IN-4882 GPIB Version 6 Software Manual

1 Manual

1.1 Disclaimer

1.2 Overview of the Manual

This manual covers all topics starting with the installation up to the usage of applications and programming the ines IEEE-488 interface.

Installation

The chapters concerning the installation should be read carefully since they contain useful information about the requirements and the installation preparation.

Using the interface

After the installation has been finished, there are different choices how to continue: One may start the favorite application like HP-VEE to check out the communication. Some useful hints will be found in the application interface section. An introduction into developing applications that control the ines GPIB bus will be found under the Programming Language Interface topic.

Cable length for IEEE-488 bus systems

To fulfill the IEEE-488 standard the maximum length of cable that shall be used to connect together a group of devices within one bus system is (1) 2 m times the number of devices (2) Or 20 m, whichever is less The maximum interconnection distance in a system depends on the transmission speed. If all devices are connected within a 2 m distance and 48 mA line drivers with open collectors are used, the bus works with an maximum transmission speed of 250 kB/s for a distance up to 20 m. Under the same conditions the maximum transmission speed can be increased to 500 kB/s with Tri-State drivers. If all devices are connected within a 1 m distance and 48 mA with Tri-State drivers used, the bus works with an maximum transmission speed of 1 MB/s for a distance up to 10 m.

For two interfaces (one controller and one device) the cable length should not exceed 4m. You can use a cable of greater length and it might work properly but it is not within the IEEE-488 standard and you must check the functionality and the signal edges very carefully.

1.3 Software Overview

The INES GPIB for Windows allows you to connect to the GPIB in many ways. You can choose from a set of programming language interface or use commercial available application development environments. In addition interfaces compatible to other vendors' GPIB interface solutions are available.

1.3.1 New Features of Version 6

While every attempt has been made to ensure compatibility to previous versions of INES GPIB for Windows the following major changes has been made.

Industry Standard Interface
The industry-standard GPIB32.DLL (ibrd, ibwrt,...) style interface is now fully supported on the Microsoft® Windows® 2000 and Windows XP/Vista® operating systems. Existing applications run directly without modifications.

Enhanced Device Operation
The software now supports standard conformant implementations of IEEE-488.2 compatible devices (instruments) via a new C++ interface class library.

Up to eight interfaces supported
The count of concurrently operating interface adaptors per computer has been increased to 8 (eight).

New C++ API
A native and easy to use C++ API has been added to the set of programming languages supported. The native C++ implementation of a GPIB API directly support the object oriented programming paradigm. The API provides classes for GPIB interfaces, devices, transfer parameters etc.

GNU Compiler Collection supported
In addition to the Microsoft Visual C++ compiler, the popular GCC compiler (see www.mingw.org) is supported.

1.3.2 Features continued in Version 6

Programming Language Interfaces
32 bit for C/C++ (Borland C 4.5, Borland C 5.0, Microsoft Visual C++ 2.0, Microsoft Visual C++ 4.0, Microsoft Visual C++ 6.0) Visual Basic (4-6), Visual C++, Delphi(2-5), HT-Basic

Application Development Environments
Agilent VEE, LabView 6+

1.3.3 Features obsoleted by Version 6

Version 5.1 is the last version supporting the 16-bit interface libraries and Windows 3.11, 95, 98, ME, NT4. Also, the WALI interface program is no more supported. The Online Manual media has been changed to HTML and PDF.

1.4 Installation

Installation background
The installation process of INES GPIB for Windows constists of two parts. First, the device drivers for the interface adaptor(s) must be installed. This process is guided by the *Hardware Wizard*. The Hardware Wizard is started whenever a new, unknown devices is mounted, inserted or attached to the computer. After the device driver software has been installed successfully, the generic software components are installed by *setup.exe*.

1.4.1 Installation of Plug&Play adaptors

Minimum requirements
In order to install INES GPIB for Windows successfully you need

- A Microsoft Windows compatible Personal Computer

- one or more ines GPIB interface adaptor(s) of any of the following type(s)
  - GPIB-USB-2
  - GPIB-PCMCIA-XL (32 bit Cardbus)
  - GPIB-PCI-XL
  - GPIB-cPCI-XL
  - GPIB-PMC-XL
  - GPIB-PCI104
  - GPIB-PCI (12.002.00)
  - GPIB-PCI (12.001.00)
  - GPIB-PCMCIA (16 bit)

Removal of previous installations

If a previous version of INES GPIB for Windows is installed:

Windows 2000/XP	Remove all previous installations of INES GPIB for Windows by using <distribution-dir-or-drive>\tools\igclreg.exe prior to installation
Windows Vista	**Never** use igclreg.exe. Instead deinstall the previous driver via the Device Manager. Then use the Uninstall item from the INES IN488.2 menu to remove the generic software components of the previous installation.

Installation procedure

Follow these steps to install a ines Plug&Play adaptor:

1. Always follow the instructions of the computer manufacturer for the installation of add-on components.
2. Insert the card into a free slot (for USB: Plug in the cable into a free USB 2.0 port). Turn on the computer and start Windows. If you have more than one card to install, insert all cards.
3. Windows recognizes the new hardware. Let the Hardware Wizard search for a driver and choose CD-ROM as source.
4. After the Hardware Wizard has finished, execute *setup.exe* from the installation CD. Follow the instructions of the installation program.

1.4.2 Installation of ISA cards

ISA cards are different to Plug&Play cards. Because the presense of hardware cannot be detected automatically, the card(s) must be added explicitely to the system.

Minimum requirements

In order to install INES GPIB for Windows successfully you need

- A Microsoft Windows compatible Personal Computer with ISA or PC/104 expansion bus
- one or more ines GPIB interface card(s) of any of the following type(s)
  - GPIB-PC104
  - GPIB-PC (NEC 7210 C installed)
  - GPIB-PC-HS (iGPIB 7210 1.1 installed)

Removal of previous installations

If a previous version of INES GPIB for Windows is installed:

Windows 2000/XP	Remove all previous installations of INES GPIB for Windows by using <distribution-dir-or-drive>\tools\igclreg.exe prior to installation

Windows Vista **Never** use igclreg.exe. Instead deinstall the previous driver via the Device Manager. Then use the Uninstall item from the INES IN488.2 menu to remove the generic software components of the previous installation.

Installation procedure

Follow these steps to install a ines ISA card:

1. Carefully configure the IRQ and IO settings of your card to use non-conflicting resources. Unresolved resource conflicts could damage your hardware. Disable all DMA channels. NOTE your settings!
2. Install (all) INES GPIB ISA card(s)
3. Power on again your computer.
4. Select ⇒Windows ⇒Settings ⇒Control Panel ⇒Add/Remove Hardware . The Add/Remove Hardware Wizard appears.
5. Select ⇒Add/Troubleshoot a device, ⇒Next . Windows does search for new Plug&Play hardware.
6. Select ⇒Add a new device ⇒Next.
7. Select ⇒No, I want to select hardware from a list ⇒Next .
8. Select ⇒Other devices .
9. Select ⇒Have Disk , direct Windows to your distribution disk or directory.
10. Select the type of your INES GPIB card
11. Enter the resources assigned in step 1. If you cannot assign a resource because Windows reports it as allocated: Quit the *Add/Remove Hardware Wizard* and see *Troubleshooting Conflicting Resources* below.
12. Select ⇒Yes
13. Select ⇒ Start Hardware Installation
14. Repeat starting with step 5 for each card not yet installed. Otherwise allow the *Add/Remove Hardware Wizard* to reboot your computer.
15. Use the Windows *Device Manager* to verify the installation. Each device installed must appear as properly functioning device.
16. After the Hardware Wizard has finished, execute setup.exe from the installation CD-ROM (or the directory containing the distribution content) and follow the instructions of the installation program. This adds the generic software components to your computer.

Troubleshooting Conflicting Resources

If Windows reports resources as being allocated:

- there might be a real resource conflict with existing hardware. Reconfigure the cards in question and start again.
- there might be remains from previously installation attempts, *icglreg.exe* was not able to remove automatically. In this case open the *Add/Remove Hardware Wizard*, Choose ⇒Deinstall Device and deinstall the devices wrongly reported as conflicting.

1.5 Diagnostics and Support

The INES GPIB for Windows provides a facility to log the interface calls and bus communication of your application. In order to create (or re-initialize) a log file, you must run the *iediag.exe* command line application from the \tools subdirectory of the INES GPIB installation directory. The iediag.exe application

creates a log file named *iediag.log* in the root directory of your disk drive *c:* and gathers system information which is written to the first lines of the file.

Whenever a GPIB function library (DLL) is used the first time by your application program it checks for a file named c:\iediag.log to be present. If it is present, interface calls and bus communication are logged by appending lines to that file.

You can use a text editor (wordpad) to view the information in the log file. No further tools are required.

Please note that logging requires computing and IO resources and makes applications using the GPIB interface run slower than they do run with logging disabled. For that reason make sure to remove the log file (or rename it to a name different to iediag.log) when logging is no more required.

When you are contacting technical support, please provide the log file with your request by following the prodedure below.

1. Before starting your application program please run the iediag.exe tool. It is located in the \tools subdirectory of the INES GPIB installation directory.
2. Then, run your application until the problem occurs.
3. After finishing your application program, the file C:\iediag.log should contain logging data. Please attach this file to your request.

1.6 Using Applications

The INES GPIB for Windows allows you to run applications designed for the GPIB-32.DLL interface, and several commercially available application development environments. This sections describes how to use this applications with the INES GPIB for Windows

1.6.1 LabView Version 6

You need to ensure that the system has access to the INES GPIB-32.DLL. Note that this DLL has the same name as other vendor's version. If you have installed such a GPIB library, you must remove or rename the other vendor's GPIB-32.DLL and make sure the INES version of GPIB-32.DLL is in the \Windows\System32 directory. Access to the INES GPIB-32.DLL will allow you to use the low level Labview GPIB I/O instructions.

Enabling the VISA drivers to recognise the INES GPIB hardware in Labview 6 requires the following procedure. Labview 6.0i uses a configuration program called "Measurement and Automation Explorer 2.0" or "MAX 2.0". You must use this program to install a NON Plug'N'Play type GPIB interface card from the list of National Instrument GPIB cards.

Select the "AT-GPIB/TNT" model. When you choose the AT-GPIB/TNT card you will be brought into a configuration procedure for the card setup. Accept the default choices.

When done, you will see a GPIB interface installed in the configuration tree of the MAX utility. If you highlight this item and then click "Scan for Devices" the program should detect the devices connected to your GPIB bus and they can now be addressed using the VISA instrument drivers.

1.6.2 Agilent/HP VEE

The INES GPIB-32.DLL library can be used with Agilent/HP VEE by following these steps:

1. Ensure your system has access to GPIB-32.DLL - either via the path or by copying the DLL to C:\WINDOWS\SYSTEM\GPIB-32.DLL.
2. Activate Agilent-VEE

3. Using the Agilent-VEE instrument manager, set your device addresses in VEE to a number of the form 14xx where xx is the GPIB address. So, for example, if your analyzer is at GPIB address 5 then set the address to 1405.

1.6.3 Agilent IO Libraries

The INES GPIB-32.DLL library can be used with the Agilent IO Libraries (later than M.01.01) by following these steps:

1. Ensure your system has access to GPIB-32.DLL - either via the path or by copying the DLL to the system directory (e.g. C:\windows\system32\GPIB-32.DLL).
2. In the system directory (e.g. c:\windows\system32) the file iegpibn32.dll must be present. In order to use the INES GPIB-32.DLL with Agilent IO Libraries rename iegpibn32.dll to gpibn32.dll by erasing the first two characters.
3. If not already done, install the Agilent IO Libraries.
4. Start Agilent IOCONFIG
5. From the list Available Interface Types select GPIB using NI-488.2. Then click Configure. Accept or modify the settings. Then click OK.
6. The INES GPIB interface(s) can now be operated using the Agilent IO Libraries.

1.6.4 HTBasic

INES GPIB adaptors may be used with HTBasic version 5 and above. Our GPIB-32.DLL is interchangeable with the like element from other vendors - and so, for GPIB support with your INES GPIB interface, you can simply use the NI driver that comes with the HT Basic package. Follow these instructions:

1. Install your GPIB adaptor and the GPIB software.
2. Select the installation option which copies GPIB-32.DLL to your system directory.
3. Activate HT Basic.
4. Load the National Instruments GPIB driver (with which the INES driver is compatible) via the command: LOAD BIN "GPIBNI". Alternatively, add line LOAD BIN "GPIBNI" to your AUTOST file so that the GPIB driver is automatically loaded when you start HTBasic.
5. Load ("GET") and run your program.

1.6.5 Applications using the GPIB-32.DLL

You need to ensure that the system has access to the INES GPIB-32.DLL. Note that this DLL has the same name as other vendor's versions. If you have installed an other vendor's GPIB library, you must remove or rename the other vendors GPIB-32.DLL and make sure the INES version of GPIB-32.DLL is in the \Windows\System32 directory. Access to the INES GPIB-32.DLL will allow you to operate these applications with the INES GPIB interfaces.

1.7 The INES driver library

In order to write programs that control the GPIB bus, there must be an interface to the GPIB hardware. There are drivers for many programming languages so that they can use the ines driver and function library.

Using the ines driver library

The ines IEEE488 interface system supplies the developer with libraries to create windows application which may control the GPIB bus. The interface to applications is designed as dynamic link libraries (DLLs), which is the typical method for Windows. These DLLs contain the GPIB functions like IeEnter(), IeOutput() or IeInpt() as described in the function reference. These functions are independent from the used hardware

interface and programming language. The GPIB functions are calling an underlying interface, the device driver. The driver and its implementation depends on the used interface hardware and the Windows Version.

The following description demonstrates the general usage of the GPIB functions while the IEEE-488 commands covers a detailed description of every function. Since the implementation of the GPIB functions is the usual Windows method by using DLLs, it is possible to control the GPIB bus via **any** application or programming language which allows to call external DLLs. Special care has to be taken for multithreaded application. The GPIB functions don't protect itself from being called concurrently by separate threads in such a environment. The application must ensure that only one GPIB function may be called at a time.

The samples files may be used for a quick introduction into accessing the GPIB bus. A device is programmed to acquire a buffer of sample data and inform the computer via SRQ when the buffer has been filled up. Then the data is read into the computers data buffer and displayed. The sample applications require a Keithley Instrument K195A DMM or compatible at primary GPIB address 9.

1.7.1 Programming in C/C++

The ines IEEE488 interface for C and C++ supplies the developer with import libraries and include files to access the DLL functions. The GPIB functions are simple C functions. For C++, the declaration is nested in an `extern "C"` declaration automatically in the include file. In order to use the correct data types and values, the file IEEE488.H must be included. This file includes other necessary files like types488.h itself and contains the necessary function declarations. It is strongly recommended to use the types shown in the command reference to write portable applications. Before including the ines IEEE488.H header file, the WINDOWS.H include file must be included and WINDOWS has to be defined:

```
#include "windows.h"
#define WINDOWS
#include "ieee488.h"
```

All functions return an error code. This error code should be evaluated for non zero, which indicates the occurrence of an error.

```
WORD devPad;
devPad = 709;
ret = IeInit(736, 0, 5, 1); if(ret) ...
ret = IeClear(devPad);
...
```

On occurrence of an error a messagebox appears with the error message. If the application handles this error by itself, this effect is not desired most times. In this case the function IeMode() may be called to turn off the error messages:

```
E488 retVal;
IeMode(99);
if((retVal = IeInit(736,0,5,1)) != 0)
  { showError(retVal); return; }
...
```

The C/C++ interface is supplied with import library files which must be linked to the application. The library names are IE488F3W.LIB and IE488H3W.LIB for 32-bit applications. The library IE488F3W.LIB is the import library for the GPIB functions and IE488H3W.LIB contain the error messages and some error related function references. For C/C++ compilers that cannot access the library functions via the supplied import libraries this library may be recreated from the DLLs by using a tool (like 'implib') which is supplied with the compiler.

1.7.2 Programming in in Visual Basic

The ines function DLL and help DLL may be accessed from Visual Basic directly. The necessary declarations are found in the IEEE488W.BAS (IEEE488.TXT for Visual Basic 1.0) file. This file has to be included in the Visual Basic project or in the GLOBAL.BAS file. GPIB functions may be called in the usual manner:

```
ierr% = IeInit(736, 0, 5, 1)
ierr% = IeSet(709, 10000, 900, 9010)
ierr% = IeRemote(709)
```

All functions return an error code as an integer value. Any nonzero value means the an error has occurred. On occurrence of an error a messagebox appears with the error message. If the application handles the error itself, this effect is not desired most times. In this case the function IeMode() may be called to turn off the error messages:

```
IeMode(99)
ierr% = IeInit(736,0,5,1)
if (ierr% <> 0) GoTo ErrHdlr
```

1.7.3 Programming in Borland Delphi

The ines function and help DLL may be accessed by Delphi directly. The file IEEE488W.PAS has to be included in the project. This file contains the unit 'ieee488w' where all necessary declarations are found to write applications that are controlling the GPIB bus. Delphi includes the ieee488w unit if the corresponding file 'IEEE488W.PAS' if specified in the project. Common data types are numeric values, which must be declared as Word and pointer to data areas, which have to be declared as array [0..nn] of Byte.

```
uses ..., ieee488w, ...;
var
  pad: Word;
 pad := 709;
  ret := IeInit(736,0,5,1);
  ret := IeSet(pad,10000,900,900);
  ret := IeClear(pad);
```

If Pascal strings are used, the StrPCopy should be used. The Send function which is part of the Delphi sample uses this conversion:

```
(* send a pascal formatted string to the device specified by pad ,sad *)
```

```
procedure Send(pad, sad: Word; s: string);
var
  len, ret : Word;
  buffer   : array [0..80] of Byte;
begin
  len := Length(s);
  StrPCopy(@buffer, s);
  ret :=IeOutput(pad,sad,@buffer,@len);
end;
```

All functions return an error code as an integer value. Any nonzero value means the an error has occurred. On occurrence of an error a messagebox appears with the error message. If the application handles the error itself, the messagebox is not desired most times. In this case the function IeMode() may be called to turn off the error messages:

```
var
  ret: Word;

 ret := IeMode(99); (* turn off error messages *)
 ret := IeInit(736,0,5,1);
  if ret <> 0 then
        ...              (* handle this error *)
```

1.7.4 Programming in HT Basic

HT Basic is considered an GPIB application because it does not use the INES function library. Please see the HT Basic subsection of the Application Interfaces section.

1.7.5 Function reference

1.7.5.1 AbortIo - Ends communication on the bus

Synopsis

- C/C++

```
#include <ieee488.h>
E488 FAR PASCAL IeAbortio (UINT16 seven);
```

- Visual Basic

```
Declare Function IeAbortio Lib [...] (ByVal seven%) As Integer
```

- Delphi

```
function IeAbortio (seven: Word): SmallInt; stdcall;
```

Parameter Description

seven → Selectcode of the interface. Values: only 7 supported yet

return ↑ 0 on success, ines IEEE488 errorcode on failure.

Abortio ends all communication on the bus according to the following priority:

- When the interface is the system controller, it sends the single line message IFC (interface clear) for at least 100 microseconds. After this, it sets the ATN line to 'TRUE', thereby becoming the active controller (CACS).
- The active controller sets the ATN line to 'TRUE' and sends the UNTalk message.
- The talker/listener function ends the asynchronous transfer procedures(DMA).

The use of this command sets all the interface functions to an initialized state. A typical use is at the start of a program or use to recover from an error. The command stops all asynchronous transfers (DMA), that may have been started. Note: This command resets the interface functions of the connected devices, but it does not affect the device function for the measurement range (see Clear).

1.7.5.2 Clear - Sets the specified devices to their initial conditions

Synopsis

- C/C++

```
#include <ieee488.h>
E488 FAR PASCAL IeClear (UINT16 pad);
```

- Visual Basic

```
Declare Function IeClear Lib [...] (ByVal pad%) As Integer
```

- Delphi

```
function IeClear (pad: Word): SmallInt; stdcall;
```

Parameter Description

pad →	Primary bus address of the device. Values: IE_NOADR (7) or 700 + bus address (0-30)
return ↑	0 on success, ines IEEE488 errorcode on failure.

Clear resets the device functions of the addressed devices to their initial conditions.

pad=IE_NOADR	Specifies all connected devices(does not perform addressing or unaddressing). Sends the universal command DCL.
pad= 700 + bus address	Specifies a single device, which is identified by it's primary address (performs appropriate addressing). Sends selective device clear SDC.

1.7.5.3 Disable - Disable asynchronous interrupts

Synopsis

- C/C++

```
#include <ieee488.h>
E488 FAR PASCAL IeDisable (void);
```

- Visual Basic

```
Declare Function IeDisable Lib [...] () As Integer
```

- Delphi

```
function IeDisable (): SmallInt; stdcall;
```

Parameter Description

return ↑ 0 on success, ines IEEE488 errorcode on failure.

Asynchronous interrupts are disabled. Disable should be used only when entire sections in a program should not be interrupted.

1.7.5.4 Enable - Enable asynchronous interrupts

Synopsis

- C/C++

```
#include <ieee488.h>
E488 FAR PASCAL IeEnable (void);
```

- Visual Basic

```
Declare Function IeEnable Lib [...] () As Integer
```

- Delphi

```
function IeEnable (): SmallInt; stdcall;
```

Parameter Description

return ↑ 0 on success, ines IEEE488 errorcode on failure.

Enables asynchronous interrupts (see Disable).

1.7.5.5 EntByte - Read a single byte from the bus into a specified program variable

Synopsis

- C/C++

```
#include <ieee488.h>
E488 FAR PASCAL IeEntByte (UINT16 pad, UNIT16 sad, LPUINT16 bval);
```

- Visual Basic

```
Declare Function IeEntByte Lib [...] (ByVal pad%, ByVal sad%, bval%)
As Integer
```

- Delphi

```
function IeEntByte (pad, sad: Word; bval: PWord): SmallInt; stdcall;
```

Parameter Description

pad →	Primary bus address of the device. Values: IE_NOADR (7) or 700 + bus address (0-30)
sad →	Secondary bus address of the device. Values 0-31 or IE_NOSAD
bval ←	Pointer to a variable which will be updated with the received byte on return.
return ↑	0 on success, ines IEEE488 errorcode on failure.

This function reads a single byte from the bus. EntByte addresses the device specified by pad and sad as talker if pad is unequal to IE_NOADR. If pad is set to IE_NOADR, no addressing will be done. Using the functions, the secondary address has to be specified. If no secondary address is needed, this parameter has to be set to IE_NOSAD. If at the same time as the data byte is received, the valid termination sequence is recognized (EOI active or EOS byte), then the highest bit of bval is set to '1'. To read more than one byte from the bus, use the Enter function.

1.7.5.6 Enter - Transfers data from the bus into a user specified buffer

Synopsis

- C/C++

```
#include <ieee488.h>
E488 FAR PASCAL IeEnter (UINT16 pad, UNIT16 sad, LPBYTE buffer, LPUINT16
count);
```

- Visual Basic

```
Declare Function IeEnter Lib [...] (ByVal pad%, ByVal sad%, ByVal
buffer$, count%) As Integer
```

- Delphi

```
function IeEnter (pad, sad: Word; buffer: Pbyte; count: PWord): SmallInt;
stdcall;
```

Parameter Description

pad →	Primary bus address of the device. Values: IE_NOADR (7) or 700 + bus address (0-30)
sad →	Secondary bus address of the device. Values 0-31 or IE_NOSAD
buffer ↔	Pointer to a user defined buffer (or 'C'-string) to place the received data in.
count ↔	Pointer to a variable which contains the maximal count of bytes to read. The variable contains the count of actual read bytes on return.
return ↑	0 on success, ines IEEE488 errorcode on failure.

This function reads data from the bus until the specified count is reached or the current termination is recognized. Enter addresses the device specified by pad and sad as talker if pad is unequal to IE_NOADR. If pad is set to IE_NOADR, no addressing will be done. The secondary address has to be specified. If no secondary address is required, this parameter has to be set to IE_NOSAD.

A termination sequence can be chosen with the Set function. If the Set function is not properly assigned, the interface cannot identify a termination sequence when the end of transmission is reached. If the error message 13 end of transfer buffer is returned, check the following:

- Is the count specified large enough?
- Are appropriate values used in the IeSet function ?

Note: Enter returns "end of transfer buffer" if the specified count has been reached and no termination is recognized. If you don't want to use any termination (i.e. want termination by specified length), this return value has to be interpreted as completion indicator and not as an error.

1.7.5.7 EntFile - Transfers data from the bus into a file

Synopsis

- C/C++

```
#include <ieee488.h>
E488 FAR PASCAL IeEntFile (UINT16 pad, UNIT16 sad, LPBYTE path);
```

- Visual Basic

```
Declare function IeEntFile Lib [...] (ByVal pad%, ByVal sad%, ByVal
path$) As Integer
```

- Delphi

```
function IeEntFile (pad, sad: Word; path: PChar): SmallInt; stdcall;
```

Parameter Description

pad →	Primary bus address of the device. Values: IE_NOADR (7) or 700 + bus address (0-30)
sad →	Secondary bus address of the device. Values 0-31 or IE_NOSAD
path →	is a valid file name
return ↑	0 on success, ines IEEE488 errorcode on failure.

This function reads data from the bus and writes it into a file until the current termination is recognized. The file will be created if it does not exists. If it exists, then the data will be appended to it. To replace an existing file, erase it first before calling this function. EntFile addresses the device specified by pad and sad as talker if pad is unequal to IE_NOADR. If pad is set to IE_NOADR, no addressing will be done. If no secondary address is needed, sad has to be set to IE_NOSAD. A termination sequence can be chosen with the Set function.

1.7.5.8 Help - Provides information about errors

Synopsis

- C/C++

```
#include <ieee488.h>
E488 FAR PASCAL IeHelp (UINT16 errno);
```

- Visual Basic

```
Declare Function IeHelp Lib [...] (ByVal errno%) As Integer
```

- Delphi

```
function IeHelp (errno: Word): SmallInt; stdcall;
```

Parameter Description

errno → ines IEEE488 errorcode

return ↑ always 0

The help function appears on the screen when an error occurs. The help message can be suppressed by using the Mode function. It can also be called by a program. This can be useful when an application wants to suppress the help messages for certain errors. Help displays the ines IEEE488 errorcode and a human readable error description. In addition the current locator value is displayed. The locator value will be set using the Loc function and can be used to find errors in complex programs.

1.7.5.9 Init - Initializes the interface

Synopsis

- C/C++

```
#include <ieee488.h>
E488 FAR PASCAL IeInit (UINT16 ioadrs, UINT16 pad, UINT16 intr, UINT16
dma);
```

- Visual Basic

```
Declare Function IeInit Lib [...] (ByVal ioadrs%, ByVal pad%, ByVal
intr%,ByVal dma%) As Integer
```

- Delphi

```
function IeInit (ioadrs, pad, intr, dma: Word): SmallInt; stdcall;
```

Parameter Description

ioadrs →	For compatiblity only: set to 0
pad →	Primary bus address of the interface. Values: 0-30
intr →	For compatiblity only: set to 0
dma →	For compatiblity only: set to 0
return ↑	0 on success, ines IEEE488 errorcode on failure.

The ines board and the software drivers will be initialized with the specified values. This has to be the first function to be called when using the ines IEEE488 interface. The current interface board becomes the system controller(SYSCTL), IFC is sent and REMOTE ENABLE is set to 'true'. After this, the interface is the active controller. With interface errors which cannot be removed by Abortio, it may be necessary to reset the entire interface system using Init. Following to an Init call the values of previous calls to Set are undefined and should be reinitialized.

1.7.5.10 LAG - Sends primary and secondary addresses of a listener

Synopsis

- C/C++

```
#include <ieee488.h>
E488 FAR PASCAL IeLAG (UINT16 pad, UINT16 sad);
```

- Visual Basic

```
Declare Function IeLAG Lib [...] (ByVal pad%,ByVal sad%) As Integer
```

- Delphi

```
function IeLAG (pad, sad: Word): SmallInt; stdcall;
```

Parameter Description

pad →	Primary bus address of the device. Values: 700 + bus address (0-30)
sad →	Secondary bus address of the device. Values 0-31 or IE_NOSAD
return ↑	0 on success, ines IEEE488 errorcode on failure.

LAG addresses the device specified by pad and sad as listener. The specified device is then able to receive data from the bus. Using the functions, the secondary address has to be specified. If no secondary address is needed, this parameter has to be set to IE_NOSAD.

1.7.5.11 LLO - Sends local lockout

Synopsis

- C/C++

```
#include <ieee488.h>
E488 FAR PASCAL IeLLO (void);
```

- Visual Basic

```
Declare Function IeLLO Lib [...] () As Integer
```

- Delphi

```
function IeLLO (): SmallInt; stdcall;
```

Parameter Description

return ↑ 0 on success, ines IEEE488 errorcode on failure.

Local lockout is a universal command that disables the operation panels of the connected devices on the IEEE488 bus.

1.7.5.12 Loc - Sets the locator value

Synopsis

- C/C++

```
#include <ieee488.h>
E488 FAR PASCAL IeLoc (UINT16 locator);
```

- Visual Basic

```
Declare Function IeLoc Lib [...] (ByVal locator%) As Integer
```

- Delphi

```
function IeLoc (locator: Word): SmallInt; stdcall;
```

Parameter Description

locator → value to set

return ↑ 0 on success, ines IEEE488 errorcode on failure.

locator is an integer value specified by the user. This value can be referenced to locate an error when it occurs and shown through the Help function. The locator value is displayed within the help function. Using this method, an error can be located in the program. Each call of Loc will update the locator value. This method is recommended when searching for an error in large programs.

1.7.5.13 Local - Disables remote control of devices

Synopsis

- C/C++

```
#include <ieee488.h>
E488 FAR PASCAL IeLocal (UINT16 pad);
```

- Visual Basic

```
Declare Function IeLocal Lib [...] (ByVal pad%) As Integer
```

- Delphi

```
function IeLocal (pad: Word): SmallInt; stdcall;
```

Parameter Description

pad →	Primary bus address of the device. Values: IE_NOADR (7) or 700 + bus address (0-30)
return ↑	0 on success, ines IEEE488 errorcode on failure.

Local disables remote control of devices connected to the bus.

pad=IE_NOADR	Affects all connected devices (i.e. does not perform addressing or unaddressing). Sets the REMOTE ENABLE line to 'FALSE' and thereby disables remote control.
pad = 700 + bus address	Affects a single device, which is identified by it's primary address (i. e. performs appropriate addressing). Returns control to the device through the GTL (GoTo Local) command.

1.7.5.14 Mode - Controls the output of the help function

Synopsis

- C/C++

```
#include <ieee488.h>
E488 FAR PASCAL IeMode (UINT16 mode);
```

- Visual Basic

```
Declare Function IeMode Lib [...] (ByVal mode%) As Integer
```

- Delphi

```
function IeMode (mode: Word): SmallInt; stdcall;
```

Parameter Description

mode →	Display mode for help function. Values: 0, 9, 90, 99
return ↑	0 on success, ines IEEE488 errorcode on failure.

The Mode function enables/disables the help message appearing when an error occurs. This is useful when an error shall be handled only by the program itself. A value of 0 or 90 enables the help message, a value of 9 or 99 disables it.

1.7.5.15 MyStat - Sets the interfaces STATUS BYTE (stb) and REQUEST SERVICE (rqs) messages

Synopsis

- C/C++

```
#include <ieee488.h>
E488 FAR PASCAL IeMyStat (LPUINT16 stb);
```

- Visual Basic

```
Declare Function IeMyStat Lib [...] (stb%) As Integer
```

- Delphi

```
function IeMyStat (stb: PWord): SmallInt; stdcall;
```

Parameter Description

stb →	Pointer to a variable containing the value of the status byte to set
return ↑	0 on success, ines IEEE488 errorcode on failure.

The MyStat function is used to set the status byte. The bits 0-5 of stb contain information specific to the device, bit 6 is the 'rqs' message. Setting bit 6 causes the interface to activate the SRQ line. The GPIB CIC (controller-in-charge) can use a SERIAL POLL to poll the device. During the poll, the interface sets the SRQ line to 'FALSE'. When MyStat is used with stb=0, it is possible to determine if a SERVICE REQUEST needs to be performed. When the SRQ does not require service, bit 6 in stb is set to 1.

Typically, MyStat is used only if the interface is not the GPIB CIC.

1.7.5.16 OnSrq - SRQ interrupt handling

Synopsis

- C/C++

```
#include <ieee488.h>
E488 FAR PASCAL IeOnSrq (UINT16 mode, LPUINT16 target1, LPUINT16
target2);
```

- Visual Basic

```
not available>
```

- Delphi

```
function IeOnSrq (mode: Word; target1, target2: PWord): SmallInt;
stdcall;
```

Parameter Description

mode → determines the action to be taken when an SRQ interrupt occurs.

target1 → mode specific information

target2 → mode specific information

return ↑ 0 on success, ines IEEE488 errorcode on failure.

OnSrq controls the handling of the asynchronous interrupt that occurs when a SERVICE REQUEST message is sent by a device. Depending on the value of mode the following handlings of an occuring SRQ interrupt are available:

mode=0	The OnSrq function is deactivated. The other parameters are ignored. This function must be called before exiting a program.
mode=1	When an SRQ interrupt occurs, the variable specified by target1 and target2 will be set to 1. The parameter target1 must be a pointer to the segment and target2 a pointer to the offset of a user defined variable. Note that the value of the variable specified by target1 and target2 will be changed asynchronously when an interrupt occurs.
mode=2	When an SRQ interrupt occurs, the user defined interrupt function specified by target1 and target2 will be executed. The argument target1 must be a pointer to the segment and target2 a pointer to the offset of the user implemented interrupt function.

1.7.5.17 OutByte - Send a data byte

Synopsis

- C/C++

```
#include <ieee488.h>
E488 FAR PASCAL IeOutByte (UINT16 pad, UNIT16 sad, LPUINT16 bval);
```

- Visual Basic

```
Declare Function IeOutByte Lib [...] (ByVal pad%, ByVal sad%, bval%)
As Integer
```

- Delphi

```
function IeOutByte (pad, sad: Word; bval: PWord): SmallInt; stdcall;
```

Parameter Description

pad →	Primary bus address of the device. Values: IE_NOADR (7) or 700 + bus address (0-30)
sad →	Secondary bus address of the device. Values 0-31 or IE_NOSAD
bval →	Pointer to a variable which contains the value to write
return ↑	0 on success, ines IEEE488 errorcode on failure.

The OutByte function writes a single data byte to the bus. OutByte addresses the device specified by pad and sad as listener if pad is unequal to IE_NOADR. If pad is set to IE_NOADR, no addressing will be done. If no secondary address is required, this parameter has to be set to IE_NOSAD. The lower byte of bval will be send as data byte over the bus. If the highest bit in the higher byte is set (by adding of 32768 to the value to be send), the termination sequence (EOI and/or EOS byte), that was selected with Set, will be sent (EOI) or appended (EOS byte) at the same time. This signals the end of a message to the addressed listener.

1.7.5.18 OutCmd - Send commands (data bytes with ATN asserted)

Synopsis

- C/C++

```
#include <ieee488.h>
E488 FAR PASCAL IeOutCmd (LPBYTE cmd);
```

- Visual Basic

```
Declare Function IeOutCmd Lib [...] (ByVal cmd$) As Integer
```

- Delphi

```
function IeOutCmd (cmd: PByte): SmallInt; stdcall;
```

Parameter Description

cmd →	Pointer to a buffer which contains a sequence of command bytes to be sent. The command sequence must be null-terminated ('C'-string).
return ↑	0 on success, ines IEEE488 errorcode on failure.

The function IeOutCmd sends a null-terminated sequence of commands to the bus. This function can generate command sequences which are not defined the standard IEEE488. Note that the interface must be the active controller.

1.7.5.19 OutCmdByte - Send a command (data byte with ATN asserted)

Synopsis

- C/C++

```
#include <ieee488.h>
E488 FAR PASCAL IeOutCmdByte (UINT16 cmdbyte);
```

- Visual Basic

```
Declare Function IeOutCmdByte Lib [...] (ByVal cmdbyte%) As Integer
```

- Delphi

```
function IeOutCmdByte (cmdbyte: Word): SmallInt; stdcall;
```

Parameter Description

cmdbyte → Data byte to be sent as bus command

return ↑ 0 on success, ines IEEE488 errorcode on failure.

The function IeOutCmdByte sends a command over the bus. This function can generate a command which is not defined by the standard IEEE488. Note that the interface must be the active controller.

1.7.5.20 OutFile - Sends a MS-DOS file to the bus

Synopsis

- C/C++

```
#include <ieee488.h>
E488 FAR PASCAL IeOutFile (UINT16 pad, UNIT16 sad, LPBYTE path);
```

- Visual Basic

```
Declare function IeOutFile Lib [...] (ByVal pad%, ByVal sad%, ByVal
path$) As Integer
```

- Delphi

```
function IeOutFile (pad, sad: Word; path: PChar): SmallInt; stdcall;
```

Parameter Description

pad →	Primary bus address of the device. Values: IE_NOADR (7) or 700 + bus address (0-30)
sad →	Secondary bus address of the device. Values 0-31 or IE_NOSAD
path →	a well-formed filename
return ↑	0 on success, ines IEEE488 errorcode on failure

This function reads data from a file and sends it over the bus. OutFile addresses the device specified by pad and sad as listener if pad is unequal to IE_NOADR. If pad is equal to IE_NOADR, no addressing will be done. If no secondary address is required, sad must be set to IE_NOSAD.

Be aware that OutFile sends the entire file, irrespective of any termination sequences which may be included in the file itself. For this reason, only EOI should be used as termination (see Set) for file transfer.

1.7.5.21 Output - Send data over the bus

Synopsis

- C/C++

```
#include <ieee488.h>
E488 FAR PASCAL IeOutput (UINT16 pad, UNIT16 sad, LPBYTE buffer, LPUINT16
count);
```

- Visual Basic

```
Declare Function IeOutput Lib [...] (ByVal pad%, ByVal sad%, ByVal
buffer$, count%) As Integer
```

- Delphi

```
function IeOutput (pad, sad: Word; buffer: Pbyte; count: PWord):
SmallInt; stdcall;
```

Parameter Description

pad →	Primary bus address of the device. Values: IE_NOADR (7) or 700 + bus address (0-30)
sad →	Secondary bus address of the device. Values 0-31 or IE_NOSAD
buffer →	Pointer to a user defined buffer containing the data to be sent
count ↔	Pointer to a variable which contains the count of bytes to write. The variable contains the count of actual written bytes on return
return ↑	0 on success, ines IEEE488 errorcode on failure

The Output function writes data to the bus until the specified count is reached. The data given in buffer will be send unmodified and unformatted to the device. Output addresses the device specified by pad and sad as listener if pad is unequal to IE_NOADR. If pad is equal to IE_NOADR, no addressing will be done. If no secondary address is required, the sad parameter has to be set to IE_NOSAD. The termination sequence for the transfer can be specified with the Set function.

1.7.5.22 PasCnt - Pass control of the system to another device

Synopsis

- C/C++

```
#include <ieee488.h>
E488 FAR PASCAL IePasCnt (UINT16 pad);
```

- Visual Basic

```
Declare Function IePasCnt Lib [..] (ByVal pad%) As Integer
```

- Delphi

```
function IePasCnt (pad: Word): SmallInt; stdcall;
```

Parameter Description

pad →	Primary bus address of the device. Values: IE_NOADR (7) or 700 + bus address (0-30)
return ↑	0 on success, ines IEEE488 errorcode on failure

Pass control of the system to another device. PasCnt addresses the device specified by pad as talker if pad is not IE_NOADR. If pad is equal to IE_NOADR, no addressing will be done (the current active talker will receive system control). The complementary function is RecCnt. If the control needs to be transferred between two ines-ieee488 systems, then RecCnt must first be activated and then the active controller can call the PasCnt function.

1.7.5.23 PPD - Deactivate parallel-poll response of one or more devices

Synopsis

- C/C++

```
#include <ieee488.h>
E488 FAR PASCAL IePPD (UINT16 pad);
```

- Visual Basic

```
Declare Function IePPD Lib [..] (ByVal pad%) As Integer
```

- Delphi

```
function IePPD (pad: Word): SmallInt; stdcall;
```

Parameter Description

pad →	Primary bus address of the device. Values: IE_NOADR (7) or 700 + bus address (0-30)
return ↑	0 on success, ines IEEE488 errorcode on failure

Deactivates the parallel-poll response of one or more devices. If pad is unequal to IE_NOADR, the specified device will be addessed as listener, otherwise all active listeners will receive this command.

1.7.5.24 PPE - Configure one or more devices for parallel polling

Synopsis

- C/C++

```
#include <ieee488.h>
E488 FAR PASCAL IePPE (UINT16 pad,UINT16 ppc);
```

- Visual Basic

```
Declare Function IePPE Lib [..] (ByVal pad%, ByVal ppc%) As Integer
```

- Delphi

```
function IePPE (pad, ppc: Word): SmallInt; stdcall;
```

Parameter Description

pad →	Primary bus address of the device. Values: IE_NOADR (7) or 700 + bus address (0-30)
ppc →	Response from a questioned device at the single bit level (parallel poll response)
return ↑	0 on success, ines IEEE488 errorcode on failure

Configures one or more devices for a parallel-poll response. If pad is unequal to IE_NOADR, the specified device will be addressed as listener, otherwise all active listeners will recognize this command. The ppc parameter is specified as follows:

Bits 0-2	specify the line on which the device shall repond to a parallel poll. Any line from 0 to 7 may be specified.
Bit 3	specifies whether there should be a response on a parallel poll for a true condition of the device or for a false condition. Using this feature a number of devices may be configured to use a single line wired-OR or wired AND, respectively

1.7.5.25 PPoll - Perform a parallel poll

Synopsis

- C/C++

```
#include <ieee488.h>
E488 FAR PASCAL IePPoll (LPUINT16 resp);
```

- Visual Basic

```
Declare Function IePPoll Lib [..] (resp%) As Integer
```

- Delphi

```
function IePPoll (resp: PWord): SmallInt; stdcall;
```

Parameter Description

resp ← Pointer to a variable which is updated with the answer to the parallel poll on return.

return ↑ 0 on success, ines IEEE488 errorcode on failure

Performs a parallel poll and returns the result in resp.

1.7.5.26 PPU - Deactivate the parallel poll function

Synopsis

- C/C++

```
#include <ieee488.h>
E488 FAR PASCAL IePPU (void);
```

- Visual Basic

```
Declare Function IePPU Lib [..] () As Integer
```

- Delphi

```
function IePPU: SmallInt; stdcall;
```

Disables all devices from responses to parallel poll.

1.7.5.27 RecCnt - Receive Control

Synopsis

- C/C++

```
#include <ieee488.h>
E488 FAR PASCAL IeRecCnt (void);
```

- Visual Basic

```
Declare Function IeRecCnt Lib [..] () As Integer
```

- Delphi

```
function IeRecCnt: SmallInt; stdcall;
```

Parameter Description

return ↑ 0 on success, ines IEEE488 errorcode on failure

Enables the ines-ieee488 to become the active controller when operating as a device (CIDS). The complementary function is PasCnt. If the control needs to be transferred from one ieee488 system to another, then RecCnt must first be activated and then the active controller can issue the PasCnt function.

1.7.5.28 Remote - Set devices in the remote control condition

Synopsis

- C/C++

```
#include <ieee488.h>
E488 FAR PASCAL IeRemote (UINT16 pad);
```

- Visual Basic

```
Declare Function IeRemote Lib [...] (ByVal pad%) As Integer
```

- Delphi

```
function IeRemote (pad: Word): SmallInt; stdcall;
```

Parameter Description

pad →	Primary bus address of the device. Values: IE_NOADR (7) or 700 + bus address (0-30)
return ↑	0 on success, ines IEEE488 errorcode on failure

Remote sets the REMOTE ENABLE line true and thereby enables remote control of the attached devices. In addition, if pad is unequal to IE_NOADR, the specified device will be addressed as listener. This command requires the interface to be the system controller.

1.7.5.29 Resume - Place an active controller in the standby status

Synopsis

- C/C++

```
#include <ieee488.h>
E488 FAR PASCAL IeResume (void);
```

- Visual Basic

```
Declare Function IeResume Lib [..] () As Integer
```

- Delphi

```
function IeResume: SmallInt; stdcall;
```

Parameter Description

return ↑ 0 on success, ines IEEE488 errorcode on failure

Place an active controller in the standby status. 'ATN' is set to FALSE.

1.7.5.30 Set - Set the interface parameters for devices on the bus.

Synopsis

- C/C++

```
#include <ieee488.h>
E488 FAR PASCAL IeSet (UINT16 pad, UINT16 tmo, UINT16 termin, UINT16
termout);
```

- Visual Basic

```
Declare Function IeSet Lib [...] (ByVal pad%, ByVal tmo%, ByVal
termin%,ByVal termout%) As Integer
```

- Delphi

```
function IeSet (pad, tmo, termin, termout: Word): SmallInt; stdcall;
```

Parameter Description

pad → Primary bus address of the device. Values: IE_NOADR (7) or 700 + bus address (0-30)

tmo → Timeout in milliseconds if acting as controller, otherwise timeout in units of 10 milliseconds. If tmo is specified as 0 no time limit will be enabled.

termin → Termination to use on input.

termout → Termination to use on output.

return ↑ 0 on success, ines IEEE488 errorcode on failure

Sets the interface parameters for devices on the bus. If pad is not IE_NOADR the parameters will be set only for the specified device. Otherwise the parameters will be set for all devices that has not been already configured by a call of Set with pad unequal to IE_NOADR.

The setting of a device will be activated if its address is sent. Only the primary address is used for device settings. An activate setting remains active until another device is addressed.

The settings available for termination values (termin/termout) are:

0	The data blocks will not be terminated. During a listener transfer (see Enter), this may result in the error message "unexpected end of transfer buffer". If using a board with FIFO buffer, the FIFO can be activated by adding a 1 to this value.
900	Data blocks will be terminated with EOI. During a talker transfer, the EOI line will be set to 'true' with the last byte of data, and signals the end of the block. During a listener transfer, the input of data is stopped at the first byte with EOI set to 'TRUE'. If using a board with FIFO buffer, the FIFO can be activated by adding a 1 to this value.
1XXX	Data blocks will be delimited by the byte specified by decimal 'XXX'. The listener transfer ends with the reception of this byte. During a talker transfer, the byte specified by 'XXX' is appended to the data. The values allowed are between 1000 (EOS byte 00) and 1255 (EOS byte 0FFH). The EOI line is ignored. If using a board with FIFO buffer, the FIFO can be activated by adding 256 (100H) to this value.
9XXX	Data blocks will be delimited by the byte specified by decimal 'XXX' and/or EOI 'TRUE'. The listener transfer ends with the reception of the EOS byte or recognition of EOI 'TRUE'. During a talker transfer, the byte specified by 'XXX' will be appended to the data and the EOI line will be set to 'TRUE' at the same time. The values allowed are between 9000 (EOS byte 00) and 9255 (EOS byte 0FFH). If using a board with FIFO buffer, the FIFO can be activated by adding 256 (100H) to this value.

1XXXX ATN mode The interface system is allowed to assert the ATN line immediately after a data transfer. This can be disabled, however, since setting ATN TRUE immediately after a data transfer can sometimes prompt a malfunction of certain devices. By adding 10.000 (decimal) to the termin/termout value you make sure the ATN line remains in the unasserted state after a data transfer. Also, ATN is asserted not earlier than the next ieee488 function call.

Initial Settings (Default Setting) After the initialization of the ines-ieee488, the following values are used as default settings: timeout = 1000 ms, termin = 900 EOI, termout = 9010 EOI/LF. Note: The assignment can be undone (set to the default) with the UnSet function.

1.7.5.31 Shadow - Performs listener handshaking without reading in data to check for valid termination

Synopsis

- C/C++

```
#include <ieee488.h>
E488 FAR PASCAL IeShadow (void);
```

- Visual Basic

```
Declare Function IeShadow Lib [..] () As Integer
```

- Delphi

```
function IeShadow: SmallInt; stdcall;
```

Parameter Description

return ↑ 0 on success, ines IEEE488 errorcode on failure

Enables the active controller to passively 'listen in' on the bus, so that it can take over control upon recognition of a termination sequence (see Set). Corresponds to the 'Enter 7' function with the exception that data is not transfered to a variable.

1.7.5.32 SPoll - Perform a Serial Poll

Synopsis

- C/C++

```
#include <ieee488.h>
E488 FAR PASCAL IeSPoll (UINT16 pad, LPUINT16 resp);
```

- Visual Basic

```
Declare Function IeSPoll Lib [..] (ByVal pad%, resp%) As Integer
```

- Delphi

```
function IeSPoll (pad:Word; resp: PWord): SmallInt; stdcall;
```

Parameter Description

pad →	Primary bus address of the device. Values: IE_NOADR (7) or 700 + bus address (0-30)
resp ←	Pointer to a variable which is updated with the answer to the serial poll on return.
return ↑	0 on success, ines IEEE488 errorcode on failure

The serial poll function allows the questioning of devices as to their momentary status. SPoll addresses the device specified by pad as talker if pad is not IE_NOADR. If pad is equal to IE_NOADR, no addressing will be done.

All devices can be set to a condition that instructs them to send an individual status byte on request. This is made possible with the universal command SPE (Serial Poll Enable). After this, the first polled device is activated by sending its talker address. The device then sends its status byte in return. After receiving this message, it is possible to poll other devices. At the end of such a sequence, the command SPD (Serial Poll Disable) should be sent. The function IeSPoll handles this entire procedure in one call. The serial poll response resp has bit 6 set if the device queried requested service (SRQ).

1.7.5.33 Status - Allows the polling of the interface status

Synopsis

- C/C++

```
#include <ieee488.h>
E488 FAR PASCAL IeStatus (LPUINT16 status);
```

- Visual Basic

```
Declare Function IeStatus Lib [..] (status%) As Integer
```

- Delphi

```
function IeStatus (status: PWord): SmallInt; stdcall;
```

Parameter Description

status ↔ Pointer to a variable which contains the selector for the status call. On return it is updated with the current interface status. This value is interpreted at bit level.

return ↑ 0 on success, ines IEEE488 errorcode on failure.

Status returns the current interface status. The selection of which status will be returned is specified by initializing status to one of the following values before calling the function:

status=1 returns the interface condition and the interface bus address. The information in the lower byte (bits 0-7) at bit level is:

Bit	Meaning
REM	interface is in a REMOTE condition
LLO	LOCAL LOCKOUT is active
ATN	the ATTENTION line is asserted
SRQ	SERVICE REQUEST line is active
LA	interface is addressed as a Listener
TA	interface is addressed as a Talker
EOI	EOI line is active

The high byte (bits 8 - 15) contains the GPIB bus address of the interface, as has been set by Init

status=2 returns the ines-ieee488 internal condition flags as follows:

LowByte

Bit	Meaning
INIT	System was initialized without any errors
CACS	Controller function is active
CSBS	Controller function is in standby
CIDS	Controller function is idle
SYSC	The interface is the system controller

HighByte

Bit	Meaning
DONE	Aynchronous data transfer has been completed (This bit is 0 after activating an asynchonous transfer)
DMA	Asynchronous data transfer was started and is still active

status=3 returns the current condition of a DMA transfer

status=4 number of bytes that are still left to transfer (0=operation is finished)

status=5 returns the event status of the interface. After reading this status value, the bits are reset to '0'.

LowByte

Bit	Meaning
DTAS	Set if the ieee488 is operating as a device and the GET(Group Execute Trigger) command has been received. See also the Trigger function.
UNC	Command that cannot be decoded was received
DCAS	DEVICE CLEAR or SELECTED DEVICE CLEAR command has been received.
ADSC	Address status changed
SRQ	SRQ has been set to 'true'

The high byte (bit 15-8) is undefined.

1.7.5.34 TAG - Send primary and secondary addresses of a talker

Synopsis

- C/C++

```
#include <ieee488.h>
E488 FAR PASCAL IeTAG (UINT16 pad, UINT16 sad);
```

- Visual Basic

```
Declare Function IeTAG Lib [...] (ByVal pad%,ByVal sad%) As Integer
```

- Delphi

```
function IeTAG (pad, sad: Word): SmallInt; stdcall;
```

Parameter Description

pad →	Primary bus address of the device. Values: IE_NOADR (7) or 700 + bus address (0-30)
sad →	Secondary bus address of the device. Values 0-31 or IE_NOSAD
return ↑	0 on success, ines IEEE488 errorcode on failure

TAG addresses the device specified by pad and sad as talker. The specified device is then able to send data on the bus. If no secondary address is needed, sad parameter must be set to IE_NOSAD. The transfer parameters are activated as specified by the Set command for that device.

1.7.5.35 Trigger - Send a GROUP EXECUTE TRIGGER

Synopsis

- C/C++

```
#include <ieee488.h>
E488 FAR PASCAL IeTrigger (UINT16 pad);
```

- Visual Basic

```
Declare Function IeTrigger Lib [...] (ByVal pad%) As Integer
```

- Delphi

```
function IeTrigger (pad: Word): SmallInt; stdcall;
```

Parameter Description

pad →	Primary bus address of the device. Values: IE_NOADR (7) or 700 + bus address (0-30)
return ↑	0 on success, ines IEEE488 errorcode on failure

Trigger addresses the device specified by pad as listener if pad is not IE_NOADR. If pad is equal to IE_NOADR, no addressing will be done. This command sends a GROUP EXECUTE TRIGGER command to all addressed devices. With devices that have a DEVICE TRIGGER function, the reception of this command causes the start of measurement sequences, etc.

1.7.5.36 UNL - Unaddress all listeners

Synopsis

- C/C++

```
#include <ieee488.h>
E488 FAR PASCAL IeUNL (void);
```

- Visual Basic

```
Declare Function IeUNL Lib [...] () As Integer
```

- Delphi

```
function IeUNL (): SmallInt; stdcall;
```

Parameter Description

return ↑ 0 on success, ines IEEE488 errorcode on failure

This command unaddresses all listener devices by sending the universal command UNL to the bus.

1.7.5.37 UnSet - Deactivate device settings.

Synopsis

- C/C++

```
#include <ieee488.h>
E488 FAR PASCAL IeUnSet (UINT16 pad);
```

- Visual Basic

```
Declare Function IeUnSet Lib [...] (ByVal pad%) As Integer
```

- Delphi

```
function IeUnSet (pad: Word): SmallInt; stdcall;
```

Parameter Description

pad → Primary bus address of the device. Values: IE_NOADR (7) or 700 + bus address (0-30)

return ↑ 0 on success, ines IEEE488 errorcode on failure

The device settings for the specified device are deactivated. This means, the standard values will be used for this device. The standard values are specified by a former call of Set with pad given as IE_NOADR. If no former call of Set with pad=IE_NOADR was made, the default parameters (see Set) are used as standard parameters.

1.7.5.38 UNT - Unaddress the active talker

Synopsis

- C/C++

```
#include <ieee488.h>
E488 FAR PASCAL IeUNT (void);
```

- Visual Basic

```
Declare Function IeUNT Lib [...] () As Integer
```

- Delphi

```
function IeUNT (): SmallInt; stdcall;
```

Parameter Description

return ↑ 0 on success, ines IEEE488 errorcode on failure

This command unaddresses the current talker device by sending the universal command UNT to the bus.

2 Appendix

2.1 Overview of the IEEE-488 GPIB

Introduction

The IEEE-488 or IEC 625 interface (GPIB) is a powerful tool for communication and control of devices used in the fields of measurement, automation, and data analysis. The acceptance of the interface as a worldwide standard had its start when Hewlett Packard created the HP-IB (Hewlett Packard Interface Bus). It allowed the company to equip completely different instruments with one standard interface.

In the following years, the HP-IB has become accepted worldwide without major changes in the form of the standards IEEE-488 and IEC 625. The only difference between IEEE-488 and IEC 625 is in the type of connector used.

The IEEE-488 uses a 24 pin amphenol connector, while the IEC 625 uses a 25 pin connector (called the MIN-D-SUB). The connection to different types can be made easily with the use of an adapter cable.

Fifteen devices can be operated on the IEEE-488 bus in a parallel manner. The management of the bus is handled by the Controller. By using a three line handshake protocol, it is assured that the slowest connected and addressed device determines the data transfer rate of the overall system. The benefit to this three line handshake is extremely reliable data transfer without the use of a separate timing parameter or clock.

The IEEE-488 Bus

The IEEE-488 bus is a communication system of up to 15 devices with a single system controller (usually a PC or similar computer). The interfaces are connected with a 24 pin cable to one another.

Eight of the 24 lines provide byte oriented, paralleled data transfer. The other sixteen lines provide signals for communication control as well as the necessary grounding (GND). Each device that is connected to the bus, as well as the system controller, is identified by a unique address. An address is a primary address only, or a primary address together with a secondary address. Secondary addressing is commonly used to address subdevices of a device. A primary address ranges from 0 to 30 and a secondary address ranges from 0 to 31.

The controller

A device that is capable of sending interface messages is called a controller. There may be only one active controller in a system. The controller which functions as the active controller once the system has been started is called the system controller. Interface messages are defined by the IEEE-488 standard and they are common to every device conforming to IEEE-488.1 or IEEE-488.2. Interface messages are used to setup the GPIB data transfer direction by specifiing a talker and one or more listener(s), reset the interface of a device and a device itself, query for status information and configure an instrument for remote control. A device receives these interface messages, which are identified by the ATN line being active. Interface messages (sometimes called commands) that affect every device connected to the IEEE-488 bus belong to the universal command group and interface messages the affect only the addressed device are called addressed command group.

Talker listener devices

A device which can send device specific messages to the bus after it has been addressed is called a talker. A device which can receive device specific data from the bus is called a listener. Most devices do combine both of these characteristics. To transfer data there must be exactly one talker and at least one listener. Since device specific messages are not specified by the IEEE-488.1 specification these messages vary between the different instruments. The more recent IEEE-488.2 specification uses the SCPI (Standard Commands for Programmable Instruments) language to control instruments. These messages bases on the IEEE-488.1 hardware protocol. For this reason it is possible to control a IEEE-488.2 device by a controller that just conforms to IEEE-488.1.

Transfering data

At the beginning of a communication sequence, the controller sends a talker and a listener address. Then, the ATN (attention) line is set to 'false' and the addressed talker starts with its data transfer. Each of the data bytes is transfered by using the three line handshake which enshures that the talker does not send another data byte before the listener has accepted the actual byte. The data transfer is terminated by a special character the EOS (End Of Sequence; usually a linefeed, ASCII code 10) or by a special control line, the EOI line (End Or Identify). Using the EOI line is commonly used for binary data transfer. After the data has been transfered, the controller sets ATN to 'true' and may initiate the next sequence. The controller can also address itself as a talker/listener which is the most usual case.

2.2 Notes on Specific Hardware

2.2.1 GPIB-PC/104

I/O address configuration

The DIP switch SW 1-2 on the card is used to select the base I/O address of the card. Position 6 is ignored. Set position 1 through 5 according to the following table. Choose an unallocated region of I/O addresses.

GPIB-PC104 address selection

HEX	DEC	Pos 1	Pos 2	Pos 3	Pos 4	Pos 5
100	256	on	on	on	off	on
120	288	off	on	on	off	on
140	320	on	off	on	off	on
160	352	off	off	on	off	on
180	384	on	on	off	off	on
1A0	416	off	on	off	off	on
1C0	448	on	off	off	off	on
1E0	480	off	off	off	off	on
200	512	on	on	on	on	off
220	544	off	on	on	on	off
240	576	on	off	on	on	off
260	608	off	off	on	on	off
280	640	on	on	off	on	off
2A0	672	off	on	off	on	off
2C0	704	on	off	off	on	off
2E0	736	off	off	off	on	off
300	768	on	on	on	off	off
320	800	off	on	on	off	off
340	832	on	off	on	off	off
360	864	off	off	on	off	off
380	896	on	on	off	off	off
3A0	928	off	on	off	off	off
3C0	960	on	off	off	off	off
3E0	992	off	off	off	off	off

IRQ address configuration

IRQ 5 is preselected. If conflicting with another card change it to an unallocated IRQ channel.

2.2.2 GPIB-PCW, GPIB-HS-NT+, GPIB-AT

I/O address configuration

The I/O address DIP switch on the card is used to select the base I/O address of the card. Position 8 must always be ON. Set position 1 through 7 according to the following table. Choose an unallated region if I/O addresses.

GPIB-PC, GPIB-HS-NT+, GPIB-AT address selection

HEX	DEC	Pos 1	Pos 2	Pos 3	Pos 4	Pos 5	Pos 6	Pos 7
100	256	ON	ON	ON	ON	ON	OFF	ON
108	264	OFF	ON	ON	ON	ON	OFF	ON
110	272	ON	OFF	ON	ON	ON	OFF	ON
118	280	OFF	OFF	ON	ON	ON	OFF	ON
120	288	ON	ON	OFF	ON	ON	OFF	ON
128	296	OFF	ON	OFF	ON	ON	OFF	ON
130	304	ON	OFF	OFF	ON	ON	OFF	ON
138	312	OFF	OFF	OFF	ON	ON	OFF	ON
140	320	ON	ON	ON	OFF	ON	OFF	ON
148	328	OFF	ON	ON	OFF	ON	OFF	ON
150	336	ON	OFF	ON	OFF	ON	OFF	ON
158	344	OFF	OFF	ON	OFF	ON	OFF	ON
160	352	ON	ON	OFF	OFF	ON	OFF	ON
168	360	OFF	ON	OFF	OFF	ON	OFF	ON
170	368	ON	OFF	OFF	OFF	ON	OFF	ON
178	376	OFF	OFF	OFF	OFF	ON	OFF	ON
180	384	ON	ON	ON	ON	OFF	OFF	ON
188	392	OFF	ON	ON	ON	OFF	OFF	ON
190	400	ON	OFF	ON	ON	OFF	OFF	ON
198	408	OFF	OFF	ON	ON	OFF	OFF	ON
1A0	416	ON	ON	OFF	ON	OFF	OFF	ON
1A8	424	OFF	ON	OFF	ON	OFF	OFF	ON
1B0	432	ON	OFF	OFF	ON	OFF	OFF	ON
1B8	440	OFF	OFF	OFF	ON	OFF	OFF	ON
1C0	448	ON	ON	ON	OFF	OFF	OFF	ON
1C8	456	OFF	ON	ON	OFF	OFF	OFF	ON
1D0	464	ON	OFF	ON	OFF	OFF	OFF	ON
1D8	472	OFF	OFF	ON	OFF	OFF	OFF	ON
1E0	480	ON	ON	OFF	OFF	OFF	OFF	ON
1E8	488	OFF	ON	OFF	OFF	OFF	OFF	ON
1F0	496	ON	OFF	OFF	OFF	OFF	OFF	ON
1F8	504	OFF	OFF	OFF	OFF	OFF	OFF	ON

HEX	DEC	Pos 1	Pos 2	Pos 3	Pos 4	Pos 5	Pos 6	Pos 7
200	512	ON	ON	ON	ON	ON	ON	OFF
208	520	OFF	ON	ON	ON	ON	ON	OFF
210	528	ON	OFF	ON	ON	ON	ON	OFF
218	536	OFF	OFF	ON	ON	ON	ON	OFF
220	544	ON	ON	OFF	ON	ON	ON	OFF
228	552	OFF	ON	OFF	ON	ON	ON	OFF
230	560	ON	OFF	OFF	ON	ON	ON	OFF
238	568	OFF	OFF	OFF	ON	ON	ON	OFF
240	576	ON	ON	ON	OF	ON	ON	OFF
248	584	OFF	ON	ON	OFF	ON	ON	OFF
250	592	ON	OFF	ON	OFF	ON	ON	OFF
258	600	OFF	OFF	ON	OFF	ON	ON	OFF
260	608	ON	ON	OFF	OFF	ON	ON	OFF
268	616	OFF	ON	OFF	OFF	ON	ON	OFF
270	624	ON	OFF	OFF	OFF	ON	ON	OFF
278	632	OFF	OFF	OFF	OFF	ON	ON	OFF
280	640	ON	ON	ON	ON	OFF	ON	OFF
288	648	OFF	ON	ON	ON	OFF	ON	OFF
290	656	ON	OFF	ON	ON	OFF	ON	OFF
298	664	OFF	OFF	ON	ON	OFF	ON	OFF
2A0	672	ON	ON	OFF	ON	OFF	ON	OFF
2A8	680	OFF	ON	OFF	ON	OFF	ON	OFF
2B0	688	ON	OFF	OFF	ON	OFF	ON	OFF
2B8	696	OFF	OFF	OFF	ON	OFF	ON	OFF
2C0	704	ON	ON	ON	OFF	OFF	ON	OFF
2C8	712	OFF	ON	ON	OFF	OFF	ON	OFF
2D0	720	ON	OFF	ON	OFF	OFF	ON	OFF
2D8	728	OFF	OFF	ON	OFF	OFF	ON	OFF
2E0	736	ON	ON	OFF	OFF	OFF	ON	OFF
2E8	744	ON	ON	OFF	OFF	OFF	ON	OFF
2F0	752	ON	OFF	OFF	OFF	OFF	ON	OFF
2F8	760	OFF	OFF	OFF	OFF	OFF	ON	OFF
300	768	ON	ON	ON	ON	ON	OFF	OFF
308	776	OFF	ON	ON	ON	ON	OFF	OFF
310	784	ON	OFF	ON	ON	ON	OFF	OFF
318	792	OFF	OFF	ON	ON	ON	OFF	OFF
320	800	ON	ON	OFF	ON	ON	OFF	OFF
328	808	OFF	ON	OFF	ON	ON	OFF	OFF
330	816	ON	OFF	OFF	ON	ON	OFF	OFF
338	824	OFF	OFF	OFF	ON	ON	OFF	OFF
340	832	ON	ON	ON	OFF	ON	OFF	OFF

HEX	DEC	Pos 1	Pos 2	Pos 3	Pos 4	Pos 5	Pos 6	Pos 7
348	840	OFF	ON	ON	OFF	ON	OFF	OFF
350	848	ON	OFF	ON	OFF	ON	OFF	OFF
358	856	OFF	OFF	ON	OFF	ON	OFF	OFF
360	864	ON	ON	OFF	OFF	ON	OFF	OFF
368	872	OFF	ON	OFF	OFF	ON	OFF	OFF
370	880	ON	OFF	OFF	OFF	ON	OFF	OFF
378	888	OFF	OFF	OFF	OFF	ON	OFF	OFF
380	896	ON	ON	ON	ON	OFF	OFF	OFF
388	904	OFF	ON	ON	ON	OFF	OFF	OFF
390	912	ON	OF	ON	ON	OFF	OFF	OFF
398	920	OFF	OFF	ON	ON	OFF	OFF	OFF
3A0	928	ON	ON	OFF	ON	OFF	OFF	OFF
3A8	936	OFF	ON	OFF	ON	OFF	OFF	OFF
3B0	944	ON	OFF	OFF	ON	OFF	OFF	OFF
3B8	952	OFF	OFF	OFF	ON	OFF	OFF	OFF
3C0	960	ON	ON	ON	OFF	OFF	OFF	OFF
3C8	968	OFF	ON	ON	OFF	OFF	OFF	OFF
3D0	976	ON	OFF	ON	OFF	OFF	OFF	OFF
3D8	984	OFF	OFF	ON	OFF	OFF	OFF	OFF
3E0	992	ON	ON	OFF	OFF	OFF	OFF	OFF
3E8	1000	OFF	ON	OFF	OFF	OFF	OFF	OFF
3F0	1008	ON	OFF	OFF	OFF	OFF	OFF	OFF
3F8	1016	OFF	OFF	OFF	OFF	OFF	OFF	OFF

IRQ address configuration

IRQ 5 is preselected. If conflicting with another card change it to an unallocated IRQ channel.

2.2.3 GPIB-USB-2

LED Signals

The GPIB-USB-2 has a LED (light-emitting diode) that signals the operating state of the adaptor.

LED off	The GPIB-USB-2 adaptor is not operated by an operating system driver (or not plugged).
LED on	The GPIB-USB-2 adaptor is in use by application software.
LED blinking fast	The GPIB-USB-2 adaptor is in use by application software and transmitting data.
LED pulse	The GPIB-USB-2 adaptor is operated by an operating system driver but not in use by application software.