 10 gesetzt ist, sollte er in der Lage sein, innerhalb von 15 Sekunden zu antworten. Wenn wir dieses Programm ausführen, dann wird im Konsolenteil des Hauptfensters folgendes angezeigt

```
F +9.99999999940247E+06
F +9.99999999939087E+06
F +9.99999999937927E+06
F +9.99999999934082E+06
F +9.99999999930847E+06
```

Picture 20

und alle zehn Sekunden wird eine neue Zeile geschrieben. Sie mögen das Scrollen nicht und möchten das 'F'-Zeug am Anfang der Zeichenfolge entfernen, damit Sie es in eine Zahl umwandeln können?

Auf geht's:

```
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.
```

Abbildung 21

Erzeugt als Ausgabe



```
+9.9999999936462E+06
```

Picture 22

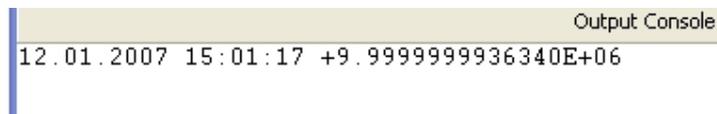
Das heißt: Ein neues Ergebnis überschreibt das vorherige Ergebnis.

Möglicherweise möchten Sie die Zeiten, zu denen der Wert gemessen wurde? Hier sind sie:

```
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.
```

Abbildung 23

Dieses Skript erzeugt die Ausgabe:



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

Picture 24

Lassen Sie mich hier anhalten und vorschlagen, dass Sie sich die Beispiele ansehen, die Ihnen noch viele andere Aspekte von EZGPIB demonstrieren. Wenn Sie feststellen, dass Sie Schwierigkeiten mit der Sprache PASCAL selbst haben, empfehle ich Ihnen, sich eine grundlegende Einführung in PASCAL zu verschaffen.

Sie müssen definitiv kein Top PASCAL Programmierer werden müssen. Sie benötigen nur ein grundlegendes Verständnis von PASCAL und Programmiergewohnheiten und -konzepten, um sehr erfolgreich mit EZGPIB arbeiten zu können.

Um ehrlich zu sein: Das Beispiel, das ich Ihnen oben gezeigt habe, zeugt aus mehreren Gründen von sehr schlechten Programmiergewohnheiten. Trotzdem hat es sich bewährt ;-))

Wenn wir schon über Programmiergewohnheiten sprechen: Das Schlimmste, was ich im obigen Beispiel getan habe, ist, **so lange auf die Antwort eines Geräts zu warten**. Es ist ohne Diskussion klar, dass ein Zähler mit einer auf 10 Sekunden eingestellten Zeitbasis nur alle 10 Sekunden einen Messwert liefern kann. Aber anstatt auf die Antwort zu warten, hätten wir in dieser Zeit andere nette Dinge tun können, zum Beispiel die Kommunikation mit anderen Geräten. **Während wir auf ein Gerät warten, können wir in dieser Zeit nichts anderes tun**. Dies führt zu der Frage, wie Sie feststellen können, wann auf einem Gerät Daten verfügbar sind, ohne nur auf diese Daten zu warten.

```
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.
```

Abbildung 25

Der GPIB hat ein schönes Konzept für den sogenannten SRQ = "Service Request". Service Request ist eine eigene Leitung am GPIB, die von jedem Gerät aktiviert werden kann, um anzuzeigen, dass es einen Service anfordert. Normalerweise fordern Geräte einen Service an, wenn neue Messwerte verfügbar sind, andere Quellen für eine Serviceanforderung sind jedoch möglich. Die Steuerung muss herausfinden, welches Gerät aktuell Service anfordert, und dann die Daten dieses Geräts lesen.

	EZGPIB Benutzerhandbuch	EZGPIB2_ger Ausgabe: 2.00 Seite 25/54
--	--------------------------------	---

EZGPIB und der Prologix Adapter unterstützen diese Strategie sehr gut. Schauen Sie sich die modifizierte Version des Zählerbeispiels oben auf der letzten Seite an. Einige Dinge wurden gegenüber der letzten Version des Beispiels geändert:

- 1) Am Beginn des Programms wurde ein Aufruf von **EZGPIB_FindAllDevices** hinzugefügt. Dies ist bei EZGPIB immer eine gute Angewohnheit. Diese Prozedur versucht, alle aktiven Geräte am Bus zu erkennen, indem das sogenannte Statusregister gelesen wird. Selbst ältere Geräte, die die IEEE-488-2 Syntax nicht verstehen können (* idn? Und Ähnliches), haben in den meisten Fällen ein Statusregister und können daher identifiziert werden, indem nach dem aktuellen Wert des Statusregisters gefragt wird
- 2) Der Konsolenbildschirm wird zu Beginn des Programms nur einmal gelöscht. Danach wird der Cursor mit dem **EZGPIB_ScreenGotoXY** Befehl an einer bestimmten x und y Position positioniert, bevor die Zeichenfolge geschrieben wird.
- 3) Beachten Sie den Aufruf von **EZGPIB_WaitForMultipleOf** in der Hauptschleife. Dies wartet, bis ein ganzzahliges Vielfaches seines Aufrufwerts abgelaufen ist. Mit einer "1" warten wir mit diesem Aufruf, bis eine neue Sekunde eingetroffen ist. Wir hätten dies weglassen und viele Male schneller durch die Hauptschleife laufen können, aber da wir die Zeitinformationen in der Hauptschleife ausdrucken möchten, besteht keine Notwendigkeit, dies schneller als einmal pro Sekunde zu tun.
- 4) Beachten Sie, dass wir die Zeitinformationen jede Sekunde ausdrucken. Zusätzlich prüfen wir jede Sekunde durch einen Aufruf von **EZGPIB_BusSrq**, ob aktuell ein SRQ aktiv ist. Nur wenn diese Bedingung gegeben ist, bitten wir den Zähler, uns seinen Messwert zu geben. Weil wir sicher sein können, dass ein Wert verfügbar ist. In dieser Situation können wir eine sehr kurze Zeitüberschreitung von 0,1 Sekunden verwenden.

Beachten Sie auch, dass wir durch Vergleichen des Ergebnisses von **EZGPIB_BusSourceOfSrq** mit Counter_Address überprüfen, ob es wirklich der Zähler ist, der den Service angefordert hat.

Dieses Beispiel zeigt, dass wir in der Zwischenzeit andere Dinge getan haben, anstatt auf den Zähler zu warten.

Beachten Sie, dass nicht alle Geräte eine neue Messung mit einem SRQ signalisieren!

Viele Geräte signalisieren dies, indem sie ein bestimmtes Bit in ihrem Statusregister setzen. Sie müssen die Handbücher überprüfen, um dies herauszufinden. In diesem Fall müssen Sie das Statusregister regelmäßig überprüfen. Schnelle Geräte, die alle paar ms einen neuen Messwert liefern können, verwenden möglicherweise weder eine Serviceanforderung noch ein Bit im Statusregister. **Wenn sie schnell sind**, wird nichts davon benötigt.

2.2.4 Datenerfassung über serielle Schnittstellen

Von Anfang an hat EZGPIB auch gelernt, mit seriellen Schnittstellen gut und einfach umzugehen. Schauen Sie sich das gut kommentierte Beispiel unten an, in dem die reguläre Ausgabe eines HP53131 Zählers gelesen wird, der als TIC (Zeitintervallzähler) verwendet wird. Danke Said, dass du mich dazu inspiriert hast! Sie finden das Programm auch unter den beigefügten Beispielen.

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
```

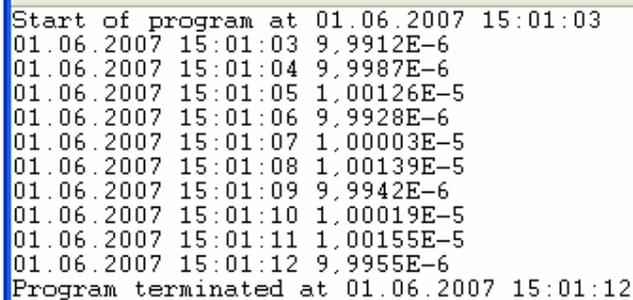
```
function Init:Boolean;  
begin;  
  if EZGPIB_FileExists(Filename)           //Comment this out if you  
  then EZGPIB_FileDelete(Filename);        //always want to append  
  EZGPIB_FileClearBuffer;                  //Clear The Filebuffer  
  EZGPIB_FileAddToBuffer('Time/MJD');      //Add two strings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

```
routine 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.
```

Wenn Sie sich die Ausgabe dieses Programms im Konsolenfenster ansehen, sieht dies wie in Abbildung 25 aus.



```
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
```

Abbildung 26

Der Inhalt der Datei, die vom Programm generiert wird (beachten Sie, dass das Verzeichnis „MyMeasurements“ automatisch erstellt wurde) ist

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

was wiederum, eingelesen mit meinem PLOTTER-Dienstprogramm, angezeigt wird als

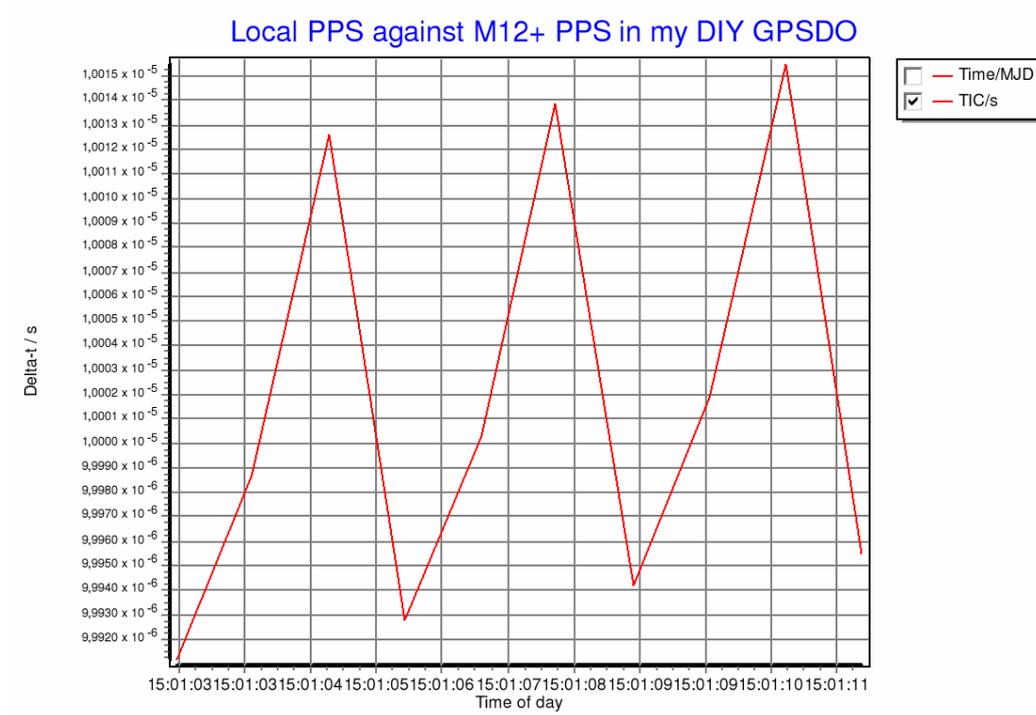


Abbildung 27

2.3 Was die Beispiele zeigen

Nachfolgend eine kurze Beschreibung dessen, was ich in den Beispielen zu demonstrieren versucht habe.

Test_01.488

Dies ist die EZGPIB Version des berühmten Programms „Hello World“.

Test_02.488

Demonstriert eine einfache **for** Schleife, grundlegende Mathematik und Konsolenausgabe.

Test_03.488

Demonstriert eine einfache **for** Schleife mit einer zeitgesteuerten **Wartebedingung**.

Test_04.488

Demonstriert eine Hauptprogrammschleife, die vom Benutzer beendet wird.

Test_05.488

Demonstriert die ersten echten Datenübertragungen vom GPIB.

Test_06.488

Funktioniert wie **Test_05.488**, verwendet jedoch Prozeduren und Funktionen, um mehr Struktur in das Source Programm zu bringen.

Test_07.488

Funktioniert wie **Test_06.488**, fügt jedoch Dateiausgabefunktionen hinzu.

Test_08.488

Funktioniert wie **Test_07.488**, fügt jedoch DDE Funktionen hinzu.

Test_09.488

Demonstriert das Konzept und die Verwendung von **“Service Requests“**.

Test_10.488

Demonstriert die Verwendung von **“include“** Dateien.

Test_11.488

Demonstriert die Kommunikation über serielle Schnittstellen.

Test_12.488

Demonstriert, wie ein Bildschirmplot von einem HP5371 empfangen wird.

Test_13.488

Demonstriert direkten Port E/A (nicht möglich bei aktuellen Windows Versionen!).

HP53131.488

Demonstriert die serielle Kommunikation mit einem HP53131.

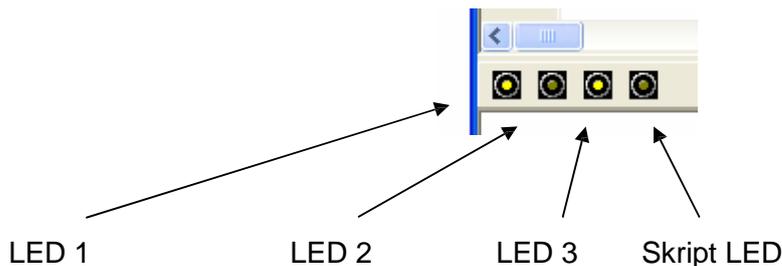
XXXX.488

Ich habe einige Skripte beigefügt, die ich für meine eigenen Anwendungen geschrieben habe. Vielleicht sind sie auch für Sie hilfreich. Sie sind nicht so gut dokumentiert, machen aber auch ihre Arbeit..

2.4 Einige neue Funktionen

2.4.1 Multi Threading

Ab der **Version 2007-07-14** haben sich die Interna von EZGPIB stark verändert. Jetzt ist EZGPIB ein Multithread Programm mit vier Threads, die ihren Zustand anhand von vier symbolisierten LEDs im Hauptfenster anzeigen..



Der erste Thread ist der **Hauptthread** des Programms, der auch dem Hauptfenster und der Windows Nachrichtenwarteschlange zugeordnet ist. Dieser Thread signalisiert seinen Arbeitszustand durch regelmäßiges Ändern des Status von **LED 1**. Beachten Sie, dass dieser Thread möglicherweise gestoppt oder verzögert ausgeführt wird, da Windows die Multitasking- und Nachrichtenwarteschlangenbehandlung verwaltet. Die anderen Threads unterliegen dem jedoch nicht, und dies war der Grund, warum EZGPIB zu einer Multithread-Anwendung wurde.

Der zweite Thread führt die gesamte **serielle Kommunikation** mit der Prologix Schnittstelle durch. Dieser signalisiert seinen Arbeitszustand durch regelmäßiges Ändern des Zustands von **LED 2**.

Der dritte Thread generiert alle Nachrichten des **GPIB Debug Fensters** und zeigt sie an. Dieser signalisiert seinen Arbeitszustand durch regelmäßiges Ändern des Zustands von **LED 3**.

Diese drei Threads sind während der gesamten Ausführung von EZGPIB aktiv. Sie sollten also sehen, dass alle drei LEDs während der gesamten Zeit in der EZGPIB aktiv ist, blinken.

Der vierte Thread ist derjenige, in dem Ihr Skript ausgeführt wird. Im Gegensatz zu den ersten drei Threads weiß ich als Autor von EZGPIB nicht genau, was in diesem Thread passiert, weil Sie das als Autor des Skripts entscheiden. Ich füge eine neue Prozedur namens **ChangeLed** hinzu, die Sie in Ihrem Skript an einer Position verwenden können, die regelmäßig ausgeführt wird, um anzuzeigen, dass auch der Skript-Thread aktiv ist.

Multithreading sollte dafür sorgen, dass Alles viel reibungsloser läuft als zuvor. Zeitgesteuerte Ereignisse in Ihrem Skript können von der GUI nicht mehr verzögert werden.

2.4.2 GPIB32.DLL Unterstützung

Ab **Version 2007-12-24** wurde Unterstützung für **GPIB32.DLL** hinzugefügt. Das bedeutet: Wenn kein Prologix Adapter gefunden wird, versucht EZGPIB, das Vorhandensein der GPIB32.DLL zu erkennen. Dies ist ein guter Indikator dafür, dass eine DLL basierte IEEE-488 Schnittstelle verfügbar ist. Anschließend testet EZGPIB, ob die DLL das Vorhandensein einer aktiven IEEE-488 Schnittstelle meldet.

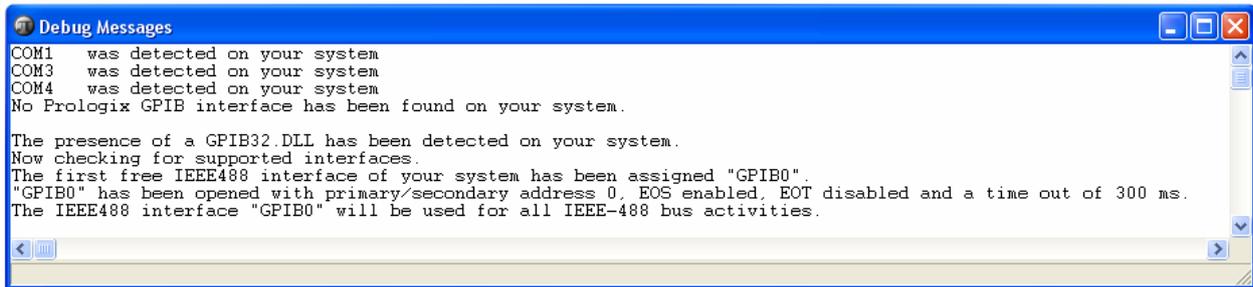


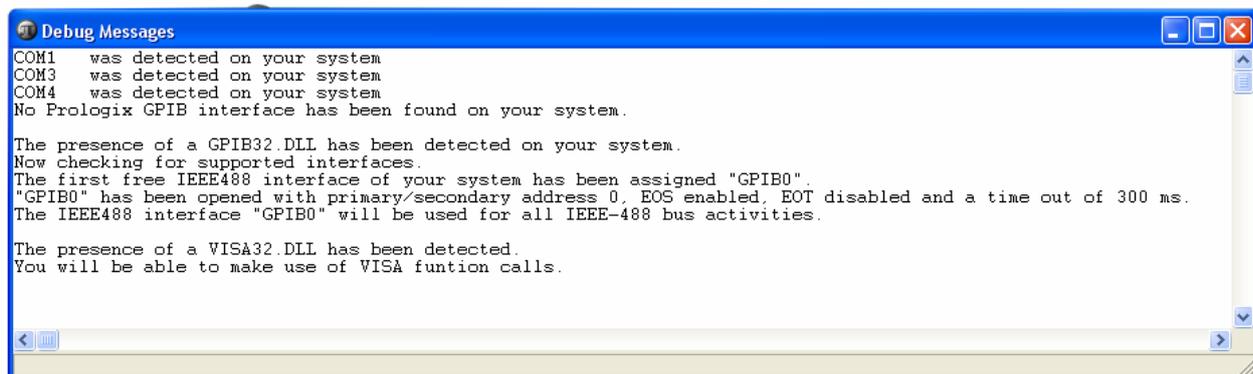
Abbildung 29

Auf meinem System sieht das Debug Fenster in diesem Fall wie oben gezeigt aus. Beachten Sie, dass die Unterstützung für GPIB32.DLL *nur mit Produkten von National Instruments getestet* wurde. Ich kann überhaupt nicht versprechen, dass diese Unterstützung mit Boards und DLL's anderer Hersteller funktioniert.

Das wirklich Schöne an dieser DLL Unterstützung ist, dass Sie nichts Neues lernen müssen. Fast alle Befehle (mit einigen geringfügigen Unterschieden, die in der Befehlsbeschreibung erläutert werden) funktionieren sowohl für den Prologix Adapter als auch für die DLL basierte Schnittstelle

2.4.3 VISA32.DLL Support

Ab **Version 2008-06-08** wurde Unterstützung für **VISA32.DLL** hinzugefügt. Das bedeutet: Wenn auf Ihrem System eine VISA-Bibliothek wie die Agilent IO-Bibliothek installiert ist, kann EZGPIB jetzt VISA Funktionsaufrufe verwenden. Es gibt nur 5 neue Funktionen, mit denen Sie lernen können, die Welt der VISA basierten Datenerfassung zu öffnen. Wenn eine VISA32.DLL erkannt wird, wird im Debug Fenster Folgendes angezeigt:



Picture 30

2.4.3.1 VISA Kommunikationsbeispiel

Im folgenden Demoskript werden alle 5 neuen Funktionen verwendet:

```
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.
```

Dieses Skript öffnet eine VISA Sitzung, stellt dann eine Verbindung zu meinem HP6632 Netzgerät an Adresse 9 des Busses her und setzt es auf 12,5 Volt Ausgangsspannung. Dann wird die Verbindung geschlossen und eine Verbindung zu meinem HP3457 Voltmeter an Adresse 10 des Busses hergestellt, um diese Spannung zurück zu lesen.

Das obige Skript zusammen mit den Debug Ausgaben befindet sich in Beispiel **VISA.488**.

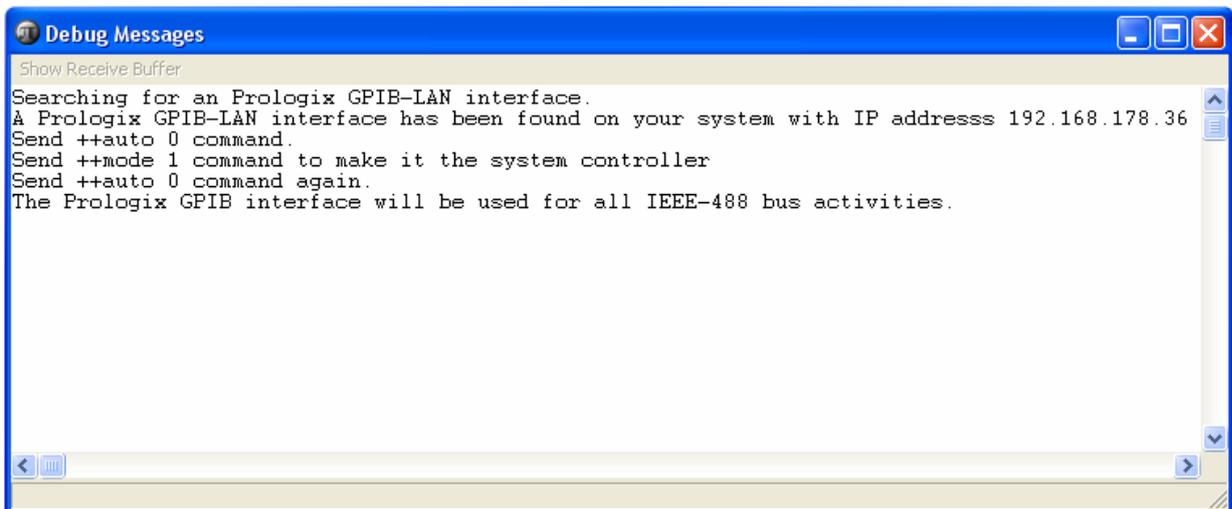
Beachten Sie, dass EZGPIB nur das Vorhandensein der VISA32.DLL erkennen kann, *aber sonst nichts!* Verwenden Sie die mit der DLL gelieferten Tools, um herauszufinden, welche Schnittstellen und welche Instrumente mit dem VISA Treiber verfügbar sind.

2.4.4 Unterstützung der Prologix LAN-GPIB Schnittstelle

Ab **Version 2008-08-02** wurde die Unterstützung für die neue Prologix LAN-GPIB-Schnittstelle integriert. Nach dem Start von EZGPIB wird als erste Aufgabe die Suche nach einer Prologix LAN-GPIB Schnittstelle durchgeführt. Beachten Sie, dass sich die IP-Adresse zur Erkennung der LAN-GPIB Schnittstelle im selben Netzwerksegment befinden muss wie die IP-Adresse Ihres PCs.

Wenn die Prologix LAN-GPIB Schnittstelle für die Verwendung von DHCP konfiguriert ist, erhält sie normalerweise eine passende IP-Adresse, wenn Sie die Schnittstelle mit Ihrem lokalen Netzwerk verbinden und ein lokaler DHCP-Server wie ein DSL-Router verfügbar ist. Wenn die Schnittstelle so konfiguriert ist, dass eine feste IP-Adresse verwendet wird, müssen Sie darauf achten, dass der IP-Adressbereich übereinstimmt. Im Prologix Handbuch finden Sie Antworten auf Fragen zur Netzwerkverwaltung, die für die LAN-GPIB-Schnittstelle erforderlich sind.

Wenn ein LAN-GPIB Adapter erkannt wird, so wird dies im Debug Fenster deutlich angezeigt, welches in diesem Fall Folgendes anzeigt:



Picture 31

Die hier angezeigte IP-Adresse unterscheidet sich natürlich auf Ihrem System. Es sind keine weiteren Maßnahmen erforderlich. Verwenden Sie EZGPIB einfach so, als ob eine USB-GPIB Schnittstelle oder eine DLL basierte PC Karte verfügbar wäre. Der einzige Unterschied, den Sie bei der Arbeit mit einer GPIB-LAN-Schnittstelle feststellen können, besteht darin, dass Sie im Debug Fenster darüber informiert werden

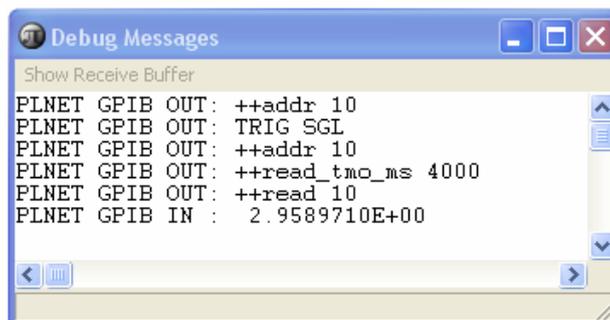


Abbildung 32

	EZGPIB Benutzerhandbuch	EZGPIB2_ger Ausgabe: 2.00 Seite 34/54
--	--------------------------------	---

Dabei bezeichnet "**PLNET**" ein netzwerkbasierendes Prologix Gerät. Bei einem USB basierten Gerät beginnt jede Zeile mit "**PLUSB**".

2.4.5 Konfiguration der 'EOI' Erkennung

Ab der **Version 2008-08-02** hat sich das Verhalten von EZGPIB hinsichtlich der Erkennung des Endes von Geräteantworten geändert. Ich hatte geglaubt, dass die Verwendung der EOI Leitung am Bus so ziemlich ein Standard ist, um das Ende einer Nachricht anzuzeigen.

Im Laufe der Zeit lernte ich immer mehr Geräte kennen, die:

- die EOI-Leitung standardmäßig NICHT verwenden! Sie müssen ausdrücklich dazu aufgefordert werden. Zu Geräten dieses Typs gehören mein Rohde & Schwarz URV-5 HF-Voltmeter und mein HP3457 Multimeter.
- die EOI Leitung *für einige Antworten* verwenden und *für andere Antworten nicht*. Was ich nicht kommentieren möchte.... Der Leistungsmesser HP437B scheint zu dieser Gruppe zu gehören.
- nicht davon überzeugt werden können, die EOI Leitung überhaupt zu nutzen. Die Racal Dana Zähler 1991 und 1992 scheinen zu dieser Gruppe zu gehören, obwohl mein eigener Zähler vom Typ 1996 EOI verwendet.

Zu erkennen, dass Sie mit einem Problem, basierend auf einer fehlenden EOI Verwendung konfrontiert sind, ist alles andere als ein einfaches Geschäft. Ich habe aus diesem Grund entschieden, dass die **Standarderkennungsmethode** für das Ende von Gerätenachrichten von nun an **EOS basiert** sein wird.

EOS basiert bedeutet, dass keine Hardware Leitung vom Bus eingelesen wird, sondern die Nachrichten selbst auf Abschlusszeichen überprüft werden. Das am häufigsten verwendete Abschlusszeichen ist der Zeilenvorschub:

<LineFeed> (ASCII Zeichen #10_{dec} oder 0A_{hex})

Für alle Schnittstellen ist EOS und <LineFeed> als EOS Zeichen jetzt Standard!

Wenn Sie dies ändern möchten, müssen Sie eine der folgenden Routinen aufrufen:

Procedure EZGPIB_BusSetEos(How:LongInt)

Diese Routine bezieht sich direkt auf den Befehl '+ + eos 0/1/2/3' des Prologix Adapters;
Anwendungsbereich: Prologix

Procedure EZGPIB_BusSetEos(How:Boolean);

Diese Routine aktiviert/deaktiviert die Verwendung des EOS Zeichens. Standardwert ist "TRUE";
Anwendungsbereich: DLL basiert

Procedure EZGPIB_BusSetEOSChar(How:Byte)

Diese Routine legt das EOS-Zeichen fest. Standardwert ist "10" (Zeilenvorschub);
Anwendungsbereich: DLL basiert

2.4.6 Verbesserungen im Menü 'File'

Ab **Version 29.10.2008** enthält das Menü Datei mehrere zusätzliche Einträge. <Parameters> und <Log Debug Messages into file> werden nachfolgend ausführlich erläutert:

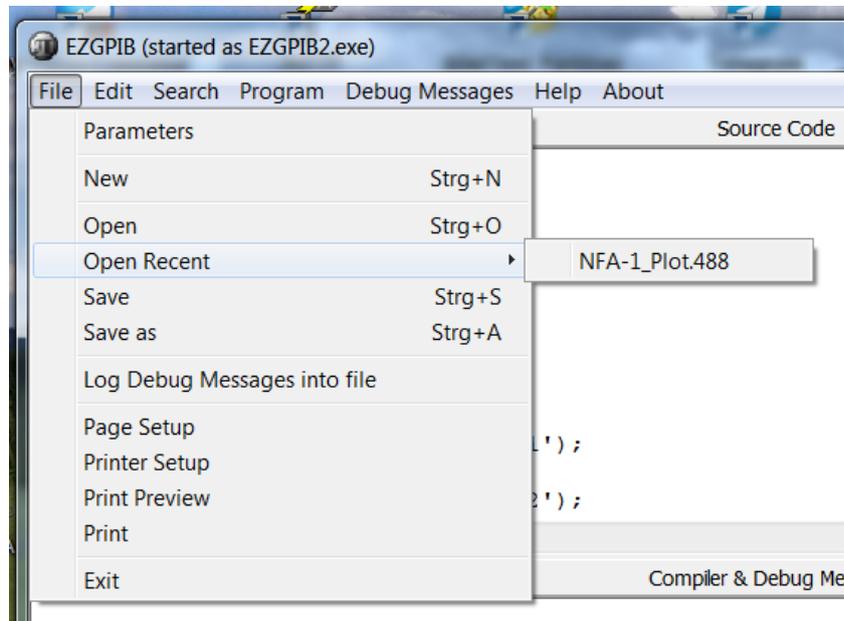


Abbildung 33

Wenn Sie <Log Debug Messages into file> aktivieren, wird jede Nachricht aus dem Fenster Debug Messages zusätzlich in der Textdatei „EZGPIB2.exe_DebugLog.Txt“ protokolliert. Suchen Sie nach dieser Datei in dem Verzeichnis, in dem sich EZGPIB2 selbst befindet.

Mit einem Mausklick auf <Parameters> wird ein Fenster geöffnet (siehe Abbildung 34).

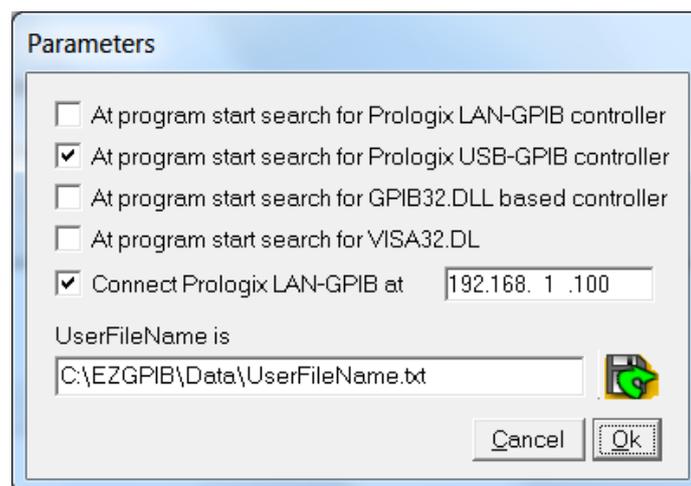
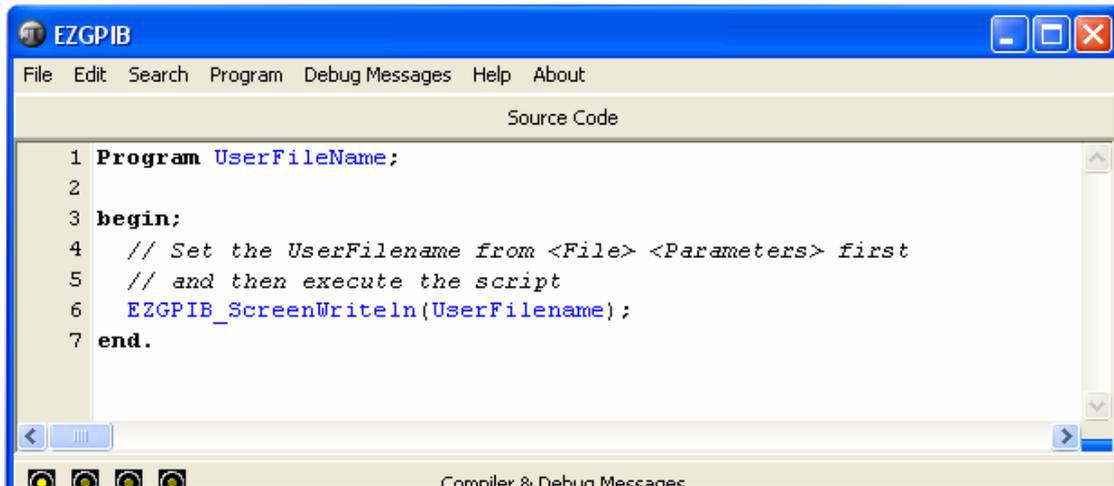


Abbildung 34

Über dieses Fenster können Sie individuell einstellen, nach welchen Arten von Schnittstellen beim Programmstart gesucht werden soll.

2.4.7 Implizite Variable 'UserFileName'

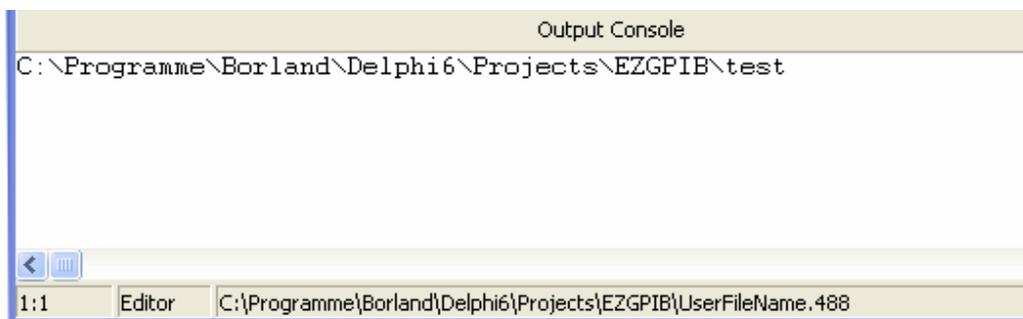
Im Parameterfenster sehen Sie den Namen einer Datei. Wenn Sie in Ihrem Skript den Variablennamen "UserFileName" verwenden, enthält der Inhalt dieser Variablen den hier angezeigten Dateinamen. Zum Beispiel das Skript



```
1 Program UserFileName;  
2  
3 begin;  
4 // Set the UserFilename from <File> <Parameters> first  
5 // and then execute the script  
6 EZGPIB_ScreenWriteln(UserFilename);  
7 end.
```

Abbildung 35

führt zum Ergebnis



```
Output Console  
C:\Programme\Borland\Delphi6\Projects\EZGPIB\test
```

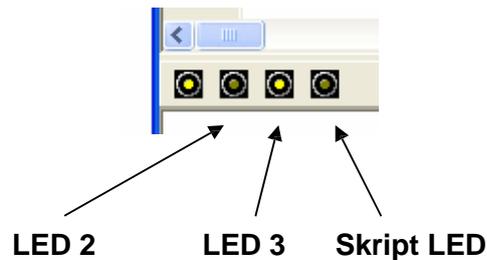
Abbildung 36

Beachten Sie, dass Sie die Variable 'UserFilename' NICHT selbst deklarieren müssen, sie ist immer mit dem richtigen Inhalt vorhanden. Sie können die Variable UserFilename direkt im Parameterfenster bearbeiten oder das Symbol rechts davon verwenden, um eine Windows Dialogbox "Speichern unter" zu öffnen

2.4.8 Einzelschritt- und Haltepunktunterstützung

Die Unterstützung für den Einzelschrittbetrieb mit F7 und F8 wurde erheblich verbessert.

Die Unterstützung für Haltepunkte im Programm mit F5 wurde erheblich verbessert. Hinweis: Wenn Ihr Programm Haltepunkte enthält, wird es jetzt bei der Ausführung anders behandelt.



Mit Haltepunkten blinken **LED 2 und LED 3 synchron** und zeigen damit an, dass das Skript im Hauptthread des Programms ausgeführt wird. Dies erleichtert die Unterstützung von Haltepunkten, hat jedoch den Nachteil, dass die Skriptausführung möglicherweise gestoppt wird, wenn Sie das Hauptfenster bearbeiten.

Ohne Haltepunkte wird das Skript in einem eigenen Thread ausgeführt und kann nicht durch Aktionen des Hauptthreads unterbrochen werden.

Haltepunkte können gesetzt/gelöscht werden, indem der Cursor in die jeweilige Zeile gesetzt und F5 mehrfach gedrückt wird.

2.4.9 Unterstützung von GPIB Haupt- und Subadressen

Normalerweise ist der GPIB Adressparameter die Adresse des Instruments auf dem GPIB Bus im Bereich von 0 bis 30. Bei älteren Instrumenten wird die Adresse häufig mit einem DIP Schalter in der Nähe der GPIB Buchse eingestellt. Neue Instrumente definieren die GPIB Adresse häufig mithilfe der Tasten auf der Vorderseite des Instruments. Bei den meisten Instrumenten ist der Adressparameter einfach die GPIB Adresse.

Manche Systeme, beispielsweise Systeme, die auf dem VXI-Bus basieren, erfordern die Verwendung einer **GPIB Subadresse**. Diese Systeme sind normalerweise Sammlungen von Instrumenten in einem einzigen Gehäuse. Das Gehäuse hat eine GPIB Adresse, und jedes einzelne Instrument darin enthält seine Adresse mit der Subadresse.

Mit EZGPIB kann die Subadresse in den oberen 8 Bits des GPIB Adressparameters angegeben werden. GPIB Subadressen liegen ebenfalls im Bereich von 0 bis 30. Die Prologix Konvention ordnet diese Adressen den Werten von 96 bis 126 zu. National Instruments unterstützt nur Werte von 1 bis 30. EZGPIB unterstützt sowohl die Prologix- als auch die National Instruments Konvention zum Definieren der Subadresse innerhalb der oberen acht Bits.

Beispiel:

Angenommen, Sie haben ein HP75000 VXI-System unter der GPIB Adresse 9 mit einem installierten Multimeter. Der Befehlsprozessor des HP75000 hat die Subadresse 0 und die Multimeter Subadresse ist 3

Bei Verwendung der **National Instruments Konvention** sind die Werte für die Subadresse

Befehlsprozessor $0 + 1 = 1$

Multimeter $3 + 1 = 4$

Um den Wert auf die oberen 8 Bits zu verschieben, multiplizieren Sie den Wert der Subadresse mit $2^8 = 256$ und fügen Sie ihn zu der GPIB Adresse hinzu

Befehlsprozessor Adresse $= 1 * 256 + 9 = 265$

Multimeter Adresse $= 4 * 256 + 9 = 1033$

Bei Verwendung der **Prologix Konvention** sind die Werte für den Subadressenbefehl

Befehlsprozessor $0 + 96 = 96$

Multimeter $3 + 96 = 99$

Um den Wert auf die oberen 8 Bits zu verschieben, multiplizieren Sie den Wert der Subadresse mit $2^8 = 256$ und fügen Sie ihn zu der GPIB Adresse hinzu

Befehlsprozessor Adresse $= 96 * 256 + 9 = 24585$

Multimeter Adresse $= 99 * 256 + 9 = 25353$

Natürlich ist es bequemer und verständlicher, den EZGPIB Compiler die Berechnung für Sie durchführen zu lassen. Hier ist der Teil des Codes, der die Subadresse aus den Beispielskripten **HP75000Test.488** und **HP75000TestA.488** definiert.

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);
```

Wenn das Fenster "Debug Messages" aktiviert ist und die Prologix Schnittstelle verwendet wird, führen diese beiden Definitionen zu den folgenden Adressbefehlen:

Für den Befehlsprozessor:

++addr 9 96

Für das Multimeter:

++addr 9 99

	EZGPIB Benutzerhandbuch	EZGPIB2_ger Ausgabe: 2.00 Seite 39/54
--	--------------------------------	---

3 EZGPIB Funktionen und Routinen

Im Folgenden finden Sie eine Liste der verfügbaren Funktionen und Routinen, die Sie mit dem Menüeintrag "Help" auflisten können. Es gibt dabei einige, die speziell für EZGPIB programmiert wurden.

Die Namen aller speziellen Funktion beginnen mit **EZGPIB_**. Die nächsten Zeichen nach dem Unterstrich versuchen anzugeben, in welche Kategorie die Routine oder Funktion fällt. Während "Bus" angibt, dass es sich um etwas GPIB spezifisches handeln muss, zeigt "Kbd" an, dass die Routine oder Funktion für die Tastatur relevant ist und so weiter.

Beachten Sie, dass Sie **anstelle der wiederholten Eingabe von EZGPIB_** im Editor die Tastenkombination **STRG-E** (oder CTRL-E auf US Tastaturen) verwenden können, um einen EZGPIB_ Eintrag zu generieren.

3.1 Buskonfiguration und -Steuerung (Prologix und DLL)

3.1.1 Abfrage von Bus- und Adapterparametern

Procedure EZGPIB_BusFindAllDevices

Diese Routine versucht alle aktiven Geräte am Bus durch Lesen ihres Statusbytes zu erkennen. Sie können das Debug Fenster öffnen, um zu sehen, was los ist. Jedes Gerät im Bereich von 0 bis 30 wird aufgefordert, sein Statusbyte zu melden.

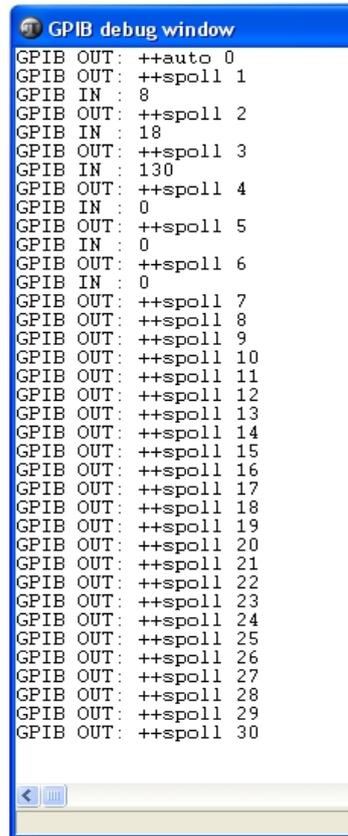
Die Routine sollte am Beginn jedes GPIB-Programms aufgerufen werden, das Sie schreiben!

Sie setzt einige interne Variablen von EZGPIB, die von anderen Funktionen verwendet werden (z. B. die höchste derzeit verwendete Geräteadresse), auf den richtigen Wert.

Jedes Gerät, das einen **Messwert liefern** kann, sollte ein Statusbyte haben. Möglicherweise gibt es einige **reine Ausgabegeräte** (Talk Only), die kein Statusbyte haben und auf diese Weise nicht erkannt werden können.

Ich habe z.B. festgestellt, dass der Wavetek 278 Funktionsgenerator kein Statusbyte hat.
Anwendungsbereich: Prologix & DLL basierend

So könnte die Ausgabe in Ihrem Debug Fenster aussehen, während die Routine "EZGPIB_BusFindAllDevices" ausgeführt wird, die den Bus nach verfügbaren GPIB-Geräten durchsucht:



```
GPIB debug window
GPIB OUT: ++auto 0
GPIB OUT: ++spoll 1
GPIB IN : 8
GPIB OUT: ++spoll 2
GPIB IN : 18
GPIB OUT: ++spoll 3
GPIB IN : 130
GPIB OUT: ++spoll 4
GPIB IN : 0
GPIB OUT: ++spoll 5
GPIB IN : 0
GPIB OUT: ++spoll 6
GPIB IN : 0
GPIB OUT: ++spoll 7
GPIB OUT: ++spoll 8
GPIB OUT: ++spoll 9
GPIB OUT: ++spoll 10
GPIB OUT: ++spoll 11
GPIB OUT: ++spoll 12
GPIB OUT: ++spoll 13
GPIB OUT: ++spoll 14
GPIB OUT: ++spoll 15
GPIB OUT: ++spoll 16
GPIB OUT: ++spoll 17
GPIB OUT: ++spoll 18
GPIB OUT: ++spoll 19
GPIB OUT: ++spoll 20
GPIB OUT: ++spoll 21
GPIB OUT: ++spoll 22
GPIB OUT: ++spoll 23
GPIB OUT: ++spoll 24
GPIB OUT: ++spoll 25
GPIB OUT: ++spoll 26
GPIB OUT: ++spoll 27
GPIB OUT: ++spoll 28
GPIB OUT: ++spoll 29
GPIB OUT: ++spoll 30
```

Abbildung 33

Function EZGPIB_BusGetAddressedDevice:LongInt

Gibt die Adresse des aktuell adressierten Busgeräts zurück.
Anwendungsbereich: Prologix

Function EZGPIB_BusGetTimeOut:Double

Fragt die aktuelle Einstellung des Zeitlimits in [ms] beim Lesen vom GPIB-Bus ab.
Der Bereich liegt zwischen 0 und 32.000 Millisekunden (32 Sekunden).
Diese Funktion verwendet den Befehl '+read_tmo_ms' des Prologix Adapters.
Anwendungsbereich: Prologix

Function EZGPIB_BusGetEoi:LongInt

Fragt die aktuelle Einstellung für die EOI Signalisierung ab.
Diese Funktion verwendet den Befehl '+eoi' des Prologix Adapters.
Anwendungsbereich: Prologix

	EZGPIB Benutzerhandbuch	EZGPIB2_ger Ausgabe: 2.00 Seite 41/54
--	--------------------------------	---

Function EZGPIB_BusSourceOfSrq:LongInt

Gibt die Adresse des Geräts zurück, das gerade den Service anfordert. Wenn mehrere Geräte einen Service anfordern, gibt die Funktion die niedrigste Geräteadresse zurück, die Service benötigt.

Es ist unbedingt notwendig, dass "**EZGPIB_BusFindAllDevices**" einmal aufgerufen wurde, bevor Sie diese Funktion verwenden.
Anwendungsbereich: Prologix & DLL basiert

Function EZGPIB_BusWaitForSrq(MaxWait:Double):Boolean

Diese Funktion wartet für "MaxWait" Sekunden oder bis eine SRQ auftritt, je nachdem, welcher Zeitpunkt früher liegt. Wenn innerhalb der artezeit ein SRQ festgestellt wurde, wird "TRUE" zurückgegeben.

Anwendungsbereich: Prologix & DLL basiert

3.1.2 Setzen von Bus- und Adapterparametern

Procedure EZGPIB_BusAddressDevice(Which:LongInt)

Setzt die GPIB-Adresse auf den Wert "Which". Die Bedeutung der GPIB-Adresse hängt von der Betriebsart des Controllers ab. Im CONTROLLER Modus bezieht sich dies auf die GPIB Adresse des zu steuernden Instruments. Im DEVICE-Modus ist es die Adresse des GPIB Peripheriegeräts, das der Prologix Adapter emuliert. Diese Funktion gibt direkt den Befehl '++ addr Which' an den Prologix Adapter aus.

Beachten Sie, dass EZGPIB keinen speziellen Sprachbefehl enthält, um den Controller in den DEVICE Modus zu bringen! Um in diese Betriebsart zu wechseln, müssen Sie den erforderlichen Befehl direkt, sozusagen "zu Fuß" senden, indem Sie Folgendes aufrufen:

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

wobei "DeviceAddr" die Adresse des GPIB Peripheriegeräts ist, das der Controller emuliert.
Anwendungsbereich: Prologix

Procedure EZGPIB_BusSetEOS(How:Booelan);

Aktiviert/deaktiviert die Verwendung eines EOS GPIB Abschlusszeichens.

Der Standardwert ist "TRUE".

Anwendungsbereich: DLL basiert

Procedure EZGPIB_BusSetEOSChar(How:Byte)

Setzt den Wert für das EOS Abschlusszeichen. Standardwert ist "10" (Zeilenvorschub).

Anwendungsbereich: DLL basiert

3.2 Program Flow Functions

Function EZGPIB_BusSPoll(Device:LongInt):LongInt

Führt eine serielle Busabfrage durch, um das Statusbyte von "Device" zurückzugeben.
Anwendungsbereich: Prologix & DLL basiert

Function EZGPIB_BusSrq:Boolean

Gibt "TRUE" zurück, wenn ein Gerät einen Service angefordert hat.
Anwendungsbereich: Prologix & DLL basiert

Function EZGPIB_BusSrqStatus:LongInt

Gibt das Statusbyte des Gerätes, das den Service angefordert hat, zurück.
Anwendungsbereich: Prologix & DLL basiert

Procedure EZGPIB_BusTrigger

Sendet einen Trigger Befehl an das aktuell adressierte Gerät. Das Gerät muss auf den Single Trigger-Modus eingestellt und vom GPIB Controller ferngesteuert werden.
Anwendungsbereich: Prologix & DLL basiert

3.3 Prologix Adapter Einstellungen

Function EZGPIB_BusGetVer:String

Gibt den Adaptertyp und die Softwareversion zurück, die vom cmd "++ ver" vom Prologix Adapter zurückgegeben wurden.
Anwendungsbereich: Prologix

Procedure EZGPIB_BusAutoOFF

Diese Routine sendet den Befehl '++auto 0' an den Prologix Adapter.
Anwendungsbereich: Prologix

Procedure EZGPIB_BusAutoON

Diese Routine sendet den Befehl '++auto 1' an den Prologix Adapter.
Anwendungsbereich: Prologix

Procedure EZGPIB_BusDisableEOI

Diese Routine sendet den Befehl '++eoi 0' an den Prologix Adapter.
Anwendungsbereich: Prologix

Procedure EZGPIB_BusEnableEOI

Diese Routine sendet den Befehl '++eoi 1' an den Prologix Adapter.
Anwendungsbereich: Prologix

Procedure EZGPIB_BusGotoLocal(Device;integer)

Diese Routine sendet den Befehl '++loc' an den Prologix Adapter.
Anwendungsbereich: Prologix & DLL basiert

Procedure EZGPIB_BusSetEos(How:LongInt)

Diese Routine sendet die Befehle '++eos 0/1/2/3' an den Prologix Adapter.
Anwendungsbereich: Prologix

3.4 Gerätesteuerung

Procedure EZGPIB_BusIFC

Rücksetzen aller Geräteschnittstellen durch Aktivierung der IFC Leitung am GPIB Bus.
Anwendungsbereich: Prologix & DLL basiert

Procedure EZGPIB_LocalLockOut

Sperrt die "Local" Betriebsart am adressierten Gerät.
Anwendungsbereich: Prologix & DLL basiert

Procedure EZGPIB_BusSetTimeOut(How:Double)

Legt den maximalen Wert in [ms] der Wartezeit beim Lesen vom GPIB-Bus fest.
Der Bereich liegt zwischen 0 und 32.000 Millisekunden (32 Sekunden)
Anwendungsbereich: Prologix & DLL basiert

3.4.1 Lesen vom Bus

Function EZGPIB_BusDataAvailable:Boolean

Gibt "TRUE" zurück, wenn ein Gerät am Bus Daten gesendet und mit EOI oder dem aktuellen EOS-Trennzeichen beendet hat.
Anwendungsbereich: Prologix

Function EZGPIB_BusGetData:string

Gibt die Zeichenfolge zurück, die zuletzt von einem Gerät über den Bus gesendet wurde.
Anwendungsbereich: Prologix

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

Wartet "MaxWait" Sekunden oder bis das Gerät "Device" antwortet, je nachdem, was früher stattfindet. Wenn "Device" innerhalb des Zeitlimits antwortet, wird die Antwort in "ForWhat" zurückgegeben und die Funktion gibt "TRUE" zurück. Das Ende der Nachricht wird an dem vom Gerät gesendeten Trennzeichen erkannt.

Verwenden Sie diese Funktion zum Einlesen einzelner Messwerte.
Anwendungsbereich: Prologix & DLL basiert

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

Wartet "MaxWait" Sekunden auf eine Antwort von gerät "Device". Wenn "Device" innerhalb des Zeitlimits antwortet, wird die Antwort in "ForWhat" zurückgegeben und die Funktion gibt "TRUE" zurück. Diese Funktion betrachtet KEINE Trennzeichen, die Teil der Antwort sein könnten, sondern wartet immer die gesamte Zeit "MaxWait" ab.

Verwenden Sie diese Funktion zum Einlesen längerer Datenblöcke wie Screenshots und Ähnliches.
Anwendungsbereich: Prologix & DLL basiert

	EZGPIB Benutzerhandbuch	EZGPIB2_ger Ausgabe: 2.00 Seite 44/54
--	--------------------------------	---

3.4.2 Schreiben am Bus

Procedure EZGPIB_BusWriteData(Device:LongInt;What:string)

Sendet die Zeichenfolge "What" über den GPIB Bus an das Gerät "Device".
Anwendungsbereich: Prologix & DLL basiert

3.5 Kommunikation über serielle Schnittstellen

3.5.1 Einrichten des seriellen Anschluss

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

Zusätzlich zur Schnittstelle, an die der Prologix Adapter angeschlossen ist, kann EZGPIB so viele serielle Schnittstellen für die Kommunikation mit anderen seriellen Geräten verarbeiten, wie Windows selbst verarbeiten kann, die Sie für Ihre eigene Kommunikation mit seriellen Geräten frei verwenden können. Diese Funktion öffnet den Port und gibt "TRUE" zurück, wenn dies ohne Fehler möglich war. Beachten Sie, dass Sie seriellen Schnittstellen, die Ihre Anwendung geöffnet hat, nicht schließen müssen. Dies wird automatisch für Sie erledigt, wenn Ihre Anwendung beendet wird.

3.5.2 Lesen vom seriellen Anschluss

Function EZGPIB_ComRead(Which:Integer):string

Diese Funktion liest jene Zeichenfolge ein, die im seriellen Eingangspuffer von COM Port "Which" verfügbar ist.

3.5.3 Schreiben zum seriellen Anschluss

Procedure EZGPIB_ComWrite(Which:Integer;What:string)

Diese Routine gibt die Zeichenfolge "What" an den seriellen COM Port "Which" aus.

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

Gibt die Zeichenfolge "What" an den seriellen COM Port "Which" aus. Nach jedem Zeichen wird eine mit "DelayMs" [ms] festgelegte Pause eingelegt, um langsamen externen Geräten Zeit zum Verarbeiten zu geben.

3.5.4 Serielle Schnittstelle Steuerpins abfragen/setzen

Function EZGPIB_ComCTS(Which:Integer):Boolean

Diese Funktion überprüft den Zustand des CTS-Pins des COM Ports „Which“

Function EZGPIB_ComDSR(Which:Integer):Boolean

Diese Funktion überprüft den Zustand des DSR-Pins des COM Ports „Which“

Procedure EZGPIB_ComSetBreak(Which:Integer;How:Boolean)

Diese Routine kann dazu verwendet werden, die Sendedatenleitung des COM Ports "Which" statisch auf "0" oder "1" zu setzen. um eine Unterbrechungsbedingung (Break) auf der seriellen Schnittstelle zu erzeugen.

Procedure EZGPIB_ComSetDTR(Which:Integer;How:Boolean)

Diese Routine kann dazu verwendet werden, die DTR Leitung des COM Ports "Which" statisch dauernd auf "0" oder "1" zu setzen.

Procedure EZGPIB_ComSetRTS(Which:Integer;How:Boolean)

Diese Routine kann dazu verwendet werden, die CTS Leitung des COM Ports "Which" statisch dauernd auf "0" oder "1" zu setzen.

3.6 String Funktionen

3.6.1 Numerisch zu String Konvertierung

Function EZGPIB_ConvertHextoInt(HexString:string):string

Konvertiert eine Zeichenfolge, die die hexadezimale Darstellung einer Zahl enthält, in eine Zeichenfolge mit der dezimalen Darstellung dieser Zahl.

Function EZGPIB_ConvertStripToNumber(V:Variant):string

Entfernt alle Zeichen aus einer Zeichenfolge, die nicht zu einer Zahlendarstellung gehören.

Function EZGPIB_ConvertToDecimalComma(Where:string):string

Konvertiert eine Zeichenfolge mit einer **Zahl mit Dezimalpunkt** in eine Zeichenfolge mit einer **Zahl mit Dezimalkomma**.

Function EZGPIB_ConvertToDecimalPoint(Where:string):string

Konvertiert eine Zeichenfolge mit einer **Zahl mit Dezimalkomma** in eine Zeichenfolge mit einer **Zahl mit Dezimalpunkt**.

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

Konvertiert den Variantenwert "V" (Zeichenfolge, Ganzzahl oder Gleitkomma) in eine Zeichenfolge im **Exponentialformat** der Gesamtlänge "TotalLength" und einer Länge des Exponenten von "ExponentLength".

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

Konvertiert den Variantenwert "V" (Zeichenfolge, Ganzzahl oder Gleitkomma) in eine Zeichenfolge im **Festkommaformat** mit einer Länge von "Digits".

3.6.2 String zu Numerisch

Function EZGPIB_ConvertToFloatNumber(Which:string):Double

Konvertiert die Zeichenfolge "Which" in eine reelle Zahl.

Function EZGPIB_ConvertToIntNumber(Which:string):LongInt

Konvertiert die Zeichenfolge "Which" in eine Ganzzahl.

	EZGPIB Benutzerhandbuch	EZGPIB2_ger Ausgabe: 2.00 Seite 46/54
--	--------------------------------	---

3.6.3 Datum zo String

Function EZGPIB_ConvertToMJD(What:Variant):string

Konvertiert den Wert von Zeit/Datum "What" (Zeichenfolge, Ganzzahl oder Gleitkomma) in eine Zeichenfolge, die die Darstellung des modifizierten Julianischen Datums enthält.

(Verwenden Sie **EZGPIB_TimeNow**, um das aktuelle Datum und die aktuelle Uhrzeit im **TDateTime** Format abzurufen).

3.6.4 Allgemeines Stringfunktionen

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

Gibt das N-te Argument aus der Zeichenfolge "Where" zurück, in der die einzelnen Argumente durch das Trennzeichen "Delimiter" getrennt sind.

Procedure EZGPIB_ConvertAddToString(Where:string;What:Variant)

Verwenden Sie diese Routine, um den Variantenwert "What" (Zeichenfolge, Ganzzahl oder Gleitkomma) an das Ende der Zeichenfolge "Where" anzuhängen.

Procedure EZGPIB_ConvertRemove(What:string;FromWhere:string)

Entfernt alle Vorkommen der Zeichenfolge "What" in der Zeichenfolge "Where".

3.7 Datei E/A

3.7.1 Allgemein

Function EZGPIB_FileExists(Which:string):Boolean

Gibt "TRUE" zurück, wenn die Datei "Which" bereits vorhanden ist.

Procedure EZGPIB_FileDelete(Which:string)

Löscht die Datei "Which". Die Zeichenfolge "Which" kann auch einen Pfad zur Datei enthalten.

Procedure EZGPIB_FileExecute(WhichProgram:string;CMDParameters:string)

Innerhalb eines EZGPIB Programms können Sie ein anderes Programm mit dem Dateinamen "WhichProgram" starten und ihm dabei die Parameter "CMDParameters" übergeben.

3.7.2 Lesen aus Dateien

Function EZGPIB_FileReadClose:Boolean

Schließt die zum Lesen geöffnete Textdatei und meldet das Ergebnis.

Function EZGPIB_FileReadEOF:Boolean

Gibt die EOF Bedingung (Dateiende) einer zum Lesen geöffneten Textdatei zurück..

Function EZGPIB_FileReadGetBuffer:string

Liest und gibt die nächste Zeile der Textdatei zurück, die zum Lesen geöffnet ist.

Function EZGPIB_FileReadOpen(Datafilename:string):Boolean

	EZGPIB Benutzerhandbuch	EZGPIB2_ger Ausgabe: 2.00 Seite 47/54
--	--------------------------------	---

Öffnet eine Textdatei zum Lesen und gibt das Ergebnis zurück.

3.7.3 Schreiben in Dateien

Procedure **EZGPIB_FileAddToBuffer(What:Variant)**

EZGPIB bietet einen sehr einfachen Mechanismus für die Dateiausgabe. Grundsätzlich gibt es eine interne Dateipuffervariable, die eine Zeichenfolge ist und eine Zeile der Datendatei darstellt. Mit dieser Routine fügen Sie am Ende des Dateipuffers ein neues Element hinzu, das Sie in die Datei schreiben möchten. Ein Tabulatorzeichen wird automatisch in den Dateipuffer eingefügt, bevor "What" an den Dateipuffer angehängt wird. Mit dieser Routine setzen Sie die nächste Zeile zusammen, die in die Datei geschrieben werden soll. Wenn die Zeile vollständig ist, verwenden Sie **EZGPIB_FileWrite**, um den Dateipuffer in die physische Datei auszugeben.

Procedure **EZGPIB_FileClearBuffer**

Diese Routine löscht die interne Dateipuffervariable, um vorherige Zeilen zu entfernen und eine neue Ausgabezeile zu erstellen.

Procedure **EZGPIB_FileWrite(Where:string)**

Schreibt den Dateipuffer in die Datei "Where". "Where" kann auch den Pfad der Datei enthalten. Wenn die Datei nicht vorhanden ist, wird sie automatisch erstellt (einschließlich des erforderlichen Pfads!). Wenn die Datei schon vorhanden ist, wird der Dateipuffer an das Ende der Datei angehängt. Danach wird die Datei geschlossen.

Bitte beachten Sie, dass dies aufgrund des Overheads des Betriebssystems eine langsam arbeitende Funktion ist.

3.8 Tastatureingaben

Function **EZGPIB_KbdKeyPressed:Boolean**

Gibt "TRUE" zurück, wenn eine Taste gedrückt wurde. Wird zusammen mit der folgenden Funktion verwendet.

Function **EZGPIB_KbdReadKey:Char**

Liest eine Taste von der Tastatur. Sie müssen vorher immer mit **EZGPIB_KbdKeyPressed** selbst testen, ob eine Taste mit gedrückt wurde oder nicht.

Function **EZGPIB_KbdReadLn:Variant**

Wartet auf eine Tastatureingabe, die mit einem <return> beendet wird.

3.9 Telnet

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

Öffnet eine Telnet Verbindung zur IP-Adresse "IPAddress" und Port "Port" und gibt zurück, ob die Verbindung hergestellt werden konnte. Diese Verbindung erhält den Namen "Name". Sie müssen die von Ihrer Anwendung erstellten Telnet Verbindungen nicht trennen oder schließen. Dies erfolgt automatisch, wenn Ihre Anwendung beendet wird.

Function **EZGPIB_TelnetRead(Name:string):string**

Liest alle verfügbaren Daten von der aktiven Telnet-Verbindung "Name".

	EZGPIB Benutzerhandbuch	EZGPIB2_ger Ausgabe: 2.00 Seite 48/54
--	--------------------------------	---

Procedure EZGPIB_TelnetWrite(Name:string;What:string)

Schreibt die Zeichenfolge "What" zur aktiven Telnet Verbindung "Name".

3.10 Datum/Zeit

Function EZGPIB_TimeNewSecond:Boolean

Gibt "TRUE" zurück, wenn seit dem letzten Aufruf der Funktion eine neue Sekunde eingetroffen ist.

Function EZGPIB_TimeNow:TDateTime

Gibt das aktuelle Datum und die aktuelle Uhrzeit im **TDateTime** Format zurück.

3.11 DDE

Procedure EZGPIB_DDEServerCreateltem(Item:string)

EZGPIB kann ein DDE Server für andere Programme sein. Verwenden Sie diese Routine, um ein neues DDE Element mit dem Namen "Item" zu erstellen.

Procedure EZGPIB_DDEServerAssignvalue(Item:string;Value:string)

Diese Routine weist einem DDE Element (das mit **EZGPIB_DDEServerCreateltem** erstellt wurde) einen Wert zu.

Procedure EZGPIB_DDEServerClearAll

Verwenden Sie diese Routine, um alle zuvor erstellten DDE Elemente zu löschen.

3.12 Verschiedenes

3.12.1 Debug Meldungen ausgeben

Procedure EZGPIB_DebugWriteLn(V:Variant)

Gibt den Variantenwert "V" (Zeichenfolge, Ganzzahl oder Gleitkomma) an das Fenster "Debug Messages" aus.

Erzeugt anscheinend keine Ausgabe im Debug Fenster? (zumindest auf meinem PC ;-)).

3.12.2 Direkter Port E/A

Direkter Port E/A verursacht sehr wahrscheinlich Probleme bei aktuellen Versionen des Windows Betriebssystems und funktioniert offensichtlich bei allen 64-Bit Windows Versionen nicht!

Procedure EZGPIB_PortOut(Port:Word;What:Byte)

Unter Verwendung von INOUT.DLL können EZGPIB-Programme direkte Port-E/A's ausführen. Diese Routine schreibt den Wert des Bytes "What" in den Port "Port", wenn dieser betriebsbereit ist.

Function EZGPIB_PortIn(Port:Word):Byte

Unter Verwendung von INOUT.DLL können EZGPIB-Programme direkte Port-E/A's ausführen. Diese Funktion gibt den aktuellen Wert von Port "Port" zurück, wenn dieser betriebsbereit ist.

	EZGPIB Benutzerhandbuch	EZGPIB2_ger Ausgabe: 2.00 Seite 49/54
--	--------------------------------	---

3.12.3 Skript LED

Procedure EZGPIB_ChangeLed

Ändert (toggelt) den Status der Skript LED ganz rechts. Verwenden Sie diese Routine, um anzuzeigen, dass Ihr Skript regelmäßig eine bestimmte Codezeile ausführt.

3.13 Zeitverzögerung

Procedure EZGPIB_TimeSleep(HowLong:Double)

Tut nichts und wartet für "HowLong" Sekunden.

Procedure EZGPIB_TimeWaitForMultipleOf(Seconds:LongWord)

Wartet, bis eine ganzzahlige Anzahl von Sekunden erreicht wurde.

Beispiel:

EZGPIB_TimeWaitForMultipleOf(60) wartet bis zum Start der nächsten Minute,
EZGPIB_TimeWaitForMultipleOf(1800) wartet bis zum Start der nächsten halben Stunde.

3.14 Ausgabe am Konsolenfenster

3.14.1 Information am Schirm schreiben

Procedure EZGPIB_ScreenWrite(V:Variant)

Schreibt den Variantenwert "V" (kann Zeichenfolge, Ganzzahl oder Gleitkomma sein) an der aktuellen Cursorposition.

Procedure EZGPIB_ScreenWriteLn(V:Variant)

Schreibt den Variantenwert "V" (Zeichenfolge, Ganzzahl oder Gleitkomma) an die aktuelle Cursorposition und setzt den Cursor an den Anfang der nächsten Zeile.

3.14.2 Bildschirm bearbeiten

Procedure EZGPIB_ScreenClear

Löscht das Konsolenfenster und positioniert den Cursor auf Zeile 1, Position 1.

Procedure EZGPIB_ScreenClearEol

Löscht in der aktuellen Cursorzeile von der Cursorposition bis zum Zeilenende.

Procedure EZGPIB_ScreenCursorOff

Schaltet den Bildschirmcursor der Konsole AUS.

Procedure EZGPIB_ScreenCursorOn

Schaltet den Bildschirmcursor der Konsole EIN.

Procedure EZGPIB_ScreenGotoXY(x:LongInt;y:LongInt)

Positioniert den Cursor des Konsolenfensters auf die Position "x" in Zeile "y".

3.15 VI Funktionalität

3.15.1 Öffnen

Function EZGPIB_viOpenDefaultRM(var RM:Integer):Integer;

Öffnet eine VISA Sitzung. Gibt ein Handle in der Variablen "RM" für die Sitzung zurück, welches in den nachfolgenden VISA Funktionsaufrufen verwendet werden muss. Das Funktionsergebnis ist "0", wenn der Aufruf erfolgreich war.

Function EZGPIB_viOpen(RM:Integer;ResourceName:String;AccessMode:Integer;Timeout:Integer;var VI:Integer):Integer;

Öffnet eine VISA Verbindung zu einem einzelnen Instrument. Gibt ein Handle für diese Verbindung in der Variablen "VI" zurück, das für alle nachfolgenden Lese-/Schreib- und Schließaufrufe verwendet werden muss. Das Funktionsergebnis ist "0", wenn der Aufruf erfolgreich war.

3.15.2 Schliessen

Function EZGPIB_viClose(VI:Integer):Integer;

Schließt die Verbindung zum Handle "VI". Das Funktionsergebnis ist "0", wenn der Aufruf erfolgreich war.

3.15.3 Lesen

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

Liest die Zeichenfolge aus dem Sendepuffer der VISA Verbindung "VI". Gibt die Anzahl der gelesenen Zeichen in "RetCount" zurück. Das Funktionsergebnis ist "0", wenn der Aufruf erfolgreich war.

3.15.4 Schreiben

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

Schreibt die Zeichenfolge in den Empfangspuffer der VISA Verbindung "VI". Gibt die Anzahl der geschriebenen Zeichen in "RetCount" zurück. Das Funktionsergebnis ist "0", wenn der Aufruf erfolgreich war.

4 Software Änderungsübersicht

- 20070330 Empfang großer Datenblöcke (Bildschirmplots) mit der neuen Routine **BusWaitForDataBlock** stark verbessert.
- 20070531 Bedient jetzt eine unbegrenzte Anzahl von seriellen Schnittstellen.
Bedient jetzt eine unbegrenzte Anzahl von Telnet Verbindungen.
Kann die Firmware des Prologix Adapters ab Firmware 4.2 aktualisieren.
Kann den neuen '++read' Befehl des Prologix Adapters vollständig nutzen, bleibt jedoch abwärtskompatibel mit Firmware Version 3.12c.
Routine **EZGPIB_ConvertRemove** hinzugefügt.
Funktion **EZGPIB_FileExists** hinzugefügt.
Routine **EZGPIB_LocalLockout** hinzugefügt.
Routine **EZGPIB_GTL** auf **EZGPIB_GotoLocal** umbenannt.
Das Debug Fenster ist jetzt vollständig Thread gesteuert.
- 20070821 Funktionen und Routinen für die Verarbeitung von Dateieingaben hinzugefügt. Schauen Sie sich Funktionen an, die mit **fileopen** beginnen, und beachten Sie auch die **NthArgument** Funktion.
Programm komplett auf Multithreading umgeschrieben.
- 20071224 Unterstützung für DLL basierte Adapter hinzugefügt.
- 20080126 Einige Fehler behoben.
- 20080608 VISA Unterstützung hinzugefügt.
- 20080619 Einige Fehler bei der Handhabung von EOS beseitigt.
- 20080802 Unterstützung für die Prologix LAN GPIB-Schnittstelle integriert. Die Erkennung von Gerätenachrichten ist nun standardmäßig auf EOS eingestellt, wobei <LF> das Standard-EOS-Zeichen ist.
- 20080809 Einige Fehler wurden behoben, die bei Einführung der LAN-GPIB Schnittstelle aufgetreten waren.
- 20081129 Einige Fehler behoben.
- 20090213 Einige Fehler entfernt. Das Kapitel „EZGPIB Funktionen und -Prozeduren“ wurde dem Handbuch hinzugefügt. Dieser Teil wurde von Jack Smith, K8ZOA, geschrieben. Vielen Dank, Jack, im Namen aller EZGPIB-Benutzer!

20090531 Unterstützung für GPIB Sekundäradressen wurde hinzugefügt. Lesen Sie Kap. [2.4.9](#) „Unterstützung von GPIB Haupt- und Subadressen“, geschrieben von David J. Holigan, daveh@essnh.com. Vielen Dank Dave, im Namen aller EZGPIB-Benutzer!

EZGPIB kann jetzt mehrmals gestartet werden wenn sich die EXE-Dateien in verschiedenen Unterverzeichnissen befinden. Eine Version kann also mit einem USB Gerät kommunizieren, eine zweite mit einem LAN basierten Gerät und eine dritte mit einem DLL basierten Gerät.

20091107 Eine der größten Verbesserungen von EZGPIB ist, dass ich einen fehlerfreien Weg für die Kommunikation zwischen EZGPIB und ProfiLab gefunden habe. ProfiLab ist ein deutsches (kann auch in englischer und französischer Sprache installiert werden) Baukastensystem für Datenerfassungssysteme, das in vielerlei Hinsicht mit LABVIEW vergleichbar ist. Es hat jedoch eine viel geringeren Fußabdruck als LABVIEW, ist in seinen Fähigkeiten gegenüber LABVIEW etwas eingeschränkt, kostet aber weniger als 100 € !!

Erfahren Sie mehr über ProfiLab unter

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

Einer der wenigen Einschränkungen von ProfiLab ist, dass es keine GPIB Kommunikation bietet. Das macht es zum idealen Partner für EZGPIB, was die GPIB Kommunikation zum Kinderspiel macht. EZGPIB hingegen fehlen die umfangreichen grafischen Steuerelemente und Anzeigen, die ProfiLab bietet. Sie ergänzen sich also ideal!

ProfiLab kann unter anderem über eine „import-DLL“ mit Hardware oder anderer Software kommunizieren. Diese import-DLL's werden auf dem Arbeitsbildschirm als Baustein mit einer Reihe von Eingaben und einer Reihe von Ausgaben angezeigt. Ich habe eine solche DLL für ProfiLab geschrieben, die so konfiguriert werden kann, dass sie 1 bis 1000 Eingänge und 1 bis 1000 Ausgänge hat. Sobald die DLL konfiguriert ist, kann sie wie gewohnt mit anderen ProfLab Komponenten verbunden werden. Im ProfiLab Projekt muss nichts mehr beachtet werden. Das Gegenstück dazu in EZGPIB sind die zwei nachstehend genannten Funktionen

```
EZGPIB_ProflabOut(Output:Integer; Value:Double)
```

und

```
EZGPIB_ProflabIn(Input:Integer): Double
```

die mittels Shared Memory direkt mit den ProfiLab Bausteinen kommunizieren.

Das ProfiLab-Verzeichnis enthält die erforderliche DLL, die zusammen mit der INI Datei in das ProfiLab Stammverzeichnis kopiert werden soll. Das Profilab-Verzeichnis enthält auch ein ProfiLab Demoprojekt, das der **ProfiLab.488** Demo von EZGPIB entspricht.

20121204 Einige Benutzer von EZGPIB haben berichtet, dass das Schreiben von Daten in eine Datei bei großen Datendateien sehr zeitaufwändig sein kann, beispielsweise 20 Minuten für eine 23 MB Datei. Wie sich herausstellte, war die Routine **EZGPIB_FileWrite(Filename)** der Engpass, da sie die Datei öffnet, die Daten an das Ende der Datei anfügt und sie anschließend wieder schließt, was gelinde gesagt bei großen Dateien nicht wirtschaftlich ist. Normalerweise sieht ein Codeausschnitt, der Daten in eine Datei schreibt, ungefähr so aus:

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

Ab sofort können Sie einen Aufruf von **EZGPIB_FileAddtoBuffer(#13+#10)** dazu verwenden, um eine Art "neue Zeile" im Puffer zu generieren, in der Sie jetzt den nächsten Satz von Variablen hinzufügen können, ohne jede Zeile einzeln zurück schreiben zu müssen. Stattdessen generieren Sie einen großen bis sehr großen Puffer und schreiben ihn einmal vollständig mit **EZGPIB_FileWrite(Filename)** zurück.

Der Neue Code sieht also so aus:

```
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);
```

und ist *viel schneller*;-))

Beachten Sie, dass **EZGPIB_FileClearBuffer** und **EZGPIB_FileWrite(Filename)** nur einmal vor und nach der Schreibschleife aufgerufen werden.

20121217 Ein kleiner Fehler, der den ersten Aufruf von EZGPIB_FileAddtoBuffer nach dem Aufruf von EZGPIB_FileClearBuffer betraf, wurde behoben.

04/01/2021 EZGPIB.EXE Ressourcen auf eine anfängliche Windows Größe von 1024x768 und die Verwendung größerer Schriftarten gepatcht, um HiRes Displays und meine alten Augen besser zu unterstützen ;-)).

Ausführbare gepatchte Datei auf EZGPIB2.EXE umbenannt.

5 Benutzerhandbuch Änderungsübersicht

04/01/2021 **Deutsches Dokument aus dem Originaldokument vom 12/17/2012 erstellt.**
 Absatzstruktur erstellt und Inhaltsverzeichnis eingefügt.
 Nachruf für OM Uli Bangert DF6JB hinzugefügt.
 Abbildungen im Kapitel [2.4.6](#) **“Verbesserungen im Menü ‘File’“** aktualisiert.
 Beschreibung der Funktionen und Routinen in eine Liste zusammengeführt.
 Kapitel [2.2.2](#) **“Skript Debugging“** hinzugefügt.
 Beschreibung der Routine **EZGPIB_BusAddressDevice** im Kapitel [3.1.2](#)
 korrigiert, Hinweise für Umschaltung in den ‘DEVICE mode’ hinzugefügt.

Version	Datum	Änderungen	von
1.nn	12/17/2012	Letzte Ausgabe, die von OM Ulrich Bangert erstellt wurde	DF6JB
2.00	04/01/2021	Diverse redaktionelle Änderungen und Aktualisierungen	KaKa