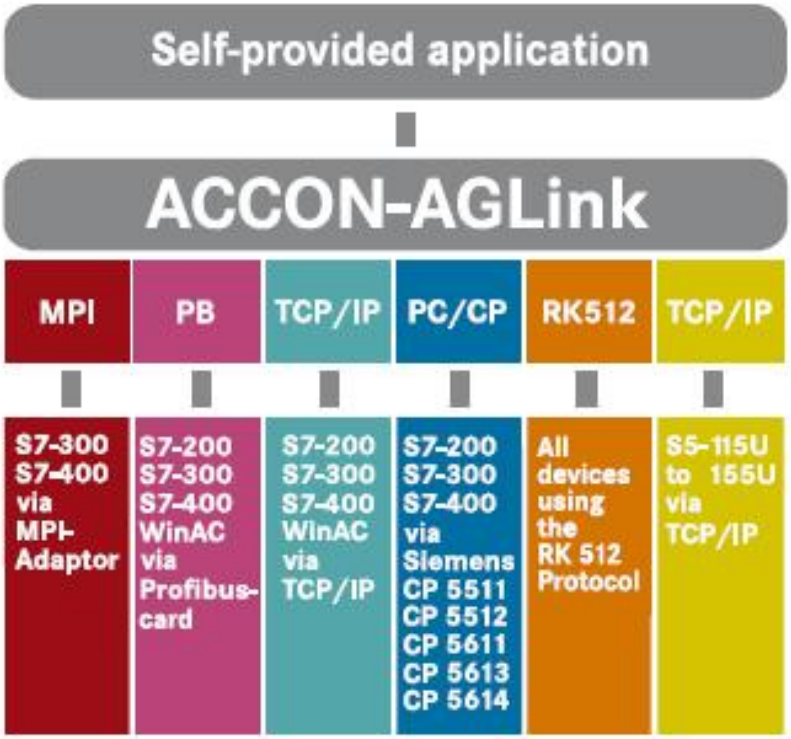
DELTALOGIC



ACCON-AGLink
User Manual

Version 4.0

The Best Solutions for PLC

DELTALOGIC Automatisierungstechnik GmbH

Stuttgarter Str. 3
73525 Schwaebisch Gmuend

Germany

Phone sale: +49-(0)7171-916-120
Phone support: +49-(0)7171-916-112

Fax sale: +49-(0)7171-916-220
Fax support: +49-(0)7171-916-212

E-Mail sale: vertrieb@deltalogic.de
E-Mail support: support@deltalogic.de

Web site: http://www.deltalogic.de

Last update 2009-03-31. All technical changes reserved.

ACCON-AGLink Version 4

with

ACCON-AGLink S7 serial

ACCON-AGLink S7 serial/TS

ACCON-AGLink S7-PB

ACCON-AGLink S7-TCP/IP

ACCON-AGLink S7-PC/CP

ACCON-AGLink S5-AS511

ACCON-AGLink S5-TCP/IP

ACCON-AGLink RK512/3964R

Preface

The quality of an automation solution depends strongly how far the different levels production control, production planning and quality assurance are integrated. One of the main problems is that production data are recorded by a PLC, but a long-term storage and a meaningful evaluation of these data isn't practicable on the PLC. PCs are excellently suitable for an economical solution of such tasks. You can save lots of more data and you are able to process these data due to the higher programming languages much more efficiently. Unfortunately,until now PLC and PC couldn't be coupled program technically so that automation solutions often weren't realized optimally.

The industrial PLC communication suite helps you. This consists of one single library that extends all common Windows programming languages by the ability to contact SIMATIC controls directly out of the PC and exchange data between PLC and PC in a simple way. You can build up the process data communication traffic from any programming language which can call functions in a 32 bit Windows DLL by handling calling simply manageable, efficient and tested functions. The developer needs no knowledge about the different used communication protocols, he/she can fully concentrate on further processing of the data (e.g. by means of a data base).

The industrial PLC communication suite offers the following highlights:

- Up to 256 devices can be controlled by one PC at the same time.
- Via every port several PLCs can be addressed at the same time depending on the protocol and the used hardware.
- The industrial PLC communication suite is written in high optimized C/C++. For this reason there is very little time needed for the communication.
- ACCON-AGLink is available as DLL for Win32, WinCE as well as a Shared Opject for Linux
- From the industrial PLC communication suite you purchase only one developer version per developer place, no further runtime charges have to be paid.
- The industrial PLC communication suite is available as 32 bit DLL for Windows. For this the industrial PLC communication suite supports a variety of programming languages enables you to concentrate on your application and not have to learn additional programming languages .
- Because of the uniform programming interface you can access the different controls without change of the PLC program.
- With the enclosed configuration tool all communication parameters can be adjusted without having to program the corresponding dialogues.

With the industrial PLC communication suite you can develop applications which collect a large quantity of process data. You can convert and store it into an arbitrary format. You can carry out meaningful evaluations and configure and steer your process therefore optimally. You can also influence the process in a simple way. For this you overwrite certain data ranges in the PLC memory and change the process parameters. With that e.g. a recipe administration is very simply feasible.

1 Table of Contents

2 ACCON-AGLink

2.1 In general

The ACCON-AGLink consists of one single library where all functions of your license are included. The possible communication modules are:

- **ACCON-AGLink S7 serial** for the access to the SIMATIC S7 300 and S7 400 via PC adapter, respectively MPI adapter as well as to the S7 200 via PPI cable.
- **ACCON-AGLink S7 serial/TS** for the access to the SIMATIC S7 300 and S7 400 via a modem and the TS adapter. This module contains the ACCON-AGLink S7 serial.
- **ACCON-AGLink S7-PB** for the access to SIMATIC S7-200, S7-300 and S7-400 via ACCON-NetLink, ACCON-NetLink PRO, ACCON-NetLink USB, ACCON-, Hilscher and Softing-Profibus cards.
- **ACCON-AGLink S7-TCP/IP** for the access to SIMATIC S7-200, S7-300 and S7-400 via TCP/IP as well as to access projected connections to the SIMATIC S7-300 and S7-400 via TCP/IP.
- **ACCON-AGLink S7-PC/CP** for the access to SIMATIC S7-200, S7-300 and S7-400 via Siemens CP 5511, CP 5512, CP 5611, CP 5613 and CP 5614.
- **ACCON-AGLink S5-AS511** for the access to the SIMATIC S5 family via AS511.
- **ACCON-AGLink S5-TCP/IP** for the access to the SIMATIC S5 family via TCP/IP.
- **ACCON-AGLink RK512/3964R** for the RK512/3964R communication.

The function library has been developed and tested in detail for Win32 (Windows 98, Windows NT, Windows 2000 and Windows XP), WinCE (4.2 and 5.0) as well as for Linux (Kernel 2.6). The test programming languages MS Visual C/C++ 6, MS Visual BASIC 6, Borland C++ Builder 5, Delphi 5, NI LabView, MS Excel, MS C# and MS VB.net were used. Examples that clarify the application and make it easier to start are provided to the function library.

Please contact us if you need support for other platforms or programming languages.

You can take the supported hardware on PC and SPS side as well as the actual availability and the planned implementation from the following table:

ACCON-AGLink Modul	PC-Hardware	SPS / CP	Win32	WinCE	Linux
S7 seriell	PC-/MPI-/TS-Adapter	S7-300/S7-400	X	*	X
	PPI-Adapter Singlemaster	S7-200	*	*	*
S7 seriell/TS	AT-Modem	S7-300/S7-400	*	*	*
		S7-200	*	*	*
	TAPI	S7-300/S7-400	*		
		S7-200	*		
S7-PB	NetLink / NetLink-S7	S7-300/S7-400	X	X	X
		S7-200	X	X	X
	NetLink-PRO	S7-300/S7-400	X	X	X
		S7-200	X	X	X
	NetLink-USB	S7-300/S7-400	X	*	X
		S7-200	X	*	X
	ACCON-/CIF-PB cards	S7-300/S7-400	X	*	*
		S7-200	X	*	*
	Softing-PB cards	S7-300/S7-400	X	*	*
		S7-200	X	*	*
S7-TCP/IP	Ethernet	CP 343-1	X	X	X
		CP 443-1	X	X	X
		CP 243-1	X	X	X
		S7-31x PN	X	X	X
S7-PC/CP	PG-PC interface	S7-300/S7-400	X		
		S7-200	X		
S5-AS511	COM-Port	S5	*	*	*
S5-TCP/IP	Ethernet	CP 1430 TCP	X	X	X
		INAT S5 TCP	X	X	X
		VIPA S5 TCP	X	X	X
RK512/3664R	COM-Port	CPs und PCs	X	*	X

Chart 1: ACCON-AGLink Synoptical Table

- X Protocol support is available
- * Protocol support is planned
- Protocol support is neither available nor planned

2.2 The Idea behind ACCON-AGLink

There are different control and communication standards in the industrial automation surrounding. If access to different products is required, this often means:

- Search for a communication toolbox supplier
- Acquisition of a new communication toolbox
- Training into the contained functions
- Hoping that the tools of the different manufacturers can be selected from one single program without problems. (Since a serial interface for the communication is often needed, this leads to complications, easily.)
- Adaptation of own programs
- Administration of different releases

This procedure is not only cost, but also time-consumming. With the ACCON-AGLink it is now possible to access different communication drivers with a uniform software interface. For sure, this can also be handled with OPC but not with such a low overhead and the compactness of ACCON-AGLink.

The mode of communication and the communication parameters can be set comfortable with a separate program and also by program code. The currently used communication driver can be utilized to investigate the run time. Furthermore, every communication module still offers additional functions, to address the controls quite specifically. For portable programming it is important that for other modules an error code can be delivered as a function result at the same point.

With this technique it is possible to modify the interface for the S7 access without program changes or to modify the local access into an access via telephone and modem.

2.3 Support for ACCON-AGLink

If questions or problems arise when using the ACCON-AGLink, please consult our technical support. You can contact us either by phone (+49)-(0)7171-916-112 or preferably by e-mail: support@deltalogic.de.

Send us your questions or the problem description in detail of the compiler release, the operating system release and the **minimum source text part** or even better with the **minimum project** with which the behavior can be reproduced. You will get support from the developer team of the ACCON-AGLink very quickly. At first check if your problem is already solved in the appendix in chapter »16 Appendix A FAQ list« and if a solution is already offered or a new release is available.

2.4 Project Support

Contact us if you have a personnel bottleneck or if you just need our competent project support. We can realize low-priced partial or also complete projects with the ACCON-AGLink according to your wishes and requests. Of course, this is possible in line with an individual training. At its end you have prepared a working standard project which can be expanded and completed. If you are interested just contact us, we will make you an offer without obligation.

2.5 Installation instructions

2.5.1 Win32

The installation setups are located on the DELTALOGIC-Automatisierungstechnik-CD or you can download it from our web site. Depending on your license please execute the following setup:

- Demo version
  Content: Demo version of the AGLink40.dll, sample programs and the manual.
  Installation setup: SetupAcconAGLink.exe
- Single license
  Content: Latest version of the AGLink40.dll and the AGLink.dll.
  Installation setup: SetupAcconAGLinkSingle.exe
  Choose the corresponding license type.
  Activation via USB-Dongle or software authorization.
- Developer license
  Content: Content: Demo version of the AGLink40.dll, sample programs and the manual.
  Installation setup: SetupAcconAGLink.exe
  When purchasing a developer license you receive a licensed DLL on data medium (e.g. floppy disc) with corresponding activated modules. The demo DLL has to be replaced by the DLL on the data medium. The activation is carried out when calling the AGLink function AGL_Activate and entering your activation key.

As basis for your application please use the available sample programs and complete them if needed.

2.5.2 WinCE

Will be added.

2.5.3 Linux

Before describing the installation process it is necessary to go into the nomen-clature's features of mutually used libraries (shared objects). The library data contains the version information in the data name e.g. libAGLink40.so.4.0.2.12. Shortcut data files belong to the proper library data which refer to it and contain only the main version as the case may be no version information e.g libAGLink40.so.4 and libAGLink40.so. The use of shortcuts always allows to refer to the same library data. The shortcut leads intrasystem in each case to the actual library.

To install do the following: When using a graphical user interface open the console (commanding input window). Go to the folder which conatins the library and the shortcut data files (CD »folder»). Copy the library data and the shortcut data files into a mutually used library folder e.g. /usr/lib. Superuser rights are required! As the case may be swap to the superuser mode via the command »su« and enter the root_ password or contact your system administrator.

To copy the data files enter the following commands and insert into the first command the appropriate data name to your actual version.

```
cp -f    libAGLink40.so.4.0.2.12 /usr/lib
cp -f -d libAGLink40.so.4        /usr/lib
cp -f -d libAGLink40.so          /usr/lib
```

Check the successful copy e.g. when entering:

```
ls /usr/lib/libAGLink40*
```

At least the above indicated data files have to be listed.

Finally the internal library cache has to be rebuild by this command

```
/sbin/ldconfig
```

The execution of this command may take considerable time.

The AGLink40 Library is now installed. If necessary leave the superuser mode by the command »exit« and if desired leave the console by typing exit again.

2.6 Demo Limitation

The demo version for Win32 is fully functional. Only a small notice appears all 5 minutes which indicates that it‘s a demo version of ACCON-AGLink. There are no further limitations e.g amount of data. So you can put the ACCON-AGLink to the acid test in your own programs and get an idea of the performance of this toolbox.

When initializing the demo version of the ACCON-AGLink for Linux it reports that this version is a demo. This message is carried out by the standard error output »stderr«. The demo has a 20-minute limit after the first use. After this limit all calls from device and communication functions will output the error code » AGL40_TIME_EXPIRED».

2.7 Communication Sequence in principle

This chapter explains the basic procedure of a communication sequence in short. Please take a detailed description of the single functions as well as its parameter and return value from the reference part.

Adjust with AGLink40_Config.EXE the desired communication path. If you want to access a S7 control via a PC or TS adapter connect it to a parametrized interface. Adjust the baud rate preferably to 38400 baud. In case you have an ACCON-MPI-adapter (with automatic baud rate detection) ACCON-AGLink will run it with its maximum communication speed.

If you are using a developer version please activate it with AGL_Activate.

Please call the function AGL_OpenDevice. It checks if the selected interface is available in the system.

Then call the function AGL_DialUp (due to compatibility reasons to the TS version) If not communicating via a modem this function always succeeds. In the other case the function tries to contact the remote station.

Now initialize the communication adapter with AGL_InitAdapter. E.g. in a S7 communication you can either retrieve the participants connected to the bus with AGL-GetLifeL or when communicating via a PC/TS adapter the directly connected PLC with AGL-GetDirectPLC.

Establish a connection to the PLC with AGL_PLCConnect as the case may be AGL_PLCConnectEx. Now you can use all functions to acces the PLC. Execute them when desired.

Disconnect with AGL_PLCDisconnect

Log off from the communication hardware with AGL_ExitAdapter

Quit a modem connection by AGL_HangUp

Release the resources once again by AGL_CloseDevice.

It is very important that for each Open, DialUp, Init, Connect a Close, HangUp, Exit, Disconnect has to be called.

2.8 Error Codes

Calling a function in ACCON-AGLink you get back an error code. When requesting please always use symbolical constants as it is possible that future versions could have different absolute values.

The error codes' meanings are in the respective header or module data.

2.9 Call convention and structure orientation (alignment)

The DLL's call convention is under Win32 __stdcall. In Visual Basic standard »call« is the default setting for DLL functions. In Delphi the call convention stdcall is listed explicit behind every function declaration.

For the data types a byte orientation is adjusted by pragmas in the header file. If possible the structures were arranged that in minimum one word alignment of the individual elements is possible (among other things because of VB). In case of needs, explicit filler bytes were implemented.

2.10 Static or dynamic loading of ACCON-AGLink

Normally, the C/C++ program is statically connected with the AGLink40 Lib library. A version from this library will be delivered for Microsoft Visual C/C++ and Borland C++ Builder in the respective demo folder. According to your program options you can also load ACCON-AGLink dynamically if you do not need the ACCON-AGLink's functionality all the time. To do this just add AGLink40.C to your project. This source code module is located in the respective demo folder of the relevant compiler, too. With this module the ACCON-AGLink-DLL will be automatically and dynamically loaded when accessing an ACCON-AGLink function for the first time. By this module it is not necessary to change your source code. Thus you can set the loading method by choosing the library(=static) or source module(=dynamic). You will receive an error if it is not possible to load the DLL dynamically or when a called function is not in the DLL e.g. the source module is newer than the used library.

3 Enclosed Programs

3.1 The Configuration Program (AGLink40_Config.EXE)

3.1.1 Parametrization

You can execute the complete parametrization or rather interface adjustment with the enclosed program AGLink40_Config.EXE. If you do this externally and not program controlled you only have to change settings when changing communication hardware e.g. from PC adapter to Profibus card. Then your application runs immediately without any program change. AGLink40_Config.EXE only uses functions from the ACCON-AGLink40.DLL that means the complete parametrization can be realized via your application. But we recommend an external configuration program due to easy hardware change.

Start the program AGLink40_Config.EXE.

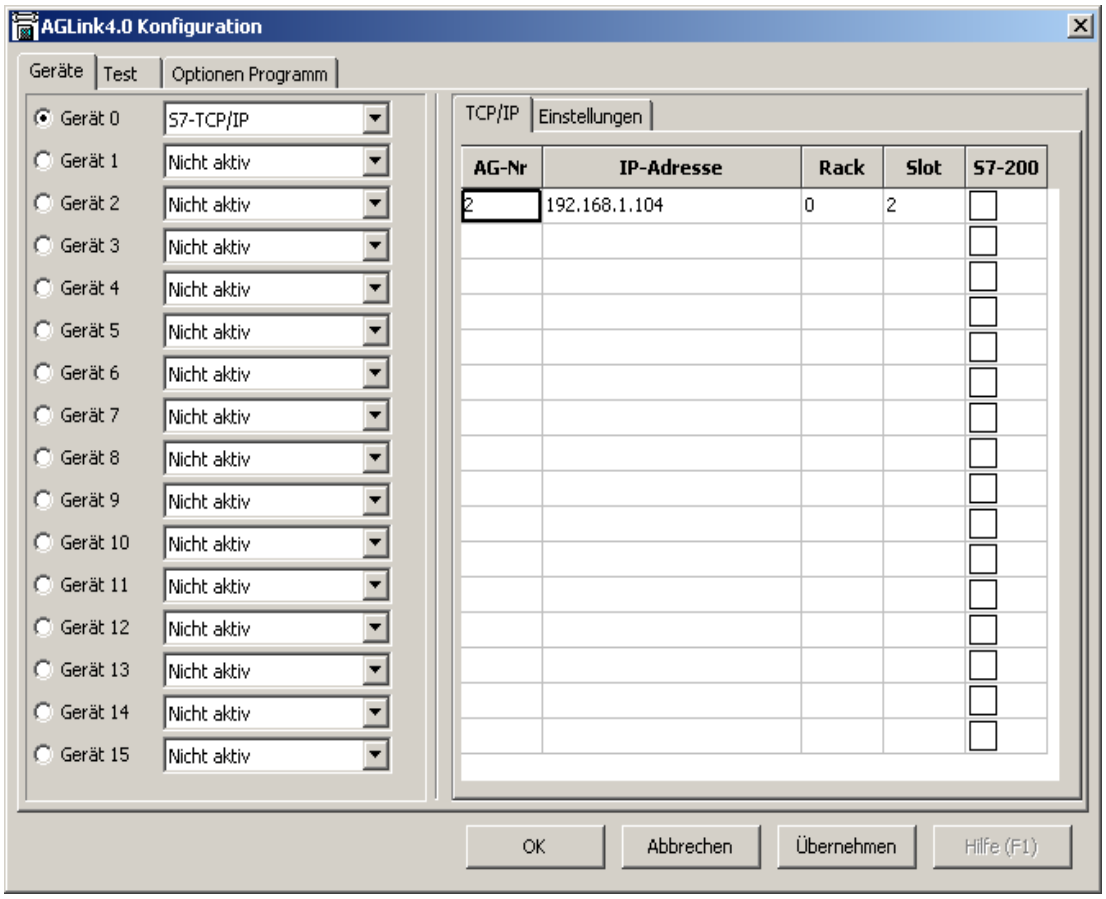


Figure 1: AGLink 4.0 Configuration

On the left side you can choose the desired device and adjust the device type in the appropriate combo box. In the tab control on the right side, the configuration possibilities suitable to the type are displayed. When changing settings you can accept them on the chosen device respectively save them in a XML data file.When exiting the programm with OK all data records of modified devices are saved in the related XML data file.

For each device all data files are saved in an own data file. Because of this they can be easily copied and duplicated. The data name for Device 0 is AGLink40CfgDev0000.xml.

The communication paths for every device are completely and independently adjustable from each other. By using the function AGL_SetDevType you can switch the communication path in the program (not possible if the device is opened). Naturally, you can parametrize e.g. Device 0 for direct access and Device 1 for modem access. Depending on which device you use program-controlled, the access takes place via a local or external adapter. Or you just always use Device 0 and let your customer decide which way he/she chooses for the actual access.

Hint:

The configuration program can be called program-controlled by using AGL_Config. For that the program just must be in the path. Thus you do not have to program out all dialogs. And of course you are allowed to pass on the configuration program with your application to your customers in the framework of our licensing requirement.

3.1.2 Device Check

In the configuration program you can perform a device check. For this choose the tab »test«. This rider is only shown if a AGLink40.DLL is in the AGLink40_Config.EXE's folder. For a clear and comprehensible error output it is important that the AGLink40_Error.TXT data is placed in the same folder.

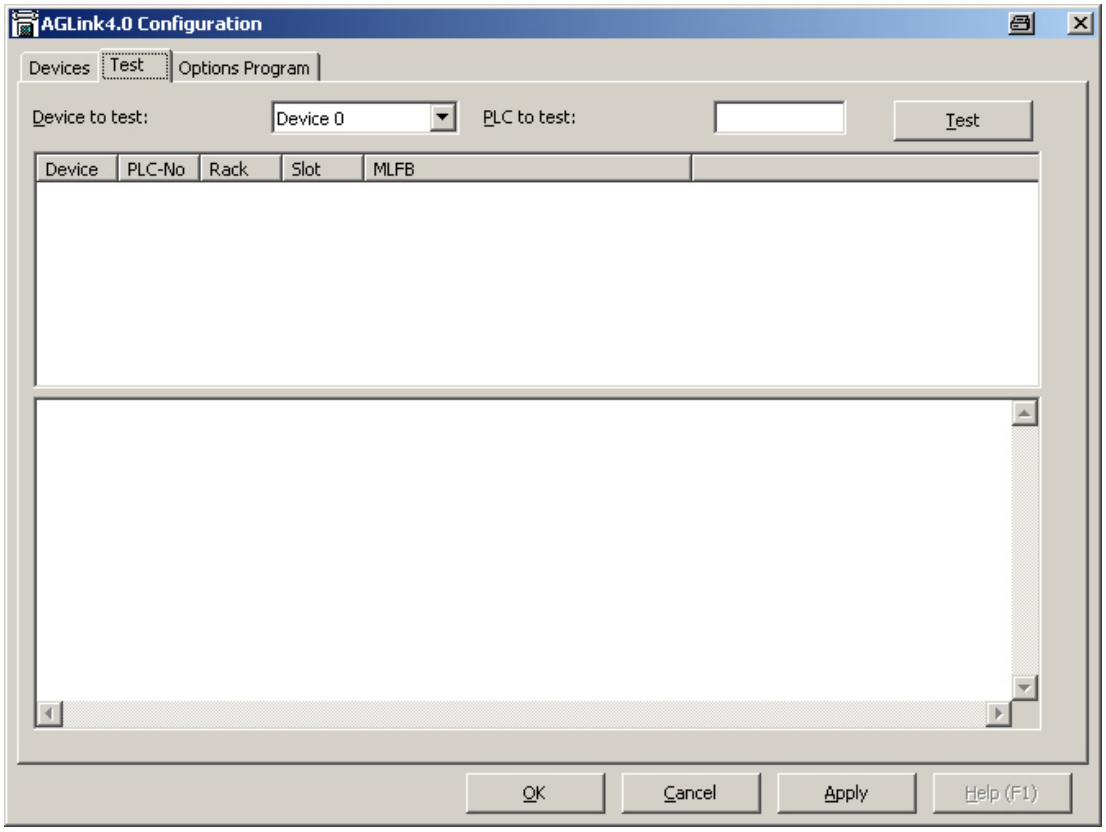


Figure 2: Device Check

Here you can check all devices or only one single device and you can limit it to one PLC. For this example the slot number of the PLC was incorrectly parametrized. The test program detects this and examines now if it can find a PLC. In this case the rack and slot number will be displayed in the table. If it is all right the columns for rack and slot number will remain blank.

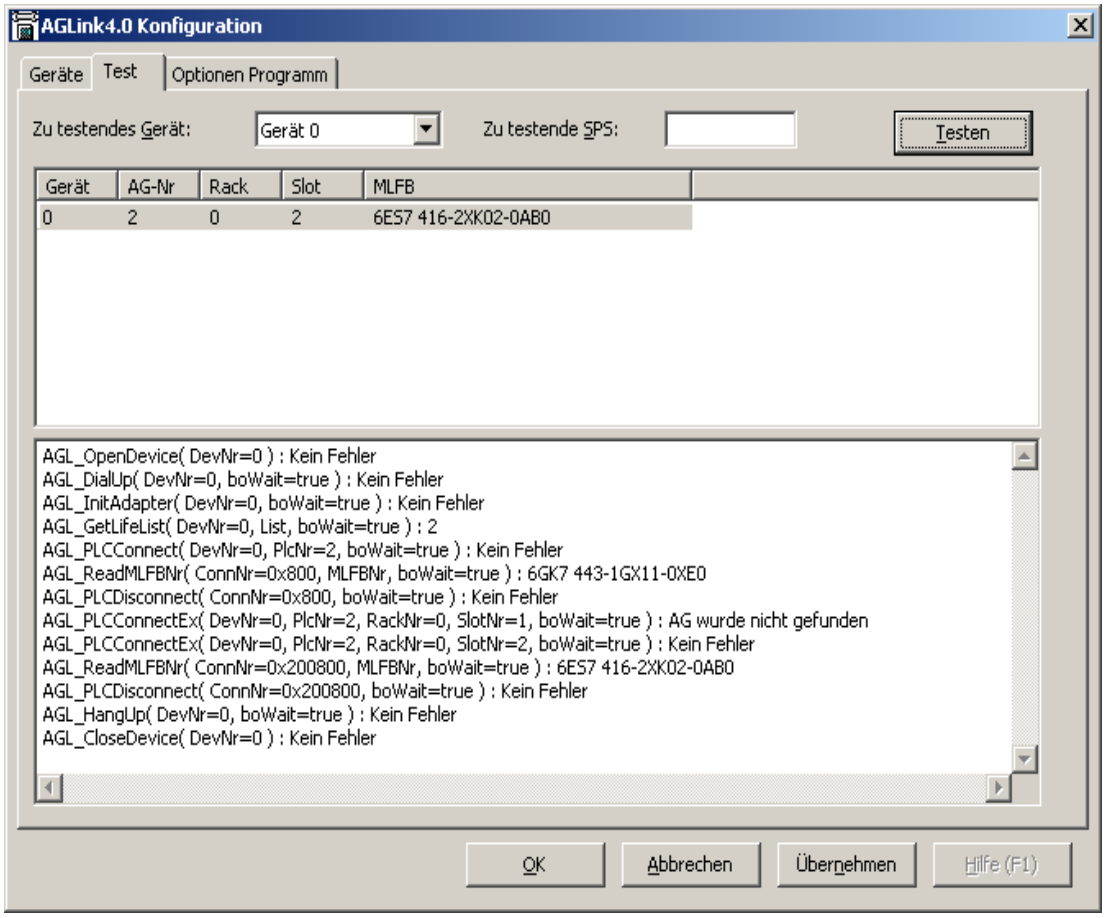


Figure 3: Example for wrong input of rack and slot number

Now accept the values to the parametrization (using TCP/IP) respectively call AGL_PLCConnectEx instead of AGL_PLCConnect (using MPI/PB) with these values.

3.1.3 Options

Naturally, you can select the language of the AGLink40_Config program. At present the languages »German« and »English« are available. For German dialogs you need the data file » AGLink40_Config_de.mo« and for English dialogs the data file »AGLink40_Config_en.mo«.

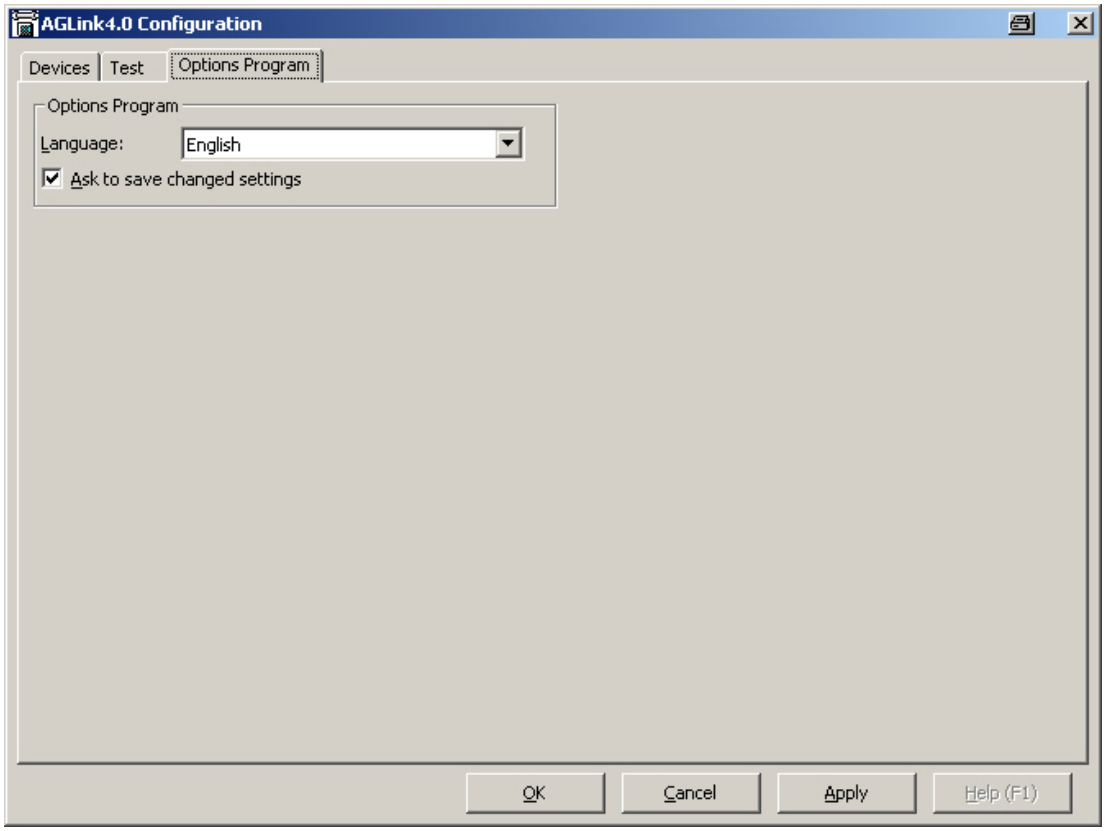


Figure 4: Language Selection

3.1.4 Features of the Linux Version

Due to the variety of spreaded Linux versions regarding kernel and scope of distribution it is only limited possible to create a binary-compatible appliance with a graphical user interface.

Now the most important system requirements are shown.

RedHat® Linux release 7.3 (Kernel 2.4.18-3) was used to create AGLink40_Config. Thus earlier systems are securely acquired.

The static Gtk2 version of the wxWidgets® library is the basis for this user interface. The target system must have installed the GTK2 package including its dependences(Atk, Pango, etc.).

AGLink40_Config was successfully tested on the following Linux systems RedHat® Linux release 7.3 and release 9.B, session type KDE and GNOME; Fedora Core 5, session type KDE and GNOME; SuSE®Linux9.3 and 10.0, only session type GNOME, **not** executable under session type KDE.

Look and handling of the Linux version correspond to the Windows version. The part »Test« is only displayed if a AGLink40 library (libAGLink40.so) can be loaded dynamically, too. The folllowing image shows the AGLink40_Config on the compiler system, session type KDE.

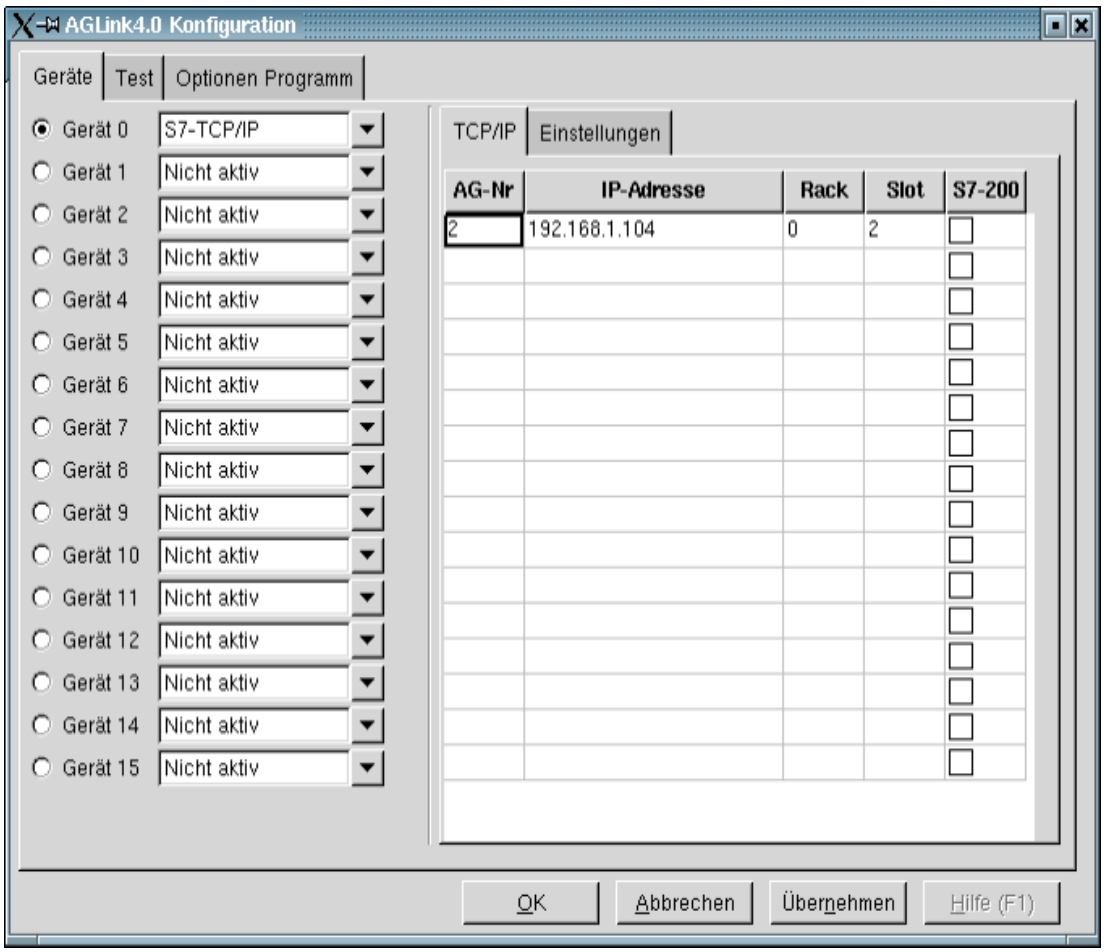


Figure 5: AGLink40_Config, compiling system session type KDE

If AGLink40_Config does not work on a determined target system e.g. due to missing libraries do the following to ascertain them.

Open a console (command input window) and switch to the folder where AGLink40_Config is located. Then enter the following command:

```
ldd ./AGLink40_Config
```

Now the list of dependencies will be output that means the libraries important for the process of AGLink40_Config. The output on the compiler system is as follows:

```
libgtk-x11-2.0.so.0 => /usr/lib/libgtk-x11-2.0.so.0 (0x40030000)
libgdk-x11-2.0.so.0 => /usr/lib/libgdk-x11-2.0.so.0 (0x4023b000)
libatk-1.0.so.0 => /usr/lib/libatk-1.0.so.0 (0x40295000)
libgdk_pixbuf-2.0.so.0 => /usr/lib/libgdk_pixbuf-2.0.so.0 (0x402ab000)
libpangox-1.0.so.0 => /usr/lib/libpangox-1.0.so.0 (0x402be000)
libpangoxft-1.0.so.0 => /usr/lib/libpangoxft-1.0.so.0 (0x402ca000)
libpango-1.0.so.0 => /usr/lib/libpango-1.0.so.0 (0x402e9000)
libgobject-2.0.so.0 => /usr/lib/libgobject-2.0.so.0 (0x4031b000)
libgmodule-2.0.so.0 => /usr/lib/libgmodule-2.0.so.0 (0x40354000)
libdl.so.2 => /lib/libdl.so.2 (0x40358000)
libgthread-2.0.so.0 => /usr/lib/libgthread-2.0.so.0 (0x4035b000)
libpthread.so.0 => /lib/i686/libpthread.so.0 (0x40360000)
libglib-2.0.so.0 => /usr/lib/libglib-2.0.so.0 (0x40374000)
libm.so.6 => /lib/i686/libm.so.6 (0x403da000)
libstdc++-libc6.2-2.so.3 => /usr/lib/libstdc++-libc6.2-2.so.3 (0x403fc000)
libc.so.6 => /lib/i686/libc.so.6 (0x42000000)
libX11.so.6 => /usr/X11R6/lib/libX11.so.6 (0x4043f000)
libXi.so.6 => /usr/X11R6/lib/libXi.so.6 (0x40514000)
libXft.so.1 => /usr/X11R6/lib/libXft.so.1 (0x4051c000)
libXrender.so.1 => /usr/X11R6/lib/libXrender.so.1 (0x40547000)
libXext.so.6 => /usr/X11R6/lib/libXext.so.6 (0x4054c000)
```

```
libfreetype.so.6 => /usr/lib/libfreetype.so.6 (0x40559000)
/lib/ld-linux.so.2 => /lib/ld-linux.so.2 (0x40000000)
```

For the missing libraries the information »not found« is displayed next to the library's name. Install the packages which contain these libraries. Normally, superuser rights are required if necessary contact your administrator. When the corresponding packages are installed please assure that the respective libraries can be found (look for manual page »ldconfig«). If libraries are missing then please start with the installation of the Gtk2 package. It has voluminous dependencies. When installing this package including the packet manager of the target system the dependent packages will be installed, too.

The indication of dependencies from AGLink40_Config and the installation of possible missing parts must be repeated until all libraries are found.

3.2 Information Program (AGLink40_Info.EXE)

To gain information about the ACCON-AGLink40-DLL on hand just call AGLink40_info. The version, the license number, the licencee and the scope of lincense are output. The ACCON-AGLink40.DLL is output in the actual folder respectively in the actual path.

With the DLL demo you will receive the following output:

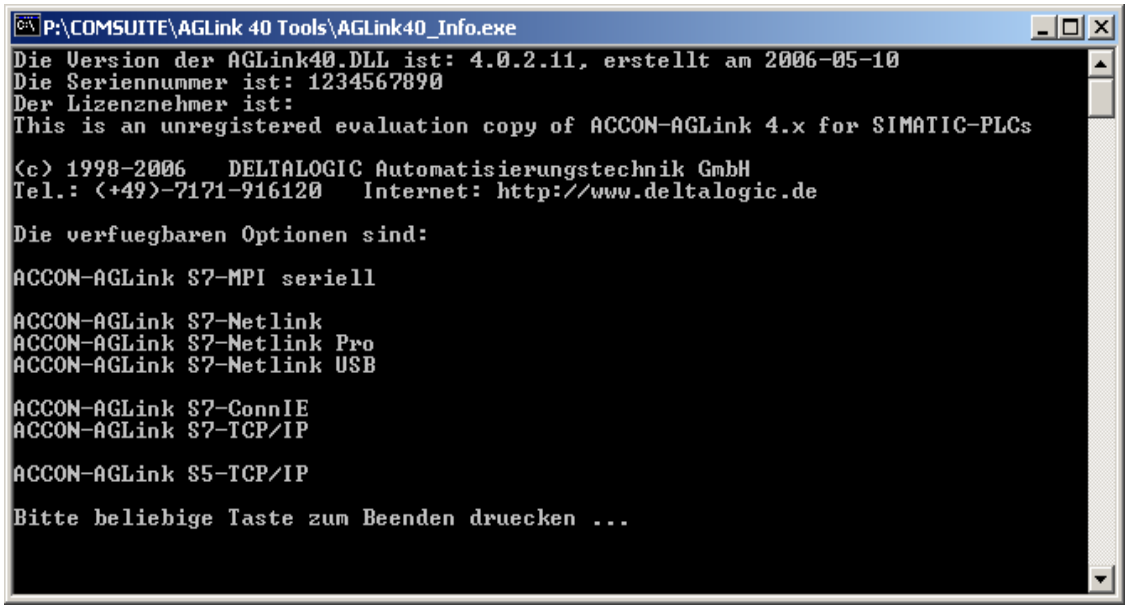


Figure 6: Output of demo version

The AGLink40_Info will be completely delivered in C/C++ sources for Microsoft C/C++ and is another example.

3.3 Performance Check Program (AGLink40_Performance.EXE)

Call AGLink40_Performance to detect the data-transfer rate of the favored connection. For this you can indicate the desired PLC as well as the rack and slot number. Then the amount of data modules and the biggest data module on the PLC are sought. It will be consulted for testing. The goal of this program is to compare the single communication ways among each other, objectively. Furthermore, with this output you can determine if the desired connection meets your requirement or if you would like to use other hardware.

To test the connection, package sizes up to 500000 bytes for each transmission are used if the chosen DB has the corresponding length.

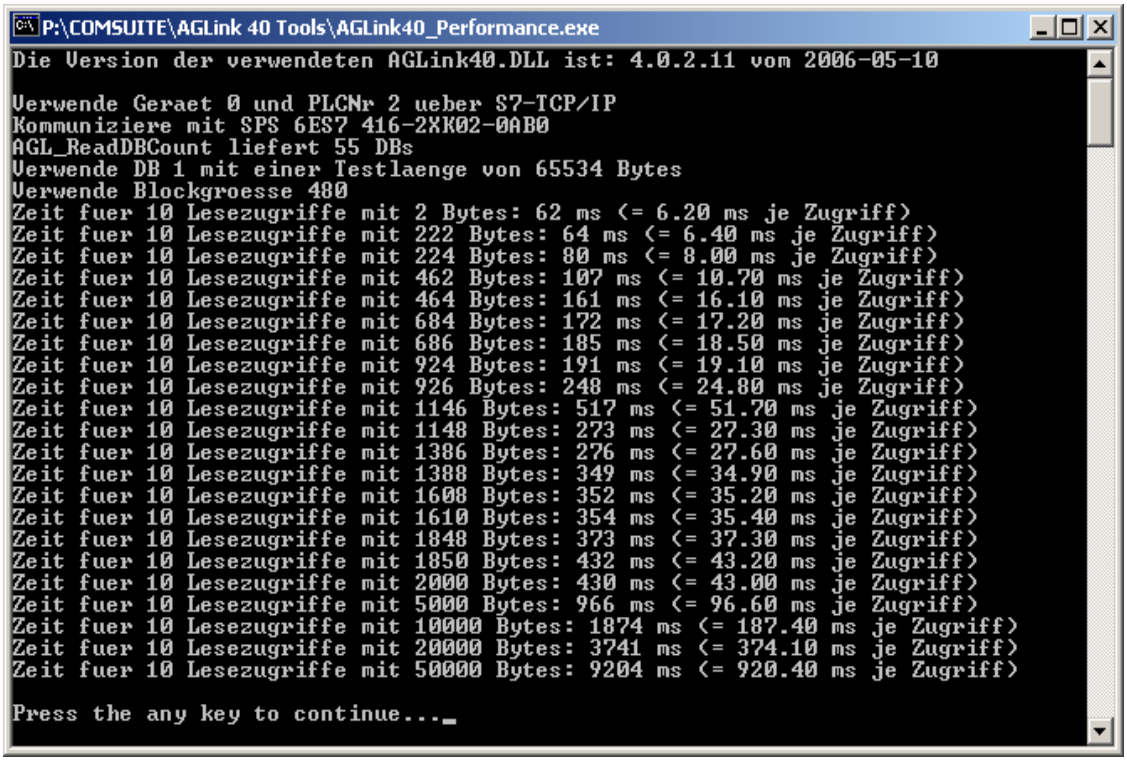


Figure 7: Testing data-transfer rate

You can indicate the following prompt/command line parameters:

-An	Indicates amount of cycles for each package size. –A100 uses 100 cycles for each package size. If not indicated 10 cycles are used
-Dn	Indicates number of used device. –D2 uses Device 2 instead of Device 0 (standard).
-Pn	Indicates number of used PLC. The connected PLCs are determined with AGL_GetLifeList by default. Tempted to test the first participant found. Use this parameter, when you only want to test a certain PLC.
-Rn	Rack number of the desired PLC. This parameter is optional.
-Sn	Slot number of the desired PLC. This parameter is optional
-Mn	Maximum read/write length tested. When using S5-TCP/IP this parameter has to be indicated because the length of data modules cannot be determined automatically.
-Nn	Number of used data module. When using S5-TCP/IP this parameter has to be indicated because the data module book keeper cannot be determined automatically.
-V-	Do not output error mesaages. This option should not be selected because important notices are suppressed.
-W+	Perform reading **and** writing test. **Attention** The data files of the used DB will be overwritten. Use this parameter with caution.

Output example with activated writing test:

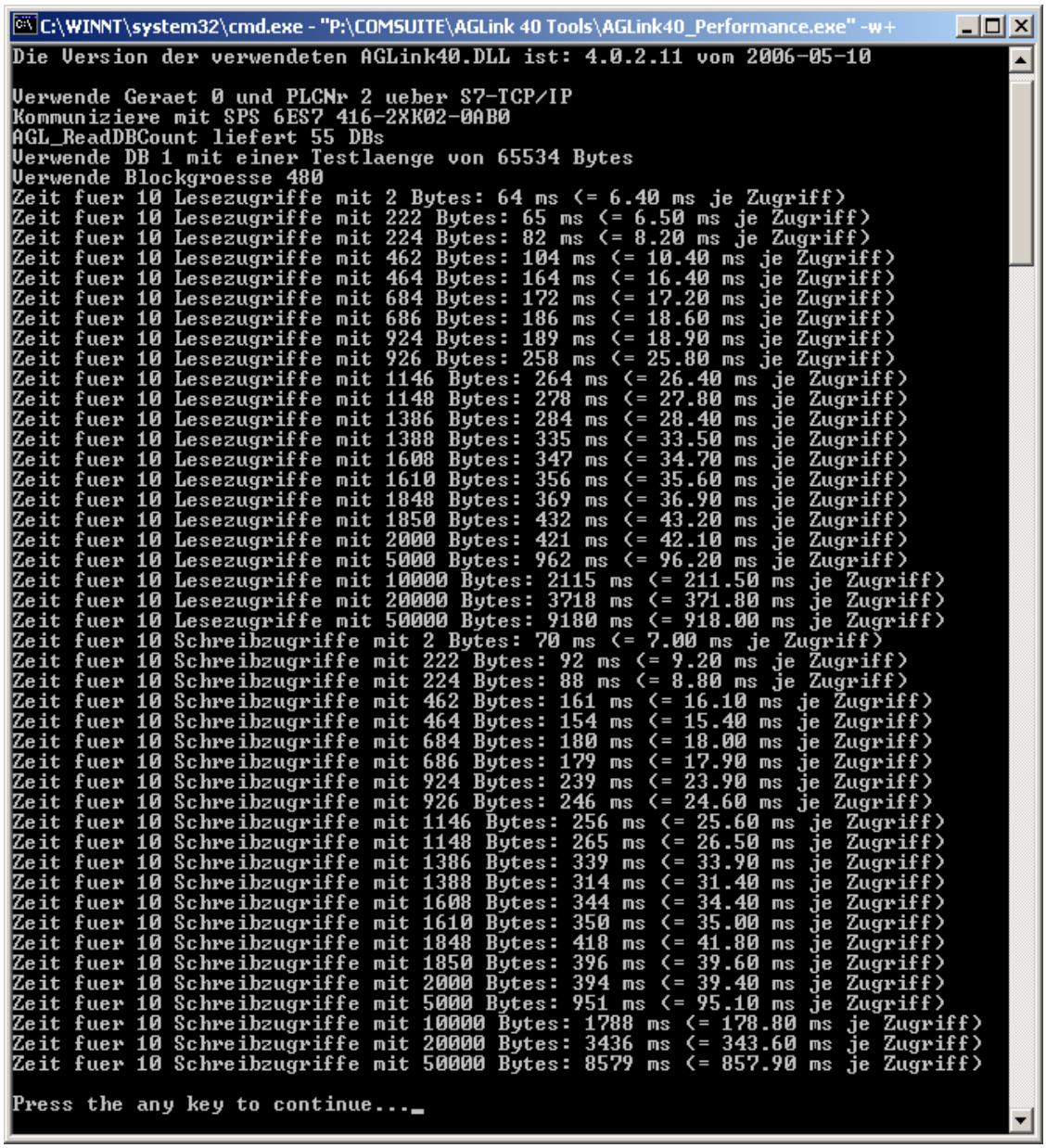


Figure 8: Output example, writing test activated

The AGLink40_Info will be completely delivered in C/C++ sources for Microsoft C/C++ and is another example.

3.4 Connection Check Program (AGLink40_ConnTest.EXE)

With the program AGLink40_ConnTest.EXE you can check how one reacts to a connection loss respectively how should you react. A connection loss can occur by pulling the plug (adapter is no longer reachable), pitch the adapter from the control (adapter is no longer reachable) or switch off the removed PLC which is accessed to (adapter is still »alive« but the PLC is »dead«). The connection check program does the following: It establishes a connection and reads the operating state in a loop. If the state changes then the actual operating state is output. Otherwise a »*« is displayed on the screen for every successful access. Does this access fail an error message is displayed and a connection will be re-established. It is possible to do this in two ways. In the first case the device will be completely closed and re-opended. Then it tries to connect to the PLC again. In the other case depending on the communiction channel only necessary measures are taken. Immediately, after that it tries to reconnect to the PLC. The method can be selected via the command line parameters.

When starting AGLink40_ConnTest you will receive e.g. the following soft copy:

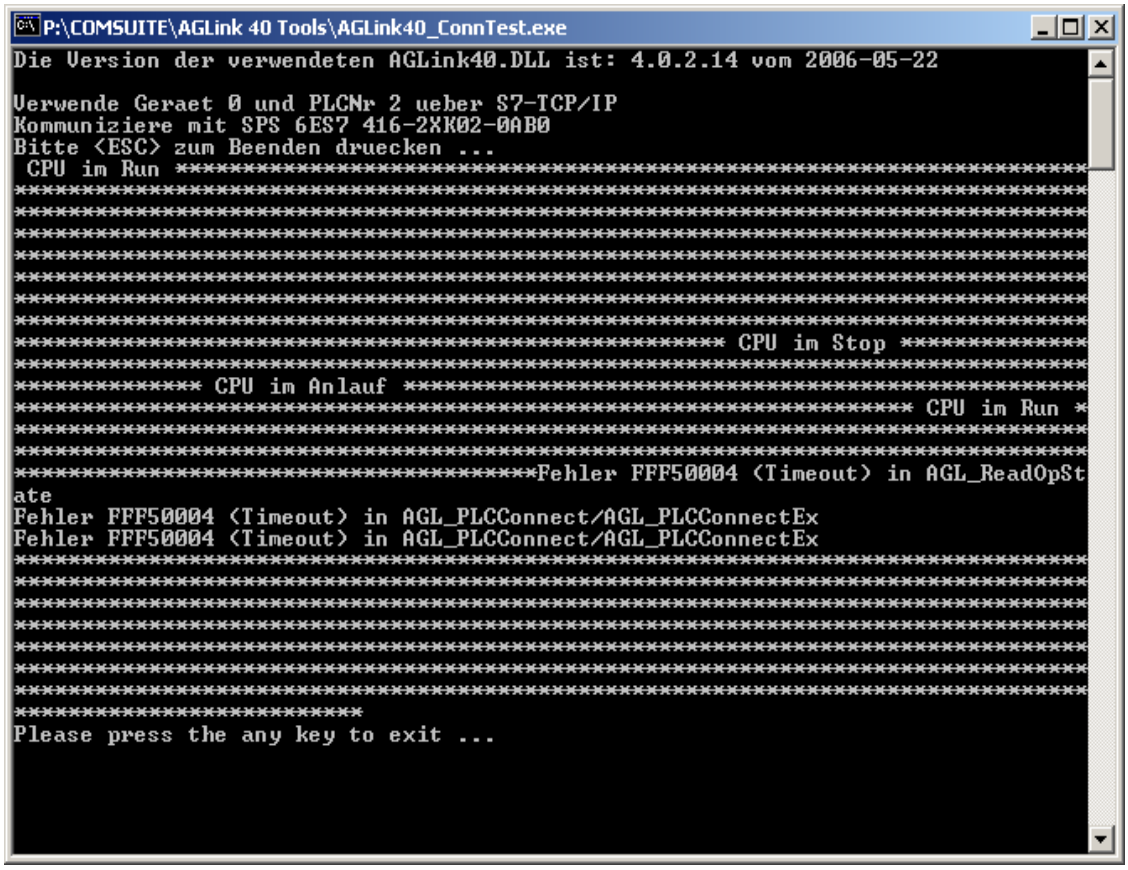


Figure 9: AGLink40_ConnTest started

You can indicate the following prompt/command line parameters:

-Dn	Indicates number of used device. –D2 uses Device 2 instead of Device 0 (standard).
-Pn	Indicates number of used PLC. The connected PLCs are determined with AGL_GetLifeList by default. Tempted to test the first found participant. Use this parameter, when you only want to test a certain PLC.
-Rn	Rack number of the desired PLC. This parameter is optional.
-Sn	Slot number of the desired PLC. This parameter is optional
-V-	Do not output error mesaages. This option should not be selected because important notices are suppressed.
-C+:.	If an error appears, close the device completely. By default only necessary steps will be performed. AGLInk40_ConnTest.CPP explains this is in detail.

AGLink40_ConnTest will be completely delivered in C/C++ sources for Microsoft C/C++ and is another example.

4 Difference between AGLink Version 4 and AGLink Version 3.x

ACCON-AGLink version 4 is not only a »normal« advancement of version 3. The complete source code has been rewritten regarding to user wishes and our gained experience. The most important requirement was the independence from operating systems. Therefore ACCON-AGLink version 4 is available for Win32, WinCE and Linux with the same software interface. Another requirement was that the basis of ACCON-AGLink version 4 could be used with ACCON-S7-Net. The communication library has already been proven hundreds of times and was put to the acid test. Furthermore a few big beta testers have accompanied the development of ACCON-AGLink and contributed to the stability and efficiency of this product. Thus ACCON-AGLink is a fully-developed product despite its changes.

ACCON-AGLink Version 3.x and ACCON-AGLink Version 4 can be used paralelly in one project. This is necessary as version 4 does not have all communication paths which are included in version 3 right from the start. For this reason all function names, structures and constants had to be changed despite of possible same meanings. From version 4 the functions are starting with AGL_ instead of AGL (e. g. AGL_OpenDevice instead of AGLOpenDevice). Normally, the constants and structures have a AGL40_ prefix.

All functions which had the parameter boWait in version 3, have now the parameter Timeout as well as Userval. Timeout is compatible to version 3 that means 1 is synchronal with standard timeout and 0 is asynchronous with standard timeout. Single functions which really take more time can be monitored with an own timeout value. E.g. the function AGL_DialUp. 60 seconds for establishing a modem connection are not really unusual. This parameter allows you to start this function with a long time monitoring and all other functions with a standard time monitoring. The parameter Userval can be any number. This number will be uninterpreted returned to the result structure. It could be possible that e.g. a class indicator is behind it which will be directly called up in the result routine.

All connection-related functions i.e. functions which require a AGL_PLCConnect or AGL_PLCConnectEx have the parameter ConnNr instead of DevNr and PlcNr. This enables connections on a multi-CPU rack. Here you have to differ between rack and slot number. An access to a PLC only by DevNr and PicNr will always work. Because of this ConnNr contains all relevant connection information to access the desired PLC.

Now an overview about the differences in detail. The specialties and things which you have to follow when porting are indicated.

4.1 Administration Functions

ACCON-AGLink Version 3.x	ACCON-AGLink Version 4
AGLActivate	AGL_Activate
AGLGetMaxDevices	AGL_GetMaxDevices
AGLGetMaxPLCPerDevice	AGL_GetMaxPLCPerDevice
AGLGetMaxQueues	AGL_GetMaxQueues
	AGL_GetTickCount
	AGL_GetMicroSecs
AGLUseSystemTime	AGL_UseSystemTime
AGLGetErrorMsg	AGL_GetErrorMsg
AGLLoadErrorFile	AGL_LoadErrorFile
AGLConfig	AGL_Config
	AGL_UnloadDyn
AGLSetWndMsgHandler	*change and use AGL_SetxxxNotification*
AGLSetThrdMsgHandler	*change and use AGL_SetxxxNotification*
AGLSetEventHandle	*change and use AGL_SetxxxNotification*
AGLSetCallback	*change and use AGL_SetxxxNotification*
	AGL_SetDevNotification
	AGL_SetConnNotification
	AGL_SetJobNotification
AGLWaitForJob	AGL_WaitForJob
AGLWaitForJobEx	AGL_WaitForJobEx
AGLDeleteJob	AGL_DeleteJob
AGLGetJobResult	AGL_GetJobResult *result structure has been changed*
AGLOpenDevice	AGL_OpenDevice
AGLCloseDevice	AGL_CloseDevice

Chart 2: Administration Functions

4.2 Communication Function

In these functions the parameter boWait has been replaced by Timeout and the parameter Userval was added. Thus a microcontrol of the time monitoring is available. The parameter Userval will be uninterpreted returned to the result structure. On connection-related functions (after AGL_PLCConnect respectively. AGL_PLCConnectEx) all additional parameters DevNr and PlcNr will be replaced by the communication handle ConnNr. Herein a complete addressing including rack and slot number is covered.

ACCON-AGLink Version 3.x	**ACCON-AGLink Version 4**
AGLDialUp	AGL_DialUp
AGLHangUp	AGL_HangUp
AGLInitAdapter	AGL_InitAdapter
AGLExitAdapter	AGL_ExitAdapter
AGLGetLifeList	AGL_GetLifeList *Here the list of all active and passive participants will be returned. And not the list of only active participants like it was in version 3.x. The list entries have other meanings*
AGLGetDirectPLC	AGL_GetDirectPLC
AGLPLCConnect	AGL_PLCConnect *This function enters the ConnNr and is for further connection-related functions available*
AGLPLCConnectEx	AGL_PLCConnectEx *This function enters the ConnNr and is for further connection-related functions available*
AGLPLCDisconnect	AGL_PLCDisconnect
AGLReadMLFBNr	AGL_ReadMLFBNr
AGLReadMLFBNrEx	AGL_ReadMLFBNrEx
AGLReadPLCInfo	AGL_ReadPLCInfo
AGLReadDiagBuffer	*change and use AGL_ReadDiagBuffer*
AGLReadDiagBufferEntry	*change and use AGL_GetDiagBufferEntry*
AGLReadDiagBufferSize	AGL_ReadDiagBufferEntrys
AGLReadDiagBufferEx	AGL_ReadDiagBuffer *parameter change: Entry is now indicator; when calling, the maximum of assigned diagnostics buffer entries for the memory area is indicated here. On function end the amount of read entries is stated (smaller or equal of the above indicated maximum)*
AGLReadDiagBufferEntryEx	AGL_GetDiagBufferEntry
AGLReadOpState	AGL_ReadOpState

ACCON-AGLink Version 3.x	**ACCON-AGLink Version 4**
AGLPLCStop	AGL_PLCStop
AGLPLCStart	AGL_PLCStart
AGLPLCResume	AGL_PLCResume
AGLReadCycleTime	AGL_ReadCycleTime
AGLReadProtLevel	AGL_ReadProtLevel
	AGL_GetPLCClock
	AGL_SetPLCClock
AGLReadSzl	AGL_ReadSzl
AGLReadDBCount	AGL_ReadDBCount *parameter list has been changed*
AGLReadDBList	AGL_ReadDBList
AGLReadDBLen	AGL_ReadDBLen
AGLReadMaxPacketSize	AGL_ReadMaxPacketSize

Chart 3: Communication Functions

4.3 Functions for Reading Data

ACCON-AGLink Version 3.x	ACCON-AGLink Version 4
AGLReadInBytes	AGL_ReadInBytes
AGLReadPInBytes	AGL_ReadPInBytes
AGLReadOutBytes	AGL_ReadOutBytes
AGLReadFlagBytes	AGL_ReadFlagBytes
AGLReadSFlagBytes	AGL_ReadSFlagBytes
AGLReadVarBytes	AGL_ReadVarBytes
AGLReadDataBytes	AGL_ReadDataBytes
AGLReadS5DataWords	AGL_ReadDataWords This function is for the S5 family, only
AGLReadTimerWords	AGL_ReadTimerWords
AGLReadCounterWords	AGL_ReadCounterWords
AGLReadMix	AGL_ReadMix Change of structure and constants for the structure elements.
AGLReadMixEx	AGL_ReadMixEx Change of structure and constants for the structure elements.
	AGL_BReceive Function is only available for parametrized connections and at present for TCP/IP. The device type which has to be assigned is »S7-ConnIE«

Chart 4: Functions for Reading Data

4.4 Functions for Writing Data

ACCON-AGLink Version 3.x	ACCON-AGLink Version 4
AGLWriteInBytes	AGL_WriteInBytes
AGLWriteOutBytes	AGL_WriteOutBytes
AGLWritePOutBytes	AGL_WritePOutBytes
AGLWriteFlagBytes	AGL_WriteFlagBytes
AGLWriteSFlagBytes	AGL_WriteSFlagBytes
AGLWriteVarBytes	AGL_WriteVarBytes
AGLWriteDataBytes	AGL_WriteDataBytes
AGLWriteS5DataWords	AGL_WriteDataWords *This function is for the S5 family, only*
AGLWriteTimerWords	AGL_WriteTimerWords
AGLWriteCounterWords	AGL_WriteCounterWords
AGLWriteMix	AGL_WriteMix *Change of structure and constants for the structure elements.e*
AGLWriteMixEx	AGL_WriteMixEx *Change of structure and constants for the structure elements.e*
	AGL_BSend *Function is only available for parametrized connections and at present for TCP/IP. The device type which has to be assigned is »S7-ConnIE«*

Chart 5: Functions for Writing Data

4.5 RK512-related Functions

ACCON-AGLink Version 3.x	ACCON-AGLink Version 4
	AGL_RKFetch
	AGL_RKFetchEx
	AGL_RKSend
	AGL_RKSendEx
	AGL_Recv_RKSend
	AGL_Recv_RKFetch
	AGL_SendRKFetch
	AGL_Send_3964
	AGL_Recv_3964

Chart 6: RK512-related Functions

4.6 Conversion Functions

ACCON-AGLink Version 3.x	ACCON-AGLink Version 4
AGLReadInt16	AGL_ReadInt16
	AGL_ReadInt32
AGLReadWord	AGL_ReadWord
AGLReadDWord	AGL_ReadDWord
AGLReadReal	AGL_ReadReal
AGLReadS5Time	AGL_ReadS5Time
	AGL_WriteInt16
	AGL_WriteInt32
AGLWriteWord	AGL_WriteWord
AGLWriteDWord	AGL_WriteDWord
AGLWriteReal	AGL_WriteReal
AGLWriteS5Time	AGL_WriteS5Time
AGLByte2Word	AGL_Byte2Word
AGLByte2DWord	AGL_Byte2DWord
AGLByte2Real	AGL_Byte2Real
AGLWord2Byte	AGL_Word2Byte
AGLDWord2Byte	AGL_DWord2Byte
AGLRead2Byte	AGL_Real2Byte
AGLBuff2String	AGL_Buff2String
AGLString2Buff	AGL_String2Buff
	AGL_Buff2WString
	AGL_WString2Buff
	AGL_S7String2String
	AGL_String2S7String
	AGL_S7DT2SysTime
	AGL_SysTime2S7DT
	AGL_TOD2SysTime
	AGL_SysTime2TOD
	AGL_BCD2Int16
	AGL_Int162BCD
	AGL_BCD2Int32
	AGL_Int322BCD
	AGL_LongAsFloat
	AGL_FloatAsLong

ACCON-AGLink Version 3.x	ACCON-AGLink Version 4
AGLGetBit	AGL_GetBit
AGLSetBit	AGL_SetBit
AGLResetBit	AGL_ResetBit
AGLSetBitVal	AGL_SetBitVal
	AGL_Text2DataRW
	AGL_DataRW2Text

Chart 7: Conversion Functions

4.7 Additional Functions

ACCON-AGLink Version 3.x	ACCON-AGLink Version 4
AGLGetDLLVersion	AGL_GetVersion
AGLGetDLLVersionEx	AGL_GetVersionEx *This function has an additional parameter*
AGLGetOptions	AGL_GetOptions *The constants for the bit masks are new defined*
AGLGetSerialNumber	AGL_GetSerialNumber
AGLGetClientName	AGL_GetClientName

Chart 8: Additional Functions

4.8 Configuration Functions

The structures for the configuration of ACCON-AGLink and the corresponding constants are completely redefined. You will find a detailed meaning of the single elements in the respective header data. Due to reasons of portability the configuration will no longer be carried out by the registry but via a XML data file. You can just copy it without any administration rights (Win32). For reasons of simplification single devices can no longer be read and written only complete devices.

ACCON-AGLink Version 3.x	**ACCON-AGLink Version 4**
AGLGetParas	AGL_GetParas
AGLSetParas	AGL_SetParas
AGLGetDevType	AGL_GetDevType
AGLSetDevType	AGL_SetDevType
AGLReadParas	AGL_ReadParas *This function reads out the parameters from the loaded XML data file*
AGLWriteParas	AGL_WriteParas *This function writes the parameters into the loaded XML data file.*
AGLReadDevice	AGL_ReadDevice
AGLWriteDevice	AGL_WriteDevice
AGLReadParasFromFile	AGL_ReadParasFromFile
AGLWriteParasToFile	AGL_WriteParasToFile
AGLReadDevType	*cancelled without substitution*
AGLWriteDevType	*cancelled without substitution*
AGLReadxxx	*cancelled without substitution*
AGLWritexxx	*cancelled without substitution*
AGLGetxxx	*Change and use AGL_GetParas*
AGLSetxxx	*Change and use AGL_GetParas*

Chart 9: Configuration Functions

5 First Steps with Win32

This chapter gives attention to the basic procedure of compiling a little program example with ACCON-AGLink version 4. But before starting you have to parametrize the used device and its communication path correctly. For this you need the following data.

```
AGLink40_Config.exe          The proper configuration program
AGLink40_Config.xml          Parameters of the configuration program
AGLink40_Config_de.mo        German language data
AGLink40_Config_en.mo        English language data
```

Optional device check:

```
AGLink40.dll                 (demo-)version of ACCON-AGLink
AGLink40_Error.txt           Data with AGLink error messages in plain text
```

First of all start the provided program »AGLink40_Config.EXE«. Then adjust Device 0 according to the used hardware. Check the settings with the »Device Check«. The configuration will be put into the data »AGLink40CfgDev0000.xml«. To start the program you only need this parameter data besides ACCON-AGLink and you can just copy this data with your application. Thus a device configuration does not have to be on the target system but can be performed on another PC.

For reasonable matters we demerge this program and perform a few little auxiliary functions. With it the complete porgram remains clearly and we can concentrate on the essential things in the examples. A possible sectioning could be:

- Output of program arguments
- Interpretation of the command line arguments (if possible)
- Output of error information
- Establishing connection to a PLC
- Connection clearing to a PLC

To analyze the command line parameter the program parameters are passed to the function, in one structure. Because of this additional parameter extensions are possible and you do not have to change the call up interface.

Naturally, an error can occur on any position at a connection establishment. Therefore, we remember our actual position in a respective variable. From this status we can close the connection, correctly. Enumeration is a suitable variable type.

5.1 First Steps with Microsoft Visual C/C++ (console application)

We need the following data for the creation and the test of a ACCON-AGLink40 application with Microsoft Visual C/C++:

AGLink40.h	Basic header data, includes further data
OSDefInc.h	Header data with definition depending on the operating system (included by AGLink40.h)
AGL_Defines.h	Header data with definition of constants (included by AGLink40.h)
AGL_Types.h	Header data with structure definitions (included by AGLink40.h)
AGL_Funcs.h	Header data with function declaration (included by AGLink40.h)
AGLink40.lib	Import library for ACCON-AGLink version 4
AGLink40.dll	The actual function library of ACCON-AGLink version 4
AGLink40_Error.txt	Data file with error messages in plain text
AGLink40CfgDev0000.xml	Data file with the configuration data, in this case from Device 0

Later only AGLink40.dll, AGLink40_Error.txt and AGLink40CfgDev0000.xml are necessary for the compiled program. In chapter »5 First Steps with Win32« you will find all essential data to change the configuration.

To ease all this we now compile a little console application. Start Microsoft Visual C/C++. Go to Data/New in the tab control then choose Win32 console application via the tab »projects«. Indicate the project name »ErsteSchritte« (first steps) and a folder where to save. Confirm with OK. Now choose an empty project. Insert a C++ source code data called »ErsteSchritte« (firststeps) via Data/New. Add the library AGLink40.Lib to the source code data files via »Add Data«.

5.1.1 The Communication's Skeletal Structure

This chapter shows how to establish a communication to a PLC. And how to detect the connected participants from the Lifelist and pick one from it. After that a message is displayed and the connection will be closed. So, this skeletal structure has all necessary tools.

For this add the new source code data file »ErsteSchritte CPP« to the project and enter the following program.

Note:
Naturally, the complete project is located in the ACCON-AGLink 40 demo folder for Microsoft Visual C/C++.

```
/*****************************************************************************

 Projekt        : New version of the AGLink library

 Data Name      : ErsteSchritte.CPP

 Description    : Introduction into the use of ACCON-AGLink version 4

 Copyright      : (c) 1998-2007
                  DELTALOGIC Automatisierungstechnik GmbH
                  Stuttgarter Str. 3
                  73525 Schwaebisch Gmuend
                  Web : http://www.deltalogic.de
                  Tel.: +49-7171-916120
                  Fax : +49-7171-916220

 Created        : 15.05.2006  RH

 Changed        : 15.05.2006  RH

 *****************************************************************************/

/*****************************************************************************

 Embedding  Header files

 *****************************************************************************/

#include <string.h>
#include <conio.h>
#include <stdio.h>

#include "AGLink40.h"


/*****************************************************************************

 Definition of Enums

 *****************************************************************************/

enum eConnState
{
  eNotInit = 0,           // Connection not initialized, yet
  eDevOpened,             // Device opened
  eDialedUp,              // Dial-up (if necessary) executed
  eInitAdapter,           // Adapter initialized
  eConnected              // Connection to PLC established
};


/*****************************************************************************

 Definition of the data types

 *****************************************************************************/

typedef struct tagProgParas
{
  int   DevNr;            // Number of the used device
  int   PlcNr;            // Number of the used PLC
  int   RackNr;           // Rack number of the PLC (0 is standard)
  int   SlotNr;           // Slot number of the PLC (0 is standard)
  int   Verbose;          // Flag if error messages should be output
  eConnState State;       // Connection status, will be registered by
                          // Open and CloseConnection
  int   ConnNr;           // Connection handle, will be registered by
                          // OpenConnection
} PROG_PARAS, *LPPROG_PARAS;
```

```
/*****************************************************************************

 Implementation of the auxiliary functions

 *****************************************************************************/

/*****************************************************************************

 Function name  : ShowError

 Parameter      : int ErrNr             Error number
                  char *Func            Function name as text

 Description    : This function outputs the error message.

 Return value   : none

 Created        : 15.05.2006  RH

 Changed        : 15.05.2006  RH

 *****************************************************************************/

void ShowError( int ErrNr, char *Func )
{
  char Fehler[256];

  if( AGL_GetErrorMsg( ErrNr, Fehler, sizeof( Fehler ) ) == 0 )
  {
    strcpy( Fehler, "Unbekannter Fehler" );
  }
  printf( "Fehler %08X (%s) in %s\n", ErrNr, Fehler, Func );
}


/*****************************************************************************

 Function name  : ShowCommandlineHelp

 Parameter      : char *InvText         Invalid argument

 Description    : This function outputs the invalid command line argument and
                  displays the valid one.

 Return Value   : none

 Created        : 15.05.2006  RH

 Changed        : 15.05.2006  RH

 *****************************************************************************/

void ShowCommandlineHelp( char *InvText )
{
  char Buff[256];
  char *FileName;

  if( InvText != NULL )
  {
    printf( "Unbekanntes Kommandozeilenargument: %s\n", InvText );
  }
  GetModuleFileName( NULL, Buff, sizeof( Buff ) );
  if( (FileName = strrchr( Buff, '\\' )) == NULL )
  {
    if( (FileName = strrchr( Buff, '/' )) == NULL )
    {
      FileName = Buff;
    }
    {
      FileName++;
```

```
    }
  }
  else
  {
    FileName++;
  }
  printf( "\nGueltige Optionen sind:\n\n" );
  printf( "%s [-dDevNr] [-pPlcNr] [-rRackNr] [-sSlotNr] [-v+|-]\n", FileName );
  printf( "  -dDevNr    Angabe der Geraetenummer      (Standard: 0)\n" );
  printf( "  -pPlcNr    Angabe der SPS-Nummer         (Standard: erste SPS aus
                                                     LifeList)\n" );
  printf( "  -rRackNr   Angabe der Racknummer         (Standard: ohne)\n" );
  printf( "  -sSlotNr   Angabe der Slotnummer         (Standard: ohne)\n" );
  printf( "  -v+        Fehlermeldungen ausgeben      (Standard)\n" );
  printf( "  -v-        Fehlermeldungen nicht ausgeben\n\n" );
}


/*****************************************************************************

 Function name  : CheckCommandline

 Parameter      : int argc              Number of arguments
                  char *argv[]          The proper command line arguments
                  LPPROG_PARAS pParas   Structure including program parameters

 Description    : This function evaluates the command line

 Return value   : none

 Created        : 15.05.2006  RH

 Changed        : 15.05.2006  RH

 *****************************************************************************/

void CheckCommandline( int argc, char *argv[], LPPROG_PARAS pParas )
{
  for( int i=1; i<argc; i++ )
  {
    if( argv[i][0] == '-' || argv[i][0] == '/' )
    {
      switch( argv[i][1] )
      {
        case 'd':
        case 'D':
        {
          pParas->DevNr = atoi( &argv[i][2] );
          break;
        }
        case 'p':
        case 'P':
        {
          pParas->PlcNr = atoi( &argv[i][2] );
          break;
        }
        case 'r':
        case 'R':
        {
          pParas->RackNr = atoi( &argv[i][2] );
          break;
        }
        case 's':
        case 'S':
        {
          pParas->SlotNr = atoi( &argv[i][2] );
          break;
        }
        case 'v':
        case 'V':
```

```
          {
            if( argv[i][2] == '+' )
            {
              pParas->Verbose = true;
            }
            else if( argv[i][2] == '-' )
            {
              pParas->Verbose = false;
            }
            break;
          }
          case '?':
          case 'h':
          case 'H':
          {
            ShowCommandlineHelp( NULL );
            break;
          }
          default:
          {
            ShowCommandlineHelp( argv[i] );
            break;
          }
        }
      }
      else
      {
        ShowCommandlineHelp( argv[i] );
      }
    }
  }

/*****************************************************************************

 Function name  : OpenConnection

 Parameter      : LPPROG_PARAS pParas   Structure with program parameters

 Description    : This function establishes a connection to a PLC depending on
                  the actual connection status. The connection establishment
                  to the PLC is synchronal (Timeout = 1 is synchronal to the
                  adjusted timeout values in the config).

 Return value   : AGL40_SUCCESS or error code

 Created        : 15.05.2006  RH

 Changed        : 15.05.2006  RH

 *****************************************************************************/

int OpenConnection( LPPROG_PARAS pParas )
{
  int RetVal;

  if( pParas->State == eNotInit )
  {
    if( (RetVal = AGL_OpenDevice( pParas->DevNr )) != AGL40_SUCCESS )
    {
      if( pParas->Verbose )
      {
        ShowError( RetVal, "AGL_OpenDevice" );
      }
      return( RetVal );
    }
    pParas->State = eDevOpened;
  }
  if( pParas->State == eDevOpened )
  {
```

```
  if( (RetVal = AGL_DialUp( pParas->DevNr, 1, 0 )) != AGL40_SUCCESS )
  {
    if( pParas->Verbose )
    {
      ShowError( RetVal, "AGL_DialUp" );
    }
    return( RetVal );
  }
  pParas->State = eDialedUp;
}
if( pParas->State == eDialedUp )
{
  if( (RetVal = AGL_InitAdapter( pParas->DevNr, 1, 0 )) != AGL40_SUCCESS )
  {
    if( pParas->Verbose )
    {
      ShowError( RetVal, "AGL_InitAdapter" );
    }
    return( RetVal );
  }
  pParas->State = eInitAdapter;
}
if( pParas->State == eInitAdapter )
{
  if( pParas->PlcNr < 0 )
  {
    BYTE LifeList[128];

    RetVal = AGL_GetLifelist( pParas->DevNr, LifeList, 1, 0 );
    if( RetVal != AGL40_SUCCESS )
    {
      if( pParas->Verbose )
      {
        ShowError( RetVal, "AGL_GetLifeList" );
      }
      return( RetVal );
    }
    for( int i=0; i<127; i++ )
    {
      if( LifeList[i] == LL_ACTIVE || LifeList[i] == LL_PASSIVE )
      {
        pParas->PlcNr = i;
        break;
      }
    }
    if( pParas->PlcNr < 0 )
    {
      RetVal = AGL40_PLC_NOT_FOUND;
      if( pParas->Verbose )
      {
        ShowError( RetVal, "AGL_GetLifeList" );
      }
      return( RetVal );
    }
  }
  if( pParas->RackNr == 0 && pParas->SlotNr == 0 )
  {
    //
    // We establish a connection with standard values
    //
    RetVal = AGL_PLCConnect( pParas->DevNr, pParas->PlcNr,
                             &pParas->ConnNr, 1, 0 );
  }
  else
  {
    //
    // We directly access the PLC on the RackNR and SlotNr
    //
    RetVal = AGL_PLCConnectEx( pParas->DevNr, pParas->PlcNr, pParas->RackNr,
                               pParas->SlotNr, &pParas->ConnNr, 1, 0 );
```

```
    }
    if( RetVal != AGL40_SUCCESS )
    {
      if( pParas->Verbose )
      {
        ShowError( RetVal, "AGL_PLCConnect/AGL_PLCConnectEx" );
      }
      return( RetVal );
    }
    pParas->State = eConnected;
  }
  return( AGL40_SUCCESS );
}

/*****************************************************************************

 Function name  : CloseConnection

 Parameter      : LPPROG_PARAS pParas   Structure including program parameters

 Description    : This function releases a connection to a PLC and closes the
                  device depending on the actual connection status. The
                  connection  establishment to the PLC is synchronal
                  (Timeout = 1 is synchronal to the adjusted timeout values in
                  the config)

 Return value   : always AGL40_SUCCESS

 Created        : 15.05.2006  RH

 Changed        : 15.05.2006  RH

 *****************************************************************************/

int CloseConnection( LPPROG_PARAS pParas )
{
  int   RetVal;

  if( pParas->State == eConnected )
  {
    RetVal = AGL_PLCDisconnect( pParas->ConnNr, 1, 0 );
    if( RetVal != AGL40_SUCCESS && pParas->Verbose )
    {
      ShowError( RetVal, "AGL_PLCDisconnect" );
    }
    pParas->State = eInitAdapter;
  }
  if( pParas->State == eInitAdapter )
  {
    RetVal = AGL_ExitAdapter( pParas->DevNr, 1, 0 );
    if( RetVal != AGL40_SUCCESS && pParas->Verbose )
    {
      ShowError( RetVal, "AGL_ExitAdapter" );
    }
    pParas->State = eDialedUp;
  }
  if( pParas->State == eDialedUp )
  {
    RetVal = AGL_HangUp( pParas->DevNr, 1, 0 );
    if( RetVal != AGL40_SUCCESS && pParas->Verbose )
    {
      ShowError( RetVal, "AGL_HangUp" );
    }
    pParas->State = eDevOpened;
  }
  if( pParas->State == eDevOpened )
  {
    RetVal = AGL_CloseDevice( pParas->DevNr );
    if( RetVal != AGL40_SUCCESS && pParas->Verbose )
    {
```

```
      ShowError( RetVal, "AGL_CloseDevice" );
    }
    pParas->State = eNotInit;
  }
  return( AGL40_SUCCESS );
}
/*****************************************************************************

 Implementation of the proper auxiliary functions

 *****************************************************************************/

/*****************************************************************************

 Function name  : main

 Parameter      : int argc              Number of arguments
                  char *argv[]          The proper command line arguments

 Description    : This function establishes a connection to a PLC. There you can
                  insert any communication functions. After that the connection
                  will be released

 Return value   : 0 if successful, otherwise error code

 Created        : 15.05.2006  RH

 Changed        : 15.05.2006  RH

 *****************************************************************************/

int __cdecl main( int argc, char *argv[] )
{
  int RetVal;
  PROG_PARAS Paras;

  //
  // Initializing parameters with standard values
  //
  Paras.DevNr   = 0;          // We are using Device 0
  Paras.PlcNr   = -1;         // We are using the first PLC on the Lifelist
  Paras.RackNr  = 0;          // We are working without ...
  Paras.SlotNr  = 0;          // ... extended addressing
  Paras.Verbose = true;       // We want to see error messages in any case

  Paras.ConnNr  = 0;          // Connection handle will be entered later
  Paras.State   = eNotInit;   // Nothing is happening, yet

  //
  // When using a developer's version please enter your license key
  //
  //
  AGL_Activate( "123456-1234-123456" );

  CheckCommandline( argc, argv, &Paras );
  if( (RetVal = OpenConnection( &Paras )) == AGL40_SUCCESS )
  {
    //
    // Hooray, we have a connection to the PLC and can start
    // At this point you can insert any communication functions
    //
    //

    //
    // First Steps Part 1
    //
    if( Paras.RackNr == 0 && Paras.SlotNr == 0 )
    {
      printf( "Verwende Geraet %d und PLCNr %d.\n",
              Paras.DevNr, Paras.PlcNr );
```

```
      }
      else
      {
        printf( "Verwende Geraet %d und PLCNr %d mit RackNr %d und SlotNr %d.\n",
                Paras.DevNr, Paras.PlcNr, Paras.RackNr, Paras.SlotNr );
      }
      RetVal = 0;
    }
    else
    {
      if( !Paras.Verbose )
      {
        //
        // Even if we should be quiet, this message appears at any case
        //
        printf( "Abbruch wegen Fehler %08X\n"
                "Bitte beliebige Taste zum Beenden druecken ...", RetVal );
      }
      RetVal = 1;
    }
    CloseConnection( &Paras );
    printf( "\nPlease press the any key to exit ..." );
    getch();
    return( RetVal );
}
```

Please compile and start the program. You will receive the following soft copy:

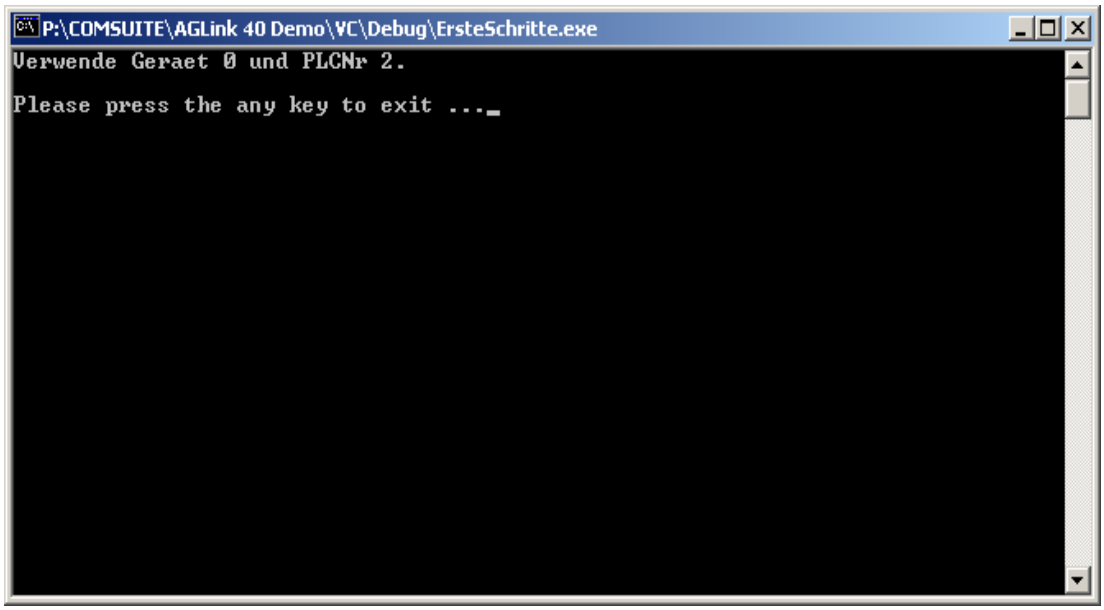


Figure 10: »ErsteSchritte.exe«, First Steps.exe

This first program example already establishes a connection to a PLC, outputs a message, releases the connection to the PLC and waits for a keypress. Now we want to breathe life into further examples. To do this we call different communication functions after the connection release. They will be explained in detail in the respective chapter.

5.1.2 Get information about the used PLC

In this chapter we want to discover to which PLC we have established a connection. Interesting is the PLC's MLFB number and which memory area is available. For this we add the following function to the module and call it after the connection establishment.

```
/*****************************************************************************

 Functio name   : ShowPLCInfo

 Parameter      : LPPROG_PARAS pParas   Structure including program parameter

 Description    : This function outputs information about the connected PLC

 Return value   : Return value of the faulty function or AGL40_SUCCESS

 Created        : 15.05.2006  RH

 Changed        : 15.05.2006  RH

 *****************************************************************************/

int ShowPLCInfo( LPPROG_PARAS pParas )
{
  int       RetVal = AGL40_SUCCESS;
  int       DevType;
  MLFB      MLFBNr;
  int       OpState;
  PLCINFO   PLCInfo;
  CYCLETIME CycleTime;
  TOD       Zeit;

  //
  // At first, we have to check if it is a S7
  //
  if( pParas->RackNr == 0 && pParas->SlotNr == 0 )
  {
    printf( "Verwende Geraet %d und PLCNr %d ueber ",
            pParas->DevNr, pParas->PlcNr );
  }
  else
  {
    printf( "Verwende Geraet %d und PLCNr %d mit RackNr %d und SlotNr %d ueber ",
            pParas->DevNr, pParas->PlcNr, pParas->RackNr, pParas->SlotNr );
  }

  DevType = AGL_GetDevType( pParas->DevNr );
  switch( DevType )
  {
    case TYPE_S7CONN_IE:
    {
      printf( "projektierte S7-Verbindung\n" );
      break;
    }
    case TYPE_S7_TCPIP:
    {
      printf( "S7-TCP/IP\n" );
      break;
    }
    case TYPE_S7_NL:
    {
      printf( "Netlink\n" );
      break;
    }
    case TYPE_S7_NLPRO:
```

```
    {
      printf( "Netlink Pro\n" );
      break;
    }
    case TYPE_S7_NLUSB:
    {
      printf( "Netlink USB\n" );
      break;
    }
    case TYPE_S7_SOFTING:
    {
      printf( "Softing Kartentreiber\n" );
      break;
    }
    case TYPE_S7_CIF:
    {
      printf( "CIF Kartentreiber\n" );
      break;
    }
    case TYPE_S7_MPI_SER:
    {
      printf( "PC-Adapter seriell\n" );
      break;
    }
    case TYPE_S7_MPI_USB:
    {
      printf( "PC-Adapter USB\n" );
      break;
    }
    case TYPE_S7_TS_AT:
    {
      printf( "AT-Modem\n" );
      break;
    }
    case TYPE_S7_TS_TAPI:
    {
      printf( "TAPI-Modem\n" );
      break;
    }
    case TYPE_S7_PCCP:
    {
      printf( "Siemens-Geraetetreiber\n" );
      break;
    }
    case TYPE_S7_PPI:
    {
      printf( "PPI-Adapter\n" );
      break;
    }
    default:
    {
      printf( "unzulaessigen Verbindungsweg (%d)!\n", DevType );
      return( AGL40_DEVICE_NOT_SUPPORTED );
    }
  }
  printf( "\n" );

  //
  // Now we can put our mind at rest and go on
  //
  RetVal = AGL_ReadMLFBNr( pParas->ConnNr, &MLFBNr, 1, 0 );
  if( RetVal == AGL40_SUCCESS )
  {
    printf( "Kommuniziere mit SPS %s\n", MLFBNr.MLFB );
    printf( "\n" );
  }
  else
  {
    if( pParas->Verbose )
    {
```

```
    ShowError( RetVal, "AGL_ReadMLFBNr" );
  }
}

RetVal = AGL_ReadOpState( pParas->ConnNr, &OpState, 1, 0 );
if( RetVal == AGL40_SUCCESS )
{
  switch( OpState )
  {
    case OPSTATE_STOP:
    {
      printf( "Die CPU ist im Stop\n" );
      break;
    }
    case OPSTATE_START:
    {
      printf( "Die CPU ist im Anlauf\n" );
      break;
    }
    case OPSTATE_RUN:
    {
      printf( "Die CPU ist im Run\n" );
      break;
    }
    default:
    case OPSTATE_UNKNOWN:
    {
      printf( "Die CPU ist in einem unbekannten Betriebszustand\n" );
      break;
    }
  }
  printf( "\n" );
}
else
{
  if( pParas->Verbose )
  {
    ShowError( RetVal, "AGL_ReadOpState" );
  }
}

RetVal = AGL_ReadPLCInfo( pParas->ConnNr, &PLCInfo, 1, 0 );
if( RetVal == AGL40_SUCCESS )
{
  printf( "Anzahl Eingangsbytes im Prozessabbild: %d\n", PLCInfo.PAE );
  printf( "Anzahl Ausgangsbytes im Prozessabbild: %d\n", PLCInfo.PAA );
  printf( "Anzahl Merkerbytes ..................: %d\n", PLCInfo.Flags );
  printf( "Anzahl Zeiten .......................: %d\n", PLCInfo.Timer );
  printf( "Anzahl Zaehler ......................: %d\n", PLCInfo.Counter );
  printf( "Groesse des Lokaldatenspeichers .....: %d\n", PLCInfo.LocalData );
  printf( "\n" );
}
else
{
  if( pParas->Verbose )
  {
    ShowError( RetVal, "AGL_ReadPLCInfo" );
  }
}

RetVal = AGL_ReadCycleTime( pParas->ConnNr, &CycleTime, 1, 0 );
if( RetVal == AGL40_SUCCESS )
{
  printf( "Die aktuelle Zyklsuzeit betraegt %d ms\n", CycleTime.AktCycleTime );
  printf( "Die minimale Zyklsuzeit betraegt %d ms\n", CycleTime.MinCycleTime );
  printf( "Die maximale Zyklsuzeit betraegt %d ms\n", CycleTime.MaxCycleTime );
  printf( "\n" );
}
else
{
```

```
    if( pParas->Verbose )
    {
      ShowError( RetVal, "AGL_ReadCycleTime" );
    }
  }

  RetVal = AGL_GetPLCClock( pParas->ConnNr, &Zeit, 1, 0 );
  if( RetVal == AGL40_SUCCESS )
  {
    //
    // We transform the BCD time into an educible format
    //
    SYSTEMTIME ST;
    AGL_TOD2SysTime( &Zeit, &ST );
    printf( "Datum auf der SPS ......: %02d.%02d.%d\n",
            ST.wDay, ST.wMonth, ST.wYear );
    printf( "Uhrzeit auf der SPS ....: %02d:%02d:%02d.%03d\n",
            ST.wHour, ST.wMinute, ST.wSecond, ST.wMilliseconds );
  }
  else
  {
    if( pParas->Verbose )
    {
      ShowError( RetVal, "AGL_GetPLCClock" );
    }
  }

  return( RetVal );
}
```

Now we change the following things in the main program:

```
  if( (RetVal = OpenConnection( &Paras )) == AGL40_SUCCESS )
  {
    //
    // Hooray, we have a connection to the PLC and can start
    // At this point you can insert any communication functions
    //

    //
    // Erste Schritte Teil 2, (First Steps Part 2)
    //
    ShowPLCInfo( &Paras );

    RetVal = 0;
  }
  else
  {
    if( !Paras.Verbose )
    {
      //
      // Even if we should be quiet, this message appears at any case
      //
      printf( "Abbruch wegen Fehler %08X\n"
              "Bitte beliebige Taste zum Beenden druecken ...", RetVal );
    }
    RetVal = 1;
  }
```

Please compile and start the program. You will receive the following soft copy:

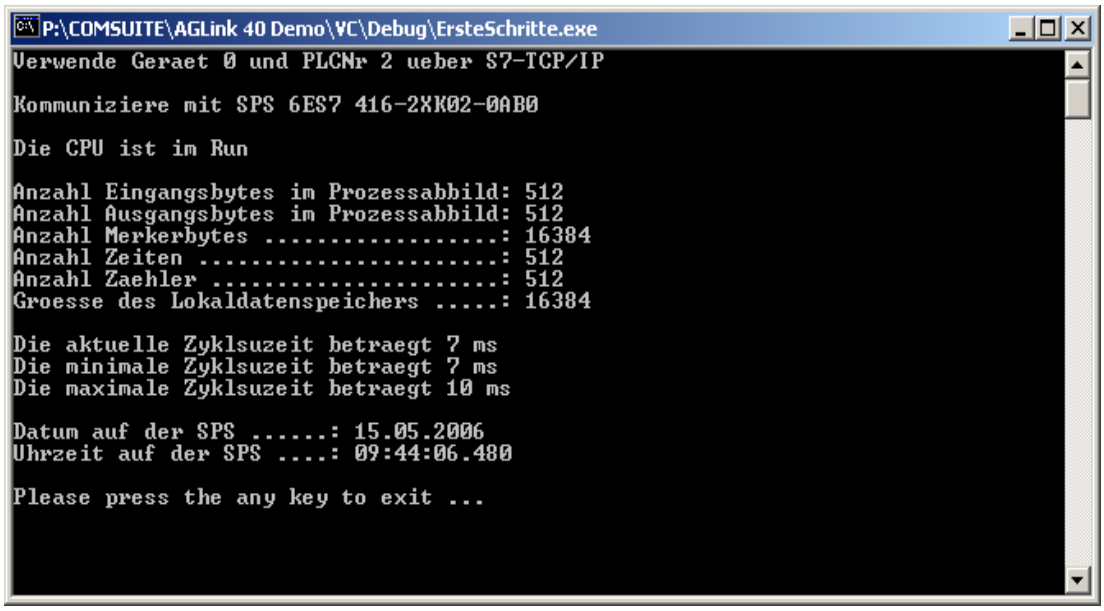


Figure 11: Extended »ErsteSchritte.exe«, First Steps.exe

5.1.3 Read out the Diagnostics Buffer

In this chapter we want to output the content of the diagnostics buffer. For this we add the following function to the module and call it after the connection establishment.

```
{*****************************************************************************

 Function name  : ShowDiagBuffer

 Parameter      : Paras : TagProgPars     Structure including program parameters

 Description    : This function outputs the content of the diagnostics buffer.

 Return value   : Return value of the faulty function or AGL40_SUCCESS

 Created        : 29.05.2006  PS

 Changed        : No change

*****************************************************************************}

Function ShowDiagBuffer(var Paras : TagProgParas) : Integer;
var
  DiagBuff  : Array[0..(8+20*30) - 1] of Byte;
  Buff1     : Array[0..255] of Char;
  Entrys, i : Integer;
begin
  //
  // There are two ways to read out the diagnostics buffer:
  // Either we inquire the amount of parametrized data, assigning the memory,
  // reading out the diagnostics buffer, output the text and release the memory.
  // Or we select how much entries are of interest to us, indicate this number
  // when reading the diagnostics buffer and output the text.
  // Each entry in the diagnostics buufer needs 20 bytes. In addition 8 bytes are
  // needed for header info. Normally, there are 120 entries, they can be changed
  // via the HW-config
  // Therefore the following method arises for the last 30 entries:
  //
  int Entrys = 30;
  BYTE DiagBuff[8+20*30];
  RetVal = AGL_ReadDiagBuffer( pParas->ConnNr, &Entrys, DiagBuff, 1, 0 );
  if( RetVal == AGL40_SUCCESS )
  {
    printf( "Anzahl gelesener Diagnosepuffereintraege: %d\n", Entrys );
    for( int i=0; i<Entrys; i++ )
    {
      char Buff1[256];

      AGL_GetDiagBufferEntry( i, DiagBuff, Buff1, sizeof( Buff1 ) );
#if defined( _CONSOLE )
      CharToOem( Buff1, Buff1 );        // Is necessary when using a console
                                        // application due to mutations
#endif
      printf( "%3d: %s\n", i+1, Buff1 );
    }
  }
  else
  {
    if( pParas->Verbose )
    {
      ShowError( RetVal, "AGL_ReadDiagBuffer" );
    }
  }
  return( RetVal );
}
```

Now we change the following things in the main program:

```
if( (RetVal = OpenConnection( &Paras )) == AGL40_SUCCESS )
{
  //
  // Hooray, we have a connection to the PLC and can start
  // At this point you can insert any communication functions
  //

  //
  // Erste Schritte Teil 3, (First Steps Part 3)
  //
  ShowDiagBuffer( &Paras );

  RetVal = 0;
}
else
{
  if( !Paras.Verbose )
  {
    //
    // Even if we should be quiet, this message appears at any case
    //
    printf( "Abbruch wegen Fehler %08X\n"
            "Bitte beliebige Taste zum Beenden druecken ...", RetVal );
  }
  RetVal = 1;
}
```

Please compile and start the program. You will receive the following soft copy.:

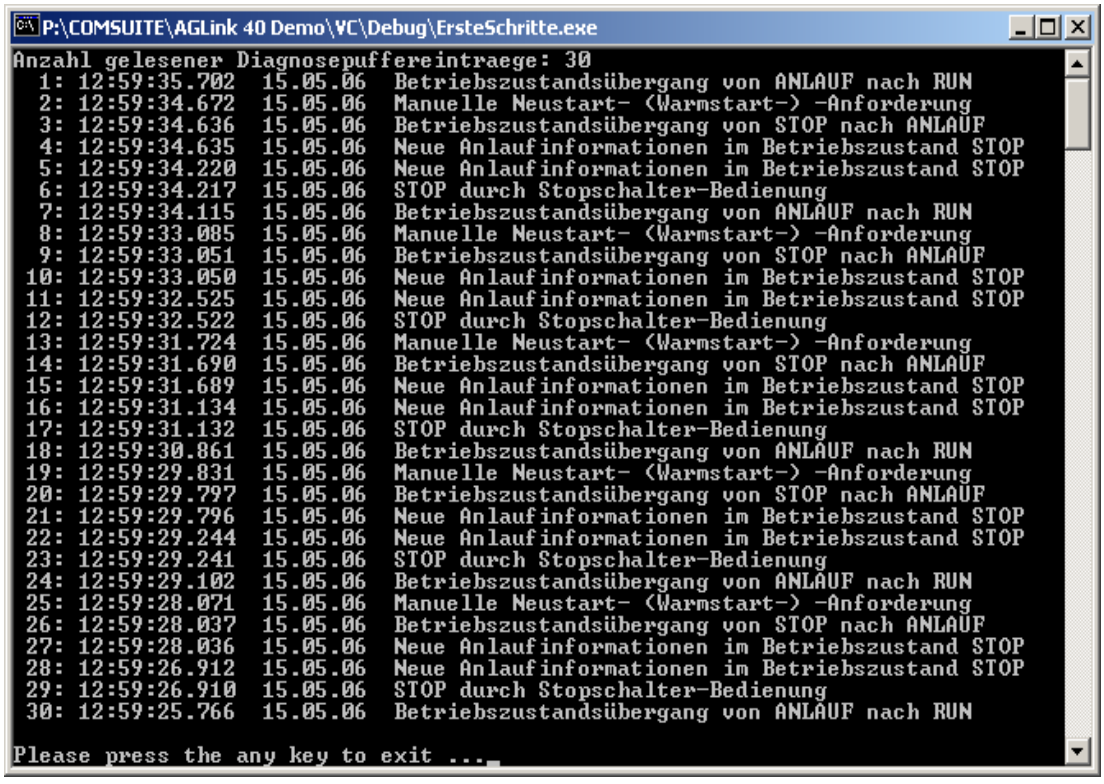


Figure 12: Read out the diagnostics buffer

5.1.4 Reading data from different sections

In this chapter we want to approach the most interesting thing in ACCON-AGLink; reading data from different sections. For this we add the following function to the module and call it after the connection establishment.

```
/*****************************************************************************

 Function name  : ShowReadMemory

 Parameter      : LPPROG_PARAS pParas   Structure including program parameters

 Description    : This function reads different memory areas with one function
                  call.

 Return value   : Return value of the faulty function or AGL40_SUCCESS

 Created        : 15.05.2006  RH

 Changed        : 15.05.2006  RH

 *****************************************************************************/

int ShowReadMemory( LPPROG_PARAS pParas )
{
  DATA_RW40 RW[8];        // We are reading here 8 different sections at most
  LPDATA_RW40 pRW;        // For always the same access
  BYTE  bBuff[8][16];     // Buffer for byte read access
  WORD  wBuff[8][8];      // Buffer for word read access
  DWORD dwBuff[8][4];     // Buffer for double word read access
  int RetVal = AGL40_SUCCESS;

  printf( "Lese Daten aus verschiedenen Bereichen in einem Aufruf...\n\n" );

  //
  // At first we are performing a AGL_ReadMixEx on different and always present
  // memory sections(E/A/M/T/Z).
  //
  pRW = &RW[0];
  pRW->OpArea = AREA_IN;      // We are reading from section PA-inputs
  pRW->OpType = TYP_BYTE;     // We are reading byte elements
  pRW->OpAnz  = 16;           // We are reading 16 elements
  pRW->DBNr   = 0;            // The DB number is assigned to 0
  pRW->Offset = 0;            // We are reading from EB 0
  pRW->BitNr  = 0;            // The bit nubmer is assigned to 0
  pRW->Buff   = bBuff[0];     // The results have to be in this section

  pRW = &RW[1];
  pRW->OpArea = AREA_OUT;     // We are reading from section PA-outputs
  pRW->OpType = TYP_BYTE;     // We are reading byte elements
  pRW->OpAnz  = 16;           // We are reading 16 elements
  pRW->DBNr   = 0;            // The DB number is assigned to 0
  pRW->Offset = 0;            // We are reading from AB 0
  pRW->BitNr  = 0;            // The bit nubmer is assigned to 0
  pRW->Buff   = bBuff[1];     // The results have to be in this section

  pRW = &RW[2];
  pRW->OpArea = AREA_FLAG;    // We are reading from section marker
  pRW->OpType = TYP_BYTE;     // We are reading byte elements
  pRW->OpAnz  = 16;           // We are reading 16 elements
  pRW->DBNr   = 0;            // The DB number is assigned to 0
  pRW->Offset = 8;            // We are reading from MB 8
  pRW->BitNr  = 0;            // The bit nubmer is assigned to 0
  pRW->Buff   = bBuff[2];     // The results have to be in this section

  pRW = &RW[3];               // And now the same word by word
  pRW->OpArea = AREA_FLAG;    // We are reading from section marker
```

```
pRW->OpType = TYP_WORD;     // We are reading word elements
pRW->OpAnz  = 8;            // We are reading 8 elements
pRW->DBNr   = 0;            // The DB number is assigned to 0
pRW->Offset = 8;            // We are reading from MB 8 !!
pRW->BitNr  = 0;            // The bit number is assigned to 0
pRW->Buff   = wBuff[3];     // The results have to be in this section

pRW = &RW[4];               // And now the same double word by double word
pRW->OpArea = AREA_FLAG;    // We are reading marker elements
pRW->OpType = TYP_DWORD;    // We are reading double word elements
pRW->OpAnz  = 4;            // We are reading 4 elements
pRW->DBNr   = 0;            // The DB number is assigned to 0
pRW->Offset = 8;            // We are reading from MB 8 !!
pRW->BitNr  = 0;            // The bit nubmer is assigned to 0
pRW->Buff   = dwBuff[4];    // The results have to be in this section

pRW = &RW[5];
pRW->OpArea = AREA_COUNTER; // We are reading from section counter
pRW->OpType = TYP_COUNTER;  // We are reading counter elements
pRW->OpAnz  = 8;            // We are reading 8 elements
pRW->DBNr   = 0;            // The DB number is assigned to 0
pRW->Offset = 0;            // We are reading from Z 0
pRW->BitNr  = 0;            // The bit nubmer is assigned to 0
pRW->Buff   = wBuff[5];     // The results have to be in this section

pRW = &RW[6];
pRW->OpArea = AREA_TIMER;   // We are reading from section timers
pRW->OpType = TYP_TIMER;    // We are reading timer elements
pRW->OpAnz  = 8;            // We are reading 8 elements
pRW->DBNr   = 0;            // The DB number is assigned to 0
pRW->Offset = 16;           // We are reading from T 16
pRW->BitNr  = 0;            // The bit nubmer is assigned to 0
pRW->Buff   = wBuff[6];     // The results have to be in this section

pRW = &RW[7];
pRW->OpArea = AREA_TIMER;   // We are reading from section timers
pRW->OpType = TYP_TIMER;    // We are reading timer elements
pRW->OpAnz  = 8;            // We are reading 8 elements
pRW->DBNr   = 0;            // The DB number is assigned to 0
pRW->Offset = 2048;         // We are reading from T 2048 => Errors occur!!
pRW->BitNr  = 0;            // The bit nubmer is assigned to 0
pRW->Buff   = wBuff[7];     // The results have to be in this section

RetVal = AGL_ReadMixEx( pParas->ConnNr, RW, 8, 1, 0 );
if( RetVal == AGL40_SUCCESS )
{
  for( int i=0; i<8; i++ )
  {
    //
    // Every structure has to be checked
    //
    if( RW[i].Result == AGL40_SUCCESS )
    {
      switch( RW[i].OpType )
      {
        case TYP_BIT:
        case TYP_BYTE:
        {
          for( int j=0; j<16; j++ )
          {
            printf( "%02X  ", bBuff[i][j] );
          }
          printf( "\n" );
          break;
        }
        case TYP_WORD:
        case TYP_COUNTER:
        case TYP_TIMER:
        {
          for( int j=0; j<8; j++ )
```

```
            {
              printf( " %04X   ", wBuff[i][j] );
            }
            printf( "\n" );
            break;
          }
          case TYP_DWORD:
          {
            for( int j=0; j<4; j++ )
            {
              printf( "   %08X     ", dwBuff[i][j] );
            }
            printf( "\n" );
            break;
          }
        }
      }
      else
      {
        ShowError( RW[i].Result, "AGL_ReadMixEx" );
      }
    }
  }
  else
  {
    if( pParas->Verbose )
    {
      ShowError( RetVal, "AGL_ReadMixEx" );
    }
  }
  printf( "\n" );

  //
  // And now we are searching for dynamic sections and reading them out (= DBs)
  //
  int DBCount;

  RetVal = AGL_ReadDBCount( pParas->ConnNr, &DBCount, 1, 0 );
  if( RetVal == AGL40_SUCCESS )
  {
    if( DBCount > 0 )
    {
      WORD DBList[32];

      printf( "%d Datenbausteine vorhanden.\n", DBCount );
      if( DBCount > sizeof(DBList)/sizeof(DBList[0]) )
      {
        //
        // Only the first modules are intersting for us
        //
        DBCount = sizeof(DBList)/sizeof(DBList[0]);
      }
      RetVal = AGL_ReadDBList( pParas->ConnNr, &DBCount, DBList, 1, 0 );
      if( RetVal == AGL40_SUCCESS )
      {
        int DBs = 0, DBLen, i;

        for( i=0; i<DBCount; i++ )
        {
          RetVal = AGL_ReadDBLen( pParas->ConnNr, DBList[i], &DBLen, 1, 0 );
          if( RetVal == AGL40_SUCCESS )
          {
            if( DBLen >= 16 )
            {
              //
              // We can read out this module
              //
              printf( "Lese 16 Bytes aus DB %d\n", DBList[i] );
              pRW = &RW[DBs];
              pRW->OpArea = AREA_DATA;    // We are reading from section data
```

```
            pRW->OpType = TYP_BYTE;     // We are reading byte elements
            pRW->OpAnz  = 16;           // We are reading 16 elements
            pRW->DBNr   = DBList[i];    // We are entering the DB number
            pRW->Offset = 0;            // We are reading from DBB 0
            pRW->BitNr  = 0;            // The bit nubmer is assigned to 0
            pRW->Buff   = bBuff[DBs];   // The results have to be in this
                                        // section
            if( ++DBs == 8 )
            {
              break;
            }
          }
        }
        else
        {
          if( pParas->Verbose )
          {
            ShowError( RetVal, "AGL_ReadDBLen" );
          }
        }
      }
      if( DBs > 0 )
      {
        RetVal = AGL_ReadMixEx( pParas->ConnNr, RW, DBs, 1, 0 );
        if( RetVal == AGL40_SUCCESS )
        {
          for( i=0; i<DBs; i++ )
          {
            //
            // Every structue has to be checked
            //
            if( RW[i].Result == AGL40_SUCCESS )
            {
              for( int j=0; j<16; j++ )
              {
                printf( "%02X  ", bBuff[i][j] );
              }
              printf( "\n" );
            }
            else
            {
              ShowError( RW[i].Result, "AGL_ReadMixEx" );
            }
          }
        }
        else
        {
          if( pParas->Verbose )
          {
            ShowError( RetVal, "AGL_ReadMixEx" );
          }
        }
      }
      else
      {
        printf( "Keine Datenbausteine für Testzwecke vorhanden\n" );
      }
    }
    else
    {
      if( pParas->Verbose )
      {
        ShowError( RetVal, "AGL_ReadDBList" );
      }
    }
  }
  else
  {
    printf( "Keine Datenbausteine für Testzwecke vorhanden\n" );
  }
```

```
  }
  else
  {
    if( pParas->Verbose )
    {
      ShowError( RetVal, "AGL_ReadDBCount" );
    }
  }
  return( RetVal );
}
```

Now we are changing the following things in the main program

```
  if( (RetVal = OpenConnection( &Paras )) == AGL40_SUCCESS )
  {
    //
    // Hooray, we have a connection to the PLC and can start
    // At this point you can insert any communication functions
    //

    //
    // Erste Schritte Teil 4, (First Steps Part 4)
    //
    ShowReadMemory( &Paras );

    RetVal = 0;
  }
  else
  {
    if( !Paras.Verbose )
    {
      //
      // Even if we should be quiet, this message appears at any case
      //
      printf( "Abbruch wegen Fehler %08X\n"
              "Bitte beliebige Taste zum Beenden druecken ...", RetVal );
    }
    RetVal = 1;
  }
```

Please compile and start the program. You will receive the following soft copy:

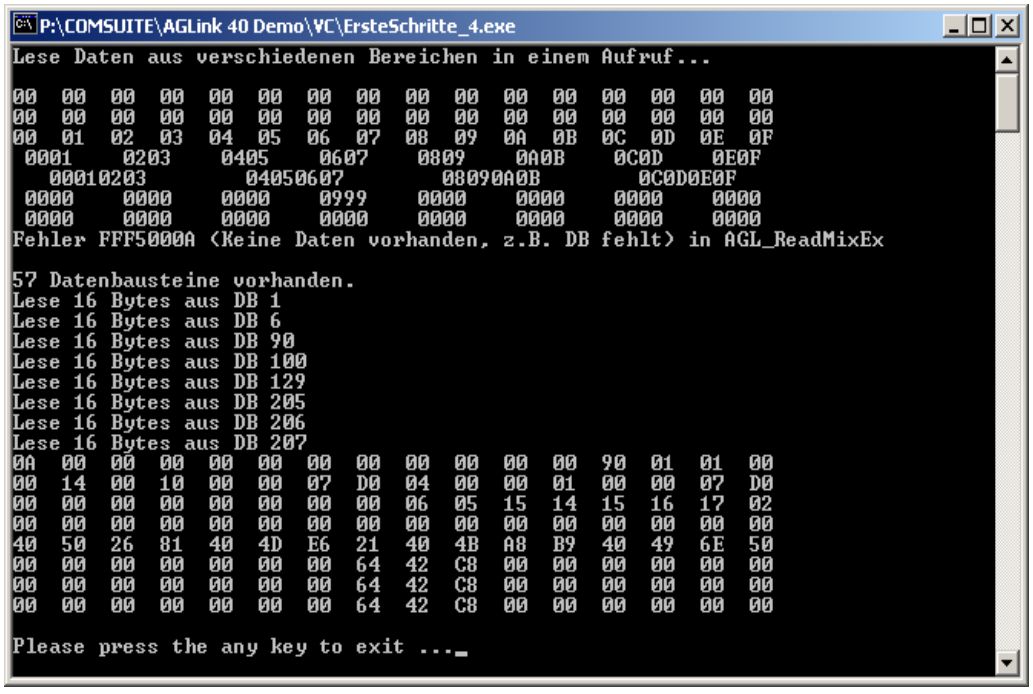


Figure 13: Reading data from different sections

5.2 First Steps with Microsoft Visual Basic

To compile and test an ACCON-AGLink40 application with Microsoft Visual basic you need the following data files:

```
AGL_Defines.bas             Module data with definitions of constants
AGL_Enums.bas               Module data with definitions of enumeration
AGL_Types.bas               Module data with definitions of types
AGL_Funcs.bas               Module data with function declarations
AGL_VBFuncs.bas             Module data with additional auxiliary functions
AGLink40.dll                The proper function library of ACCON-AGLink
                            version 4

AGLink40_Error.txt          Data with error messages in plain text
AGLink40CfgDev0000.xml      Data with the configuration data, in this case
                            from Device 0
```

Later only AGLink40.dll, AGLink40_Error.txt and AGLink40CfgDev0000.xml are necessary for the compiled program. In chapter »5 First Steps with Win32« you will find all essential data to change the configuration.

To ease all this we create a little form with just one text box. So please start Microsoft Visual Basic. Select a new project, choose »save as…« and enter the project name »ErsteSchritte» (First Steps) as well as the save folder. Confirm with OK. Name the new form »frmErsteSchritte«. Add the above BAS data files successively to the project via »Add« and »Module«.

And keep in mind that all modules and UserForms should be preferably provided with the following entry:

```
Option Explicit                 All variables have to be declared
```

By this entry you will be directly pointed to not declared variables when programming. Thus a slip of the pen has immediately an effect when compiling and not even sometime during the running time.

5.2.1 The Communication's Skeletal Structure

This chapter shows how to establish a communication to a PLC. And how to detect the connected participants from the Lifelist and pick one from it. After that a message is displayed and the connection will be closed. So this skeletal structure has all necessary tools.

Enter the following program

Note:
Naturally, the complete project is located in the ACCON-AGLink 40 demo folder for Microsoft Visual Basic.

```
Option Explicit

'*****************************************************************************
'
' Project        : New version of the AGLink library
'
' Data name      : ErsteSchritte.frm
'
' Description    : Introduction into the use of ACCON-AGLink Version 4
'
' Copyright      : (c) 1998-2007
'                  DELTALOGIC Automatisierungstechnik GmbH
'                  Stuttgarter Str. 3
'                  73525 Schwaebisch Gmuend
'                  Web : http://www.deltalogic.de
'                  Tel.: +49-7171-916120
'                  Fax : +49-7171-916220
'
' Created        : 16.05.2006  RH
'
' Changed        : 16.05.2006  RH
'
'*****************************************************************************

'*****************************************************************************
'
' Definition of Enums
'
'*****************************************************************************

Private Enum eConnState
  eNotInit = 0            ' Connection not initialized, yet
  eDevOpened              ' Device opened
  eDialedUp               ' Dial-up (if necessary) executed
  eInitAdapter            ' Adapter initialized
  eConnected              ' Connection to PLC established
End Enum


'*****************************************************************************
'
' Definition of data types
'
'*****************************************************************************

Private Type tProgParas
  DevNr     As Long       ' Number of used device
  PlcNr     As Long       ' Number of used PLC
  RackNr    As Long       ' Rack number of the PLC (0 is standard)
  SlotNr    As Long       ' Slot number of the PLC (0 is standard)
  Verbose   As Long       ' Flag if error messages should be output
  State     As eConnState ' Connection status, will be registered by
                          ' Open and CloseConnection
  ConnNr    As Long       ' Connection handle, will be registered by
                          ' OpenConnection
End Type


'*****************************************************************************
'
' Declaration of module global variables
'
'*****************************************************************************
```

```
'*****************************************************************************
'
' Implementation of the auxiliary functions
'
'*****************************************************************************

'*****************************************************************************
'
' Function name  : ShowTextCrLf
'
' Parameter      : Text As String        Text displayed
'
' Description    : This function adds the text including vbCrLf to the textbox
'
' Return value   : none
'
' Created        : 16.05.2006  RH
'
' Chasnged       : 16.05.2006  RH
'
'*****************************************************************************

Private Sub ShowTextCrLf(Text As String)
  ShowText Text & vbCrLf
End Sub


'*****************************************************************************
'
' Function name  : ShowText
'
' Parameter      : Text As String        Text displayed
'
' Description    : This function adds the text to the text bow
'
' Return value   : none
'
' Created        : 16.05.2006  RH
'
' Changed        : 16.05.2006  RH
'
'*****************************************************************************

Private Sub ShowText(Text As String)
  With Text1
    .Text = .Text & Text
    .SelStart = Len(.Text)
  End With
End Sub


'*****************************************************************************
'
' Function name  : ShowError
'
' Parameter      : ErrNr As Long         Error number
'                  Func As String        Function name as text
'
' Description    : This function displays the error text
'
' Return value   : none
'
' Created        : 16.05.2006  RH
'
' Changed        : 16.05.2006  RH
'
'*****************************************************************************

Private Sub ShowError(ByVal ErrNr As Long, Func As String)
  Dim Fehler As String
```

```
  Fehler = "Fehler " & Right$("00000000" & Hex$(ErrNr), 8) & " (" &
AGL_GetVBErrorMsg(ErrNr) & ") in " & Func
  ShowTextCrLf Fehler
End Sub


'*****************************************************************************
'
' Function name  : ShowCommandlineHelp
'
' Parameter      : InvText As String     Invalid argument
'
' Description    : This function outputs the invalid command line argument
'                  and displays the valid
'
' Return value   : none
'
' Created        : 16.05.2006  RH
'
' Changed        : 16.05.2006  RH
'
'*****************************************************************************

Private Sub ShowCommandlineHelp(InvText As String)
  Dim Text As String

  If Len(InvText) > 0 Then
    ShowTextCrLf "Unbekanntes Kommandozeilenargument: " & InvText & vbCrLf
  End If
  ShowTextCrLf "Gültige Optionen sind:" & vbCrLf & vbCrLf & App.EXEName & _
    " [-dDevNr] [-pPlcNr] [-rRackNr] [-sSlotNr] [-v+|-]" & vbCrLf & _
    "  -dDevNr    Angabe der Gerätenummer       (Standard: 0)" & vbCrLf & _
    "  -pPlcNr    Angabe der SPS-Nummer         (Standard: erste SPS aus
                                                LifeList)" & vbCrLf & _
    "  -rRackNr   Angabe der Racknummer         (Standard: ohne)" & vbCrLf & _
    "  -sSlotNr   Angabe der Slotnummer         (Standard: ohne)" & vbCrLf & _
    "  -v+        Fehlermeldungen ausgeben      (Standard)" & vbCrLf & _
    "  -v-        Fehlermeldungen nicht ausgeben"
End Sub


'*****************************************************************************
'
' Function name  : CheckCommandline
'
' Parameter      : Paras As tProgParas    Structure including program parameters
'
' Description    : This function evaluates the command line arguments
'
' Return value   : none
'
' Created        : 16.05.2006  RH
'
' Changed        : 16.05.2006  RH
'
'*****************************************************************************

Private Sub CheckCommandline(Paras As tProgParas)
  Dim CmdParas() As String
  Dim Cmd As String
  Dim I As Integer

  On Error Resume Next      ' We are careful again ;-)

  Cmd = Command$
  If Len(Cmd) > 0 Then
    CmdParas = Split(Cmd)
    For I = 0 To UBound(CmdParas)
      If Len(CmdParas(I)) > 0 Then
        If Left$(CmdParas(I), 1) = "/" Or Left$(CmdParas(I), 1) = "-" Then
```

```
          Select Case UCase$(Mid$(CmdParas(I), 2, 1))
            Case "D"
              Paras.DevNr = Val(Mid$(CmdParas(I), 3))
            Case "P"
              Paras.PlcNr = Val(Mid$(CmdParas(I), 3))
            Case "R"
              Paras.RackNr = Val(Mid$(CmdParas(I), 3))
            Case "S"
              Paras.SlotNr = Val(Mid$(CmdParas(I), 3))
            Case "V"
              If Mid$(CmdParas(I), 3, 1) = "+" Then
                Paras.Verbose = True
              ElseIf Mid$(CmdParas(I), 3, 1) = "-" Then
                Paras.Verbose = False
              End If
            Case "H", "?"
              ShowCommandlineHelp ""
            Case Else
              ShowCommandlineHelp CmdParas(I)
          End Select
        Else
          ShowCommandlineHelp CmdParas(I)
        End If
      End If
    Next
  End If
End Sub


'*****************************************************************************
'
' Function name  : OpenConnection
'
' Parameter      : Paras As tProgParas    Structure including program parameters
'
' Description    : This function establishes a connection to a PLC depending on
'                  the actual connection status. The connection establishment
'                  to the PLC is synchronal (Timeout = 1 is synchronal to the
'                  adjusted timeout values in the config).
'
' Return value   : AGL40_SUCCESS or error code
'
' Created        : 16.05.2006  RH
'
' Changed        : 16.05.2006  RH
'
'*****************************************************************************

Private Function OpenConnection(Paras As tProgParas) As AGL40_Error
  Dim RetVal As AGL40_Error

  If Paras.State = eNotInit Then
    RetVal = AGL_OpenDevice(Paras.DevNr)
    If RetVal <> eAGL40_SUCCESS Then
      If Paras.Verbose Then
        ShowError RetVal, "AGL_OpenDevice"
      End If
      OpenConnection = RetVal
      Exit Function
    End If
    Paras.State = eDevOpened
  End If
  If Paras.State = eDevOpened Then
    RetVal = AGL_DialUp(Paras.DevNr, 1, 0)
    If RetVal <> eAGL40_SUCCESS Then
      If Paras.Verbose Then
        ShowError RetVal, "AGL_DialUp"
      End If
      OpenConnection = RetVal
      Exit Function
```

```
      End If
      Paras.State = eDialedUp
    End If
    If Paras.State = eDialedUp Then
      RetVal = AGL_InitAdapter(Paras.DevNr, 1, 0)
      If RetVal <> eAGL40_SUCCESS Then
        If Paras.Verbose Then
          ShowError RetVal, "AGL_InitAdapter"
        End If
        OpenConnection = RetVal
        Exit Function
      End If
      Paras.State = eInitAdapter
    End If
    If Paras.State = eInitAdapter Then
      If Paras.PlcNr < 0 Then
        Dim LifeList(127) As Byte
        Dim I As Integer

        RetVal = AGL_GetLifelist(Paras.DevNr, LifeList(0), 1, 0)
        If RetVal <> AGL40_SUCCESS Then
          If Paras.Verbose Then
            ShowError RetVal, "AGL_GetLifeList"
          End If
          OpenConnection = RetVal
          Exit Function
        End If
        For I = 0 To 126
          If LifeList(I) = LL_ACTIVE Or LifeList(I) = LL_PASSIVE Then
            Paras.PlcNr = I
            Exit For
          End If
        Next
        If Paras.PlcNr < 0 Then
          If Paras.Verbose Then
            ShowError eAGL40_PLC_NOT_FOUND, "AGL_GetLifeList"
          End If
          OpenConnection = eAGL40_PLC_NOT_FOUND
          Exit Function
        End If
      End If
      If Paras.RackNr = 0 And Paras.SlotNr = 0 Then
        '
        ' We are establishing the connection with standard values
        '
        RetVal = AGL_PLCConnect(Paras.DevNr, Paras.PlcNr, Paras.ConnNr, 1, 0)
      Else
        '
        ' We directly access the PLC on the RackNr and SlotNr
        '
        RetVal = AGL_PLCConnectEx(Paras.DevNr, Paras.PlcNr, Paras.RackNr,
  Paras.SlotNr, Paras.ConnNr, 1, 0)
      End If
      If RetVal <> AGL40_SUCCESS Then
        If Paras.Verbose Then
          ShowError RetVal, "AGL_PLCConnect/AGL_PLCConnectEx"
        End If
        OpenConnection = RetVal
        Exit Function
      End If
      Paras.State = eConnected
    End If
    OpenConnection = eAGL40_SUCCESS
  End Function
```

```
'*****************************************************************************
'
' Function name  : CloseConnection
'
' Parameter      : Paras As tProgParas    Structure including program parameters
'
' Description    : This function releases the connection to a PLC and closes the
'                  Device depending on the acutal connection status. The
'                  connection establishment to the PLC is synchronal
                   (Timeout = 1 is synchronal to the adjusted timeout
                   values in the config).
'
' Return value   : always AGL40_SUCCESS
'
' Created        : 16.05.2006  RH
'
' Changed        : 16.05.2006  RH
'
'*****************************************************************************

Private Function CloseConnection(Paras As tProgParas) As AGL40_Error
  Dim RetVal As Long

  If Paras.State = eConnected Then
    RetVal = AGL_PLCDisconnect(Paras.ConnNr, 1, 0)
    If RetVal <> AGL40_SUCCESS And Paras.Verbose <> False Then
      ShowError RetVal, "AGL_PLCDisconnect"
    End If
    Paras.State = eInitAdapter
  End If
  If Paras.State = eInitAdapter Then
    RetVal = AGL_ExitAdapter(Paras.DevNr, 1, 0)
    If RetVal <> AGL40_SUCCESS And Paras.Verbose <> False Then
      ShowError RetVal, "AGL_ExitAdapter"
    End If
    Paras.State = eDialedUp
  End If
  If Paras.State = eDialedUp Then
    RetVal = AGL_HangUp(Paras.DevNr, 1, 0)
    If RetVal <> AGL40_SUCCESS And Paras.Verbose <> False Then
      ShowError RetVal, "AGL_HangUp"
    End If
    Paras.State = eDevOpened
  End If
  If Paras.State = eDevOpened Then
    RetVal = AGL_CloseDevice(Paras.DevNr)
    If RetVal <> AGL40_SUCCESS And Paras.Verbose <> False Then
      ShowError RetVal, "AGL_CloseDevice"
    End If
    Paras.State = eNotInit
  End If
  CloseConnection = eAGL40_SUCCESS
End Function
```

```
'*************************************************************************
'
' Implementation of the proper auxiliary function
'
'*************************************************************************

Private Sub Form_Load()
  Dim RetVal As AGL40_Error
  Dim Paras As tProgParas

  Text1.Locked = True       ' No output handling possible
  KeyPreview = True         ' Keypresses first to the form then to the control

  '
  ' Initializing parameters with standard values
  '
  Paras.DevNr = 0           ' We are using Device 0
  Paras.PlcNr = -1          ' We are using the first PLC from the Lifelist
  Paras.RackNr = 0          ' We are working without...
  Paras.SlotNr = 0          ' ... extended addressing
  Paras.Verbose = True      ' We want to see error messages in any case

  Paras.ConnNr = 0          ' Connection handle will be entered later
  Paras.State = eNotInit    ' Nothing is happening, yet

  '
  ' When using a developer's version please enter your license key
  '
  AGL_Activate "123456-1234-123456"

  CheckCommandline Paras
  RetVal = OpenConnection(Paras)
  If RetVal = eAGL40_SUCCESS Then
    '
    // Hooray, we have a connection to the PLC and can start
    // At this point you can insert any communication functions
    '

    '
    ' Erste Schritte Teil 1, (First Steps Part 1)
    '
    If Paras.RackNr = 0 And Paras.SlotNr = 0 Then
      ShowTextCrLf "Verwende Gerät " & Paras.DevNr & " und PLCNr " & Paras.PlcNr
    Else
      ShowTextCrLf "Verwende Gerät " & Paras.DevNr & " und PLCNr " & Paras.PlcNr
& _
                   " mit RackNr " & Paras.RackNr & " und SlotNr " & Paras.SlotNr
    End If
  Else
    If Paras.Verbose = False Then
      '
      ' Even if we should be quiet, this message appears at any case
      '
      ShowTextCrLf "Abbruch wegen Fehler " & Right$("00000000" & Hex$(RetVal), 8)
    End If
  End If
  CloseConnection Paras
  ShowText vbCrLf & "Bitte <ESC> zum Beenden drücken ..."
End Sub
```

```
'*************************************************************************
'
' Implementation of the needed auxiliary event functions
'
'*************************************************************************

Private Sub Form_KeyPress(KeyAscii As Integer)
  If KeyAscii = 27 Then     ' We quit when pressing Escape
    Unload Me
  End If
End Sub

Private Sub Form_Resize()
  '
  ' We always fill out the complete form with a text box
  '
  Text1.Move 0, 0, ScaleWidth, ScaleHeight
End Sub
```

Please start the program wirh »F5«. You will receive the following soft copy:

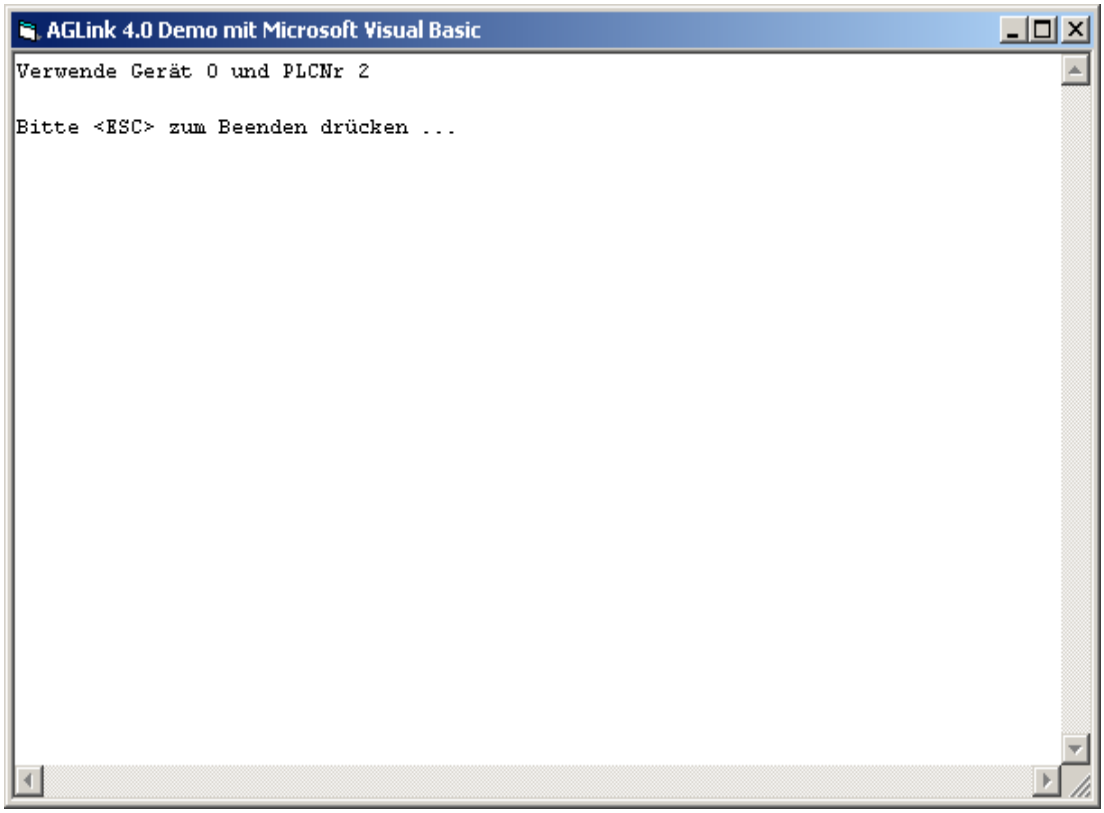


Figure 14: AGLink 4.0 Microsoft Visual Basi

This first program example already establishes a connection to a PLC, outputs a message, releases the connection to the PLC and waits for a keypress. Now we want to breathe life into further examples. To do this we call different communication functions after the connection release. They will be explained in detail in the respective chapter.

5.2.2 Get information about the used PLC

In this chapter we want to discover to which PLC we have established a connection. Interesting is the PLC's MLFB number and which memory area is available. For this we add the following function to the module and call it after the connection establishment.

```
'************************************************************************
'
' Function name  : ShowPLCInfo
'
' Parameter      : Paras As tProgParas    Structure including program parameters
'
' Description    : This function outputs information about the connected PLC
'
' Return value   : AGL40_SUCCESS or the error message
'
' Created        : 16.05.2006  RH
'
' Changed        : 16.05.2006  RH
'
'************************************************************************

Private Function ShowPLCInfo(Paras As tProgParas) As AGL40_Error
  Dim RetVal  As AGL40_Error
  Dim DevType As Long
  Dim MLFBNr  As MLFBT
  Dim OpState As Long
  Dim PLCInfo As PLCInfo
  Dim CycleT  As CYCLETIME
  Dim Zeit    As TOD

  '
  ' At first, we have to check if it is a S7
  '
  If Paras.RackNr = 0 And Paras.SlotNr = 0 Then
    ShowText "Verwende Gerät " & Paras.DevNr & " und PLCNr " & Paras.PlcNr & "
über "
  Else
    ShowText "Verwende Gerät " & Paras.DevNr & " und PLCNr " & Paras.PlcNr & _
             " mit RackNr " & Paras.RackNr & " und SlotNr " & Paras.SlotNr & "
über "
  End If

  DevType = AGL_GetDevType(Paras.DevNr)
  Select Case DevType
    Case TYPE_S7CONN_IE
      ShowTextCrLf "projektierte S7-Verbindung"
    Case TYPE_S7_TCPIP:
      ShowTextCrLf "S7-TCP/IP"
    Case TYPE_S7_NL:
      ShowTextCrLf "Netlink"
    Case TYPE_S7_NLPRO:
      ShowTextCrLf "Netlink Pro"
    Case TYPE_S7_NLUSB:
      ShowTextCrLf "Netlink USB"
    Case TYPE_S7_SOFTING:
      ShowTextCrLf "Softing Kartentreiber"
    Case TYPE_S7_CIF:
      ShowTextCrLf "CIF Kartentreiber"
    Case TYPE_S7_MPI_SER:
      ShowTextCrLf "PC-Adapter seriell"
    Case TYPE_S7_MPI_USB:
      ShowTextCrLf "PC-Adapter USB"
    Case TYPE_S7_TS_AT:
      ShowTextCrLf "AT-Modem"
```

```
    Case TYPE_S7_TS_TAPI:
      ShowTextCrLf "TAPI-Modem"
    Case TYPE_S7_PCCP:
      ShowTextCrLf "Siemens-Geraetetreiber"
    Case TYPE_S7_PPI:
      ShowTextCrLf "PPI-Adapter"
    Case Else:
      ShowTextCrLf "unzulässigen Verbindungsweg (" & DevType & ")!"
      ShowPLCInfo = eAGL40_DEVICE_NOT_SUPPORTED
      Exit Function
  End Select
  ShowText vbCrLf

  '
  ' Now we can put our mind at rest and go on
  '
  RetVal = AGL_ReadMLFBNr(Paras.ConnNr, MLFBNr, 1, 0)
  If RetVal = eAGL40_SUCCESS Then
    ShowTextCrLf "Kommuniziere mit SPS " & Left$(MLFBNr.MLFB, 20)
    ShowText vbCrLf
  Else
    If Paras.Verbose <> False Then
      ShowError RetVal, "AGL_ReadMLFBNr"
    End If
  End If

  RetVal = AGL_ReadOpState(Paras.ConnNr, OpState, 1, 0)
  If RetVal = eAGL40_SUCCESS Then
    Select Case OpState
      Case OPSTATE_STOP:
        ShowTextCrLf "Die CPU ist im Stop"
      Case OPSTATE_START:
        ShowTextCrLf "Die CPU ist im Anlauf"
      Case OPSTATE_RUN:
        ShowTextCrLf "Die CPU ist im Run"
      Case Else:
        ShowTextCrLf "Die CPU ist in einem unbekannten Betriebszustand"
    End Select
    ShowText vbCrLf
  Else
    If Paras.Verbose <> False Then
      ShowError RetVal, "AGL_ReadOpState"
    End If
  End If

  RetVal = AGL_ReadPLCInfo(Paras.ConnNr, PLCInfo, 1, 0)
  If RetVal = eAGL40_SUCCESS Then
    ShowTextCrLf "Anzahl Eingangsbytes im Prozessabbild: " & PLCInfo.PAE
    ShowTextCrLf "Anzahl Ausgangsbytes im Prozessabbild: " & PLCInfo.PAA
    ShowTextCrLf "Anzahl Merkerbytes ...................: " & PLCInfo.Flags
    ShowTextCrLf "Anzahl Zeiten ........................: " & PLCInfo.Timer
    ShowTextCrLf "Anzahl Zaehler .......................: " & PLCInfo.Counter
    ShowTextCrLf "Größe des Lokaldatenspeichers .......: " & PLCInfo.LocalData
    ShowText vbCrLf
  Else
    If Paras.Verbose <> False Then
      ShowError RetVal, "AGL_ReadPLCInfo"
    End If
  End If

  RetVal = AGL_ReadCycleTime(Paras.ConnNr, CycleT, 1, 0)
  If RetVal = eAGL40_SUCCESS Then
    ShowTextCrLf "Die aktuelle Zyklsuzeit betraegt " & CycleT.AktCycleTime & "
ms"
    ShowTextCrLf "Die minimale Zyklsuzeit betraegt " & CycleT.MinCycleTime & "
ms"
    ShowTextCrLf "Die maximale Zyklsuzeit betraegt " & CycleT.MaxCycleTime & "
ms"
    ShowText vbCrLf
  Else
```

```
    If Paras.Verbose <> False Then
      ShowError RetVal, "AGL_ReadCycleTime"
    End If
  End If

  RetVal = AGL_GetPLCClock(Paras.ConnNr, Zeit, 1, 0)
  If RetVal = eAGL40_SUCCESS Then
    '
    ' We transform the BCD time into an educible format
    '
    Dim ST As SYSTEMTIME
    AGL_TOD2SysTime Zeit, ST
    ShowTextCrLf "Datum auf der SPS ......: " & _
                 Format$(ST.wDay, "00") & "." & _
                 Format$(ST.wMonth, "00") & "." & _
                 Format$(ST.wYear, "0000")
    ShowTextCrLf "Uhrzeit auf der SPS ....: " & _
                 Format$(ST.wHour, "00") & ":" & _
                 Format$(ST.wMinute, "00") & ":" & _
                 Format$(ST.wSecond, "00") & "." & _
                 Format$(ST.wMilliseconds, "000")
  Else
    If Paras.Verbose <> False Then
      ShowError RetVal, "AGL_GetPLCClock"
    End If
  End If

  ShowPLCInfo = RetVal
End Function
```

Now we change the following things in the main progam:

```
  RetVal = OpenConnection(Paras)
  If RetVal = eAGL40_SUCCESS Then
    '
    ' Hooray, we have a connection to the PLC and can start
    ' At this point you can insert any communication functions
    '

    '
    ' Erste Schritte Teil 2, (First Steps Part 2)
    '
    ShowPLCInfo Paras

  Else
    If Paras.Verbose = False Then
      '
      ' Even if we should be quiet, this message appears at any case
      '
      ShowTextCrLf "Abbruch wegen Fehler " & Right$("00000000" & Hex$(RetVal), 8)
    End If
  End If
```

Please start the programm with »F5«. You will receive the following soft copy:

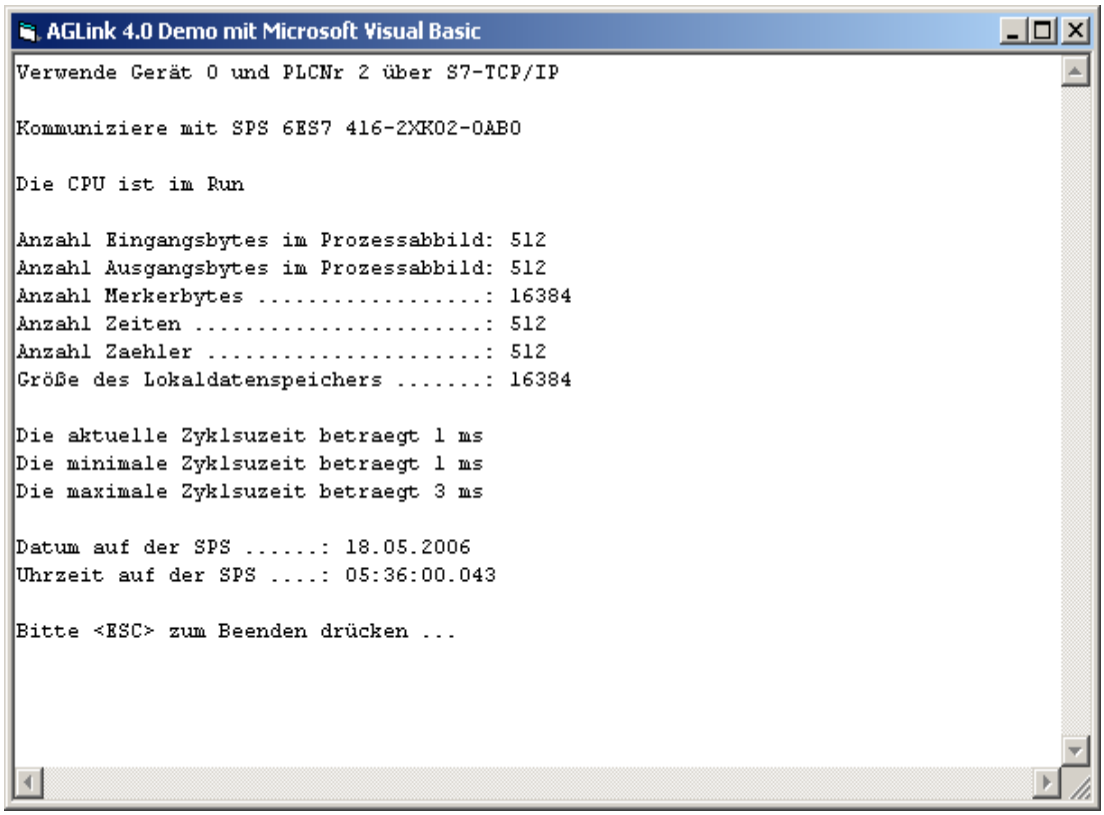


Figure 15: AGLink 4.0 Microsoft Visual Basic

5.2.3 Read out the diagnostics buffer

In this chapter we want to output the content of the diagnostics buffer. For this we add the following function into the module and call it after the connection establishment.

```
'************************************************************************
'
' Function name  : ShowDiagBuffer
'
' Parameter      : Paras As tProgParas    Structure including program parameters
'
' Description    : This function outputs the content of the diagnostics buffer
'
' Return value   : AGL40_SUCCESS or the error message
'
' Created        : 16.05.2006  RH
'
' Changed        : 16.05.2006  RH
'
'************************************************************************

Private Function ShowDiagBuffer(Paras As tProgParas) As AGL40_Error
  Dim RetVal  As AGL40_Error

  '
  // There are two ways to read out the diagnostics buffer:
  // Either we inquire the amount of parametrized data, assigning the memory,
  // reading out the diagnostics buffer, output the text and release the memory.
  // Or we select how much entries are of interest to us, indicate this number
  // when reading the diagnostics buffer and output the text.
  // Each entry in the diagnostics buufer needs 20 bytes. In addition 8 bytes are
  // needed for header info. Normally, there are 120 entries, they can be changed
  // via the HW-config
  // Therefore the following method arises for the last 30 entries:
  '
  Dim Entrys As Long, I As Long
  Dim DiagBuff(8 + 20 * 30 - 1) As Byte ' VB assigend up to the index but not
                                        ' with this size, so - 1

  Entrys = 30
  RetVal = AGL_ReadDiagBuffer(Paras.ConnNr, Entrys, DiagBuff(0), 1, 0)
  If RetVal = AGL40_SUCCESS Then
    ShowTextCrLf "Anzahl gelesener Diagnosepuffereinträge: " & Entrys
    For I = 0 To Entrys - 1
      ShowTextCrLf Format$(I + 1, "000:  ") & AGL_GetVBDiagBufferEntry(I,
DiagBuff(0))
    Next
  Else
    If Paras.Verbose <> False Then
      ShowError RetVal, "AGL_ReadDiagBuffer"
    End If
  End If
  ShowDiagBuffer = RetVal
End Function
```

Now we change the following things in the main progam:

```
  RetVal = OpenConnection(Paras)
  If RetVal = eAGL40_SUCCESS Then
    '
    // Hooray, we have a connection to the PLC and can start
    // At this point you can insert any communication functions
    '
```

```
    '
    ' Erste Schritte Teil 3, (First Steps Part 3)
    '
    ShowDiagBuffer Paras

  Else
    If Paras.Verbose = False Then
      '
      ' Even if we should be quiet, this message appears at any case
      '
      ShowTextCrLf "Abbruch wegen Fehler " & Right$("00000000" & Hex$(RetVal), 8)
    End If
  End If
```

Please start the programm with »F5«. You will receive the following soft copy:

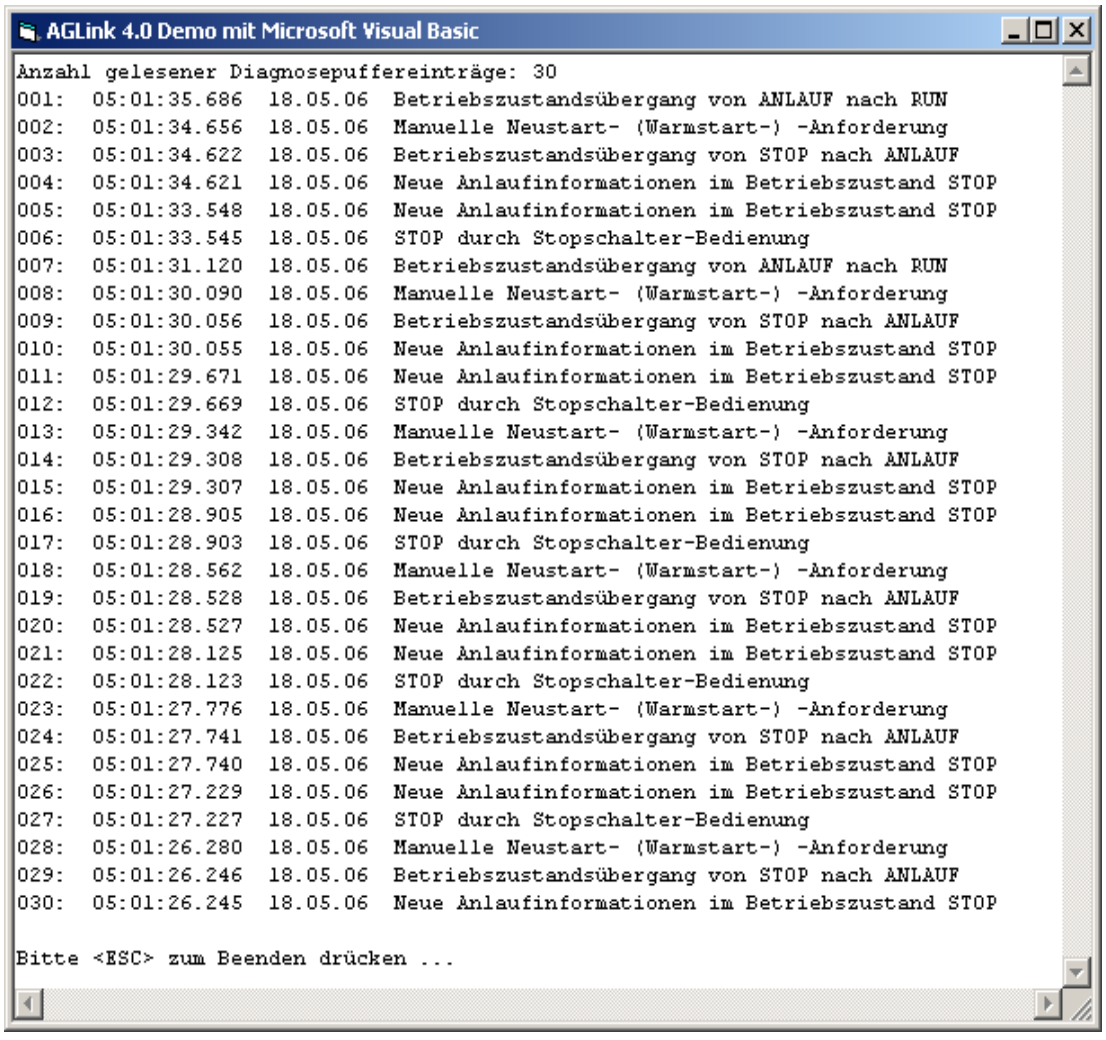


Figure 16: AGLink 4.0 Microsoft Visual Basic

5.2.4 Reading data from different sections

In this chapter we want to approach the most interesting thing in ACCON-AGLink; reading data from different sections. For this we add the following function to the module and call it after the connection establishment.

```
'*****************************************************************************
'
' Function name  : ShowReadMemory
'
' Parameter      : Paras As tProgParas    Structure including program parameters
'
' Description    : This function outputs the content of the diagnostics buffer
'
' Return value   : AGL40_SUCCESS or error code
'
' Created        : 16.05.2006  RH
'
' Changed        : 16.05.2006  RH
'
'*****************************************************************************

Private Function ShowReadMemory(Paras As tProgParas) As AGL40_Error
  Dim RetVal As AGL40_Error
  Dim i As Integer, j As Integer
  Dim RW(7) As DATA_RW40      ' We are reading here a maximum of 8 different
                                sections
  '
  ' Attention:  In VB the array indices are arranged backwards to C/C++
  '             To achieve the same result in a AGLink call using a
                multidimensional array you have to swap the barriers. As well as
                the other dimension when accessing an array.
  ' Attention   In VB arrays are dimensioned with the maximum index and not with
  '             the array size like in C/C++.
  '
  Dim bBuff(15, 7) As Byte    ' Buffer for byte read access
  Dim wBuff(7, 7) As Integer  ' Buffer for word read access
  Dim dwBuff(3, 7) As Long    ' Buffer for double word read access

  ShowTextCrLf "Lese Daten aus verschiedenen Bereichen in einem Aufruf..." &
vbCrLf

  '
  // At first we are performing a AGL_ReadMixEx on different and always present
  // memory sections(E/A/M/T/Z).
  '
  With RW(0)
    .OpArea = AREA_IN             ' We are reading from section PA inputs
    .OpType = TYP_BYTE            ' We are reading byte elements
    .OpAnz = 16                   ' We are reading 16 elements
    .DBNr = 0                     ' The DB number is assigned to 0
    .Offset = 0                   ' We are reading from EB 0
    .BitNr = 0                    ' The bit number is assigned to 0
    .Value = VarPtr(bBuff(0, 0))  ' The results have to be in this section
  End With

  With RW(1)
    .OpArea = AREA_OUT            ' Wir lesen aus dem Bereich PA-Ausgänge
    .OpType = TYP_BYTE            ' We are reading byte elements
    .OpAnz = 16                   ' We are reading 16 elements
    .DBNr = 0                     ' The DB number is assigned to 0
    .Offset = 0                   ' We are reading from AB 0
    .BitNr = 0                    ' The bit number is assigned to 0
    .Value = VarPtr(bBuff(0, 1))  ' The results have to be in this section
  End With

  With RW(2)
```

```
  .OpArea = AREA_FLAG           ' We are reading from section marker
  .OpType = TYP_BYTE            ' We are reading byte elements
  .OpAnz = 16                   ' We are reading 16 elements
  .DBNr = 0                     ' The DB number is assigned to 0
  .Offset = 8                   ' We are reading from MB 8
  .BitNr = 0                    ' The bit number is assigned to 0
  .Value = VarPtr(bBuff(0, 2))  ' The results have to be in this section
End With

With RW(3)
  .OpArea = AREA_FLAG           ' We are reading from section marker
  .OpType = TYP_WORD            ' We are reading word elements
  .OpAnz = 8                    ' We are reading 8 elements
  .DBNr = 0                     ' The DB number is assigned to 0
  .Offset = 8                   ' We are reading from MB 8
  .BitNr = 0                    ' The bit number is assigned to 0
  .Value = VarPtr(wBuff(0, 3))  ' The results have to be in this section
End With

With RW(4)
  .OpArea = AREA_FLAG           ' We are reading from section marker
  .OpType = TYP_DWORD           ' We are reading double word elements
  .OpAnz = 4                    ' We are reading 4 elements
  .DBNr = 0                     ' The DB number is assigned to 0
  .Offset = 8                   ' We are reading from MB 8
  .BitNr = 0                    ' The bit number is assigned to 0
  .Value = VarPtr(dwBuff(0, 4)) ' The results have to be in this section
End With

With RW(5)
  .OpArea = AREA_COUNTER        ' We are reading from section counter
  .OpType = TYP_COUNTER         ' We are reading counter elements
  .OpAnz = 8                    ' We are reading 8 elements
  .DBNr = 0                     ' The DB number is assigned to 0
  .Offset = 0                   ' We are reading from Z 0
  .BitNr = 0                    ' The bit number is assigned to 0
  .Value = VarPtr(wBuff(0, 5))  ' The results have to be in this section
End With

With RW(6)
  .OpArea = AREA_TIMER          ' We are reading from section timers
  .OpType = TYP_TIMER           ' We are reading timer elements
  .OpAnz = 8                    ' We are reading 8 elements
  .DBNr = 0                     ' The DB number is assigned to 0
  .Offset = 16                  ' We are reading from T 16
  .BitNr = 0                    ' The bit number is assigned to 0
  .Value = VarPtr(wBuff(0, 6))  ' The results have to be in this section
End With

With RW(7)
  .OpArea = AREA_TIMER          ' We are reading from section timers
  .OpType = TYP_TIMER           ' We are reading timer elements
  .OpAnz = 8                    ' We are reading 8 elements
  .DBNr = 0                     ' The DB number is assigned to 0
  .Offset = 2048                ' We are reading from T 2048=> Errors occur!!
  .BitNr = 0                    ' The bit number is assigned to 0
  .Value = VarPtr(wBuff(0, 7))  ' The results have to be in this section
End With

RetVal = AGL_ReadMixEx(Paras.ConnNr, RW(0), 8, 1, 0)
If RetVal = AGL40_SUCCESS Then
  For i = 0 To 7
    '
    ' Every structure has to be checked
    '
    If RW(i).Result = eAGL40_SUCCESS Then
      Select Case RW(i).OpType
        Case TYP_BIT, TYP_BYTE:
          For j = 0 To 15
            ShowText AGL_Hex(bBuff(j, i), 2) & "  "
```

```
          Next
          ShowText vbCrLf
        Case TYP_WORD, TYP_COUNTER, TYP_TIMER:
          For j = 0 To 7
            ShowText " " & AGL_Hex(wBuff(j, i), 4) & "   "
          Next
          ShowText vbCrLf
        Case TYP_DWORD:
          For j = 0 To 3
            ShowText "   " & AGL_Hex(dwBuff(j, i), 8) & "     "
          Next
          ShowText vbCrLf
      End Select

    Else
      ShowError RW(i).Result, "AGL_ReadMixEx"
    End If
  Next
Else
  If Paras.Verbose <> False Then
    ShowError RetVal, "AGL_ReadMixEx"
  End If
End If
ShowText vbCrLf

'
' And now we are searching for dynamic sections and reading them out (= DBs)
'
Dim DBCount As Long

RetVal = AGL_ReadDBCount(Paras.ConnNr, DBCount, 1, 0)
If RetVal = eAGL40_SUCCESS Then
  If DBCount > 0 Then
    Dim DBList(31) As Integer

    ShowTextCrLf DBCount & " Datenbausteine vorhanden."
    If DBCount > UBound(DBList) Then
      '
      ' Only the first modules are intersting for us
      '
      DBCount = UBound(DBList)
    End If
    RetVal = AGL_ReadDBList(Paras.ConnNr, DBCount, DBList(0), 1, 0)
    If RetVal = eAGL40_SUCCESS Then
      Dim DBs As Long, DBLen As Long

      For i = 0 To DBCount - 1
        RetVal = AGL_ReadDBLen(Paras.ConnNr, DBList(i), DBLen, 1, 0)
        If RetVal = eAGL40_SUCCESS Then
          If DBLen >= 16 Then
            '
            ' We can read out this module
            '
            ShowTextCrLf "Lese 16 Bytes aus DB " & DBList(i)
            With RW(DBs)
              .OpArea = AREA_DATA             ' We are reading from section
                                              ' data
              .OpType = TYP_BYTE              ' We are reading byte elements
              .OpAnz = 16                     ' We are reading 16 elements
              .DBNr = DBList(i)               ' We are entering the DB number
              .Offset = 0                     ' We are reading from DBB 0
              .BitNr = 0                      ' The bit number is assigned to 0
              .Value = VarPtr(bBuff(0, DBs))  ' The results are here
            End With
            DBs = DBs + 1
            If DBs = 8 Then
              Exit For
            End If
          End If
        Else
```

```
            If Paras.Verbose <> False Then
              ShowError RetVal, "AGL_ReadDBLen"
            End If
          End If
        Next
        If DBs > 0 Then
          RetVal = AGL_ReadMixEx(Paras.ConnNr, RW(0), DBs, 1, 0)
          If RetVal = eAGL40_SUCCESS Then
            For i = 0 To DBs - 1
              '
              ' Every structue has to be checked
              '
              If RW(i).Result = eAGL40_SUCCESS Then
                For j = 0 To 15
                  ShowText AGL_Hex(bBuff(j, i), 2) & "  "
                Next
                ShowText vbCrLf
              Else
                ShowError RW(i).Result, "AGL_ReadMixEx"
              End If
            Next
          Else
            If Paras.Verbose <> False Then
              ShowError RetVal, "AGL_ReadMixEx"
            End If
          End If
        Else
          ShowTextCrLf "Keine Datenbausteine für Testzwecke vorhanden"
        End If
      Else
        If Paras.Verbose <> False Then
          ShowError RetVal, "AGL_ReadDBList"
        End If
      End If
    Else
      ShowTextCrLf "Keine Datenbausteine für Testzwecke vorhanden"
    End If
  Else
    If Paras.Verbose <> False Then
      ShowError RetVal, "AGL_ReadDBCount"
    End If
  End If

  ShowReadMemory = RetVal
End Function
```

Now we are changing the following things in the main program:

```
  RetVal = OpenConnection(Paras)
  If RetVal = eAGL40_SUCCESS Then
    '
    // Hooray, we have a connection to the PLC and can start
    // At this point you can insert any communication functions
    '

    '
    ' Erste Schritte Teil 4, (First Steps Part 4)
    '
    ShowReadMemory Paras

  Else
    If Paras.Verbose = False Then
      '
      ' Even if we should be quiet, this message appears at any case
      '
      ShowTextCrLf "Abbruch wegen Fehler " & Right$("00000000" & Hex$(RetVal), 8)
    End If
  End If
```

Please start the program with »F5«. You will receive the following soft copy:

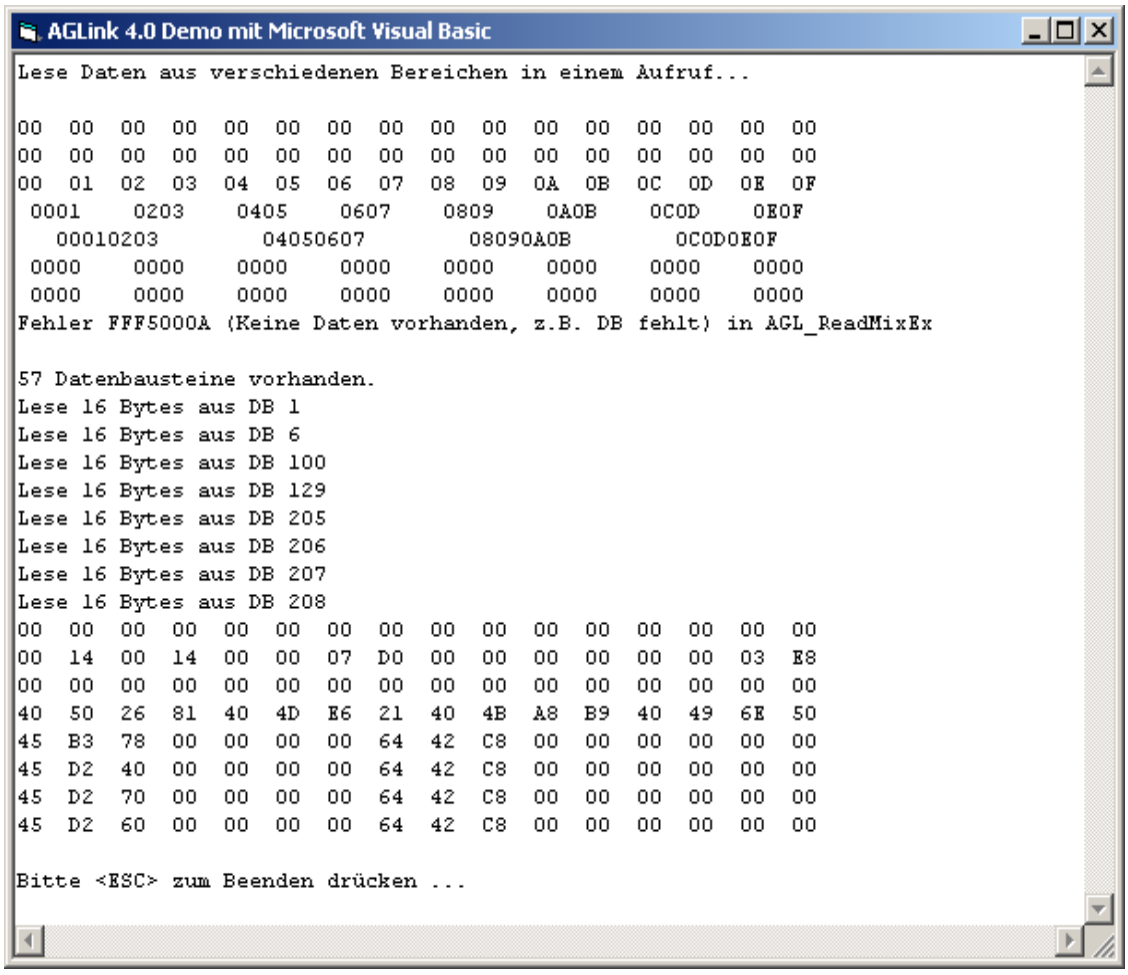


Figure 17: AGLink 4.0 Microsoft Visual Basic

5.3 First steps with Borland Delphi (console application)

We need the following data for the creation and the test of a ACCON-AGLink40 application with Borland Delphi:

```
AGLink40.PAS                Basic header data AGLink40.h integrated
AGLink40.dll                The proper function library of ACCON-AGLink
                            version 4
AGLink40_Error.txt          Data file with error messages in plain text
AGLink40CfgDev0000.xml      Data file with the configuration data, in this
                            case from Device 0
```

Later only AGLink40.dll, AGLink40_Error.txt and AGLink40CfgDev0000.xml are necessary for the compiled program. In chapter »5 First Steps with Win32« you will find all essential data to change the configuration.

To ease all this we now compile a little console application. Start Borland Delphi and create a new console project.

5.3.1 The Communication's Skeletal Structure

This chapter shows how to establish a communication to a PLC. And how to detect the connected participants from the Lifelist and pick one from it. After that a message is displayed and the connection will be closed. So this skeletal structure has all necessary tools.

Add the source code into the »ErsteSchritte DPR» FirstSteps data file.

Note:
Naturally, the complete project is located in the ACCON-AGLink 40 demo folder for Microsoft Visual Basic.

```
{*****************************************************************************

 Project        : New version of the AGLink library

 Data name      : ErsteSchritte.Pas

 Description    : Introduction into the use of ACCON-AGLink version 4

 Copyright      : (c) 1998-2007
                  DELTALOGIC Automatisierungstechnik GmbH
                  Stuttgarter Str. 3
                  73525 Schwaebisch Gmuend
                  Web : http://www.deltalogic.de
                  Tel.: +49-7171-916120
                  Fax : +49-7171-916220

 Created        : 29.05.2006  PS

 Changed        : 06.06.2006  RH

*****************************************************************************}

program ErsteSchritte;

{$APPTYPE CONSOLE}
{$X+}
uses
  SysUtils,
  Windows,
  AGLink40;         // It is absolutely necessary to integrate the DLL-Wrapper

{*****************************************************************************

 Definition of constants

*****************************************************************************}

Const
  eNotInit     = 0;         // Connection not initialized, yet
  eDevOpened   = 1;         // Device opened
  eDialedUp    = 2;         // Dial-up (if necessary) executed
  eInitAdapter = 3;         // Adapter initialized
  eConnected   = 4;         // Connection to PLC established

{*****************************************************************************

 Definition der Datentypen

*****************************************************************************}

type
 TagProgParas = record
  DevNr    : Integer;      // Number of the used device
  PlcNr    : Integer;      // Number of the used PLC
  RackNr   : Integer;      // Rack number of the PLC (0 is standard)
  SlotNr   : Integer;      // Slot number of the PLC (0 is standard)
  Verbose  : BOOLEAN;      // Flag if error messages should be output
  State    : Integer;      // Connection status, will be registered by
                           // Open and CloseConnection
  ConnNr   : Integer;      // Connection handle, will be registered by
                           // OpenConnection
 end;

var
  Paras : TagProgParas;
  Rslt  : Integer;         // Buffering of function results
```

```
{******************************************************************************

 Implementation of the auxiliary functions

******************************************************************************}

{******************************************************************************

 Procedure name : ShowError

 Parameter      : ErrNr : Integer            Error number
                  var Func : String          function name as text

 Description    : This procedure outputs the error text.

 Return value   : none

 Created        : 29.05.2006  PS

 Changed        : No change

******************************************************************************}

Procedure ShowError(ErrNr : Integer; Func : PChar);
var
  Fehler : Array[0..255] of Char;
begin
  if AGL_GetErrorMsg(ErrNr, Fehler, SizeOf(Fehler)) = 0 then
     Fehler := 'Unbekannter Fehler'
     else
     WRITELN('Fehler ' + IntToHex(ErrNr, 8) + ' <' + Fehler + '> in ' + Func);
end;

{******************************************************************************

 Procedure name : ShowCommandlineHelp

 Parameter      : none

 Description    : This function outputs the invalid command line argument and
                  displays the valid.

 Return value   : none

 Created        : 29.05.2006  PS

 Changed        : No change

******************************************************************************}

Procedure ShowCommandLineHelp;
begin
  WRITELN(Paramstr(0));
  WRITELN('Gueltige Optionen sind:');
  WRITELN;
  WRITELN('  [-dDevNr] [-pPlcNr] [-rRackNr] [-sSlotNr] [-v+|-], FileName');
  WRITELN('  -dDevNr    Angabe der Geraetenummer      (Standard: 0)');
  WRITELN('  -pPlcNr    Angabe der SPS-Nummer         (Standard: erste SPS aus
LifeList)');
  WRITELN('  -rRackNr   Angabe der Racknummer         (Standard: ohne)');
  WRITELN('  -sSlotNr   Angabe der Slotnummer         (Standard: ohne)');
  WRITELN('  -v+        Fehlermeldungen ausgeben      (Standard)');
  WRITELN('  -v-        Fehlermeldungen nicht ausgeben');
  WRITELN;
end;
```

```
{*****************************************************************************

 Funvtion name  : CheckCommandline

 Parameter      : Param              Structure including program parameters

 Description    : This function evaluates the command line

 Return value   : none

 Created        : 29.05.2006  PS

 Changed        : No change

*****************************************************************************}

Procedure CheckCommandline(var Param : TagProgParas);
var
  i : Integer;
begin
  for i := 1 to ParamCount do
    begin
      if (ParamStr(i)[1] = '-') OR (ParamStr(i)[1] = '/')then
        begin
          case ParamStr(i)[2] of
            'd', 'D' : Param.DevNr   := StrToInt(ParamStr(i)[3]);
            'p', 'P' : Param.PlcNr   := StrtoInt(ParamStr(i)[3]);
            'r', 'R' : Param.RackNr  := StrToint(ParamStr(i)[3]);
            's', 'S' : Param.SlotNr  := StrToint(ParamStr(i)[3]);
            'v', 'V' : if (ParamStr(i)[4] = '-') then Param.Verbose := FALSE
                                                 else Param.Verbose := TRUE;
            '?', 'h', 'H' : ShowCommandLineHelp;
          else ShowCommandLineHelp;
          end;
        end;
    end;
end;

{*****************************************************************************

 Function name  : OpenConnection

 Parameter      : Paras : TagProgPars     Structure including program parameters

 Description    : This function establishes a connection to a PLC depending on
'                 the actual connection status. The connection establishment
                  to the PLC is synchronal (Timeout = 1 is synchronal to the
                  adjusted timeout values in the config).
'

 Return value   : AGL40_SUCCESS or error code

 Created        : 29.05.2006  PS

 Changed        : No change

*****************************************************************************}

Function OpenConnection(var Paras : TagProgParas) : Integer;
var
  i        : Integer;
  Lifelist : Array[0..127] of Byte;
begin
  if (Paras.State = eNotInit) then
    begin
      Result := AGL_OpenDevice(Paras.DevNr);
      if (Result <> AGL40_SUCCESS) then
        begin
          if Paras.Verbose then ShowError(Result, 'AGL_OpenDevice');
          exit;
```

```
        end
        else Paras.State := eDevopened;
    end;
  if (Paras.State = eDevOpened) then
    begin
      Result := AGL_DialUp(Paras.DevNr, 1, 0);
      if (Result <> AGL40_SUCCESS) then
        begin
          if Paras.Verbose then ShowError(Result, 'AGL_DialUp');
          exit;
        end
        else Paras.State := eDialedUp;
    end;
  if (Paras.State = eDialedUp) then
    begin
      Result := AGL_InitAdapter(Paras.DevNr, 1, 0);
      if (Result <> AGL40_SUCCESS) then
        begin
          if Paras.Verbose then ShowError(Result, 'AGL_InitAdapter');
          exit;
        end
        else Paras.State := eInitAdapter;
    end;
  if (Paras.State = eInitAdapter) then
    begin
      if (Paras.PlcNr < 0) then
        begin
          Result := AGL_GetLifeList(Paras.DevNr, Lifelist[0], 1, 0);
          if (Result <> AGL40_SUCCESS) then
            begin
              if Paras.Verbose then ShowError(Result, 'AGL_GetLifeList');
              exit;
            end;
          for i := 0 to 127 do
            if (LifeList[i] = LL_ACTIVE) OR (LifeList[i] = LL_PASSIVE) then
              begin
                Paras.PlcNr := i;
                break;
              end;
          if (Paras.PlcNr < 0) then
            begin
              if Paras.Verbose then ShowError(Result, 'AGL_GetLifeList');
              exit;
            end;
        end;
      if (Paras.RackNr = 0) AND (Paras.SlotNr = 0) then
        // We are establishing the connection with standard values
        Result := AGL_PLCconnect(Paras.DevNr, Paras.PlcNr, Paras.ConnNr, 1, 0)
        else
        // We directly access the PLC on the RackNr and SlotNr an
        Result := AGL_PLCConnectEx(Paras.DevNr, Paras.PlcNr, Paras.RackNr,
                                   Paras.SlotNr, Paras.ConnNr, 1, 0);
      if (Result <> AGL40_SUCCESS) then
        begin
          if Paras.Verbose then ShowError(Result,
'AGL_AGL_PLCConnect/AGL_PLCConnectEx');
          exit;
        end
        else Paras.State := eConnected;
    end;
  Result := AGL40_SUCCESS;
end;
```

```
{*****************************************************************************

 Function name  : CloseConnection

 Parameter      : Paras : TagProgPars     Structure including programp arameters

 Description    : This function releases the connection to a PLC and closes the
'                 Device depending on the acutal connection status. The
'                 connection  establishment to the PLC is synchronal
                  (Timeout = 1 is synchronal to the adjusted timeout
                  values in the config).
'

 Return value   : always AGL40_SUCCESS

 Created        : 29.05.2006  PS

 Changed        : No change

*****************************************************************************}

Function CloseConnection(var Paras : TagProgParas) : Integer;
begin
  if (Paras.State = eConnected) then
    begin
      Result := AGL_PLCDisconnect(Paras.ConnNr, 1, 0);
      if (Result <> AGL40_SUCCESS) then
        if Paras.Verbose then ShowError(Result, 'AGL_PLCDisConnect');
      Paras.State := eInitAdapter;
    end;
  if (Paras.State = eInitAdapter) then
    begin
      Result := AGL_ExitAdapter(Paras.DevNr, 1, 0);
      if (Result <> AGL40_SUCCESS) then
        if Paras.Verbose then ShowError(Result, 'AGL_ExitAdapter');
      Paras.State := edialedUp
    end;
  if (Paras.State = eDialedUp) then
    begin
      Result := AGL_HangUp(Paras.DevNr, 1, 0);
      if (Result <> AGL40_SUCCESS) then
        if Paras.Verbose then ShowError(Result, 'AGL_HangUp');
      Paras.State := eDevOpened;
    end;
  if (Paras.State = eDevOpened) then
    begin
      Result := AGL_CloseDevice(Paras.DevNr);
      if (Result <> AGL40_SUCCESS) then
        if Paras.Verbose then ShowError(Result, 'AGL_CloseDevice');
      Paras.State := eNotinit;
    end;
  Result := AGL40_SUCCESS;
end;
```

```
{*****************************************************************************

The main program starts here

*****************************************************************************}

begin
  //
  // Initialization of the parameters including standard values
  //
  Paras.DevNr   := 0;          // We are using Device 0
  Paras.PlcNr   := -1;         // We are using the first PLC on the Lifelist
  Paras.RackNr  := 0;          // We are working without ...
  Paras.SlotNr  := 0;          // ... extended addressing
  Paras.Verbose := true;       // We want to see error messages in any case
  Paras.ConnNr  := 0;          // Connection handle will be entered later
  Paras.State   := eNotInit;   // Nothing is happening, yet
  //
  // When using a developer's license please enter your registry key
  //
  AGL_Activate( '123456-1234-123456');

  CheckCommandLine(Paras);
  Rslt := OpenConnection(Paras);
  if(Rslt = AGL40_SUCCESS) then
    begin
      //
      // Hooray, we have a connection to the PLC and can start
      // At this point you can insert any communication functions
      //

      //
      // Erste Schritte Teil 1, (First Steps Part 1)
      //
      if(Paras.RackNr = 0) AND (Paras.SlotNr = 0) then
        WRITELN('Verwende Geraet ' + IntToStr(Paras.DevNr) +
                ' und PLCNr ' + IntToStr(Paras.PlcNr))
      else
        WRITELN('Verwende Geraet ' + IntToStr(Paras.DevNr) +
                ' und PLCNr '  + IntToStr(Paras.PlcNr)  +
                ' mit RackNr ' + IntToStr(Paras.RackNr) +
                ' und SlotNr ' + IntToStr(Paras.SlotNr));
    end
    else ShowError(Rslt, 'OpenConnection');
  CloseConnection(Paras);
  WRITELN;
  WRITELN('Please press the "Enter" key to exit ...');
  READLN;
end.
```

Please start the program. You will receive the following soft copy:

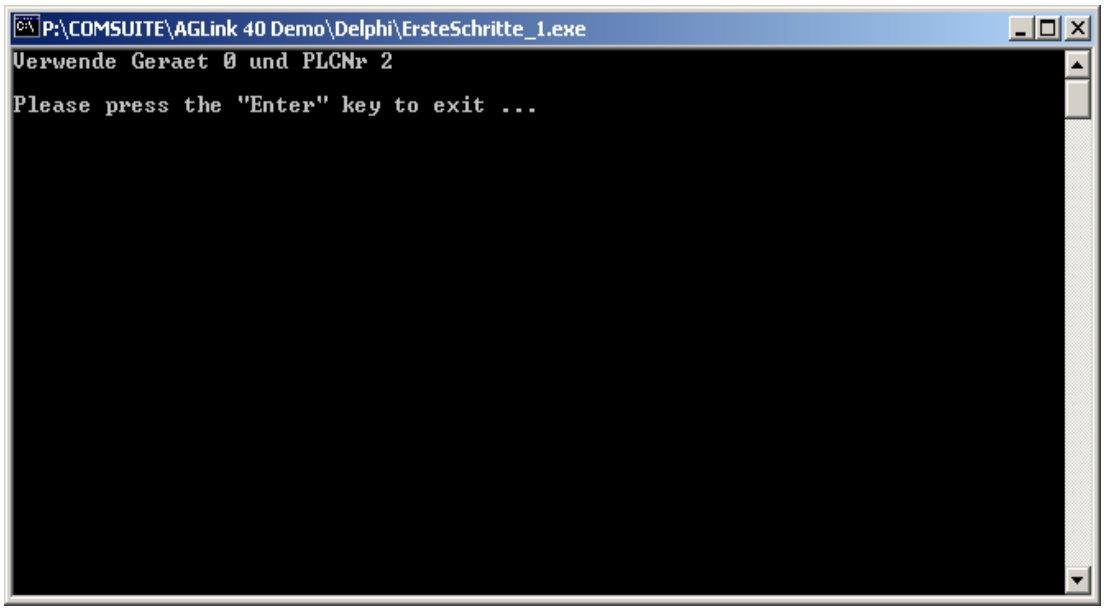


Figure 18: »ErsteSchritte_1.exe« FirstSteps

This first program example already establishes a connection to a PLC, outputs a message, releases the connection to the PLC and waits for a keypress. Now we want to breathe life into further examples. To do this we call different communication functions after the connection release. They will be explained in detail in the respective chapter.

5.3.2 Get information about the used PLC

In this chapter we want to discover to which PLC we have established a connection. Interesting is the PLC's MLFB number and which memory area is available. For this we add the following function to the module and call it after the connection establishment.

```
{*****************************************************************************

 Function name  : ShowPLCInfo

 Parameter      : Paras : TagProgPars     Structure including program parameters

 Description    : This function outputs information about the connected PLC.

 Return value   : The return value of the faulty function or AGL40_SUCCESS

 Created        : 29.05.2006  PS

 Changed        : No change

*****************************************************************************}

Function ShowPLCInfo(var Paras : TagProgParas) : Integer;
var
  DevType   : Integer;
  MLFB      : TagMLFB;
  PMLFB     : PChar;
  OpState   : Integer;
  PLCInfo   : TagPLCInfo;
  CycleTime : TagCycleTime;
  Zeit      : TagTOD;
  ST        : SystemTime;
begin
  //
  // At first we check if it is a S7
  //
  if(Paras.RackNr = 0) AND (Paras.SlotNr = 0) then
    WRITE('Verwende Geraet ' + IntToStr(Paras.DevNr) + ' und PLCNr ' +
IntToStr(Paras.PlcNr) + ' ueber ')
    else
    WRITE('Verwende Geraet ' + IntToStr(Paras.DevNr) + ' und PLCNr '  +
IntToStr(Paras.PlcNr) +
          ' mit RackNr ' + IntToStr(Paras.RackNr)    + ' und SlotNr ' +
IntToStr(Paras.SlotNr) +
          ' ueber ');
  DevType := AGL_GetDevType(Paras.DevNr);
  case DevType of
    TYPE_S7CONN_IE   : WRITELN('projektierte S7-Verbindung');
    TYPE_S7_TCPIP    : WRITELN('S7-TCP/IP');
    TYPE_S7_NL       : WRITELN('Netlink');
    TYPE_S7_NLPRO    : WRITELN('Netlink Pro');
    TYPE_S7_NLUSB    : WRITELN('Netlink USB');
    TYPE_S7_SOFTING  : WRITELN('Softing Kartentreiber');
    TYPE_S7_CIF      : WRITELN('CIF Kartentreiber');
    TYPE_S7_MPI_SER  : WRITELN('PC-Adapter seriell');
    TYPE_S7_MPI_USB  : WRITELN('PC-Adapter USB');
    TYPE_S7_TS_AT    : WRITELN('AT_Modem');
    TYPE_S7_TS_TAPI  : WRITELN('TAPI-Modem');
    TYPE_S7_PCCP     : WRITELN('Siemens-Geraetetreiber');
    TYPE_S7_PPI      : WRITELN('PPI-Adapter');
    else WRITELN('unzulaessigen Verbindungsweg' + IntToStr(Paras.DevNr) + ' !');
  end;  // case DevType of
  //
  // Now we can go on
  // Reading the MLFB No. of the CPU e.g.
```

```
  Result := AGL_ReadMLFBNr(Paras.ConnNr, MLFB, 1, 0 );
  PMLFB  := @MLFB;
  if Result = AGL40_SUCCESS then
    begin
      WRITELN('Kommuniziere mit SPS ' + (PMLFB));
    end
    else if Paras.Verbose then ShowError(Result, 'AGL_ReadMLFB');
  WRITELN;
  // Now we read out the status of the PLC
  Result := AGL_ReadOpState(Paras.ConnNr, OpState, 1, 0);
  if Result = AGL40_SUCCESS then
    begin
      case OpState of
        OPSTATE_STOP  : WRITELN('Die CPU ist im Stop');
        OPSTATE_START : WRITELN('Die CPU ist im Anlauf');
        OPSTATE_RUN   : WRITELN('Die CPU ist im Run');
      else WRITELN('Die CPU ist einem unbekannten Betriebszustand');
      end;
    end
    else if Paras.Verbose then ShowError(Result, 'AGL_ReadOpState');
  WRITELN;
  // Now we read out some information about the PLC
  Result := AGL_ReadPLCInfo(Paras.ConnNr, PLCInfo, 1, 0);
  if Result = AGL40_SUCCESS then
    begin
      WRITELN(Format('Anzahl Eingangsbytes im Prozessabbild: %d',
[PLCInfo.PAE]));
      WRITELN(Format('Anzahl Ausgangsbytes im Prozessabbild: %d',
[PLCInfo.PAA]));
      WRITELN(Format('Anzahl Merkerbytes...................: %d',
[PLCInfo.Flags]));
      WRITELN(Format('Anzahl Zeiten........................: %d',
[PLCInfo.Timer]));
      WRITELN(Format('Anzahl Zaehler.......................: %d',
[PLCInfo.Counter]));
      WRITELN(Format('Groesse des Lokaldatenspeichers......: %d',
[PLCInfo.LocalData]));
    end
    else if Paras.Verbose then ShowError(Result, 'AGL_ReadPLCInfo');
  WRITELN;
  // Now we read out some information about the cycle time
  Result := AGL_ReadCycleTime(Paras.ConnNr, CycleTime, 1, 0);
  if Result = AGL40_SUCCESS then
    begin
      WRITELN(Format('Die aktuelle Zykluszeit betraegt %d ms',
[CycleTime.AktCycleTime]));
      WRITELN(Format('Die minimale Zykluszeit betraegt %d ms',
[CycleTime.MinCycleTime]));
      WRITELN(Format('Die maximale Zykluszeit betraegt %d ms',
[CycleTime.MaxCycleTime]));
    end
    else if Paras.Verbose then ShowError(Result, 'AGL_ReadCycleTime');
  WRITELN;
  // In the end we read out the time from the PLC
  Result := AGL_getPLCClock(Paras.ConnNr, Zeit, 1, 0);
  if Result = AGL40_SUCCESS then
    begin
      //
      // We change the BCD tme into an educible format
      //
      AGL_TOD2SysTime(Zeit, ST);
      WRITELN(Format('Datum auf der SPS ......: %0.2d.%0.2d.%0.4d', [ST.wDay,
ST.wMonth, ST.wYear]));
      WRITELN(Format('Uhrzeit auf der SPS ....: %0.2d:%0.2d:%0.2d:%0.3d',
[ST.wHour, ST.wMinute, ST.wSecond, ST.wMilliseconds]));
    end
    else if Paras.Verbose then ShowError(Result, 'AGL_GetPLCClock');
  WRITELN;
  Result := AGL40_SUCCESS;
end;
```

Now we change the following in the main program:

```
Rslt := OpenConnection(Paras);
if(Rslt = AGL40_SUCCESS) then
  begin
    //
    // Hooray, we have a connection to the PLC and can start.
    // At this point you can insert any communication functions.
    //

    //
    // Erste Schritte Teil 2 (First Steps part 2)
    //
    ShowPLCInfo(Paras);

  end
  else ShowError(Rslt, 'OpenConnection');
CloseConnection(Paras);
```

Please start the programm with »F5«. You will receive the following soft copy:

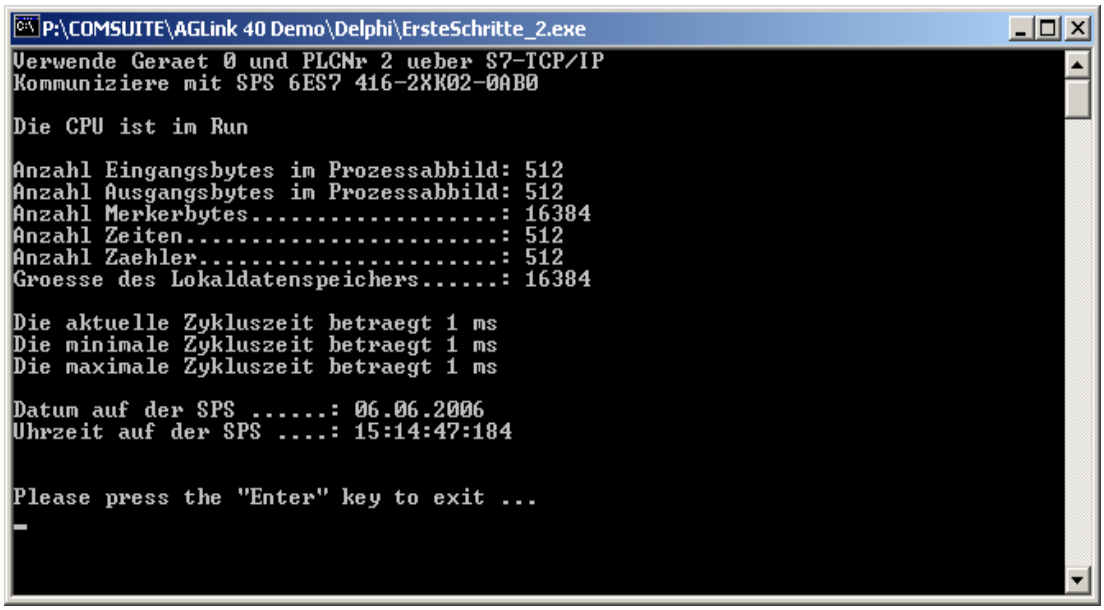


Figure 19: »ErsteSchritte_2.exe« FirstSteps

5.3.3 Reading out the diagnostics buffer

In this chapter we want to output the content of the diagnostics buffer. For this we add the following function into the module and call it after the connection establishment.

```
{*****************************************************************************

 Function name  : ShowDiagBuffer

 Parameter      : Paras : TagProgPars     Structure including program parameters

 Description    : This function outputs the content of the diagnostics buffer

 Return value   : Return value of the faulty function or AGL40_SUCCESS

 Created        : 29.05.2006  PS

 Changed        : No change

*****************************************************************************}

Function ShowDiagBuffer(var Paras : TagProgParas) : Integer;
var
  DiagBuff  : Array[0..(8+20*30) - 1] of Byte;
  Buff1     : Array[0..255] of Char;
  Entrys, i : Integer;
begin
  //
  // There are two ways to read out the diagnostics buffer:
  // Either we inquire the amount of parametrized data, assigning the memory,
  // reading out the diagnostics buffer, output the text and release the memory.
  // Or we select how much entries are of interest to us, indicate this number
  // when reading the diagnostics buffer and output the text.
  // Each entry in the diagnostics buffer needs 20 bytes. In addition 8 bytes are
  // needed for header info. Normally, there are 120 entries, they can be changed
  // via the HW-config.
  // Therefore the following method arises for the last 30 entries:
  //
  Entrys := 30;
  Result := AGL_ReadDiagBuffer(Paras.ConnNr, Entrys, DiagBuff[0], 1, 0);
  if Result = AGL40_SUCCESS then
    begin
      WRITELN(Format('Anzahl gelesener Diagnosepuffereintraege : %d', [Entrys]));
      WRITELN;
      for i := 0 to Entrys - 1 do
        begin
          AGL_GetDiagBufferEntry(i, DiagBuff[0], Buff1, SizeOf(Buff1));
          if IsConsole then
             CharToOem(Buff1, Buff1);     // Ist bei einer Konsolenapplikation
                                          // wegen den Umlauten notwendig
          WRITELN(Format('%0.3d: %s', [i + 1, Buff1]));
        end;
    end
    else if Paras.Verbose then ShowError(Result, 'AGL_ReadDiagBuffer');
end;
```

Now we change the following things in the main program:

```
Rslt := OpenConnection(Paras);
if(Rslt = AGL40_SUCCESS) then
  begin
    //
  // Hooray, we have a connection to the PLC and can start
  // At this point you can insert any communication functions
    //

    //
    // Erste Schritte Teil 3, (First Steps Part 3)
    //
    ShowDiagBuffer(Paras);

  end
  else ShowError(Rslt, 'OpenConnection');
CloseConnection(Paras);
```

Please start the programm. You will receive the following soft copy:

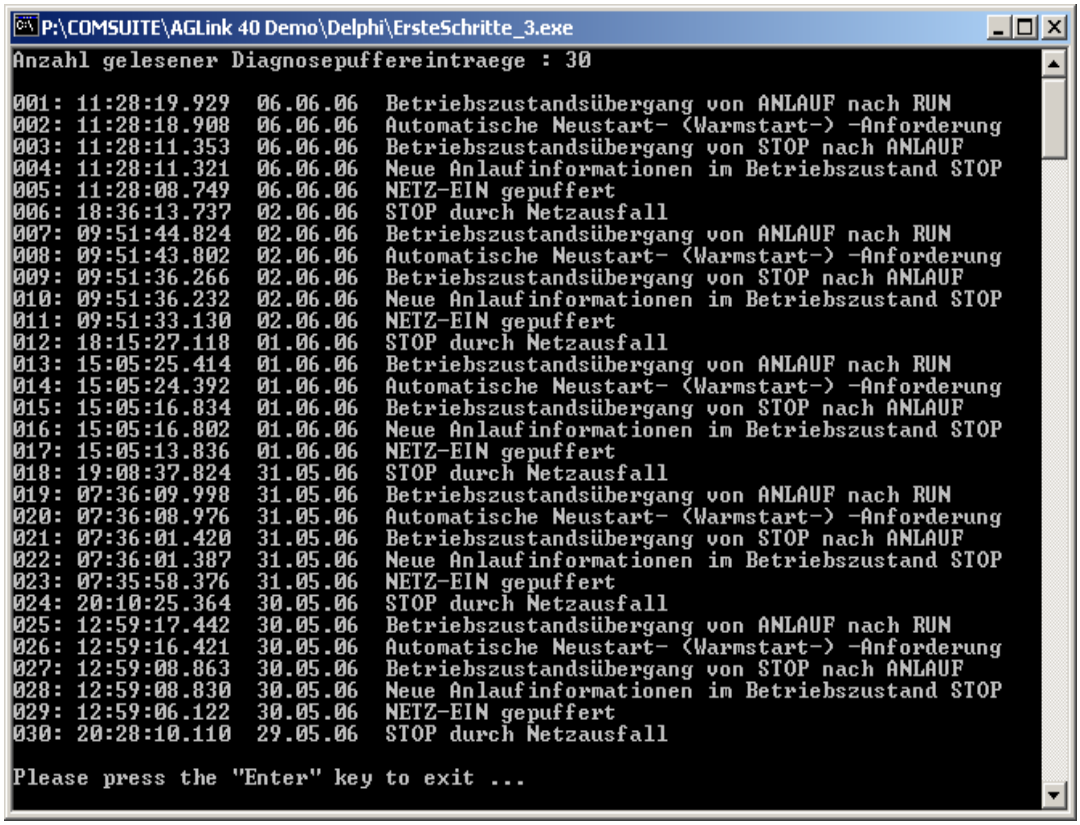


Figure 20: »ErsteSchritte_3.exe«, FirstSteps

5.3.4 Reading data from different sections

In this chapter we want to approach the most interesting thing in ACCON-AGLink; reading data from different sections. For this we add the following function to the module and call it after the connection establishment.

```
{*****************************************************************************

 Function name  : ShowReadMemory

 Parameter      : Paras : TagProgPars     Structure including program parameters

 Description    : This function reads out different memory sections with a
                  function call.

 Return value   : Return value of the faulty function or AGL40_SUCCESS

 Created        : 02.06.2006  PS

 Changed        : No change

*****************************************************************************}

Function ShowReadMemory(var Paras : TagProgParas) : Integer;
var
  i, j, DBCount, DBs, DBLen   : Integer;
  RW     : Array[0..7] of TagDATA_RW40;  // We are reading here 8 different
                                         // sections at most
  bBuff  : array[0..7, 0..15] of Byte;   // Buffer for byte read access
  wBuff  : array[0..7, 0..7]  of Word;   // Buffer for word read access
  dwBuff : Array[0..7, 0..3]  of DWord;  // Buffer for double word read access
  DBList : Array[0..31] of Word;         // List of numbers from the data modules
begin
  WRITELN('Lese Daten aus verschiedenen Bereichen in einem Aufruf...');
  WRITELN;
  //
  // At first we are performing a AGL_ReadMixEx on different and always present
  // memory sections(E/A/M/T/Z).
  //
  With RW[0] do
    begin
      OpArea := AREA_IN;      // We are reading from section PA inputs
      OpType := TYP_BYTE;     // We are reading byte elements
      OpAnz  := 16;           // We are reading 16 elements
      DBNr   := 0;            // The DB number is assigned to 0
      Offset := 0;            // We are reading from EB 0
      Bitnr  := 0;            // The bit number is assigned to 0
      Buff   := @bBuff[0];    // The results have to be in this section
    end;
  With RW[1] do
    begin
      OpArea := AREA_OUT;     // We are reading from section PA outputs
      OpType := TYP_BYTE;     // We are reading byte elements
      OpAnz  := 16;           // We are reading 16 elements
      DBNr   := 0;            // The DB number is assigned to 0
      Offset := 0;            // We are reading from AB 0
      Bitnr  := 0;            // The bit number is assigned to 0
      Buff   := @bBuff[1];    // The results have to be in this section
    end;
  With RW[2] do
    begin
      OpArea := AREA_FLAG;    // We are reading from section marker
      OpType := TYP_BYTE;     // We are reading byte elements
      OpAnz  := 16;           // We are reading 16 elements
      DBNr   := 0;            // The DB number is assigned to 0
      Offset := 8;            // We are reading from MB 8
      Bitnr  := 0;            // The bit number is assigned to 0
```

```
    Buff   := @bBuff[2];    // The results have to be in this section
  end;
With RW[3] do
  begin
    OpArea := AREA_FLAG;    // And now the same word by word
    OpType := TYP_WORD;     // We are reading word elements
    OpAnz  := 8;            // We are reading 8 elements
    DBNr   := 0;            // The DB number is assigned to 0
    Offset := 8;            // We are reading from MB 8
    Bitnr  := 0;            // The bit number is assigned to 0
    Buff   := @wBuff[3];    // The results have to be in this section
  end;
With RW[4] do
  begin
    OpArea := AREA_FLAG;    // And now the same double word by double word
    OpType := TYP_DWORD;    // We are reading double word elements
    OpAnz  := 4;            // We are reading 4 elements
    DBNr   := 0;            // The DB number is assigned to 0
    Offset := 8;            // We are reading from MB 8 !!
    Bitnr  := 0;            // The bit number is assigned to 0
    Buff   := @dwBuff[4];   // The results have to be in this section
  end;
With RW[5] do
  begin
    OpArea := AREA_COUNTER; // We are reading from section counter
    OpType := TYP_COUNTER;  // We are reading counter elements
    OpAnz  := 8;            // We are reading 8 elements
    DBNr   := 0;            // The DB number is assigned to 0
    Offset := 0;            // We are reading from Z 0
    Bitnr  := 0;            // The bit number is assigned to 0
    Buff   := @wBuff[5];    // The results have to be in this section
  end;
With RW[6] do
  begin
    OpArea := AREA_TIMER;   // We are reading from section timers
    OpType := TYP_TIMER;    // We are reading timer elements
    OpAnz  := 8;            // We are reading 8 elements
    DBNr   := 0;            // The DB number is assigned to 0
    Offset := 16;           // We are reading from T 16
    Bitnr  := 0;            // The bit number is assigned to 0
    Buff   := @wBuff[6];    // The results have to be in this section
  end;
With RW[7] do
  begin
    OpArea := AREA_TIMER;   // We are reading from section timers
    OpType := TYP_TIMER;    // We are reading timer elements
    OpAnz  := 8;            // We are reading 8 elements
    DBNr   := 0;            // The DB number is assigned to 0
    Offset := 2048;         // We are reading from T 2048 => Errors occur!!
    Bitnr  := 0;            // The bit number is assigned to 0
    Buff   := @wBuff[7];    // The results have to be in this section
  end;
Result := AGL_ReadMixEx(Paras.ConnNr, RW[0], 8, 1, 0);
if Result = AGL40_SUCCESS then
  begin
    for i := 0 to 7 do
      begin
        //
        // Every structure has to be checked
        //
        if RW[i].Result  = AGL40_SUCCESS then
          begin
            case RW[i].OpType of
              TYP_BIT,
              TYP_BYTE  : BEGIN
                            for j := 0 to 15 do
                              WRITE(Format('%0.2X  ', [bBuff[i][j]]));
                            WRITELN;
                          END;
              TYP_WORD,
```

```
                  TYP_COUNTER,
                  TYP_TIMER : BEGIN
                                for j := 0 to 7 do
                                  WRITE(Format('%0.4X  ', [wBuff[i][j]]));
                                WRITELN;
                              END;
                  TYP_DWORD : BEGIN
                                for j := 0 to 3 do
                                  WRITE(Format('%0.8X  ', [dwBuff[i][j]]));
                                WRITELN;
                              END;
              end;  // Case RW[i].OpType
            end   // if RW[i].Result
            else if Paras.Verbose then ShowError(RW[i].Result, 'AGL_ReadMixEx');
        end;  // For i := 0 to 7 do
    end
    else if Paras.Verbose then ShowError(Result, 'AGL_ReadMixEx');
  WRITELN;
  //
  // And now we are searching for dynamic sections and reading them out (= DBs)
  //
  i   := 0;  // i has already been used as loop counter, so re-initialize
  DBs := 0;
  Result := AGL_ReadDBCount(Paras.ConnNr, DBCount, 1, 0);
  if Result = AGL40_SUCCESS then
    begin
      if (DBCount > 0) then
        begin
          WRITELN(Format('%d Datenbausteine vorhanden.', [DBCount]));
          WRITELN;
          if(DBCount > SizeOf(DBList) DIV SizeOf(DBList[0])) then
            //
            // Only the first modules are interesting for us
            //
            DBCount := SizeOf(DBList) DIV SizeOf(DBList[0]);
          Result := AGL_ReadDBList(Paras.ConnNr, DBCount, DBList[0], 1, 0);
          if Result = AGL40_SUCCESS then
            begin
              for i := 0 to DBCount - 1 do
                begin
                  Result := AGL_ReadDBLen(Paras.ConnNr, DBList[i], DBLen, 1 , 0);
                  if Result = AGL40_SUCCESS then
                    begin
                      if (DBLen > 16) AND (DBs < 8) then
                        //
                        // IF the minimum length of 16 bytes is met you can read
                        // out this data module
                        begin
                          WRITELN(Format('Lese 16 Bytes aus DB %d',
[DBList[i]]));
                          With RW[DBs] do
                            begin
                              OpArea := AREA_DATA;    // We are reading from the
                                                      // section data files
                              OpType := TYP_BYTE;     // We are reading byte
                                                      // elements
                              OpAnz  := 16;           // We are reading 16
                                                      // elements
                              DBNr   := DBList[i];    // We are entering the DB
                                                      // number
                              Offset := 0;            // We are reading from
                                                      // DBB 0
                              Bitnr  := 0;            // The bit number is
                                                      // assigned to 0
                              Buff   := @bBuff[DBs];  // The results have to be
                                                      // in this section
                            end;  // With RW[DBs] do
                          INC(DBs);
                        end;  // if (DBLen > 16)
                    end
```

```
                    else if Paras.Verbose then ShowError(RW[i].Result,
'AGL_ReadDBLen');
                end;  // for i := 0  to DBCount - 1 do
              WRITELN;
              i := 0;  // i has already been used as loop counter, so re-
                       // initialize
              if (DBs > 0) then
                begin
                  Result := AGL_ReadMixEx(Paras.ConnNr, RW[0], DBs, 1, 0);
                  for i := 0 to DBs - 1 do
                    begin
                      //
                      // Every structue has to be checked
                      //
                      if RW[i].Result = AGL40_SUCCESS then
                        begin
                          for j := 0 to 15 do WRITE(Format('%0.2X ', [bBuff [i]
[j]]));
                          WRITELN;
                        end  // if RW[i].Result
                        else if Paras.Verbose then ShowError(RW[i].Result,
'AGL_ReadMixEx');
                    end; // for i := 0 to DBs - 1
                end  // if DBs > 0
                else if Paras.Verbose then ShowError(RW[i].Result,
'AGL_ReadMixEx');
            end  // AGL_ReadDBList succeeded
            else if Paras.Verbose then ShowError(RW[i].Result, 'AGL_ReadDBList');
        end  // if DBCount > 0
        else WRITELN('Keine Datenbausteine für Testzwecke gefunden');
    end
    else if Paras.Verbose then ShowError(Result, 'AGL_DBCount');
  WRITELN;
end;
```

Now we change the foolowing things in the main program:

```
  Rslt := OpenConnection(Paras);
  if(Rslt = AGL40_SUCCESS) then
    begin
      //
      // Hooray, we have a connection to the PLC and can start
      // At this point you can insert any communication functions
      //

      //
      // Erste Schritte Teil 4, (First Steps Part 4)
      //
      ShowReadMemory(Paras);

    end
    else ShowError(Rslt, 'OpenConnection');
  CloseConnection(Paras);
```

Please compile and start the program. You will receive the following soft copy:

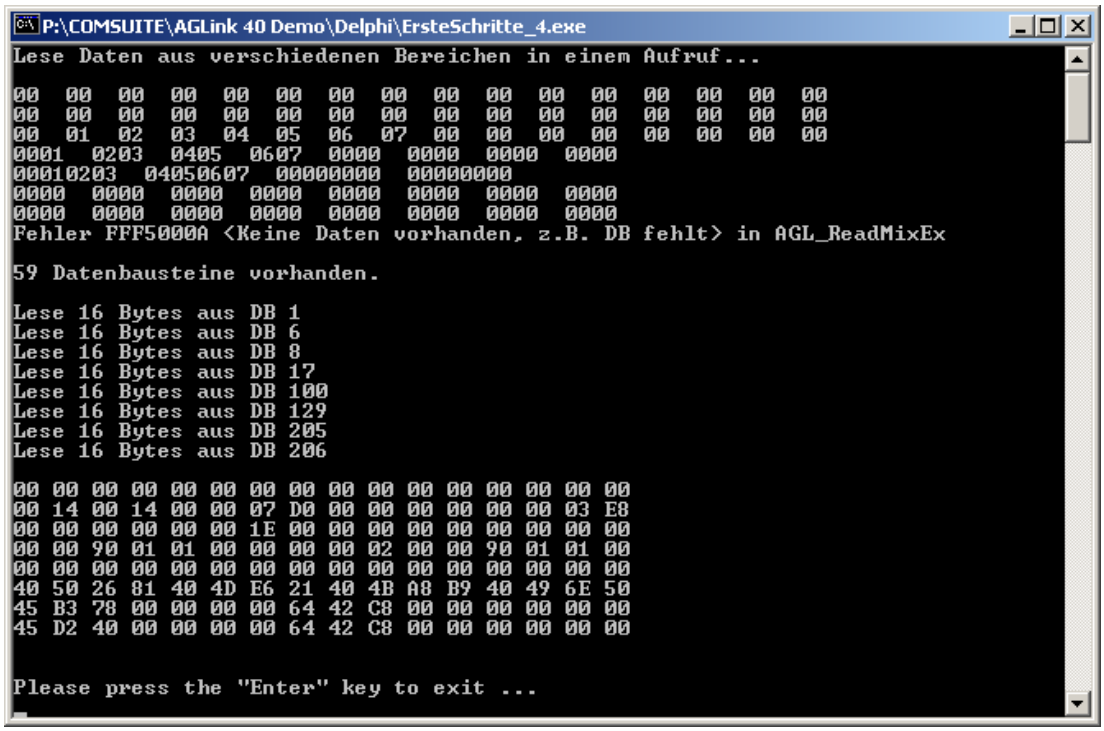


Figure 21: »ErsteSchritte_4.exe«, FirstSteps

6 First Steps with Linux

This chapter gives attention to the basic procedure of compiling a litlle program example with ACCON-AGLink version 4. But before starting you have to parametrize the used device and its communication path correctly. For this you need the following data:

```
AGLink40_Config             The proper configuration program
AGLink40_Config.xml         Parameters of the configuration program
AGLink40_Config_de.mo       German lanuage data
AGLink40_Config_en.mo       English language data
```

Optional device check:

```
libAGLink40.so              (Demo-)version of ACCON-AGLink
AGLink40_Error.txt          Data file wiht AGLink error messages in plain
                            text
```

ACCON-AGLink consists of the proper library data e.g. libAGLink40.so.4.0.2.12 and the appropriate shortcut data files libAGLink40.so.4 and libAGLink40.so. Consecutively, the shortcut libAGLink40.so will be named vicariously for the proper library data file.

First of all start the provided program »AGLink40_Config.EXE«. Then adjust Device 0 according to the used hardware. Check the settings with the »Device Check«. The configuration will be put into the data »AGLink40CfgDev0000.xml«. To start the program you only need this parameter data besides ACCON-AGLink and you can just copy this data with your application. Thus a device configuration does not have to be on the target system but can be performed on another PC.

For reasonable matters we demerge this program and perform a few little auxiliary functions. With it the complete porgram remains clearly and we can concentrate on the essential things in the examples. A possible sectioning could be:

- Output of program arguments
- Interpretation of the command line arguments (if possible)
- Output of error information
- Establishing connection to a PLC
- Connection clearing to a PLC

To analyze the command line parameter the program parameters, in one structure, are passed to the function. Because of this additional parameter extensions are possible and you do not have to change the call up interface.

Naturally, an error can occur anywhere at a connection establishment, therefore we remember our actual position in a respective variable. From this status we can close the connection correctly. Enumeration is a suitable variable type.

6.1 First Steps with g++ (console application)

We need the following data for the creation and the test of a ACCON-AGLink40 application with g++:

AGLink40.h	Basic header data, integrates further data files
OSDefInc.h	Header data with definitions depending on the oprating system (integrated by AGLink40.h)
AGL_Defines.h	Header data with constant definitions (integrated by AGLink40.h)
AGL_Types.h	Header data with structure definitions (integrated by AGLink40.h)
AGL_Funcs.h	Header data with function declarations (integrated by AGLink40.h)
libAGLink40.so	The proper function library of ACCON-AGLink version 4
AGLink40_Error.txt	Data file with error messages in plain text
AGLink40CfgDev0000.xml	Data file with configuration data in plain text, in this case Device 0

Later only AGLink40.so, AGLink40_Error.txt and AGLink40CfgDev0000.xml are necessary for the compiled program. In chapter »6 First Steps with Linux« you will find all essential data to change the configuration.

To ease all this we now compile a little console application. Open a new console (command input window) and create a new folder with the data above-mentioned. Create a new C++ source code data file called ErsteSchritte.cpp, FirstSteps. Translate this source data file using the following command line:

```
g++ -DLINUX -lAGLink40 -oErsteSchritte ErsteSchritte.cpp
```

The command line parameters mean in detail:

- Define LINUX symbol (-DLINUX)
- Use the external and dynamic library libAGLink40. (-lAGLink40)
- Created program data file ErsteSchritte (-oErsteSchritte) FirstSteps
- Translated source data file ErsteSchritte.cpp FirstSteps

6.1.1 The Communication's Skeletal Structure

This chapter shows how to establish a communication to a PLC. And how to detect the connected participants from the Lifelist and pick one from it. After that a message is displayed and the connection will be closed. So this skeletal structure has all necessary tools.

Add the following program into the new source code data file into the »ErsteSchritte cpp», (FirstSteps).

Note:
Naturally, the complete project is located in the ACCON-AGLink 40 demo folder for Linux g++. The source code data file ErsteSchritte.cpp, FirstSteps can be translated automatically when using the included creation data file »makefile«. For this enter the command »make«. The source code data file ErsteSchritte.cpp, FirstSteps includes the whole program code for the subsequent steps. The automatic creation generates all programs in one cycle via the command »make«.

```
/*****************************************************************************

 Project        : New version of the AGLink library

 Data name      : ErsteSchritte.CPP

 Description    : Introduction into use of ACCON-AGLink version 4

 Copyright      : (c) 1998-2007
                  DELTALOGIC Automatisierungstechnik GmbH
                  Stuttgarter Str. 3
                  73525 Schwaebisch Gmuend
                  Web : http://www.deltalogic.de
                  Tel.: +49-7171-916120
                  Fax : +49-7171-916220

 Created        : 15.05.2006  RH

 Changed        : 15.05.2006  RH

 *****************************************************************************/

/*****************************************************************************

 Integrating header data files

 *****************************************************************************/

#include <string.h>
#include <stdio.h>

#include "kbhit.h"
#include "AGLink40.h"


/*****************************************************************************

 Definition of Enums

 *****************************************************************************/

enum eConnState
{
  eNotInit = 0,           // Connection not initialized, yet
  eDevOpened,             // Device opened
  eDialedUp,              // Dial-up (if necessary) executed
  eInitAdapter,           // Adapter initialized
  eConnected              // Connection to PLC established
};


/*****************************************************************************

 Definition of data file types

 *****************************************************************************/

typedef struct tagProgParas
{
  int   DevNr;            // Number of used device
  int   PlcNr;            // Number of used PLC
  int   RackNr;           // Rack number of the PLC (0 is standard)
  int   SlotNr;           // Slot number of the PLC (0 is standard)
  int   Verbose;          // Flag if error messages should be output
  eConnState State;       // Connection status, will be registered by
                          // Open and CloseConnection
  int   ConnNr;           // Connection handle will be registered by
OpenConnection
} PROG_PARAS, *LPPROG_PARAS;
```

```
/*****************************************************************************

 Implementation of the auxiliarty functions

 *****************************************************************************/

/*****************************************************************************

 Function name  : ShowError

 Parameter      : int ErrNr             Error number
                  char *Func            Function name as text

 Description    : This function outputs the error message

 Return value   : none

 Created        : 15.05.2006  RH

 Changed        : 15.05.2006  RH

 *****************************************************************************/

void ShowError( int ErrNr, char *Func )
{
  char Fehler[256];

  if( AGL_GetErrorMsg( ErrNr, Fehler, sizeof( Fehler ) ) == 0 )
  {
    strcpy( Fehler, "Unbekannter Fehler" );
  }
  printf( "Fehler %08X (%s) in %s\n", ErrNr, Fehler, Func );
}


/*****************************************************************************

 Function name  : ShowCommandlineHelp

 Parameter      : char *PrgName         Name of program data file
                  char *InvText         Invalid argument

 Description    : This function outputs the invalid command line argument and
                  displays the valid.

 Retrun value   : none

 Created        : 15.05.2006  RH

 Changed        : 15.05.2006  RH

 *****************************************************************************/

void ShowCommandlineHelp( char *PrgName, char *InvText )
{
  char Buff[256];
  char *FileName;

  if( InvText != NULL )
  {
    printf( "Unbekanntes Kommandozeilenargument: %s\n", InvText );
  }

  strncpy( Buff, PrgName, sizeof( Buff ) );

  if( (FileName = strrchr( Buff, '\\' )) == NULL )
  {
    if( (FileName = strrchr( Buff, '/' )) == NULL )
```

```
    {
      FileName = Buff;
    }
    else
    {
      FileName++;
    }
  }
  else
  {
    FileName++;
  }
  printf( "\nGueltige Optionen sind:\n\n" );
  printf( "%s [-dDevNr] [-pPlcNr] [-rRackNr] [-sSlotNr] [-v+|-]\n", FileName );
  printf( "  -dDevNr    Angabe der Geraetenummer      (Standard: 0)\n" );
  printf( "  -pPlcNr    Angabe der SPS-Nummer         (Standard: erste SPS aus
                                                     LifeList)\n" );
  printf( "  -rRackNr   Angabe der Racknummer         (Standard: ohne)\n" );
  printf( "  -sSlotNr   Angabe der Slotnummer         (Standard: ohne)\n" );
  printf( "  -v+        Fehlermeldungen ausgeben      (Standard)\n" );
  printf( "  -v-        Fehlermeldungen nicht ausgeben\n\n" );
}


/******************************************************************************

 Function name  : CheckCommandline

 Parameter      : int argc              Number of arguments
                  char *argv[]          The proper command line arguments
                  LPPROG_PARAS pParas   Structure including programparameters

 Description    : This function evaluates the command line

 Return value   : none

 Created        : 15.05.2006  RH

 Changed        : 15.05.2006  RH

 ******************************************************************************/

void CheckCommandline( int argc, char *argv[], LPPROG_PARAS pParas )
{
  for( int i=1; i<argc; i++ )
  {
    if( argv[i][0] == '-' || argv[i][0] == '/' )
    {
      switch( argv[i][1] )
      {
        case 'd':
        case 'D':
        {
          pParas->DevNr = atoi( &argv[i][2] );
          break;
        }
        case 'p':
        case 'P':
        {
          pParas->PlcNr = atoi( &argv[i][2] );
          break;
        }
        case 'r':
        case 'R':
        {
          pParas->RackNr = atoi( &argv[i][2] );
          break;
        }
        case 's':
        case 'S':
```

```
          {
            pParas->SlotNr = atoi( &argv[i][2] );
            break;
          }
          case 'v':
          case 'V':
          {
            if( argv[i][2] == '+' )
            {
              pParas->Verbose = true;
            }
            else if( argv[i][2] == '-' )
            {
              pParas->Verbose = false;
            }
            break;
          }
          case '?':
          case 'h':
          case 'H':
          {
            ShowCommandlineHelp( argv[0], NULL );
            break;
          }
          default:
          {
            ShowCommandlineHelp( argv[0], argv[i] );
            break;
          }
        }
      }
      else
      {
        ShowCommandlineHelp( argv[0], argv[i] );
      }
    }
  }

  /*****************************************************************************

   Function name  : OpenConnection

   Parameter      : LPPROG_PARAS pParas   Structure including program parameters

  Description      : This function establishes a connection to a PLC depending on
                     the actual connection status. The connection establishment
                     to the PLC is synchronal (Timeout = 1 is synchronal to the
                     adjusted timeout values in the config).

   Return value   : AGL40_SUCCESS or the error message

   Created        : 15.05.2006  RH

   Changed        : 15.05.2006  RH

   *****************************************************************************/

  int OpenConnection( LPPROG_PARAS pParas )
  {
    int RetVal;

    if( pParas->State == eNotInit )
    {
      if( (RetVal = AGL_OpenDevice( pParas->DevNr )) != AGL40_SUCCESS )
      {
        if( pParas->Verbose )
        {
          ShowError( RetVal, "AGL_OpenDevice" );
        }
```

```
      return( RetVal );
    }
    pParas->State = eDevOpened;
  }
  if( pParas->State == eDevOpened )
  {
    if( (RetVal = AGL_DialUp( pParas->DevNr, 1, 0 )) != AGL40_SUCCESS )
    {
      if( pParas->Verbose )
      {
        ShowError( RetVal, "AGL_DialUp" );
      }
      return( RetVal );
    }
    pParas->State = eDialedUp;
  }
  if( pParas->State == eDialedUp )
  {
    if( (RetVal = AGL_InitAdapter( pParas->DevNr, 1, 0 )) != AGL40_SUCCESS )
    {
      if( pParas->Verbose )
      {
        ShowError( RetVal, "AGL_InitAdapter" );
      }
      return( RetVal );
    }
    pParas->State = eInitAdapter;
  }
  if( pParas->State == eInitAdapter )
  {
    if( pParas->PlcNr < 0 )
    {
      BYTE LifeList[128];

      RetVal = AGL_GetLifelist( pParas->DevNr, LifeList, 1, 0 );
      if( RetVal != AGL40_SUCCESS )
      {
        if( pParas->Verbose )
        {
          ShowError( RetVal, "AGL_GetLifeList" );
        }
        return( RetVal );
      }
      for( int i=0; i<127; i++ )
      {
        if( LifeList[i] == LL_ACTIVE || LifeList[i] == LL_PASSIVE )
        {
          pParas->PlcNr = i;
          break;
        }
      }
      if( pParas->PlcNr < 0 )
      {
        RetVal = AGL40_PLC_NOT_FOUND;
        if( pParas->Verbose )
        {
          ShowError( RetVal, "AGL_GetLifeList" );
        }
        return( RetVal );
      }
    }
    if( pParas->RackNr == 0 && pParas->SlotNr == 0 )
    {
      //
      // We are establishing the connection with standard values
      //
      RetVal = AGL_PLCConnect( pParas->DevNr, pParas->PlcNr,
                               &pParas->ConnNr, 1, 0 );
    }
    else
```

```
    {
      //
      // We directly access the PLC on the RackNr and SlotNr
      //
      RetVal = AGL_PLCConnectEx( pParas->DevNr, pParas->PlcNr, pParas->RackNr,
                                 pParas->SlotNr, &pParas->ConnNr, 1, 0 );
    }
    if( RetVal != AGL40_SUCCESS )
    {
      if( pParas->Verbose )
      {
        ShowError( RetVal, "AGL_PLCConnect/AGL_PLCConnectEx" );
      }
      return( RetVal );
    }
    pParas->State = eConnected;
  }
  return( AGL40_SUCCESS );
}


/*****************************************************************************

 Function name  : CloseConnection

 Parameter      : LPPROG_PARAS pParas   Structure including program parameters

 Description    : This function releases the connection to a PLC and closes the
                  Device depending on the acutal connection status. The
                  connection  establishment to the PLC is synchronal
                  (Timeout = 1 is synchronal to the adjusted timeout
                  values in the config).

 Return value   : always AGL40_SUCCESS

 Created        : 15.05.2006  RH

 Changed        : 15.05.2006  RH

 *****************************************************************************/

int CloseConnection( LPPROG_PARAS pParas )
{
  int   RetVal;

  if( pParas->State == eConnected )
  {
    RetVal = AGL_PLCDisconnect( pParas->ConnNr, 1, 0 );
    if( RetVal != AGL40_SUCCESS && pParas->Verbose )
    {
      ShowError( RetVal, "AGL_PLCDisconnect" );
    }
    pParas->State = eInitAdapter;
  }
  if( pParas->State == eInitAdapter )
  {
    RetVal = AGL_ExitAdapter( pParas->DevNr, 1, 0 );
    if( RetVal != AGL40_SUCCESS && pParas->Verbose )
    {
      ShowError( RetVal, "AGL_ExitAdapter" );
    }
    pParas->State = eDialedUp;
  }
  if( pParas->State == eDialedUp )
  {
    RetVal = AGL_HangUp( pParas->DevNr, 1, 0 );
    if( RetVal != AGL40_SUCCESS && pParas->Verbose )
    {
      ShowError( RetVal, "AGL_HangUp" );
    }
```

```
    pParas->State = eDevOpened;
  }
  if( pParas->State == eDevOpened )
  {
    RetVal = AGL_CloseDevice( pParas->DevNr );
    if( RetVal != AGL40_SUCCESS && pParas->Verbose )
    {
      ShowError( RetVal, "AGL_CloseDevice" );
    }
    pParas->State = eNotInit;
  }
  return( AGL40_SUCCESS );
}


/*****************************************************************************

 Implementation of the proper main function

 *****************************************************************************/

/*****************************************************************************

 Function name  : main

 Parameter      : int argc              Number of arguments
                  char *argv[]          The proper command line arguments

 Description    : This function establishes a connection to a PLC. There you can
                  insert any coomunication functions. After that the connection
                  will be released

 Return value   : always 0 if successful, otherwise error code

 Created        : 15.05.2006  RH

 Changed        : 15.05.2006  RH

 *****************************************************************************/

int __cdecl main( int argc, char *argv[] )
{
  int RetVal;
  PROG_PARAS Paras;

  //
  // Initializing parameters with standard values
  //
  Paras.DevNr   = 0;          // We are using Device 0
  Paras.PlcNr   = -1;         // We are using the first PLC on the Lifelist
  Paras.RackNr  = 0;          // We are working without ...
  Paras.SlotNr  = 0;          // ... extendedaddressing
  Paras.Verbose = true;       // We want to see error messages in any case
  Paras.ConnNr  = 0;          // Connection handle will be entered later
  Paras.State   = eNotInit;   // Nothing is happening,yet

  //
  // When using a developer's license please enter your registry key
  //
  AGL_Activate( "123456-1234-123456" );

  CheckCommandline( argc, argv, &Paras );
  if( (RetVal = OpenConnection( &Paras )) == AGL40_SUCCESS )
  {
    //
    / Hooray, we have a connection to the PLC and can start
    / At this point you can insert any communication functions
    //

    //
```

```
    // Erste Schritte Teil 1, (First Steps Part 1)
    //
    if( Paras.RackNr == 0 && Paras.SlotNr == 0 )
    {
      printf( "Verwende Geraet %d und PLCNr %d.\n",
              Paras.DevNr, Paras.PlcNr );
    }
    else
    {
      printf( "Verwende Geraet %d und PLCNr %d mit RackNr %d und SlotNr %d.\n",
              Paras.DevNr, Paras.PlcNr, Paras.RackNr, Paras.SlotNr );
    }
    RetVal = 0;
  }
  else
  {
    if( !Paras.Verbose )
    {
      //
      // Even if we should be quiet, this message appears at any case
      //
      printf( "Abbruch wegen Fehler %08X\n"
              "Bitte beliebige Taste zum Beenden druecken ...", RetVal );
    }
    RetVal = 1;
  }
  CloseConnection( &Paras );
  printf( "\nPlease press the any key to exit ..." );
  getch();
  return( RetVal );
}
```

Please compile and start the program. You will receive the following soft copy:

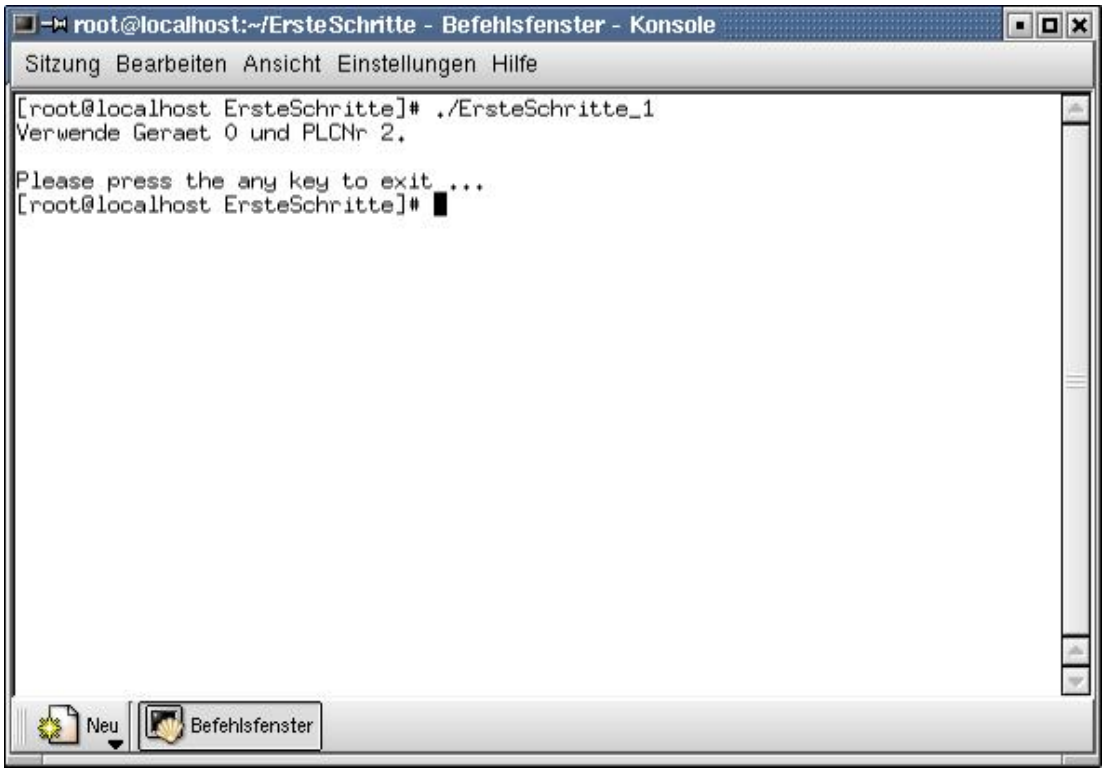


Figure 22: »ErsteSchritte_1«, FirstSteps Linux

This first program example already establishes a connection to a PLC, outputs a message, releases the connection to the PLC and waits for a keypress. Now we

want to breathe life into further examples. To do this we call different communication functions after the connection release. They will be explained in detail in the respective chapter.

Please remember that not every user has the rights to access certain devices e.g. `/dev/ttyS0` (first serial interface) when using Linux. If you cannot connect despite a correct configuration and function calls, log on as superuser root try to connect again. If it works you have to grant this user access to the respective device. The easiest way is to add this user to the group to which the device belongs to. If necessary contact your system administrator.

6.1.2 Get information about the used PLC

In this chapter we want to discover to which PLC we have established a connection. Interesting is the PLC's MLFB number and which memory area is available. For this we add the following function to the module and call it after the connection establishment.

```
/*****************************************************************************

 Function name  : ShowPLCInfo

 Parameter      : LPPROG_PARAS pParas   Structure including program parameters

 Description    : This function outputs information about the connected PLC

 Return value   : Return value of the faulty function or AGL40_SUCCESS

 Created        : 15.05.2006  RH

 Changed        : 15.05.2006  RH

 *****************************************************************************/

int ShowPLCInfo( LPPROG_PARAS pParas )
{
  int       RetVal = AGL40_SUCCESS;
  int       DevType;
  MLFB      MLFBNr;
  int       OpState;
  PLCINFO   PLCInfo;
  CYCLETIME CycleTime;
  TOD       Zeit;

  //
  // At first, we have to check if it is a S7
  //
  if( pParas->RackNr == 0 && pParas->SlotNr == 0 )
  {
    printf( "Verwende Geraet %d und PLCNr %d ueber ",
            pParas->DevNr, pParas->PlcNr );
  }
  else
  {
    printf( "Verwende Geraet %d und PLCNr %d mit RackNr %d und SlotNr %d ueber ",
            pParas->DevNr, pParas->PlcNr, pParas->RackNr, pParas->SlotNr );
  }

  DevType = AGL_GetDevType( pParas->DevNr );
  switch( DevType )
  {
    case TYPE_S7CONN_IE:
    {
      printf( "projektierte S7-Verbindung\n" );
      break;
    }
    case TYPE_S7_TCPIP:
    {
      printf( "S7-TCP/IP\n" );
      break;
    }
    case TYPE_S7_NL:
    {
      printf( "Netlink\n" );
      break;
    }
    case TYPE_S7_NLPRO:
    {
      printf( "Netlink Pro\n" );
```

```
      break;
    }
    case TYPE_S7_NLUSB:
    {
      printf( "Netlink USB\n" );
      break;
    }
    case TYPE_S7_SOFTING:
    {
      printf( "Softing Kartentreiber\n" );
      break;
    }
    case TYPE_S7_CIF:
    {
      printf( "CIF Kartentreiber\n" );
      break;
    }
    case TYPE_S7_MPI_SER:
    {
      printf( "PC-Adapter seriell\n" );
      break;
    }
    case TYPE_S7_MPI_USB:
    {
      printf( "PC-Adapter USB\n" );
      break;
    }
    case TYPE_S7_TS_AT:
    {
      printf( "AT-Modem\n" );
      break;
    }
    case TYPE_S7_TS_TAPI:
    {
      printf( "TAPI-Modem\n" );
      break;
    }
    case TYPE_S7_PCCP:
    {
      printf( "Siemens-Geraetetreiber\n" );
      break;
    }
    case TYPE_S7_PPI:
    {
      printf( "PPI-Adapter\n" );
      break;
    }
    default:
    {
      printf( "unzulaessigen Verbindungsweg (%d)!\n", DevType );
      return( AGL40_DEVICE_NOT_SUPPORTED );
    }
  }
  printf( "\n" );

  //
  // Now we can put our mind at rest and go on
  //
  RetVal = AGL_ReadMLFBNr( pParas->ConnNr, &MLFBNr, 1, 0 );
  if( RetVal == AGL40_SUCCESS )
  {
    printf( "Kommuniziere mit SPS %s\n", MLFBNr.MLFB );
    printf( "\n" );
  }
  else
  {
    if( pParas->Verbose )
    {
      ShowError( RetVal, "AGL_ReadMLFBNr" );
    }
```

```
}

RetVal = AGL_ReadOpState( pParas->ConnNr, &OpState, 1, 0 );
if( RetVal == AGL40_SUCCESS )
{
  switch( OpState )
  {
    case OPSTATE_STOP:
    {
      printf( "Die CPU ist im Stop\n" );
      break;
    }
    case OPSTATE_START:
    {
      printf( "Die CPU ist im Anlauf\n" );
      break;
    }
    case OPSTATE_RUN:
    {
      printf( "Die CPU ist im Run\n" );
      break;
    }
    default:
    case OPSTATE_UNKNOWN:
    {
      printf( "Die CPU ist in einem unbekannten Betriebszustand\n" );
      break;
    }
  }
  printf( "\n" );
}
else
{
  if( pParas->Verbose )
  {
    ShowError( RetVal, "AGL_ReadOpState" );
  }
}

RetVal = AGL_ReadPLCInfo( pParas->ConnNr, &PLCInfo, 1, 0 );
if( RetVal == AGL40_SUCCESS )
{
  printf( "Anzahl Eingangsbytes im Prozessabbild: %d\n", PLCInfo.PAE );
  printf( "Anzahl Ausgangsbytes im Prozessabbild: %d\n", PLCInfo.PAA );
  printf( "Anzahl Merkerbytes ..................: %d\n", PLCInfo.Flags );
  printf( "Anzahl Zeiten .......................: %d\n", PLCInfo.Timer );
  printf( "Anzahl Zaehler ......................: %d\n", PLCInfo.Counter );
  printf( "Groesse des Lokaldatenspeichers .....: %d\n", PLCInfo.LocalData );
  printf( "\n" );
}
else
{
  if( pParas->Verbose )
  {
    ShowError( RetVal, "AGL_ReadPLCInfo" );
  }
}

RetVal = AGL_ReadCycleTime( pParas->ConnNr, &CycleTime, 1, 0 );
if( RetVal == AGL40_SUCCESS )
{
  printf( "Die aktuelle Zykluszeit betraegt %d ms\n", CycleTime.AktCycleTime );
  printf( "Die minimale Zykluszeit betraegt %d ms\n", CycleTime.MinCycleTime );
  printf( "Die maximale Zykluszeit betraegt %d ms\n", CycleTime.MaxCycleTime );
  printf( "\n" );
}
else
{
  if( pParas->Verbose )
  {
```

```
      ShowError( RetVal, "AGL_ReadCycleTime" );
    }
  }

  RetVal = AGL_GetPLCClock( pParas->ConnNr, &Zeit, 1, 0 );
  if( RetVal == AGL40_SUCCESS )
  {
    //
    // We transform the BCD time into a educible format
    //
    SYSTEMTIME ST;
    AGL_TOD2SysTime( &Zeit, &ST );
    printf( "Datum auf der SPS ......: %02d.%02d.%d\n",
            ST.wDay, ST.wMonth, ST.wYear );
    printf( "Uhrzeit auf der SPS ....: %02d:%02d:%02d.%03d\n",
            ST.wHour, ST.wMinute, ST.wSecond, ST.wMilliseconds );
  }
  else
  {
    if( pParas->Verbose )
    {
      ShowError( RetVal, "AGL_GetPLCClock" );
    }
  }

  return( RetVal );
}
```

Now we change the following things in the main program:

```
  if( (RetVal = OpenConnection( &Paras )) == AGL40_SUCCESS )
  {
    //
    ' Hooray, we have a connection to the PLC and can start
    ' At this point you can insert any communication functions
    //

    //
    // Erste Schritte Teil 2, (First Steps Part 2)
    //
    ShowPLCInfo( &Paras );

    RetVal = 0;
  }
  else
  {
    if( !Paras.Verbose )
    {
      //
      // Even if we should be quiet, this message appears at any case
      //
      printf( "Abbruch wegen Fehler %08X\n"
              "Bitte beliebige Taste zum Beenden druecken ...", RetVal );
    }
    RetVal = 1;
  }
```

Please start and compile the program. You will receive the following soft copy:

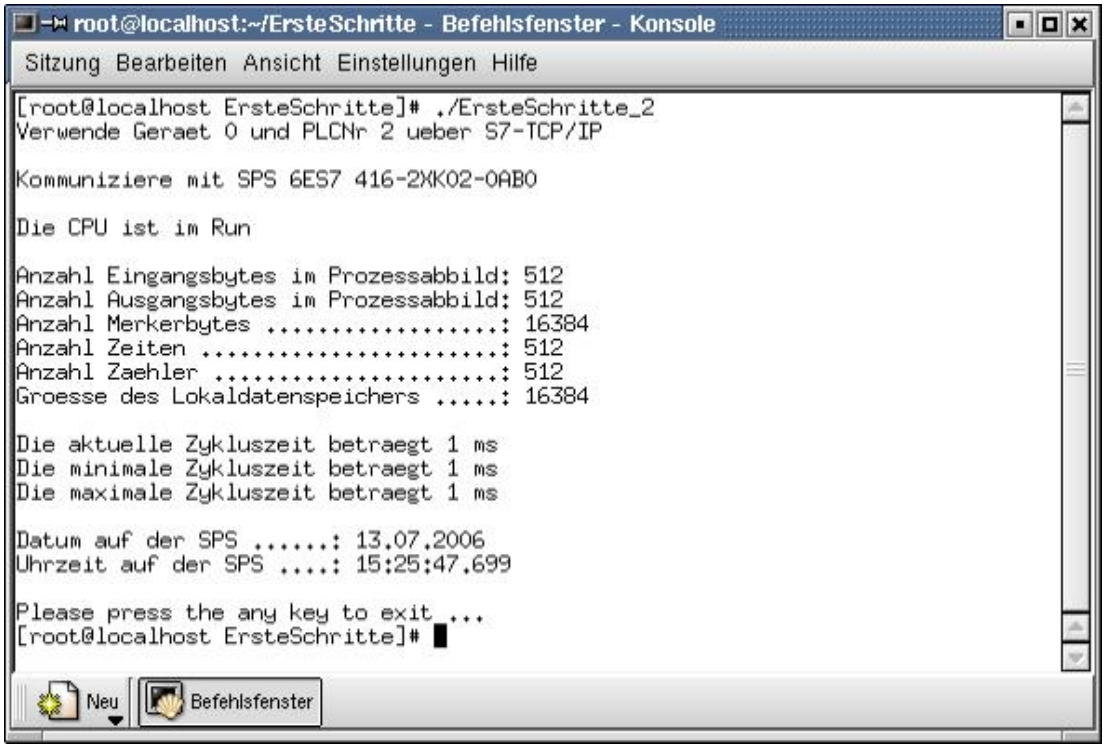


Figure 23: »ErsteSchritte_2», FirstSteps Linux

6.1.3 Reading out the diagnostics buffer

In this chapter we want to output the content of the diagnostics buffer. For this we add the following function into the module and call it after the connection establishment.

```
/*****************************************************************************

 Funktionsname  : ShowDiagBuffer

 Parameter      : LPPROG_PARAS pParas   Struktur mit den Programmparametern

 Beschreibung   : Diese Funktion gibt den Inhalt des Diagnosepuffers aus.

 Rückgabewert   : Den Rückgabewert der fehlerhaften Funktion oder AGL40_SUCCESS

 Erstellt       : 15.05.2006  RH

 Geändert       : 15.05.2006  RH

 *****************************************************************************/

int ShowDiagBuffer( LPPROG_PARAS pParas )
{
  int       RetVal = AGL40_SUCCESS;

  //
  // There are two ways to read out the diagnostics buffer:
  // Either we inquire the amount of parametrized data, assigning the memory,
  // reading out the diagnostics buffer, output the text and release the memory.
  // Or we select how much entries are of interest to us, indicate this number
  // when reading the diagnostics buffer and output the text.
  // Each entry in the diagnostics buufer needs 20 bytes. In addition 8 bytes are
  // needed for header info. Normally, there are 120 entries, they can be changed
  // via the HW-config
  // Therefore the following method arises for the last 30 entries:
  //
  int Entrys = 30;
  BYTE DiagBuff[8+20*30];
  RetVal = AGL_ReadDiagBuffer( pParas->ConnNr, &Entrys, DiagBuff, 1, 0 );
  if( RetVal == AGL40_SUCCESS )
  {
    printf( "Anzahl gelesener Diagnosepuffereintraege: %d\n", Entrys );
    for( int i=0; i<Entrys; i++ )
    {
      char Buff1[256];

      AGL_GetDiagBufferEntry( i, DiagBuff, Buff1, sizeof( Buff1 ) );
#if defined( _CONSOLE )
      CharToOem( Buff1, Buff1 );        // Is necessary when using a console
                                        // application due to mutations
#endif
      printf( "%3d: %s\n", i+1, Buff1 );
    }
  }
  else
  {
    if( pParas->Verbose )
    {
      ShowError( RetVal, "AGL_ReadDiagBuffer" );
    }
  }
  return( RetVal );
}
```

Now we change the follwing things in the main program:

```
if( (RetVal = OpenConnection( &Paras )) == AGL40_SUCCESS )
{
  //
  // Hooray, we have a connection to the PLC and can start
  // At this point you can insert any communication functions
  //
  //

  //
  // Erste Schritte Teil 3, (First Steps Part 3)
  //
  ShowDiagBuffer( &Paras );

  RetVal = 0;
}
else
{
  if( !Paras.Verbose )
  {
    //
    // Even if we should be quiet, this message appears at any case
    //
    printf( "Abbruch wegen Fehler %08X\n"
            "Bitte beliebige Taste zum Beenden druecken ...", RetVal );
  }
  RetVal = 1;
}
```

Please compile and start the program. You will receive the following soft copy:

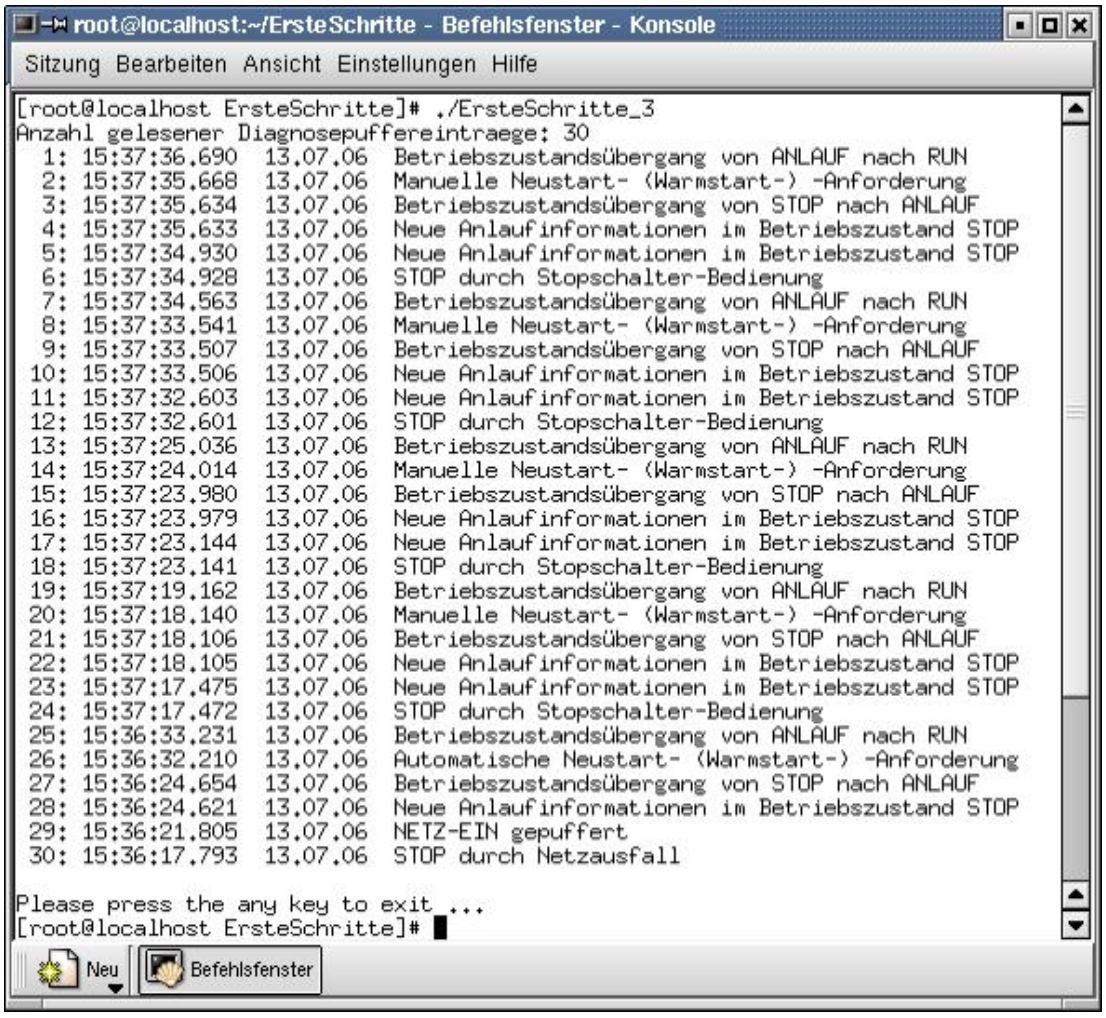


Figure 24: »ErsteSchritte_3», FirstSteps Linux

6.1.4 Reading data from different sections

In this chapter we want to approach the most interesting thing in ACCON-AGLink; reading data from different sections. For this we add the following function to the module and call it after the connection establishment.

```
/*****************************************************************************

 Function name  : ShowReadMemory

 Parameter      : LPPROG_PARAS pParas   Structure including program parameters

 Description    : This function reads out different memory sections with a
                  function call.

 Return value   : Return value of the faulty function or AGL40_SUCCESS

 Created        : 15.05.2006  RH

 Changed        : 15.05.2006  RH

 *****************************************************************************/

int ShowReadMemory( LPPROG_PARAS pParas )
{
  DATA_RW40 RW[8];        // We are reading here 8 different sections at most
  LPDATA_RW40 pRW;        // For always the same access
  BYTE  bBuff[8][16];     // Buffer for byte read access
  WORD  wBuff[8][8];      // Buffer for word read access
  DWORD dwBuff[8][4];     // Buffer for double word access
  int RetVal = AGL40_SUCCESS;

  printf( "Lese Daten aus verschiedenen Bereichen in einem Aufruf...\n\n" );

  //
  // At first we are performing a AGL_ReadMixEx on different and always present
  // memory sections(E/A/M/T/Z).
  //
  pRW = &RW[0];
  pRW->OpArea = AREA_IN;      // We are reading from section PA inputs
  pRW->OpType = TYP_BYTE;     // We are reading byte elements
  pRW->OpAnz  = 16;           // We are reading 16 elements
  pRW->DBNr   = 0;            // The DB number is assigned to 0
  pRW->Offset = 0;            // We are reading from EB 0
  pRW->BitNr  = 0;            // The bit number is assigned to 0
  pRW->Buff   = bBuff[0];     // The results have to be in this section

  pRW = &RW[1];
  pRW->OpArea = AREA_OUT;     // We are reading from section PA outputs
  pRW->OpType = TYP_BYTE;     // We are reading byte elements
  pRW->OpAnz  = 16;           // We are reading 16 elements
  pRW->DBNr   = 0;            // The DB number is assigned to 0
  pRW->Offset = 0;            // We are reading from AB 0
  pRW->BitNr  = 0;            // The bit number is assigned to 0
  pRW->Buff   = bBuff[1];     // The results have to be in this section

  pRW = &RW[2];
  pRW->OpArea = AREA_FLAG;    // We are reading from section marker
  pRW->OpType = TYP_BYTE;     // We are reading byte elements
  pRW->OpAnz  = 16;           // We are reading 16 elements
  pRW->DBNr   = 0;            // The DB number is assigned to 0
  pRW->Offset = 8;            // We are reading from MB 8
  pRW->BitNr  = 0;            // The bit number is assigned to 0
  pRW->Buff   = bBuff[2];     // The results have to be in this section

  pRW = &RW[3];               // And now the same word by word
  pRW->OpArea = AREA_FLAG;    // We are reading from section marker
```

```
pRW->OpType = TYP_WORD;     // We are reading word elements
pRW->OpAnz  = 8;            // We are reading 8 elements
pRW->DBNr   = 0;            // The DB number is assigned to 0
pRW->Offset = 8;            // We are reading from MB 8!!
pRW->BitNr  = 0;            // The bit number is assigned to 0
pRW->Buff   = wBuff[3];     // The results have to be in this section

pRW = &RW[4];               // And now the same double word by doubl word
pRW->OpArea = AREA_FLAG;    // We are reading from section marker
pRW->OpType = TYP_DWORD;    // We are reading double word elements
pRW->OpAnz  = 4;            // We are reading 4 elements
pRW->DBNr   = 0;            // The DB number is assigned to 0
pRW->Offset = 8;            // We are reading from MB 8 !!
pRW->BitNr  = 0;            // The bit number is assigned to 0
pRW->Buff   = dwBuff[4];    // The results have to be in this section

pRW = &RW[5];
pRW->OpArea = AREA_COUNTER; // We are reading from section counter
pRW->OpType = TYP_COUNTER;  // We are reading counter elements
pRW->OpAnz  = 8;            // We are reading 8 elements
pRW->DBNr   = 0;            // The DB number is assigned to 0
pRW->Offset = 0;            // We are reading from Z 0
pRW->BitNr  = 0;            // The bit number is assigned to 0
pRW->Buff   = wBuff[5];     // The results have to be in this section

pRW = &RW[6];
pRW->OpArea = AREA_TIMER;   // We are reading from section timers
pRW->OpType = TYP_TIMER;    // We are reading timer elements
pRW->OpAnz  = 8;            // We are reading 8 elements
pRW->DBNr   = 0;            // The DB number is assigned to 0
pRW->Offset = 16;           // We are reading from b T 16
pRW->BitNr  = 0;            // The bit number is assigned to 0
pRW->Buff   = wBuff[6];     // The results have to be in this section

pRW = &RW[7];
pRW->OpArea = AREA_TIMER;   // We are reading from section timers
pRW->OpType = TYP_TIMER;    // We are reading timer elements
pRW->OpAnz  = 8;            // We are reading 8 elements
pRW->DBNr   = 0;            // The DB number is assigned to 0
pRW->Offset = 2048;         // We are reading from T 2048 => Errors occur!!
pRW->BitNr  = 0;            // The bit number is assigned to 0
pRW->Buff   = wBuff[7];     // The results have to be in this section

RetVal = AGL_ReadMixEx( pParas->ConnNr, RW, 8, 1, 0 );
if( RetVal == AGL40_SUCCESS )
{
  for( int i=0; i<8; i++ )
  {
    //
    // Every structure has to be checked
    //
    if( RW[i].Result == AGL40_SUCCESS )
    {
      switch( RW[i].OpType )
      {
        case TYP_BIT:
        case TYP_BYTE:
        {
          for( int j=0; j<16; j++ )
          {
            printf( "%02X  ", bBuff[i][j] );
          }
          printf( "\n" );
          break;
        }
        case TYP_WORD:
        case TYP_COUNTER:
        case TYP_TIMER:
        {
          for( int j=0; j<8; j++ )
```

```
            {
              printf( " %04X   ", wBuff[i][j] );
            }
            printf( "\n" );
            break;
          }
          case TYP_DWORD:
          {
            for( int j=0; j<4; j++ )
            {
              printf( "   %08X     ", dwBuff[i][j] );
            }
            printf( "\n" );
            break;
          }
        }
      }
      else
      {
        ShowError( RW[i].Result, "AGL_ReadMixEx" );
      }
    }
  }
  else
  {
    if( pParas->Verbose )
    {
      ShowError( RetVal, "AGL_ReadMixEx" );
    }
  }
  printf( "\n" );

  //
  // And now we are searching for dynamic sections and reading them out (= DBs)
  //
  int DBCount;

  RetVal = AGL_ReadDBCount( pParas->ConnNr, &DBCount, 1, 0 );
  if( RetVal == AGL40_SUCCESS )
  {
    if( DBCount > 0 )
    {
      WORD DBList[32];

      printf( "%d Datenbausteine vorhanden.\n", DBCount );
      if( DBCount > sizeof(DBList)/sizeof(DBList[0]) )
      {
        //
        // Only the first modules are interesting for us
        //
        DBCount = sizeof(DBList)/sizeof(DBList[0]);
      }
      RetVal = AGL_ReadDBList( pParas->ConnNr, &DBCount, DBList, 1, 0 );
      if( RetVal == AGL40_SUCCESS )
      {
        int DBs = 0, DBLen, i;

        for( i=0; i<DBCount; i++ )
        {
          RetVal = AGL_ReadDBLen( pParas->ConnNr, DBList[i], &DBLen, 1, 0 );
          if( RetVal == AGL40_SUCCESS )
          {
            if( DBLen >= 16 )
            {
              //
              // We can read out this module
              //
              printf( "Lese 16 Bytes aus DB %d\n", DBList[i] );
              pRW = &RW[DBs];
              pRW->OpArea = AREA_DATA;    // We are reading from section data
```

```
            pRW->OpType = TYP_BYTE;     // We are reading byte elements
            pRW->OpAnz  = 16;           // We are reading 16 elements
            pRW->DBNr   = DBList[i];    // We are entering the DB number here
            pRW->Offset = 0;            // We are reading from DBB 0
            pRW->BitNr  = 0;            // The bit number is assigned to 0
            pRW->Buff   = bBuff[DBs];   // The results have to be in this
                                           section
            if( ++DBs == 8 )
            {
              break;
            }
          }
        }
        else
        {
          if( pParas->Verbose )
          {
            ShowError( RetVal, "AGL_ReadDBLen" );
          }
        }
      }
      if( DBs > 0 )
      {
        RetVal = AGL_ReadMixEx( pParas->ConnNr, RW, DBs, 1, 0 );
        if( RetVal == AGL40_SUCCESS )
        {
          for( i=0; i<DBs; i++ )
          {
            //
            // Jede Struktur muss geprüft werden
            //
            if( RW[i].Result == AGL40_SUCCESS )
            {
              for( int j=0; j<16; j++ )
              {
                printf( "%02X  ", bBuff[i][j] );
              }
              printf( "\n" );
            }
            else
            {
              ShowError( RW[i].Result, "AGL_ReadMixEx" );
            }
          }
        }
        else
        {
          if( pParas->Verbose )
          {
            ShowError( RetVal, "AGL_ReadMixEx" );
          }
        }
      }
      else
      {
        printf( "Keine Datenbausteine für Testzwecke vorhanden\n" );
      }
    }
    else
    {
      if( pParas->Verbose )
      {
        ShowError( RetVal, "AGL_ReadDBList" );
      }
    }
  }
  else
  {
    printf( "Keine Datenbausteine für Testzwecke vorhanden\n" );
  }
```

```
  }
  else
  {
    if( pParas->Verbose )
    {
      ShowError( RetVal, "AGL_ReadDBCount" );
    }
  }
  return( RetVal );
}
```

Now we change the following things in the main progam:

```
  if( (RetVal = OpenConnection( &Paras )) == AGL40_SUCCESS )
  {
    //
      // Hooray, we have a connection to the PLC and can start
      // At this point you can insert any communication functions
      //
    //

    //
    // Erste Schritte Teil 4, (First Steps Part 4)
    //
    ShowReadMemory( &Paras );

    RetVal = 0;
  }
  else
  {
    if( !Paras.Verbose )
    {
      //
      // Even if we should be quiet, this message appears at any case!
      //
      printf( "Abbruch wegen Fehler %08X\n"
              "Bitte beliebige Taste zum Beenden druecken ...", RetVal );
    }
    RetVal = 1;
  }
```

Please compile and start the program. You will receive the following soft copy:

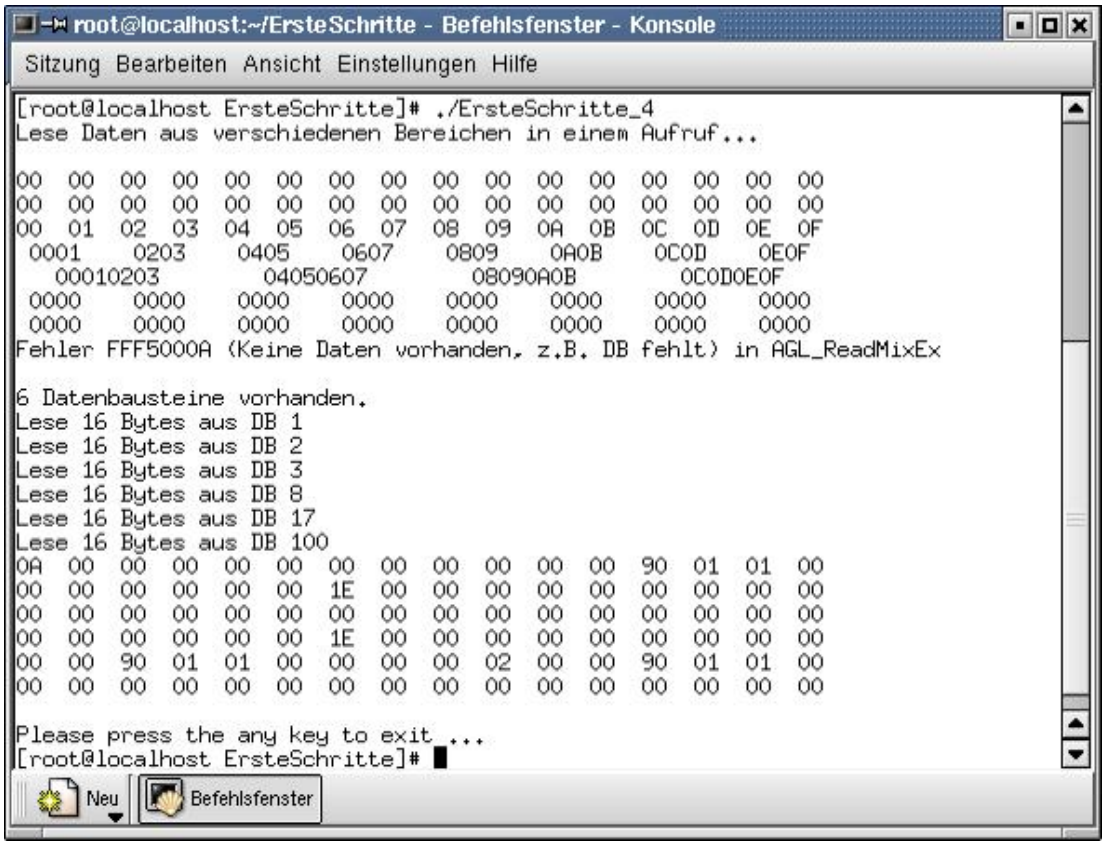


Figure 25: »ErsteSchritte_4«, FirstSteps Linux

7 Programming reference administration

7.1 General Functions

7.1.1 Unlock respectively activate AGLink (AGL_Activate)

This function activates the functionality of the ACCON-AGLink-DLL you purchased. It has to be called only once on propgram start with the provided key. This makes sure that your provided ACCON-AGLink-DLL including applications can only be used by you.

Note:
This is only valid for the developer's license, not for Einzelplatz license.

C/C++-Syntax:

```
void WINAPI AGL_Activate( LPSTR Key );
```

Visual Basic-Syntax:

```
Declare Sub AGL_Activate Lib "AGLink40.DLL" (ByVal Key As String)
```

Delphi-Syntax:

```
procedure AGL_Activate( Key: PChar ); stdcall;
```

Parameter:

Key	Activation key of your license

Return value:

none

7.1.2 Calling maximum amount of devices (AGL_GetMaxDevices)

This function returns the maximum available device amount of your ACCON-AGLink version. This number can change in the future as it is not advisable to use a constant.

C/C++-Syntax:

```
int WINAPI AGL_GetMaxDevices( void );
```

Visual Basic-Syntax:

```
Declare Function AGL_GetMaxDevices Lib "AGLink40.DLL" () As Long
```

Delphi-Syntax:

```
function AGL_GetMaxDevices: Integer; stdcall;
```

Parameter:

none

Return value:

The maximum device amount

7.1.3 Calling maximum amount of PLCs (AGL_GetMaxPLCPerDevice)

This function returns the maximum of simultaneously opened PLC connections for each device of your ACCON-AGLink version. But in reality this number can be lower because of possible hardware limitations. E.g. a PC adapter only supports four connections at the same time. The number of the maximum simultaneously opened PLC connections can change in future ACCON-AGLink versions. So it is not advisable to use a constant.

C/C++-Syntax:

```
int WINAPI AGL_GetMaxPLCPerDevice( void );
```

Visual Basic-Syntax:

```
Declare Function AGL_GetMaxPLCPerDevice Lib "AGLink40.DLL" () As
Long
```

Delphi-Syntax:

```
function AGL_GetMaxPLCPerDevice: Integer; stdcall;
```

Parameter:

none

Return value

The maximum PLC number for each device

7.1.4 Calling maximum amount of queues (AGL_GetMaxQueues)

This function returns the maximum number of queues which can be buffered by your version of ACCON-AGLink. This number can change in future versions. So it is not advisable to use a constant.

C/C++-Syntax:

```
int WINAPI AGL_GetMaxQueues( void );
```

Visual Basic-Syntax:

```
Declare Function AGL_GetMaxQueues Lib "AGLink40.DLL" () As Long
```

Delphi-Syntax:

```
function AGL_GetMaxQueues: Integer; stdcall;
```

Parameter:

none

Return value:

The maximum queue number for each device.

7.1.5 Calling Tickcount (AGL_GetTickCount)

This function returns the time period in miliseconds since the computer's last start. Unlike the Windows function GetTickCount(), this function calculates with milisecond granularity and not depending on the operating system with a granularity of 10 or 15 miliseconds. This function can also be used for time measurements. For this purpose the computer must have a high-performance-counter.This is a standard since the introduction of the Pentium CPUs. If not, the value of GetTickCount() will be returned.

C/C++-Syntax:

```
DWORD WINAPI AGL_GetTickCount( void );
```

Visual Basic-Syntax:

```
Declare Function AGL_GetTickCount Lib "AGLink40.DLL" () As Long
```

Delphi-Syntax:

```
function AGL_GetTickCount: LongInt; stdcall;
```

Parameter:

none

Return value

Time period in miliseconds since the last computer start.

7.1.6 Calling microseconds (AGL_GetMicroSecs)

This function returns the time period in microseconds since the computer's last start. This function can also be used for time measurements. For this purpose the computer must have a high-performance-counter.This is a standard since the introduction of the Pentium CPUs. If not, the value of GetTickCount()*1000 will be returned.

C/C++-Syntax:

```
DWORD WINAPI AGL_GetMicroSecs( void );
```

Visual Basic-Syntax:

```
Declare Function AGL_GetMicroSecs Lib "AGLink40.DLL" () As Long
```

Delphi-Syntax:

```
function AGL_GetMicroSecs: LongInt; stdcall;
```

Parameter:

none

Return value

Time period in microseconds since the last computer start.

7.1.7 Adjusting time function for result (AGL_UseSystemTime)

With this function you can adjust if the time in the result structure should be determined with GetSystemTime (Flag != 0) or GetLocalTime (Flag == 0). If function not called GetLocalTime will be used by default.

C/C++-Syntax:

```
void WINAPI AGL_UseSystemTime( int Flag );
```

Visual Basic-Syntax:

```
Declare Sub AGL_UseSystemTime Lib "AGLink40.DLL" (ByVal Flag As
Long)
```

Delphi-Syntax:

```
procedure AGL_UseSystemTime ( Flag: Integer ); stdcall;
```

Parameter:

Flag	Flag if GetSystemTime or GetLocalTime is used

Return value

none

7.1.8 Determining error message to error number (AGL_GetErrorMsg)

This function determines the error message to the respective error number. In doing so the content of the data file »AGLink40_Error.TXT« will be evaluated. On the first function call this will be read in and is available without any time lag. This data file will be searched for in the folder where the Glink40.DLL respectively the Shared Object is located.

Note:
Alternatively, VB programers can look for the function AGL_GetVBErrorMsg in chapter »12 Additional functions for VB respectively VBA«.

C/C++-Syntax:

```
int WINAPI AGL_GetErrorMsg( int ErrNr, LPSTR Msg, int MaxLen );
```

Visual Basic-Syntax:

```
Declare Function AGL_GetErrorMsg Lib "AGLink40.DLL" (ByVal ErrNr As
Long, ByVal Msg As String, ByVal MaxLen As Long) As Long
```

Delphi-Syntax:

```
function AGL_GetErrorMsg( ErrNr: Integer; Msg: PChar; MaxLen:
Integer ): Integer; stdcall;
```

Parameter:

ErrNr	Error number
Msg	Indicator on String for error message
MaxLen	Length of Strings

Return value

Number of characters copied into Msg

7.1.9 Loading error message data file (AGL_LoadErrorFile)

This function loads the error message data file. Normally, AGLink40_Error.TXT will be used. But you can also indicate AGLink40_Error.ENG if you want to output error messages in English.

C/C++-Syntax:

```
int WINAPI AGL_LoadErrorFile( LPSTR FileName );
```

Visual Basic-Syntax:

```
Declare Function AGL_LoadErrorFile Lib "AGLink40.DLL" (ByVal
FileName As String) As Long
```

Delphi-Syntax:

```
function AGL_LoadErrorFile ( FileName: PChar ): Integer; stdcall;
```

Parameter:

FileName	The error data file's name. At present AGLink40_Error.TXT and AGLink40_Error.ENG are the provided error message data files.

Return value:

AGL40_SUCCESS	If all worked properly
< 0	An error number (s. AGL_Defines.H, AGL_Defines.BAS or AGLink40.PAS)

7.1.10 Calling external configuration program (AGL_Config)

This function calls the program »AGLink40_Config.EXE«. It has to be in the searching path or in the actual working directory of your application.

C/C++-Syntax:

```
int WINAPI AGL_Config( int DevNr );
```

Visual Basic-Syntax:

```
Declare Function AGL_Config Lib "AGLink40.DLL" (ByVal DevNr As Long)
As Long
```

Delphi-Syntax:

```
function AGL_Config( DevNr: Integer ): Integer; stdcall;
```

Parameter:

DevNrDie	Number of Devices (0 ... MAX_DEVICES-1) if only a certain device should be configured or –1 when all devices should be configured.

Return value

AGL40_SUCCESS	If all worked properly
< 0	An error number (s. AGL_Defines.H, AGL_Defines.BAS or AGLink40.PAS)

7.1.11 Removing DLL during dynamic laoding (AGL_UnloadDyn)

This is a very secure function when linking AGLink40.lib statically. Because it does absolutely nothing. When the ACCON-AGLink40.DLL was integrated and loaded dynamically via AGLink40.C, then it will be eradicated from the memory.

C/C++-Syntax:

```
void WINAPI AGL_UnloadDyn( void );
```

Visual Basic-Syntax:

```
Declare Sub AGL_UnloadDyn Lib "AGLink40.DLL" ()
```

Delphi-Syntax:

```
procedure AGL_UnloadDyn: stdcall;
```

Parameter:

none

Return value:

none

7.2 Job administration

If the communication functions are called with a Timeout <= 0, they will be returned immediately. Then the complete communication is asynchronous, in the background, to your application. If >= 0 the return value is the internal job number. With this you can call the status of your job, cancel the job (if not in progress, yet) or install another notification handler. If the return value is <0 you will receive an error.

When processing asynchronous you can generate a meesage with the job number at processing end. Alternatively, you can set a notification on the connection or on the whole device. These methods can be mixed, too. If a notification is set on the job there will be no messages transmitted to the connection or the device. If a notification is put on the connection, there will be no notification set on the device at job end.

The notification structure has the following build up:

```
//
// Structure for notification mechanisms
//
// Attention: Not used elements have to be initialized with 0!!!
//
typedef struct tagNOTIFICATION
{
  HWND        hWnd;                   // Window handle for notifiaction
                                      // via
                                      // Message
  UINT        WndMessage;             // Message number
  DWORD       dwThreadID;             // If message should be send directly to
                                      // a thread
  UINT        ThreadMessage;          // Desired message for thread
  HANDLE      hEvent;                 // Notification about event
  CB_FUNC     CB;                     // Callback implementation
} NOTIFICATION, *LPNOTIFICATION;
```

Please remember that an own notification structure will be generated and set by the functions AGL_WaitForJob and AGL_WaitForJobEx. This strucutre overwrites a possible prior notification info set by AGL_SetJobNotification.

Due to reasons of simplification an used event should be generated as Autoreset. Otherwise ResetEvent has to be called all the time.

Note:
At present only the callback method is implemented in Linux. The result structure has the following build up:

```
//
// Structure for the communication's result
//
typedef struct tagRESULT40
{
  long        State;                  // Communication status
  long        ErrCode;                // Error code when canceling
                                      // communication
  SYSTEMTIME  SysTime;                // Date and time at the end of
                                      // communication
  long        SError;                 // If necessary hardware error code
  UINT        UserVal;                // Value for free use by the programer
} RESULT40, *LPRESULT40;
```

7.2.1 Setting notification on device (AGL_SetDevNotification)

By this function you inform the ACCON-AGLink that you want to receive a notification when a job has been completed or canceled on this device.

C/C++-Syntax:

```
int WINAPI AGL_SetDevNotification
( int DevNr, LPNOTIFICATION pN );
```

Visual Basic-Syntax:

```
Declare Function AGL_SetDevNotification Lib "AGLink40.DLL" (ByVal
DevNr As Long, ByRef pN as NOTIFICATION) As Long
```

Delphi-Syntax:

```
function AGL_SetDevNotification( DevNr: Integer; var pN:
NOTIFICATION ): Integer; stdcall;
```

Parameter:

DevNr	Device number
pN	Structure including notification info

Return value:

AGL40_SUCCESS	If all worked properly
< 0	Error number (s. AGL_Defines.H, AGL_Defines.BAS or AGLink40.PAS)

7.2.2 Setting notification on connection (AGL_SetConnNotification)

By this function you inform the ACCON-AGLink that you want to receive a notification when a job has been completed or canceled on this connection. If this function and AGL_SetDevNotification are called, then only the notifiaction of AGL_SetConnNotification will be performed on connection orientated functions. Thus this function covers AGL_SetDevNotification.

C/C++-Syntax:

```
int WINAPI AGL_SetConnNotification ( int ConnNr,
LPNOTIFICATION pN );
```

Visual Basic-Syntax:

```
Declare Function AGL_SetConnNotification Lib "AGLink40.DLL" (ByVal
ConnNr As Long, ByRef pN as NOTIFICATION) As Long
```

Delphi-Syntax:

```
function AGL_SetConnNotification( ConnNr: Integer; var pN:
NOTIFICATION ): Integer; stdcall;
```

Parameter:

ConnNr	Connection handle of AG_PLCConnect bzw. AGL_PLCConnectEx
pN	Structure including notification info

Return value:

AGL40_SUCCESS	If all worked properly
< 0	Error number (s. AGL_Defines.H, AGL_Defines.BAS or AGLink40.PAS)

7.2.3 Setting notification on job (AGL_SetJobNotification)

By this function you inform the ACCON-AGLink that you want to receive a notification when a job has been completed or canceled on this job. If this function and AGL_SetDevNotification or AGL_SetConnNotification are called, then on this job only the notifiaction of AGL_SetConnNotification will be performed. Thus this function covers AGL_SetDevNotification and AGL_SetConnNotification.

C/C++-Syntax:

```
int WINAPI AGL_SetJobNotification ( int JobNr, LPNOTIFICATION pN );
```

Visual Basic-Syntax:

```
Declare Function AGL_SetJobNotification Lib "AGLink40.DLL" (ByVal
JobNr As Long, ByRef pN as NOTIFICATION) As Long
```

Delphi-Syntax:

```
function AGL_SetJobNotification( JobNr: Integer; var pN:
NOTIFICATION ): Integer; stdcall;
```

Parameter:

DevNr	Device number
JobNr	Job number
pN	Structure including notification info

Return value:

AGL40_SUCCESS	If all worked properly
< 0	An error number (s. AGL_Defines.H, AGL_Defines.BAS or AGLink40.PAS)

7.2.4 Waiting for job (AGL_WaitForJob)

By this function you wait for the end of an asynchronous-started job. This function returns the communication's error code automatically.

C/C++-Syntax:

```
int WINAPI AGL_WaitForJob( int DevNr, int JobNr );
```

Visual Basic-Syntax:

```
Declare Function AGL_WaitForJob Lib "AGLink40.DLL" (ByVal DevNr As
Long, ByVal JobNr As Long) As Long
```

Delphi-Syntax:

```
function AGL_WaitForJob( DevNr: Integer; JobNr: Integer ): Integer;
stdcall;
```

Parameter:

DevNr	Device number
JobNr	Job number from the asynchronous call

Return value:

AGL40_SUCCESS	If all worked properly
< 0	an error number (s. AGL_Defines.H, AGL_Defines.BAS or AGLink40.PAS)

7.2.5 Waiting for job and fill result structure (AGL_WaitForJobEx)

By this function you wait for the end of an asynchronous-started job. This function returns the communication's error code automatically and fills in the result structure.

C/C++-Syntax:

```
int WINAPI AGL_WaitForJobEx( int DevNr, int JobNr, LPRESULT40 pR );
```

Visual Basic-Syntax:

```
Declare Function AGL_WaitForJobEx Lib "AGLink40.DLL" (ByVal DevNr As
Long, ByVal JobNr As Long, pR as Result40) As Long
```

Delphi-Syntax:

```
function AGL_WaitForJobEx( DevNr: Integer; JobNr: Integer; var pR:
Result40 ): Integer; stdcall;
```

Parameter:

DevNr	Device number
JobNr	Job number from the asynchronous call
pR	Indicator on result structure

Return value:

AGL40_SUCCESS	If all worked properly
< 0	An error messge (s. AGL_Defines.H, AGL_Defines.BAS or AGLink40.PAS)

7.2.6 Deleting job (AGL_DeleteJob)

This function deletes a job with a certain number from the job processing queue.

C/C++-Syntax:

```
int WINAPI AGL_DeleteJob( int DevNr, int JobNr );
```

Visual Basic-Syntax:

```
Declare Function AGL_DeleteJob Lib "AGLink40.DLL" (ByVal DevNr As
Long, ByVal JobNr As Long) As Long
```

Delphi-Syntax:

```
function AGL_DeleteJob( DevNr: Integer; JobNr: Integer ): Integer;
stdcall;
```

Parameter:

DevNr	Device number
JobNr	Job number from the asynchronous call

Return value:

AGL40_SUCCESS	If all worked properly
< 0	An error number (s. AGL_Defines.H, AGL_Defines.BAS or AGLink40.PAS)

7.2.7 Calling a communication result from a job (AGL_GetJobResult)

This function provides the communication result.

C/C++-Syntax:

```
int WINAPI AGL_GetJobResult( int DevNr, int JobNr, LPRESULT40 pR );
```

Visual Basic-Syntax:

```
Declare Function AGL_GetJobResult Lib "AGLink40.DLL" (ByVal DevNr As
Long, ByVal JobNr As Long, pR As Result40) As Long
```

Delphi-Syntax:

```
function AGL_GetJobResult( DevNr: Integer; JobNr: Integer; var pR:
Result ): Integer; stdcall;
```

Parameter:

DevNr	Device number
JobNr	Job number from the asynchronous call
pR	Indicator on result structure

Return value:

AGL40_SUCCESS	If all worked properly
< 0	An error number (s. AGL_Defines.H, AGL_Defines.BAS or AGLink40.PAS)

7.3 Open and close the device

7.3.1 Opening device (AGL_OpenDevice)

This function opens the indicated Device for later use. Through this the memory capacity will be assigned and the protocol threads started according to the settings made.

C/C++-Syntax:

```
int WINAPI AGL_OpenDevice( int DevNr );
```

Visual Basic-Syntax:

```
Declare Function AGL_OpenDevice Lib "AGLink40.DLL" (ByVal DevNr As
Long) As Long
```

Delphi-Syntax:

```
function AGL_OpenDevice( DevNr: Integer ): Integer; stdcall;
```

Parameter:

DevNr	Device number

Return value:

AGL40_SUCCESS	If all worked properly
< 0	An error number (s. AGL_Defines.H, AGL_Defines.BAS or AGLink40.PAS)

7.3.2 Closing device (AGL_CloseDevice)

This function closes the indicated Device. Through this the protocol threads will be quit and the assigned memory capacity released.

C/C++-Syntax:

```
int WINAPI AGL_CloseDevice( int DevNr );
```

Visual Basic-Syntax:

```
Declare Function AGL_CloseDevice Lib "AGLink40.DLL" (ByVal DevNr As
Long) As Long
```

Delphi-Syntax:

```
function AGL_CloseDevice( DevNr: Integer ): Integer; stdcall;
```

Parameter:

DevNr	Device number

Return value:

AGL40_SUCCESS	If all worked properly
< 0	An error number (s. AGL_Defines.H, AGL_Defines.BAS or AGLink40.PAS)

8 Programming reference communication functions

The functions of this chapter can only be executed, if the device has been opened with the function AGL_OpenDevice.

For the functions in this chapter you can submit the parameter Timeout. Thereby you ascertain whether the function shall be returned immediately or only after the complete function processing. When you don't want to wait until the function finished, you get a value greater or equal zero in the success case back. This is an internal job number for checking the status of the job. If you want to wait until the function has finished (Timeout > 0), you directly get the error code of the communication. Waiting till the function has finished is absolutely harmless for the computing time.

The parameter Timeout can be used as follows. Timeout== 0 means asynchronous to the timeout value adjusted in the configuration. Timeout== 1 means synchronous to the timeout value adjusted in the configuration. Timeout value==INFINITE (defined as –1) means asynchronous without timeout monitoring. This can be reasonable e.g. for AGL_Breceive. A Timeout value <0 will be monitored asynchronous to this value. E.g. a Timeout value of-1000 means that the function will be called asynchronous and if this function hasn't finished after the above-mentioned period it will be canceled. Using a Timeout value >1, this period will be waited for the function end. If the function is not finished until then it will be canceled.

Attention:
When using asynchronous functionality you have to be sure that all buffers handed over to the communication driver are valid during the entire processing time. And assure that the buffers are only used by the communication driver. Furthermore you have to implement and guarantee the reentrance ability into your application. Otherwise strange results can occur up to programm crashes.

8.1 Function reference adapter

8.1.1 Establishing connection to remote station (AGL_DialUp)

This function dials into the remote station if the device is TYPE_S7_TS_AT or TYPE_S7_TS_TAPI. In all other cases the function sends back a success, immediately. And due to reasons of compatibility it can always be called after AGL_OpenDevice and before AGL_InitAdapter.

C/C++-Syntax:

```
int WINAPI AGL_DialUp( int DevNr, int Timeout, long UserVal );
```

Visual Basic-Syntax:

```
Declare Function AGL_DialUp Lib "AGLink40.DLL" (ByVal DevNr As Long,
ByVal Timeout As Long, ByVal UserVal As Long) As Long
```

Delphi-Syntax:

```
function AGL_DialUp( DevNr: Integer; Timeout: Integer; UserVal:
LongInt ): Integer; stdcall;
```

Parameter:

DevNr	Device number
Timeout	Timeout value which has to be used
UserVal	Value for free usage

Return value:

>= 0	Job number asynchronous call respectively AGL40_SUCCESS if synchronous call
< 0	Error number (s. AGL_Defines.H, AGL_Defines.BAS or AGLink40.PAS)

8.1.2 Releasing connection to remote station (AGL_HangUp)

This function quits the connection to the remote station if the device is TYPE_S7_TS_AT or TYPE_S7_TS_TAPI. In all other cases the function sends back a success, immediately. And due to reasons of compatibility it can always be called after AGL_ExitAdapter and before AGL_CloseDevice.

C/C++-Syntax:

```
int WINAPI AGL_HangUp( int DevNr, int Timeout, long UserVal );
```

Visual Basic-Syntax:

```
Declare Function AGL_HangUp Lib "AGLink40.DLL" (ByVal DevNr As Long,
ByVal Timeout As Long, ByVal UserVal as Long) As Long
```

Delphi-Syntax:

```
function AGL_HangUp( DevNr: Integer; Timeout: Integer; UserVal:
LongInt ): Integer; stdcall;
```

Parameter:

DevNr	Device number
Timeout	Timeout value which has to be used
UserVal	Value for free usage

Return value:

>= 0	Job number asynchronous call respectively AGL40_SUCCESS if synchronous call
< 0	Error number (s. AGL_Defines.H, AGL_Defines.BAS or AGLink40.PAS)

8.1.3 Initializing communications adapter (AGL_InitAdapter)

This function initializes the communications adapter with the adjusted values.

C/C++-Syntax:

```
int WINAPI AGL_InitAdapter( int DevNr, int Timeout, long UserVal );
```

Visual Basic-Syntax:

```
Declare Function AGL_InitAdapter Lib "AGLink40.DLL" (ByVal DevNr As
Long, ByVal Timeout As Long, ByVal UserVal as Long) As Long
```

Delphi-Syntax:

```
function AGL_InitAdapter( DevNr: Integer; Timeout: Integer; UserVal:
LongInt ): Integer; stdcall;
```

Parameter:

DevNr	Device number
Timeout	Timeout value which has to be used
UserVal	Value for free usage

Return value:

>= 0	Job number asynchronous call respectively AGL40_SUCCESS if synchronous call
< 0	Error number (s. AGL_Defines.H, AGL_Defines.BAS or AGLink40.PAS)

8.1.4 Deinitializing communications adapter (AGL_ExitAdapter)

This function logs off from the communications adapter.

C/C++-Syntax:

```
int WINAPI AGL_ExitAdapter( int DevNr, int Timeout, long UserVal );
```

Visual Basic-Syntax:

```
Declare Function AGL_ExitAdapter Lib "AGLink40.DLL" (ByVal DevNr As
Long, ByVal Timeout As Long, ByVal UserVal as Long) As Long
```

Delphi-Syntax:

```
function AGL_ExitAdapter( DevNr: Integer; Timeout: Integer; UserVal:
LongInt ): Integer; stdcall;
```

Parameter:

DevNr	Device number
Timeout	Timeout value which has to be used
UserVal	Value for free usage

Return value:

>= 0	Job number asynchronous call respectively AGL40_SUCCESS if synchronous call
< 0	Error number (s. AGL_Defines.H, AGL_Defines.BAS or AGLink40.PAS)

8.1.5 Querrying active bus participants (AGL_GetLifelist)

This function determines active and passive participants on MPI or Profibus when accessing a S7. The parameter list is an array with at least 127 entries. After the function's completion there will be for each byte 0x10 (= LL_NONE) for every nonexistent participant, 0x00 (= LL_PASSIVE) for every passive participant, 0x30 (= LL_ACTIVE) for every active participant and 0x20 (= LL_ACTIVE_READY) for every active participant who is prepared to go on the bus.

Attention:
When communicating via TCP/IP the parametrized PLCs will be registered without checking if they are really available.

C/C++-Syntax:

```
int WINAPI AGL_GetLifeList( int DevNr, PBYTE List, int Timeout, long
UserVal );
```

Visual Basic-Syntax:

```
Declare Function AGL_GetLifeList Lib "AGLink40.DLL" (ByVal DevNr As
Long, List As Byte, ByVal Timeout As Long, ByVal UserVal as Long) As
Long
```

Delphi-Syntax:

```
function AGL_GetLifeList( DevNr: Integer; var List: Byte; Timeout:
Integer; UserVal: LongInt ): Integer; stdcall;
```

Parameter:

DevNr	Device number
List	Array for bus participants, at lest 127 bytes
Timeout	Timeout value which has to be used
UserVal	Value for free usage

Return value:

>= 0	Job number asynchronous call respectively AGL40_SUCCESS if synchronous call
< 0	Error number (s. AGL_Defines.H, AGL_Defines.BAS or AGLink40.PAS)

8.1.6 Querrying a directly connected PLC (AGL_GetDirectPLC)

This function determines the directly connected PLC when accessing S7 control via a PC or TS adapter. If using other communication paths you will receive PLC number 255 despite a faultless job processing. Because the directly connected PLC could not be detected.

Attention:
This function only outputs valid values for the directly connected PLC with the device types ACCON-AGLink S7 serial and ACCON-AGLink S7 serial/TS (Device types:TYPE_S7_MPI_SER, TYPE_S7_MPI_USB, TYPE_S7_TS_AT and TYPE_S7_TS_TAPI). All other modules will return PLC number 255.

C/C++-Syntax:

```
int WINAPI AGL_GetDirectPLC( int DevNr, PBYTE pPlc, int Timeout,
long UserVal );
```

Visual Basic-Syntax:

```
Declare Function AGL_GetDirectPLC Lib "AGLink40.DLL" (ByVal DevNr As
Long, pPlc As Byte, ByVal Timeout As Long, ByVal UserVal as Long) As
Long
```

Delphi-Syntax:

```
function AGL_GetDirectPLC( DevNr: Integer; var pPlc: Byte; Timeout:
Integer; UserVal: LongInt ): Integer; stdcall;
```

Parameter:

DevNr	Device number
pPlc	Variable for PLC number
Timeout	Timeout value which has to be used
UserVal	Value for free usage

Return value:

>= 0	Job number asynchronous call respectively AGL40_SUCCESS if synchronous call
< 0	Error number (s. AGL_Defines.H, AGL_Defines.BAS or AGLink40.PAS)

8.2 Function reference PLC

8.2.1 Establishing connection to a PLC (AGL_PLCConnect)

With this function you can establish a connection to a PLC. By calling this function multiple times and with different PLC numbers you can have several active connections, simultaneously. The maximum amount of simultaneous connections for each Device is limited by the constant MAX_PLCS in the communication driver. But it could be possible that the used communications hardware does only support a certain amount of simultaneous connections. Every connection established by AGL_PLCConnect must be released by the function AGL_PLCDisconnect.

C/C++-Syntax:

```
int WINAPI AGL_PLCConnect( int DevNr, int PlcNr, int *ConnNr, int
Timeout, long UserVal );
```

Visual Basic-Syntax:

```
Declare Function AGL_PLCConnect Lib "AGLink40.DLL" (ByVal DevNr As
Long, ByVal PlcNr As Long, ByRef ConnNr as Long, ByVal Timeout As
Long, Byval UserVal As Long) As Long
```

Delphi-Syntax:

```
function AGL_PLCConnect( DevNr: Integer; PlcNr: Integer; var ConnNr:
Integer; TimeOut: Integer; UserVal: LongInt ): Integer; stdcall;
```

Parameter:

DevNr	Device number
PlcNr	Variable for PLC number
ConnNr	Variable for connection handle
Timeout	Timeout value which has to be used
UserVal	Value for free usage

Return value:

>= 0	Job number asynchronous call respectively AGL40_SUCCESS if synchronous call
< 0	Error number (s. AGL_Defines.H, AGL_Defines.BAS or AGLink40.PAS).

8.2.2 Establishing connection to a PLC (AGL_PLCConnectEx)

With this function you can establish a connection to a PLC. By calling this function multiple times and with different PLC numbers you can have several active connections, simultaneously. The maximum amount of simultaneous connections for each device is limited by the constant MAX_PLCS in the communications driver. But it could be possible that the used communications hardware does only support a certain amount of simultaneous connections. Every connection established by AGL_PLCConnect must be released by the function AGL_PLCDisconnect.

The difference between the function AGL_PLCConnect and AGL_PLCConnectEx is that you can indicate the rack and slot number of the CPU. This is necessary e.g. when accessing a S7 control via a CP 342-5. Normally, the rack number is 0 and the slot number is 2. For detailed values please check your hardware configuration.

You can easily check if you have established a connection to the correct CPU, just call the function AGL_ReadMLFBN after AGL_PLCConnect respectively AGL_PLCConnectEx. Compare the MLBF number with the control. Alternatively you can use the check function in the AGLink40_Config.EXE.

C/C++-Syntax:

```
int WINAPI AGL_PLCConnectEx( int DevNr, int PlcNr, int RackNr, int
SlotNr, int *ConnNr, int Timeout, long UserVal );
```

Visual Basic-Syntax:

```
Declare Function AGL_PLCConnectEx Lib "AGLink40.DLL" (ByVal DevNr As
Long, ByVal PlcNr As Long, ByVal RackNr As Long, ByVal SlotNr As
Long, ByRef ConnNr as Long, ByVal Timeout As Long, Byval UserVal As
Long) As Long
```

Delphi-Syntax:

```
function AGL_PLCConnectEx( DevNr: Integer; PlcNr: Integer; RackNr:
Integer; SlotNr: Integer; var ConnNr: Integer; TimeOut: Integer;
UserVal: LongInt ): Integer; stdcall;
```

Parameter:

DevNr	Device number
PlcNr	Number of bus participant
RackNr	PLC rack number
SlotNr	PLC slot number
ConnNr	Variable for connection handle
Timeout	Timeout value which has to be used
UserVal	Value for free usage

Return value:

>= 0	Job number asynchronous call respectively AGL40_SUCCESS if synchronous call
< 0	Error number (s. AGL_Defines.H, AGL_Defines.BAS or AGLink40.PAS)

8.2.3 Releasing connection to a PLC (AGL_PLCDisconnect)

This function releases the connection established by AGL_PLCConnect or AGL_PLCConnectEx.

C/C++-Syntax:

```
int WINAPI AGL_PLCDisconnect( int ConnNr, int TimeOut, long UserVal
);
```

Visual Basic-Syntax:

```
Declare Function AGL_PLCDisconnect Lib "AGLink40.DLL" (ByVal ConnNr
As Long, ByVal TimeOut As Long, ByVal UserVal As Long) As Long
```

Delphi-Syntax:

```
function AGL_PLCDisconnect( ConnNr: Integer; TimeOut: Integer;
UserVal: LongInt ): Integer; stdcall;
```

Parameter:

ConnNr	Variable for connection handle
Timeout	Timeout value which has to be used
UserVal	Value for free usage

Return value:

>= 0	Job number asynchronous call respectively AGL40_SUCCESS if synchronous call
< 0	Error number (s. AGL_Defines.H, AGL_Defines.BAS or AGLink40.PAS)

8.2.4 Reading the PLC's MLFB number (AGL_ReadMLFBNr)

This function determines the MLFB number of the PLC with the indicated connection by the connection handle. There must be an already established connection to the PLC.

C/C++-Syntax:

```
int WINAPI AGL_ReadMLFBNr( int ConnNr, LPMLFB MLFBNr, int Timeout,
long UserVal );
```

Visual Basic-Syntax:

```
Declare Function AGL_ReadMLFBNr Lib "AGLink40.DLL" (ByVal ConnNr As
Long, MLFB As MLFBT, ByVal Timeout As Long, ByVal UserVal As Long)
As Long
```

Delphi-Syntax:

```
function AGL_ReadMLFBNr( ConnNr: Integer; var MLFBNr: TagMLFB;
Timeout: Integer; UserVal: LongInt ): Integer; stdcall;
```

Parameter:

ConnNr	Variable for connection handle
MLFBNr	Indicator on MLFB struture (String with 21 characters)
Timeout	Timeout value which has to be used
UserVal	Value for free usage

Return value:

>= 0	Job number asynchronous call respectively AGL40_SUCCESS if synchronous call
< 0	Error number (s. AGL_Defines.H, AGL_Defines.BAS or AGLink40.PAS)

The MLFBT strucutre has the following build up:

```
typedef struct tagMLFB
{
  BYTE  MLFB[21];          // MLFB number as a zero-terminated string
} MLFB, *LPMLFB;
```

8.2.5 Reading the PLC's extended MLFB number (AGL_ReadMLFBNrEx)

This function determines the MLFB number and the version number of the PLC with the indicated connection by the connection handle as well as the activation. For this the PLC must be already connected.

C/C++-Syntax:

```
int WINAPI AGL_ReadMLFBNrEx( int ConnNr, LPMLFBEX MLFBNr, int
Timeout, int UserVal );
```

Visual Basic-Syntax:

```
Declare Function AGL_ReadMLFBNrEx Lib "AGLink40.DLL" (ByVal ConnNr
As Long, MLFB As MLFBEX, ByVal Timeout As Long, ByVal UserVal As
Long) As Long
```

Delphi-Syntax:

```
function AGL_ReadMLFBNrEx( ConnNr: Integer; var MLFBNr: TagMLFBEX;
Timeout: Integer; UserVal: LongInt ): Integer; stdcall;
```

Parameter:

ConnNr	Variable for connection handle
MLFBNr	Indicator on extended MLFB structure
Timeout	Timeout value which has to be used
UserVal	Value for free usage

Return value:

>= 0	Job number asynchronous call respectively AGL40_SUCCESS if synchronous call
< 0	Error number (s. AGL_Defines.H, AGL_Defines.BAS or AGLink40.PAS)

The MLFBEX structure has the following build up:

```
typedef struct tagMLFBEX
{
  WORD  PLCVer;              // Output state PLC
  WORD  PGASVer;             // Output state PG activation
  BYTE  MLFB[21];            // MLFB number as a zero-terminated string
} MLFBEX, *LPMLFBEX;
```

8.2.6 Reading PLC info (AGL_ReadPLCInfo)

With this function you can querry how much inputs, outputs, markers, timers and counters the connected CPU has. For this, the PLC must be already connected.

Attention:
This function is not possible with all controls. It only works with controls of the S7-300 and S7-400 family. Therefore, you must reckon that you will receive AGL40_FUNC_NOT_IMPLEMENTED as function result.

C/C++-Syntax:

```
int WINAPI AGL_ReadPLCInfo( int ConnNr, LPPLCINFO PLCInfo, int
Timeout, int UserVal );
```

Visual Basic-Syntax:

```
Declare Function AGL_ReadPLCInfo Lib "AGLink40.DLL" (ByVal ConnNr As
Long, Info As PLCInfo, ByVal Timeout As Long, ByVal UserVal As Long)
As Long
```

Delphi-Syntax:

```
function AGL_ReadPLCInfo( ConnNr: Integer; var PLCInfo: TagPLCINFO;
Timeout: Integer; UserVal: LongInt ): Integer; stdcall;
```

Parameter:

ConnNr	Variable for connection handle
PLCInfo	Buffer for maximum value
Timeout	Timeout value which has to be used
UserVal	Value for free usage

Return value:

>= 0	Job number asynchronous call respectively AGL40_SUCCESS if synchronous call
< 0	Error number (s. AGL_Defines.H, AGL_Defines.BAS or AGLink40.PAS)

The PLCINFO structure has the following build up:

```
typedef struct tagPLCINFO
{
  DWORD PAE;                    //  Number of input bytes
  DWORD PAA;                    //  Number of output bytes
  DWORD Flags;                  //  Number of marker bytes
```

```
  DWORD Timer;                    //  Number of timers
  DWORD Counter;                  //  Number of counters
  DWORD LogAddress;               //  Size of logical address space
  DWORD LocalData;                //  Size of local data memory
} PLCINFO, *LPPLCINFO;
```

8.2.7 Determining the number of parametrized diagnostics buffer entries (AGL_ReadDiagBufferEntrys)

This function reads out the number of the PLC's parametrized diagnostics buffer entries. For this the PLC must be already connected. The diagnostics buffer will be read by the function AGL_ReadDiagBuffer. The transformation into text will be executed by the function AGL_GetDiagBufferEntry.

Note:
You do not have to call this function. But you can assign the memory capacity and call the function AGL_ReadDiagBuffer, immediately. Normally, the PLCs have up to 120 diagnostics buffer entries.

C/C++-Syntax:

```
int WINAPI AGL_ReadDiagBufferEntrys( int ConnNr, int *Entrys, int
Timeout, int UserVal );
```

Visual Basic-Syntax:

```
Declare Function AGL_ReadDiagBufferEntrys Lib "AGLink40.DLL" (ByVal
ConnNr As Long, ByRef Entrys As Long, ByVal Timeout As Long, ByVal
UserVal As Long) As Long
```

Delphi-Syntax:

```
function AGL_ReadDiagBufferEntrys( ConnNr: Integer; var Entrys:
Long; Timeout: Integer; UserVal: LongInt ): Integer; stdcall;
```

Parameter:

ConnNr	Variable for connection handle
Entrys	Indicator on parametrized number of entries
Timeout	Timeout value which has to be used
UserVal	Value for free usage

Return value

>= 0	Job number asynchronous call respectively AGL40_SUCCESS if synchronous call
< 0	Error number (s. AGL_Defines.H, AGL_Defines.BAS or AGLink40.PAS)

8.2.8 Reading the PLC's diagnostics buffer (AGL_ReadDiagBuffer)

This function reads out the PLC'S diagnostics buffer. For this the PLC must be already connected. You assign the variable entries, how much entries can be placed into the buffer. After a successful function call you can see how much diagnostics buffer entries have been read there. The translation of the binary content in plain text will be effected by the function AGL_GetDiagBufferEntry. Call this function with an index from 0 until entrys –1. Other values are invalid and will output errror messages.

Note:
Every entry in the diagnostics buffer reserves 20 bytes. In addition 8 bytes for header information are required in the communications buffer. To read 100 entries you have to assign 100 * 20 + 8 = 2008 memory bytes.

C/C++-Syntax:

```
int WINAPI AGL_ReadDiagBuffer( int ConnNr, int *Entrys, PBYTE
pDiagBuff, int Timeout, int UserVal );
```

Visual Basic-Syntax:

```
Declare Function AGL_ReadDiagBuffer Lib "AGLink40.DLL" (ByVal ConnNr
As Long, ByRef Entrys As Long, pDiagBuff As Byte, ByVal Timeout As
Long, ByVal UserVal As Long) As Long
```

Delphi-Syntax:

```
function AGL_ReadDiagBuffer( ConnNr: Integer; var Entrys: Integer;
var pDiagBuff: BYTE; Timeout: Integer; UserVal: LongInt ): Integer;
stdcall;
```

Parameter:

ConnNr	Variable for connection handle
Entrys	Number of assigned entries when calling and number of read entries on function end
pDiagBuff	Indicator on buffer section
Timeout	Timeout value which has to be used
UserVal	Value for free usage

Return value:

>= 0	Job number asynchronous call respectively AGL40_SUCCESS if synchronous call
< 0	Error number (s. AGL_Defines.H, AGL_Defines.BAS or AGLink40.PAS)

8.2.9 Transforming content of diagnostics buffer into text (AGL_GetDiagBufferEntry)

This function transforms the diagnostic's buffer content into plain text entry by entry. Call this function with an index of 0 until entry –1. Other values are invalid and will output an error message.

Note:
Alternatively, VB programers should look for the function AGL_GetVBDiagBufferEntry in chapter »12 Additional functions for VB respectively VBA«.

C/C++-Syntax:

```
int WINAPI AGL_GetDiagBufferEntry( int Index, PBYTE pDiagBuff, LPSTR
Text, int TextLen );
```

Visual Basic-Syntax:

```
Declare Function AGL_GetDiagBufferEntry Lib "AGLink40.DLL" (ByVal
Index As Long, pDiagBuff As Byte, ByVal Text As String, ByVal
TextLen as Long) As Long
```

Delphi-Syntax:

```
function AGL_GetDiagBufferEntry( Index: Long; var pDiagBuff: BYTE;
Text: Pchar; TextLen: Long ): Integer; stdcall;
```

Parameter:

Index	Index of diagnostics buffer entry
pDiagBuff	Indicator on buffer
Text	Indicator on string
TextLen	The string's maximum length

Return value:

>= 0	Length of the diagnsotics buffer message
< 0	Error number (s. AGL_Defines.H, AGL_Defines.BAS or AGLink40.PAS)

8.2.10 Calling the PLC's operating state (AGL_ReadOpState)

With this function you can call the PLC's current operating state. For this the PLC must be already connected. In State the actual operating state will be returned. This can be the following constant: OPSTATE_STOP, OPSTATE_START, OPSTATE_RUN and OPSTATE_UNKNOWN.

C/C++-Syntax:

```
int WINAPI AGL_ReadOpState( int ConnNr, int *State, int Timeout, int
UserVal );
```

Visual Basic-Syntax:

```
Declare Function AGL_ReadOpState Lib "AGLink40.DLL" (ByVal ConnNr As
Long, State As Long, ByVal Timeout As Long, ByVal UserVal As Long)
As Long
```

Delphi-Syntax:

```
function AGL_ReadOpState( ConnNr: Integer; var State: Integer;
Timeout: Integer; UserVal: LongInt ): Integer; stdcall;
```

Parameter:

ConnNr	Variable for connection handle
State	Variable for the PLC's operating state
Timeout	Timeout value which has to be used
UserVal	Value for free usage

Return value:

>= 0	Job number asynchronous call respectively AGL40_SUCCESS if synchronous call
< 0	Error number (s. AGL_Defines.H, AGL_Defines.BAS or AGLink40.PAS)

8.2.11 Stopping PLC (AGL_PLCStop)

With this function you can set the PLC in stop mode. For this the PLC must be already connected.

C/C++-Syntax:

```
int WINAPI AGL_PLCStop( int ConnNr, int Timeout, int UserVal );
```

Visual Basic-Syntax:

```
Declare Function AGL_PLCStop Lib "AGLink40.DLL" (ByVal ConnNr As
Long, ByVal Timeout As Long, ByVal UserVal As Long) As Long
```

Delphi-Syntax:

```
function AGL_PLCStop( ConnNr: Integer; Timeout: Integer; UserVal:
LongInt ): Integer; stdcall;
```

Parameter:

ConnNr	Variable for connection handle
Timeout	The used Timout value
UserVal	Value for free usage

Return value:

>= 0	Job number asynchronous call respectively AGL40_SUCCESS if synchronous call
< 0	Error number (s. AGL_Defines.H, AGL_Defines.BAS or AGLink40.PAS)

8.2.12 Restarting PLC (AGL_PLCStart)

With this function you can set the PLC in run mode. A restart will be performed. For this the PLC must be already connected.

C/C++-Syntax:

```
int WINAPI AGL_PLCStart( int ConnNr, int Timeout, int UserVal );
```

Visual Basic-Syntax:

```
Declare Function AGL_PLCStart Lib "AGLink40.DLL" (ByVal ConnNr As
Long, ByVal Timeout As Long, ByVal UserVal As Long) As Long
```

Delphi-Syntax:

```
function AGL_PLCStart( ConnNr: Integer; Timeout: Integer; UserVal:
LongInt ): Integer; stdcall;
```

Parameter:

ConnNr	Variable for connection handle
Timeout	Timeout value which has to be used
UserVal	Value for free usage

Return value:

>= 0	Job number asynchronous call respectively AGL40_SUCCESS if synchronous call
< 0	Error number (s. AGL_Defines.H, AGL_Defines.BAS or AGLink40.PAS)

8.2.13 Resuming PLC (AGL_PLCResume)

With this function you can set the PLC in run mode. The PLC will be resumed This function cannot be performed with all controls in all PLC parametrizations. Therefore it can output an error. For this the PLC must be already connected.

C/C++-Syntax:

```
int WINAPI AGL_PLCResume( int ConnNr, int Timeout, int UserVal );
```

Visual Basic-Syntax:

```
Declare Function AGL_PLCResume Lib "AGLink40.DLL" (ByVal ConnNr As
Long, ByVal Timeout As Long, ByVal UserVal As Long) As Long
```

Delphi-Syntax:

```
function AGL_PLCResume( ConnNr: Integer; Timeout: Integer; UserVal:
LongInt ): Integer; stdcall;
```

Parameter:

ConnNr	Variable for connection handle
Timeout	Timeout value which has to be used
UserVal	Value for free usage

Return value:

>= 0	Job number asynchronous call respectively AGL40_SUCCESS if synchronous call
< 0	Error number (s. AGL_Defines.H, AGL_Defines.BAS or AGLink40.PAS)

8.2.14 Reading the PLC'S cycle times (AGL_ReadCycleTime)

This function reads the PLC's current, minimum and maximum cycle time. For this the PLC must be already connected.

C/C++-Syntax:

```
int WINAPI AGL_ReadCycleTime( int ConnNr, LPCYCLETIME pCycleTime,
int Timeout, int UserVal );
```

Visual Basic-Syntax:

```
Declare Function AGL_ReadCycleTime Lib "AGLink40.DLL" (ByVal ConnNr
As Long, pCycleTime As CYCLETIME, ByVal Timeout As Long, ByVal
UserVal As Long) As Long
```

Delphi-Syntax:

```
function AGL_ReadCycleTime( ConnNr: Integer; var pCycleTime:
TagCYCLETIME; Timeout: Integer; UserVal: LongInt ): Integer;
stdcall;
```

Parameter:

ConnNr	Variable for connection handle
pCycleTime	Indicator on CYCLETIME structure
Timeout	Timeout value which has to be used
UserVal	Value for free usage

Return value:

>= 0	Job number asynchronous call respectively AGL40_SUCCESS if synchronous call
< 0	Error number (s. AGL_Defines.H, AGL_Defines.BAS or AGLink40.PAS)

The CYCLETIME structure has the following build up:

```
typedef struct tagCYCLETIME
{
  int AktCycleTime;              //  Current cycle time
  int MinCycleTime;              //  Minimum cycle time
  int MaxCycleTime;              //  Maximum cycle time
} CYCLETIME, *LPCYCLETIME;
```

8.2.15 Reading the PLC's protection levels (AGL_ReadProtLevel)

This function reads the protection level adjusted by the operation mode switch, the parametrized protection level, the CPU's valid protection level, the position of the operation mode switch as well as the position of the starting switch indicated by the PLC's number. For this the PLC must be already connected.

The protection level adjusted by the operation mode switch allows the values 1,2 and 3. The parametrized protection level enables 0,1,2 or 3, but 0 menas that no password is given respectively the parametrized protection level is invalid. The operation mode switch positions have the following meanings 1 = RUN, 2 = RUN-P, 3 = STOP, 4 = MRES and 0 = undefined respectively not determinable. The starting switch positions have the following meanings 1 = CRST, 2 = WRST and 0 = undefined, not present or not determinable.

C/C++-Syntax:

```
int WINAPI AGL_ReadProtLevel( int ConnNr, LPPROTLEVEL pProtLevel,
int Timeout, int UserVal );
```

Visual Basic-Syntax:

```
Declare Function AGL_ReadProtLevel Lib "AGLink40.DLL" (ByVal ConnNr
As Long, pProtLevel As PROTLEVEL, ByVal Timeout As Long, ByVal
UserVal As Long) As Long
```

Delphi-Syntax:

```
function AGL_ReadProtLevel( ConnNr: Integer; var pProtLevel:
TagPROTLEVEL; Timeout: Integer; UserVal: LongInt ): Integer;
stdcall;
```

Parameter:

ConnNr	Variable for connection handle
pProtLevel	Indicator on PROTLEVEL structure
Timeout	Timeout value which has to be used
UserVal	Value for free usage

Return value:

>= 0	Job number asynchronous call respectively AGL40_SUCCESS if synchronous call
< 0	Error number (s. AGL_Defines.H, AGL_Defines.BAS or AGLink40.PAS)

The PROTLEVEL structure has the following build up:

```
typedef struct tagPROTLEVEL
{
  int KeyProtLevel;              //  Protection level adjusted by
                                     operation mode switch
  int ParaProtLevel;             //  Parametrized protection level
  int CPUProtLevel;              //  The CPU's valid protection level
  int ModeSelector;              //  The operation mode's position
  int StartupSwitch;             //  The starting switche's position
    } PROTLEVEL, *LPPROTLEVEL;
```

8.2.16 Reading out the PLC's clock (AGL_GetPLCClock)

This function reads out the PLC's system time. For this the PLC must be already connected.

C/C++-Syntax:

```
int WINAPI AGL_GetPLCClock( int ConnNr, LPTOD pTOD, int Timeout, int
UserVal );
```

Visual Basic-Syntax:

```
Declare Function AGL_GetPLCClock Lib "AGLink40.DLL" (ByVal ConnNr As
Long, ByRef pTO as TOD, ByVal Timeout As Long, ByVal UserVal As
Long) As Long
```

Delphi-Syntax:

```
function AGL_GetPLCClock( ConnNr: Integer; var pTOD: TagTOD;
Timeout: Integer; UserVal: LongInt ): Integer; stdcall;
```

Parameter:

ConnNr	Variable for connection handle
pTOD	Indicator on TimeOfDay structure
Timeout	Timeout value which has to be used
UserVal	Value for free usage

Return value:

>= 0	Job number asynchronous call respectively AGL40_SUCCESS if synchronous call
< 0	Error number (s. AGL_Defines.H, AGL_Defines.BAS or AGLink40.PAS)

The TOD structure has the following build up:

```
typedef struct tagTOD
{
  WORD        ClockStatus;              // Look for PLC description
  BYTE        Year;                     // Year in BCD (2-digit)
  BYTE        Month;                    // Month in BCD (2-digit)
  BYTE        Day;                      // Day in BCD (2-digit)
  BYTE        Hour;                     // Hour in BCD (2-digit)
  BYTE        Minute;                   // Minute in BCD (2-digit)
```

```
  BYTE        Second;                 // Second in BCD (2-digit)
  BYTE        Zero;                   // Should always be 0
  BYTE        Weekday;                // Weekday, Sunday = 1
} TOD, *LPTOD;
```

8.2.17 Setting the PLC's clock (AGL_SetPLCClock)

This function sets the PLC's system time. For this the PLC must be already connected.

C/C++-Syntax:

```
int WINAPI AGL_SetPLCClock( int ConnNr, LPTOD pTOD, int Timeout, int
UserVal );
```

Visual Basic-Syntax:

```
Declare Function AGL_SetPLCClock Lib "AGLink40.DLL" (ByVal ConnNr As
Long, ByRef pTOD As TOD, ByVal Timeout As Long, ByVal UserVal As
Long) As Long
```

Delphi-Syntax:

```
function AGL_SetPLCClock( ConnNr: Integer; var pTOD: TagTOD;
Timeout: Integer; UserVal: LongInt ): Integer; stdcall;
```

Parameter:

ConnNr	Variable for connection handle
pTOD	Indicator on TimeOfDay structure
Timeout	Timeout value which has to be used
UserVal	Valuie for free usage

Return value:

>= 0	Job number asynchronous call respectively AGL40_SUCCESS if synchronous call
< 0	Error number (s. AGL_Defines.H, AGL_Defines.BAS or AGLink40.PAS)

The TOD structure has the following build up:

```
typedef struct tagTOD
{
  WORD        ClockStatus;              // Look for PLC description
  BYTE        Year;                     // Year in BCD (2-digit)
  BYTE        Month;                    // Month in BCD (2-digit)
  BYTE        Day;                      // Day in BCD (2-digit)
  BYTE        Hour;                     // Hour in BCD (2-digit)
  BYTE        Minute;                   // Minute in BCD (2-digit)
```

```
  BYTE        Second;                 // Second in BCD (2-digit)
  BYTE        Zero;                   // Should always be 0
  BYTE        Weekday;                // Weekday, Sunday = 1
} TOD, *LPTOD;
```

8.2.18 Reading the PLC's system status list (AGL_ReadSzl)

This function reads the PLC's system status list. You can look for the meaning of the parameters and the valid values for SzlId and index in the descriptions of the respective PLCs. For this the PLC must be already connected.

C/C++-Syntax:

```
int WINAPI AGL_ReadSzl( int ConnNr, int SzlId, ind Index, PBYTE
Buff, int *BuffLen, int Timeout, int UserVal );
```

Visual Basic-Syntax:

```
Declare Function AGL_ReadSzl Lib "AGLink40.DLL" (ByVal ConnNr As
Long, ByVal SzlId as Long, ByVal Index As Long, Buff As Byte,
BuffLen As Long, ByVal Timeout As Long, ByVal UserVal As Long) As
Long
```

Delphi-Syntax:

```
function AGL_ReadSzl( ConnNr: Integer; SzlId: Integer; Index:
Integer; var Buff: Byte; var BuffLen: Long; Timeout: Integer;
UserVal: LongInt ): Integer; stdcall;
```

Parameter:

ConnNr	Variable for connnection handle
SzlId	Number of the system status list.
Index	Index of the system status list
Buff	Indicator on buffer
BuffLen	Length of buffer when calling function, number of characters read after executing function
Timeout	Timeout value which has to be used
UserVal	Value for free usage

Return value:

>= 0	Job number asynchronous call respectively AGL40_SUCCESS if synchronous call
< 0	Error number (s. AGL_Defines.H, AGL_Defines.BAS or AGLink40.PAS)

8.2.19 Determining number of data modules (AGL_ReadDBCount)

With this function you can determine how much data modules are in one PLC.

C/C++-Syntax:

```
int WINAPI AGL_ReadDBCount( int ConnNr, int *DBCount, int Timeout,
int UserVal );
```

Visual Basic-Syntax:

```
Declare Function AGL_ReadDBCount Lib "AGLink40.DLL" (ByVal ConnNr As
Long, DBCount As Long, ByVal Timeout As Long, ByVal UserVal As Long)
As Long
```

Delphi-Syntax:

```
function AGL_ReadDBCount( ConnNr: Integer; var DBCount: Integer;
Timeout: Integer; UserVal: LongInt ): Integer; stdcall;
```

Parameter:

ConnNr	Variable for connection handle
DBCount	Indicator on variable for number of data modules
Timeout	Timeout value which has to be used
UserVal	Value for free usage

Return value:

>= 0	Job number asynchronous call respectively AGL40_SUCCESS if synchronous call
< 0	Error number (s. AGL_Defines.H, AGL_Defines.BAS or AGLink40.PAS)

8.2.20 Reading data module book keeper (AGL_ReadDBList)

With this function you can read the data modules' book keeper. The memory capacity for the data module numbers must be assigned sufficiently. When calling a function you indicate the number of assigned elements in DBCount. The number of entered D numbers will be returned in DBCount on function end. For a complete list call the function AGL_ReadDBCount and then assign the memory capacity. For this the PLC must be already connected.

C/C++-Syntax:

```
int WINAPI AGL_ReadDBList( int ConnNr, int *DBCount, PWORD DBList,
int Timeout, int UserVal );
```

Visual Basic-Syntax:

```
Declare Function AGL_ReadDBList Lib "AGLink40.DLL" (ByVal ConnNr As
Long, DBCount As Long, DBList As Integer, ByVal Timeout As Long,
ByVal UserVal As Long) As Long
```

Delphi-Syntax:

```
function AGL_ReadDBList( ConnNr: Integer; var DBCount: Integer; var
DBList: WORD; Timeout: Integer; UserVal: LongInt ): Integer;
stdcall;
```

Parameter:

ConnNr	Variable for connection handle
DBCount	On function call, number of assigned elements in DBList and on function endnumber of entereddata module numbers
DBList	Indicator on Array for data module numbers
Timeout	Timeout value which has to be used
UserVal	Value for free usage

Return value:

>= 0	Job number asynchronous call respectively AGL40_SUCCESS if synchronous call
< 0	Error number (s. AGL_Defines.H, AGL_Defines.BAS or AGLink40.PAS)

8.2.21 Reading the length of data module (AGL_ReadDBLen)

With this function you can determine the length of a certain data module. For this the PLC must be already connected.

C/C++-Syntax:

```
int WINAPI AGL_ReadDBLen( int ConnNr, int DBNr, int *DBLen, int
Timeout, int UserVal );
```

Visual Basic-Syntax

```
Declare Function AGL_ReadDBLen Lib "AGLink40.DLL" (ByVal ConnNr As
Long, ByVal DBNr As Long, DBLen As Long, ByVal Timeout As Long,
ByVal UserVal As Long) As Long
```

Delphi-Syntax:

```
function AGL_ReadDBLen( ConnNr: Integer; DBNr: Integer; var DBLen:
Integer; Timeout: Integer; UserVal: LongInt ): Integer; stdcall;
```

Parameter:

ConnNr	Variable for connection handle
DBNr	Number of data module
DBLen	Indicator on variable for length of data module
Timeout	Timeout value which has to be used
UserVal	Value for free usage

Return value:

>= 0	Job number asynchronous call respectively AGL40_SUCCESS if synchronous call
< 0	Error number (s. AGL_Defines.H, AGL_Defines.BAS or AGLink40.PAS)

8.2.22 Reading the PLC's communication package size (AGL_ReadMaxPacketSize)

This function returns the PLC's maximum communication package size. At present the following values are possible 240bytes (generally using 300 PLCs) and 480bytes (normally using 400 PLCs). For this the PLC must be already connected.

C/C++-Syntax:

```
int WINAPI AGL_ReadMaxPacketSize( int ConnNr );
```

Visual Basic-Syntax:

```
Declare Function AGL_ReadMaxPacketSize Lib "AGLink40.DLL" (ByVal
ConnNr As Long) As Long
```

Delphi-Syntax:

```
function AGL_ReadMaxPacketSize( ConnNr: Integer ): Integer; stdcall;
```

Parameter:

ConnNr	Variable for connection handle

Return value:

> 0	The maximum package size
< 0	Error number (s. AGL_Defines.H, AGL_Defines.BAS or AGLink40.PAS)

8.3 Function reference reading data

8.3.1 Reading input bytes (AGL_ReadInBytes)

With this function you can read the PLC's input bytes from the process image. For this the PLC must be already connected.

C/C++-Syntax:

```
int WINAPI AGL_ReadInBytes( int ConnNr, int Start, int Num, PBYTE
Buff, int Timeout, int UserVal );
```

Visual Basic-Syntax:

```
Declare Function AGL_ReadInBytes Lib "AGLink40.DLL" (ByVal ConnNr As
Long, ByVal Start As Long, ByVal Num As Long, Buff As Byte, ByVal
Timeout As Long, ByVal UserVal As Long) As Long
```

Delphi-Syntax:

```
function AGL_ReadInBytes( ConnNr: Integer; Start: Integer; Num:
Integer; var Buff: Byte; Timeout: Integer; UserVal: LongInt ):
Integer; stdcall;
```

Parameter:

ConnNr	Variable for connection handle
Start	From which byte should be read
Num	Number of bytes which have to be read
Buff	Buffer for bytes read
Timeout	Timeout value which has to be used
UserVal	Value for free usage

Return value:

>= 0	Job number asynchronous call respectively AGL40_SUCCESS if synchronous call
< 0	Error number (s. AGL_Defines.H, AGL_Defines.BAS or AGLink40.PAS)

8.3.2 Reading periphery input bytes (AGL_ReadPInBytes)

With this function you can read the PLC's periphery input byte. For this the PLC must be already connected.

C/C++-Syntax:

```
int WINAPI AGL_ReadPInBytes( int ConnNr, int Start, int Num, PBYTE
Buff, int Timeout, int UserVal );
```

Visual Basic-Syntax:

```
Declare Function AGL_ReadPInBytes Lib "AGLink40.DLL" (ByVal ConnNr
As Long, ByVal Start As Long, ByVal Num As Long, Buff As Byte, ByVal
Timeout As Long, ByVal UserVal As Long) As Long
```

Delphi-Syntax:

```
function AGL_ReadPInBytes( ConnNr: Integer; Start: Integer; Num:
Integer; var Buff: Byte; Timeout: Integer; UserVal: LongInt ):
Integer; stdcall;
```

Parameter:

ConnNr	Variable for connection handle
Start	From which byte should be read
Num	Number of bytes which have to be read
Buff	Buffer for bytes read
Timeout	Timeout value which has to be used
UserVal	Value for free usage

Return value:

>= 0	Job number asynchronous call respectively AGL40_SUCCESS if synchronous call
< 0	Error number (s. AGL_Defines.H, AGL_Defines.BAS or AGLink40.PAS)

8.3.3 Reading output bytes (AGL_ReadOutBytes)

With this function you can read the PLC's output byte from the process image. For this the PLC must be already connected.

C/C++-Syntax:

```
int WINAPI AGL_ReadOutBytes( int ConnNr, int Start, int Num, PBYTE
Buff, int Timeout, int UserVal );
```

Visual Basic-Syntax:

```
Declare Function AGL_ReadOutBytes Lib "AGLink40.DLL" (ByVal ConnNr
As Long, ByVal Start As Long, ByVal Num As Long, Buff As Byte, ByVal
Timeout As Long, ByVal UserVal As Long) As Long
```

Delphi-Syntax:

```
function AGL_ReadOutBytes( ConnNr: Integer; Start: Integer; Num:
Integer; var Buff: Byte; Timeout: Integer; UserVal: LongInt ):
Integer; stdcall;
```

Parameter:

ConnNr	Variable for connection handle
Start	From which byte should be read
Num	Number of bytes which have to be read
Buff	Buffer for bytes read
Timeout	Timeout value which has to be used
UserVal	Value for free usage

Return value:

>= 0	Job number asynchronous call respectively AGL40_SUCCESS if synchronous call
< 0	Error number (s. AGL_Defines.H, AGL_Defines.BAS or AGLink40.PAS)

8.3.4 Reading marker bytes (AGL_ReadFlagBytes)

With this function you can read the PLC's marker byte. For this the PLC must be already connected.

C/C++-Syntax:

```
int WINAPI AGL_ReadFlagBytes( int ConnNr, int Start, int Num, PBYTE
Buff, int Timeout, int UserVal );
```

Visual Basic-Syntax:

```
Declare Function AGL_ReadFlagBytes Lib "AGLink40.DLL" (ByVal ConnNr
As Long, ByVal Start As Long, ByVal Num As Long, Buff As Byte, ByVal
Timeout As Long, ByVal UserVal As Long) As Long
```

Delphi-Syntax:

```
function AGL_ReadFlagBytes( ConnNr: Integer; Start: Integer; Num:
Integer; var Buff: Byte; Timeout: Integer; UserVal: LongInt ):
Integer; stdcall;
```

Parameter:

ConnNr	Variable for connection handle
Start	From which byte should be read
Num	Number of bytes which have to be read
Buff	Buffer for bytes read
Timeout	Timeout value which has to be used
UserVal	Value for free usage

Return value:

>= 0	Job number asynchronous call respectively AGL40_SUCCESS if synchronous call
< 0	Error number (s. AGL_Defines.H, AGL_Defines.BAS or AGLink40.PAS)

8.3.5 Reading special marker bytes (AGL_ReadSFlagBytes)

With this function you can read the PLC's special marker bytes of the 200 series with the desired number. For this the PLC must be already connected. For these PLCs you can alternatively access markers from 256. They are internally mapped on the special markers. Through this your programs will be hardware independent.

Attention:
This function is not implemented in all PLCs. If not it will output the return value AGL40_FUNC_NOT_IMPLEMENTED.

C/C++-Syntax:

```
int WINAPI AGL_ReadSFlagBytes( int ConnNr, int Start, int Num, PBYTE
Buff, int Timeout, int UserVal );
```

Visual Basic-Syntax:

```
Declare Function AGL_ReadSFlagBytes Lib "AGLink40.DLL" (ByVal ConnNr
As Long, ByVal Start As Long, ByVal Num As Long, Buff As Byte, ByVal
Timeout As Long, ByVal UserVal As Long) As Long
```

Delphi-Syntax:

```
function AGL_ReadSFlagBytes( ConnNr: Integer; Start: Integer; Num:
Integer; var Buff: Byte; Timeout: Integer; UserVal: LongInt ):
Integer; stdcall;
```

Parameter:

ConnNr	Variable for connection handle
Start	From which byte should be read
Num	Number of bytes which have to be read
Buff	Buffer for bytes read
Timeout	Timeout value which has to be used
UserVal	Value for free usage

Return value:

>= 0	Job number asynchronous call respectively AGL40_SUCCESS if synchronous call
< 0	Error number (s. AGL_Defines.H, AGL_Defines.BAS or AGLink40.PAS)

8.3.6 Reading variable bytes (AGL_ReadVarBytes)

With this function you can read the PLC's variable bytes of the 200 series with the desired number. For this the PLC must be already connected. For these PLCs you can alternatively access on DB 1. It is internally mapped on the variables. Through this your programs will be hardware independent.

Attention:
This function is not implemented in all PLCs. If not it will output the return value AGL40_FUNC_NOT_IMPLEMENTED.

C/C++-Syntax:

```
int WINAPI AGL_ReadVarBytes( int ConnNr, int Start, int Num, PBYTE
Buff, int Timeout, int UserVal );
```

Visual Basic-Syntax:

```
Declare Function AGL_ReadVarBytes Lib "AGLink40.DLL" (ByVal ConnNr
As Long, ByVal Start As Long, ByVal Num As Long, Buff As Byte, ByVal
Timeout As Long, ByVal UserVal As Long) As Long
```

Delphi-Syntax:

```
function AGL_ReadVarBytes( ConnNr: Integer; Start: Integer; Num:
Integer; var Buff: Byte; Timeout: Integer; UserVal: LongInt ):
Integer; stdcall;
```

Parameter:

ConnNr	Variable for connection handle
Start	From which byte should be read
Num	Number of bytes which have to be read
Buff	Buffer for bytes read
Timeout	Timeout value which has to be used
UserVal	Value for free usage

Return value:

>= 0	Job number asynchronous call respectively AGL40_SUCCESS if synchronous call
< 0	Error number (s. AGL_Defines.H, AGL_Defines.BAS or AGLink40.PAS)

8.3.7 Reading data bytes (AGL_ReadDataBytes)

With this function you can read the PLC's data bytes with the desired number. For this the PLC must be already connected. Using a S5 you can access DX modules when indicating a DB number >255.

C/C++-Syntax:

```
int WINAPI AGL_ReadDataBytes( int ConnNr, int DBNr, int Start, int
Num, PBYTE Buff, int Timeout, int UserVal );
```

Visual Basic-Syntax:

```
Declare Function AGL_ReadDataBytes Lib "AGLink40.DLL" (ByVal ConnNr
As Long, ByVal DBNr As Long, ByVal Start As Long, ByVal Num As Long,
Buff As Byte, ByVal Timeout As Long, ByVal UserVal As Long) As Long
```

Delphi-Syntax:

```
function AGL_ReadDataBytes( ConnNr: Integer; DBNr: Integer; Start:
Integer; Num: Integer; var Buff: Byte; Timeout: Integer; UserVal:
LongInt ): Integer; stdcall;
```

Parameter:

ConnNr	Variable for connection handle
DBNr	DB number which should be read from
Start	From which byte should be read
Num	Number of bytes which have to be read
Buff	Buffer for bytes read
Timeout	Timeout value which has to be used
UserVal	Value for free usage

Return value:

>= 0	Job number asynchronous call respectively AGL40_SUCCESS if synchronous call
< 0	Error number (s. AGL_Defines.H, AGL_Defines.BAS or AGLink40.PAS)

8.3.8 Reading S5 data words (AGL_ReadDataWords)

With this function you can read data words from an PLC's S5 addressing with the desired number. For this the PLC must be already connected. When accessing data word 10 on a S5 it causes that DW 10 will be read. But if using a S7 DBB 20 and DBB 21 will be read. Indicating a DB number >255 on a S5 you can access DX modules.

Note:
This function is only implemented for the S5 family. In other cases you will receive the following error message AGL40_FUNC_NOT_IMPLEMENTED.

C/C++-Syntax:

```
int WINAPI AGL_ReadDataWords( int ConnNr, int DBNr, int Start, int
Num, PWORD Buff, int Timeout, int UserVal );
```

Visual Basic-Syntax:

```
Declare Function AGL_ReadDataWords Lib "AGLink40.DLL" (ByVal ConnNr
As Long, ByVal DBNr As Long, ByVal Start As Long, ByVal Num As Long,
Buff As Integer, ByVal Timeout As Long, ByVal UserVal As Long) As
Long
```

Delphi-Syntax:

```
function AGL_ReadDataWords( ConnNr: Integer; DBNr: Integer; Start:
Integer; Num: Integer; var Buff: Word; Timeout: Integer; UserVal:
LongInt ): Integer; stdcall;
```

Parameter:

ConnNr	Variable for connection handle
DBNr	DB number which should be read from
Start	From which word should be read
Num	Number of words which have to be read
Buff	Buffer for words read
Timeout	Timeout value which has to be used
UserVal	Value for free usage

Return value:

>= 0	Job number asynchronous call respectively AGL40_SUCCESS if synchronous call
< 0	Error number (s. AGL_Defines.H, AGL_Defines.BAS or AGLink40.PAS)

8.3.9 Reading timer words (AGL_ReadTimerWords)

With this function you can read the PLC's timer words with the desired number. For this the PLC must be already connected.

C/C++-Syntax:

```
int WINAPI AGL_ReadTimerWords( int ConnNr, int Start, int Num, PWORD
Buff, int Timeout, int UserVal );
```

Visual Basic-Syntax:

```
Declare Function AGL_ReadTimerWords Lib "AGLink40.DLL" (ByVal ConnNr
As Long, ByVal Start As Long, ByVal Num As Long, Buff As Integer,
ByVal Timeout As Long, ByVal UserVal As Long) As Long
```

Delphi-Syntax:

```
function AGL_ReadTimerWords( ConnNr: Integer; Start: Integer; Num:
Integer; var Buff: WORD; Timeout: Integer; UserVal: LongInt ):
Integer; stdcall;
```

Parameter:

ConnNr	Variable for connection handle
Start	From which timer should be read
Num	Number of timers which have to be read
Buff	Buffer for timers read
Timeout	Timeout value which has to be used
UserVal	Value for free usage

Return value:

>= 0	Job number asynchronous call respectively AGL40_SUCCESS if synchronous call
< 0	Error number (s. AGL_Defines.H, AGL_Defines.BAS or AGLink40.PAS)

8.3.10 Reading counter words (AGL_ReadCounterWords)

With this function you can read the PLC's counter words with the desired number. For this the PLC must be already connected.

C/C++-Syntax:

```
int WINAPI AGL_ReadCounterWords( int ConnNr, int Start, int Num,
PWORD Buff, int Timeout, int UserVal );
```

Visual Basic-Syntax:

```
Declare Function AGL_ReadCounterWords Lib "AGLink40.DLL" (ByVal
ConnNr As Long, ByVal Start As Long, ByVal Num As Long, Buff As
Integer, ByVal Timeout As Long, ByVal UserVal As Long) As Long
```

Delphi-Syntax:

```
function AGL_ReadCounterWords( ConnNr: Integer; Start: Integer; Num:
Integer; var Buff: WORD; Timeout: Integer; UserVal: LongInt ):
Integer; stdcall;
```

Parameter:

ConnNr	Variable for connection handle
Start	From which counter should be read
Num	Number of counters which have to bread
Buff	Buffer for counters read
Timeout	Timeout value which has to be used
UserVal	Value for free usage

Return value:

>= 0	Job number asynchronous call respectively AGL40_SUCCESS if synchronous call
< 0	Error number (s. AGL_Defines.H, AGL_Defines.BAS or AGLink40.PAS)

8.3.11 Mixed reading job (AGL_ReadMix)

With this function you can read different data file types and data sizes in one job. For this the PLC must be already connected.

C/C++-Syntax:

```
int WINAPI AGL_ReadMix( int ConnNr, LPDATA_RW40 Buff, int Num, int
Timeout, int UserVal );
```

Visual Basic-Syntax:

```
Declare Function AGL_ReadMix Lib "AGLink40.DLL" (ByVal ConnNr As
Long, Buff As DATA_RW40, ByVal Num As Long, ByVal Timeout As Long,
ByVal UserVal As Long) As Long
```

Delphi-Syntax:

```
function AGL_ReadMix( ConnNr: Integer; var Buff: TagDATA_RW40; Num:
Integer; Timeout: Integer; UserVal: LongInt ): Integer; stdcall;
```

Parameter:

ConnNr	Variable for connection handle
Buff	Indicator on reading structures
Num	Number of structures
Timeout	Timeout value which has to be used
UserVal	Value for free usage

Return value

>= 0	Job number asynchronous call respectively AGL40_SUCCESS if synchronous call
< 0	Error number (s. AGL_Defines.H, AGL_Defines.BAS or AGLink40.PAS)

The DATA_RW40 structure has the following build up:

```
typedef struct tagDATA_RW40
{
  WORD  OpArea;    // Section of operand which should accessed to
  WORD  OpType;    // Data type

  WORD  OpAnz;     // Only important when using ReadMixEx and
                   // WriteMixEx

  WORD  DBNr;      // Number of data module
```

```
  WORD  Offset;    // Offset for memory access (when using T+Z number)
  WORD  BitNr;     // Bit number (if necessary)
  int   Result;    // Result (error code) of the reading respectively
                   // operation writing
  union
  {
    DWORD Value;   // Operand value as  DWORD (ReadMix and WriteMix)
    WORD  W[2];    // For an easier access (ReadMix and WriteMix)
    BYTE  B[4];    // For an easier access (ReadMix and WriteMix)
    float fValue;  // Operand value as single (ReadMix and WriteMix)
    PVOID Buff;    // Indicator on buffer (ReadMixEx and WriteMixEx)
  };
} DATA_RW40, *LPDATA_RW40;
```

These fields have the following meaning:

OpArea	Operand type, valid values are AREA_IN (processing image inputs), AREA_OUT (processing image outputs), AREA_FLAG (marker), AREA_DATA (data files), AREA_TIMER (timers), AREA_COUNTER (counter), AREA_SFLAG_200 (special marker of series 200), AREA_VAR_200 (variable memory of series 200), AREA_TIMER_200 (timers of series 200), AREA_COUNTER_200 (counter of series 200) and AREA_PERIPHERIE (periphery inputs)
OpType	Operand type. Valid values are TYP_BIT, TYP _BYTE, TYP _WORD and TYP _DWORD. Timer must have TYP_TIMER there and counter must have TYP_COUNTER there (respectively the corresponding version of the 200 series). TYP _BIT is not valid when using AREA_PERIPHERIE and leads to an error return.
DBNr	The DB number, when it is about a data file access.
ByteNr	Offset for memory access
BitNr	Bit number, when it is about SIZE_BIT using the access size.
Result	Result of reading operation.
Value, W, B, fValue	Value read if no error occured.

8.3.12 Extended mixed reading job (AGL_ReadMixEx)

With this function you can read different data file types and data sizes in one job. For this the PLC must be already connected. The element Buff of the DATA_RW40 structure is used here. There is an indicator on the memory in which the read data files should be written into.

C/C++-Syntax:

```
int WINAPI AGL_ReadMixEx( int ConnNr, LPDATA_RW40 Buff, int Num, int
Timeout, int UserVal );
```

Visual Basic-Syntax:

```
Declare Function AGL_ReadMixEx Lib "AGLink40.DLL" (ByVal ConnNr As
Long, Buff As DATA_RW40, ByVal Num As Long, ByVal Timeout As Long,
ByVal UserVal As Long) As Long
```

Delphi-Syntax:

```
function AGL_ReadMixEx( ConnNr: Integer; var Buff: DATA_RW40; Num:
Integer; Timeout: Integer; UserVal: LongInt ): Integer; stdcall;
```

Parameter:

ConnNr	Variable for connection handle
Buff	Indicator on reading structures
Num	Number of structures
Timeout	Timeout value which has to be used
UserVal	Value for free usage

Return value:

>= 0	Job number asynchronous call respectively AGL40_SUCCESS if synchronous call
< 0	Error number (s. AGL_Defines.H, AGL_Defines.BAS or AGLink40.PAS)

The DATA_RW40 structure has the following build up:

```
typedef struct tagDATA_RW40
{
  WORD  OpArea;    // Section of operand which should be accessed to
  WORD  OpType;    // Data type
  WORD  OpAnz;     // Only important when using ReadMixEx and WriteMixEx
```

```
  WORD  DBNr;      // Number of data module
  WORD  Offset;    // Offset for memory access (when using T+Z number)
  WORD  BitNr;     // Bit number (if necessary)
  int   Result;    // Result (error code) of reading respectively writing
                   // operation
  union
  {
    DWORD Value;   // Operand value as DWORD (ReadMix and WriteMix)
    WORD  W[2];    // For easier access (ReadMix and WriteMix)
    BYTE  B[4];    // For easier access (ReadMix and WriteMix)
    float fValue;  // Operand value as single (ReadMix and WriteMix)
    PVOID Buff;    // Indicator on buffer (ReadMixEx and WriteMixEx)
  };
} DATA_RW40, *LPDATA_RW40;
```

These fields have the following meaning:

OpArea	Operand type, valid values are AREA_IN (processing image inputs), AREA_OUT (processing image outputs), AREA_FLAG (marker), AREA_DATA (data files), AREA_TIMER (timers), AREA_COUNTER (counter), AREA_SFLAG_200 (special marker of series 200), AREA_VAR_200 (variable memory of series 200), AREA_TIMER_200 (timers of series 200), AREA_COUNTER_200 (counter of series 200) and AREA_PERIPHERIE (periphery inputs)
OpType	Operand type. Valid values are TYP_BIT, TYP _BYTE, TYP _WORD and TYP _DWORD. Timer must have TYP_TIMER there and counter must have TYP_COUNTER there (respectively the corresponding version of the 200 series). TYP _BIT is not valid when using AREA_PERIPHERIE and leads to an error return.
DBNr	The DB number, when it is about a data file access.
ByteNr	Offset for memory access
BitNr	Bit number, when it is about SIZE_BIT using the access size.
Result	Result of reading operation.
Buff	Indicator on data file buffer

8.3.13 Reading data block (AGL_BReceive)

With this function you can receive a data block which is sent by the PLC with Bsend. For this a connection has to be parametrized. At present only Industrial Ethernet with the ACCON-AGLink communication setting TYPE_S7CONN_IE is supported. For this the PLC must be already connected.

C/C++-Syntax:

```
int WINAPI AGL_BReceive( int ConnNr, LPWSABUF pwsa, int Timeout, int
UserVal );
```

Visual Basic-Syntax:

```
Declare Function AGL_BReceive Lib "AGLink40.DLL" (ByVal ConnNr As
Long, pwsa As WSABUFF, ByVal Timeout As Long, ByVal UserVal As Long)
As Long
```

Delphi-Syntax:

```
function AGL_BReceive( ConnNr: Integer; var pwsa: TagWSABUFF;
Timeout: Integer; UserVal: LongInt ): Integer; stdcall;
```

Parameter:

ConnNr	Variable for connection handle
pwsa	Indicator on WSA buffer
Timeout	Timeout value which has to be used
UserVal	Valuie for free usage

Return value:

>= 0	Job number asynchronous call respectively AGL40_SUCCESS if synchronous call
< 0	Error number (s. AGL_Defines.H, AGL_Defines.BAS or AGLink40.PAS)

The WSABUF structure has the following build up:

```
typedef struct tagWSABUF
{
  ULONG  len;      // buffer length
  char  *buf;      // pointer to buffer
} WSABUF, *LPWSABUF;
```

8.3.14 Reading data block with R_ID (AGL_BReceiveEx)

With this function you can receive a data block which is sent by the PLC using BSend. Compared to the function AGL_BReceive, the R_ID of the connection is returned here, too. Also a connection has to be parametrized. At present only Industrial Ethernet with the AGLink communication setting TYPE_S7CONN_IE is supported. To this PLC a connection has to be already established to.

C/C++-Syntax:

```
int WINAPI AGL_BReceiveEx( int ConnNr, LPWSABUF pwsa, int *R_ID, int
Timeout, int UserVal );
```

Visual Basic-Syntax:

```
Declare Function AGL_BReceiveEx Lib "AGLink40.DLL" (ByVal ConnNr As
Long, pwsa As WSABUFF, ByRef R_ID as Long, ByVal Timeout As Long,
ByVal UserVal As Long) As Long
```

Delphi-Syntax:

```
function AGL_BReceiveEx( ConnNr: Integer; var pwsa: TagWSABUFF; var
R_ID: Integer; Timeout: Integer; UserVal: LongInt ): Integer;
stdcall;
```

Parameter:

ConnNr	Variable for connection handle
pwsa	Indicator on WSA buffer
R_ID	Indicator on variable for the RemoteID of the connection
Timeout	Timeout value which has to be used
UserVal	Valuie for free usage

Return value:

>= 0	Job number asynchronous call respectively AGL40_SUCCESS if synchronous call
< 0	Error number (s. AGL_Defines.H, AGL_Defines.BAS or AGLink40.PAS)

8.4 Function reference writing data

8.4.1 Writing input bytes (AGL_WriteInBytes)

With this function you can write input bytes into the PLC's processing image with the desired number. For this the PLC must be already connected. At the system point the written bytes will be immediately overwritten by the values of the physically present inputs depending on the PLC.

C/C++-Syntax:

```
int WINAPI AGL_WriteInBytes( int ConnNr, int Start, int Num, PBYTE
Buff, int Timeout, int UserVal );
```

Visual Basic-Syntax:

```
Declare Function AGL_WriteInBytes Lib "AGLink40.DLL" (ByVal ConnNr
As Long, ByVal Start As Long, ByVal Num As Long, Buff As Byte, ByVal
Timeout As Long, ByVal UserVal As Long) As Long
```

Delphi-Syntax:

```
function AGL_WriteInBytes( ConnNr: Integer; Start: Integer; Num:
Integer; var Buff: Byte; Timeout: Integer; UserVal: LongInt ):
Integer; stdcall;
```

Parameter:

ConnNr	Variable for connection handle
Start	From which byte should be read
Num	Number of bytes which have to be read
Buff	Buffer for the number of bytes which have to be read
Timeout	Timeout value which has to be used
UserVal	Value for free usage

Return value:

>= 0	Job number asynchronous call respectively AGL40_SUCCESS if synchronous call
< 0	Error number (s. AGL_Defines.H, AGL_Defines.BAS or AGLink40.PAS)

8.4.2 Writing output bytes (AGL_WriteOutBytes)

With this function you can write output bytes into the PLC's processing image with the desired number. For this the PLC must be already connected.

C/C++-Syntax:

```
int WINAPI AGL_WriteOutBytes( int ConnNr, int Start, int Num, PBYTE
Buff, int Timeout, int UserVal );
```

Visual Basic-Syntax:

```
Declare Function AGL_WriteOutBytes Lib "AGLink40.DLL" (ByVal ConnNr
As Long, ByVal Start As Long, ByVal Num As Long, Buff As Byte, ByVal
Timeout As Long, ByVal UserVal As Long) As Long
```

Delphi-Syntax:

```
function AGL_WriteOutBytes( ConnNr: Integer; Start: Integer; Num:
Integer; var Buff: Byte; Timeout: Integer; UserVal: LongInt ):
Integer; stdcall;
```

Parameter:

ConnNr	Variable for connection handle
Start	From which byte should be read
Num	Number of bytes which have to be read
Buff	Buffer for the number of bytes which have to be read
Timeout	Timeout value which has to be used
UserVal	Value for free usage

Return value:

>= 0	Job number asynchronous call respectively AGL40_SUCCESS if synchronous call
< 0	Error number (s. AGL_Defines.H, AGL_Defines.BAS or AGLink40.PAS)

8.4.3 Writing periphery output bytes (AGL_WritePOutBytes)

With this function you can write the PLC's periphery output bytes with the desired number. For this the PLC must be already connected.

C/C++-Syntax:

```
int WINAPI AGL_WritePOutBytes( int ConnNr, int Start, int Num, PBYTE
Buff, int Timeout, int UserVal );
```

Visual Basic-Syntax:

```
Declare Function AGL_WritePOutBytes Lib "AGLink40.DLL" (ByVal ConnNr
As Long, ByVal Start As Long, ByVal Num As Long, Buff As Byte, ByVal
Timeout As Long, ByVal UserVal As Long) As Long
```

Delphi-Syntax:

```
function AGL_WritePOutBytes( ConnNr: Integer; Start: Integer; Num:
Integer; var Buff: Byte; Timeout: Integer; UserVal: LongInt ):
Integer; stdcall;
```

Parameter:

ConnNr	Variable for connection handle
Start	From which byte should be read
Num	Number of bytes which have to be read
Buff	Buffer of the number of bytes which have to be read
Timeout	Timeout value which has to be used
UserVal	Value for free usage

Return value:

>= 0	Job number asynchronous call respectively AGL40_SUCCESS if synchronous call
< 0	Error number (s. AGL_Defines.H, AGL_Defines.BAS or AGLink40.PAS)

8.4.4 Writing marker bytes (AGL_WriteFlagBytes)

With this function you can write the PLC's marker bytes with the desired number. For this the PLC must be already connected.

C/C++-Syntax:

```
int WINAPI AGL_WriteFlagBytes( int ConnNr, int Start, int Num, PBYTE
Buff, int Timeout, int UserVal );
```

Visual Basic-Syntax:

```
Declare Function AGL_WriteFlagBytes Lib "AGLink40.DLL" (ByVal ConnNr
As Long, ByVal Start As Long, ByVal Num As Long, Buff As Byte, ByVal
Timeout As Long, ByVal UserVal As Long) As Long
```

Delphi-Syntax:

```
function AGL_WriteFlagBytes( ConnNr: Integer; Start: Integer; Num:
Integer; var Buff: Byte; Timeout: Integer; UserVal: LongInt ):
Integer; stdcall;
```

Parameter:

ConnNr	Variable for connection handle
Start	From which byte should be read
Num	Number of bytes which have to be read
Buff	Buffer of the number of bytes which have to be read
Timeout	Timeout value which has to be used
UserVal	Value for free usage

Return value:

>= 0	Job number asynchronous call respectively AGL40_SUCCESS if synchronous call
< 0	Error number (s. AGL_Defines.H, AGL_Defines.BAS or AGLink40.PAS)

8.4.5 Writing special marker bytes (AGL_WriteSFlagBytes)

With this function you can write special marker bytes of a series 200 PLC with the desired number. For this the PLC must be already connected. For these PLCs you can alternatively access markers from 256. They are internally mapped on the special markers. Through this your programs will be hardware independent.

Attention:
This function is not implemented in all PLCs. If not it will output the return value AGL40_FUNC_NOT_IMPLEMENTED.

C/C++-Syntax:

```
int WINAPI AGL_WriteSFlagBytes( int ConnNr, int Start, int Num,
PBYTE Buff, int Timeout, int UserVal );
```

Visual Basic-Syntax:

```
Declare Function AGL_WriteSFlagBytes Lib "AGLink40.DLL" (ByVal
ConnNr As Long, ByVal Start As Long, ByVal Num As Long, Buff As
Byte, ByVal Timeout As Long, ByVal UserVal As Long) As Long
```

Delphi-Syntax:

```
function AGL_WriteSFlagBytes( ConnNr: Integer; Start: Integer; Num:
Integer; var Buff: Byte; Timeout: Integer; UserVal: LongInt ):
Integer; stdcall;
```

Parameter:

ConnNr	Variable for connection handle
Start	From which byte should be read
Num	Number of bytes which have to be read
Buff	Buffer of the number of bytes which have to be read
Timeout	Timeout value which has to be used
UserVal	Value for free usage

Return value:

>= 0	Job number asynchronous call respectively AGL40_SUCCESS if synchronous call
< 0	Error number (s. AGL_Defines.H, AGL_Defines.BAS or AGLink40.PAS)

8.4.6 Writing variable bytes (AGL_WriteVarBytes)

With this function you can write the PLC's variable bytes of the 200 series with the desired number. The PLC must be already connected. For these PLCs you can alternatively access on DB 1. It is internally mapped on the variables. Through this your programs will be hardware independent.

Attention:
This function is not implemented in all PLCs. If not it will output the return value AGL40_FUNC_NOT_IMPLEMENTED.

C/C++-Syntax:

```
int WINAPI AGL_WriteVarBytes( int ConnNr, int Start, int Num, PBYTE
Buff, int Timeout, int UserVal );
```

Visual Basic-Syntax:

```
Declare Function AGL_WriteVarBytes Lib "AGLink40.DLL" (ByVal ConnNr
As Long, ByVal Start As Long, ByVal Num As Long, Buff As Byte, ByVal
Timeout As Long, ByVal UserVal As Long) As Long
```

Delphi-Syntax:

```
function AGL_WriteVarBytes( ConnNr: Integer; Start: Integer; Num:
Integer; var Buff: Byte; Timeout: Integer; UserVal: LongInt ):
Integer; stdcall;
```

Parameter:

ConnNr	Variable for connection handle
Start	From which byte should be read
Num	Number of bytes which have to be read
Buff	Buffer of the number of bytes which have to be read
Timeout	Timeout value which has to be used
UserVal	Value for free usage

Return value:

>= 0	Job number asynchronous call respectively AGL40_SUCCESS if synchronous call
< 0	Error number (s. AGL_Defines.H, AGL_Defines.BAS or AGLink40.PAS)

8.4.7 Writing data bytes (AGL_WriteDataBytes)

With this function you can write the PLC's data bytes with the desired number. For this the PLC must be already connected. Using a S5 you can access DX modules when indicating a DB number >255.

C/C++-Syntax:

```
int WINAPI AGL_WriteDataBytes( int ConnNr, int DBNr, int Start, int
Num, PBYTE Buff, int Timeout, int UserVal );
```

Visual Basic-Syntax:

```
Declare Function AGL_WriteDataBytes Lib "AGLink40.DLL" (ByVal ConnNr
As Long, ByVal DBNr As Long, ByVal Start As Long, ByVal Num As Long,
Buff As Byte, ByVal Timeout As Long, ByVal UserVal As Long) As Long
```

Delphi-Syntax:

```
function AGL_WriteDataBytes( ConnNr: Integer; DBNr: Integer; Start:
Integer; Num: Integer; var Buff: Byte; Timeout: Integer; UserVal:
LongInt ): Integer; stdcall;
```

Parameter:

ConnNr	Variable for connection handle
DBNr	DB number which should be read from
Start	From which byte should be read
Num	Number of bytes which have to be read
Buff	Buffer for bytes read
Timeout	Timeout value which has to be used
UserVal	Value for free usage

Return value:

>= 0	Job number asynchronous call respectively AGL40_SUCCESS if synchronous call
< 0	Error number (s. AGL_Defines.H, AGL_Defines.BAS or AGLink40.PAS)

8.4.8 Writing S5 data words (AGL_WriteDataWords)

With this function you can write data words from a PLC's S5 addressing with the desired number. For this the PLC must be already connected. When accessing data word 10 on a S5 it causes that DW 10 will be read. But if using a S7 DBB 20 and DBB 21 will be read. Indicating a DB number >255 on a S5 you can access DX modules.

Note:
This function is only implemented for the S5 family. In other cases you will receive the following error message AGL40_FUNC_NOT_IMPLEMENTED.

C/C++-Syntax:

```
int WINAPI AGL_WriteDataWords( int ConnNr, int DBNr, int Start, int
Num, PWORD Buff, int Timeout, int UserVal );
```

Visual Basic-Syntax:

```
Declare Function AGL_WriteDataWords Lib "AGLink40.DLL" (ByVal ConnNr
As Long, ByVal DBNr As Long, ByVal Start As Long, ByVal Num As Long,
Buff As Integer, ByVal Timeout As Long, ByVal UserVal As Long) As
Long
```

Delphi-Syntax:

```
function AGL_WriteDataWords( ConnNr: Integer; DBNr: Integer; Start:
Integer; Num: Integer; var Buff: Word; Timeout: Integer; UserVal:
LongInt ): Integer; stdcall;
```

Parameter:

ConnNr	Variable for connection handle
DBNr	DB number which should be read from
Start	From which byte should be read
Num	Number of bytes which have to be read
Buff	Buffer for bytes read
Timeout	Timeout value which has to be used
UserVal	Value for free usage

Return value:

>= 0	Job number asynchronous call respectively AGL40_SUCCESS if synchronous call
< 0	Error number (s. AGL_Defines.H, AGL_Defines.BAS or AGLink40.PAS)

8.4.9 Writing timer words (AGL_WriteTimerWords)

With this function you can write the PLC's timer words with the desired number. For this the PLC must be already connected.

C/C++-Syntax:

```
int WINAPI AGL_WriteTimerWords( int ConnNr, int Start, int Num,
PWORD Buff, int Timeout, int UserVal );
```

Visual Basic-Syntax:

```
Declare Function AGL_WriteTimerWords Lib "AGLink40.DLL" (ByVal
ConnNr As Long, ByVal Start As Long, ByVal Num As Long, Buff As
Integer, ByVal Timeout As Long, ByVal UserVal As Long) As Long
```

Delphi-Syntax:

```
function AGL_WriteTimerWords( ConnNr: Integer; Start: Integer; Num:
Integer; var Buff: WORD; Timeout: Integer; UserVal: LongInt ):
Integer; stdcall;
```

Parameter:

ConnNr	Variable for connection handle
Start	From which byte should be read
Num	Number of bytes which have to be read
Buff	Buffer for bytes read
Timeout	Timeout value which has to be used
UserVal	Value for free usage

Return value:

>= 0	Job number asynchronous call respectively AGL40_SUCCESS if synchronous call
< 0	Error number (s. AGL_Defines.H, AGL_Defines.BAS or AGLink40.PAS)

8.4.10 Writing counter words (AGL_WriteCounterWords)

With this function you can write the PLC's counter words with the desired number. For this the PLC must be already connected.

C/C++-Syntax:

```
int WINAPI AGL_WriteCounterWords( int ConnNr, int Start, int Num,
PWORD Buff, int Timeout, int UserVal );
```

Visual Basic-Syntax:

```
Declare Function AGL_WriteCounterWords Lib "AGLink40.DLL" (ByVal
ConnNr As Long, ByVal Start As Long, ByVal Num As Long, Buff As
Integer, ByVal Timeout As Long, ByVal UserVal As Long) As Long
```

Delphi-Syntax:

```
function AGL_WriteCounterWords( ConnNr: Integer; Start: Integer;
Num: Integer; var Buff: WORD; Timeout: Integer; UserVal: LongInt ):
Integer; stdcall;
```

Parameter:

ConnNr	Variable for connection handle
Start	From which byte should be read
Num	Number of bytes which have to be read
Buff	Buffer for bytes read
Timeout	Timeout value which has to be used
UserVal	Value for free usage

Return value:

>= 0	Job number asynchronous call respectively AGL40_SUCCESS if synchronous call
< 0	Error number (s. AGL_Defines.H, AGL_Defines.BAS or AGLink40.PAS)

8.4.11 Mixed writing job (AGL_WriteMix)

With this function you can write different data file types and data file sizes into one job. For this the PLC must be already connected.

C/C++-Syntax:

```
int WINAPI AGL_WriteMix( int ConnNr, LPDATA_RW40 Buff, int Num, int
Timeout, int UserVal );
```

Visual Basic-Syntax:

```
Declare Function AGL_WriteMix Lib "AGLink40.DLL" (ByVal ConnNr As
Long, Buff As DATA_RW40, ByVal Num As Long, ByVal Timeout As Long,
ByVal UserVal As Long) As Long
```

Delphi-Syntax:

```
function AGL_WriteMix( ConnNr: Integer; var Buff: DATA_RW40; Num:
Integer; Timeout: Integer; UserVal: LongInt ): Integer; stdcall;
```

Parameter:

ConnNr	Variable for connection handle
Buff	Indicator on writing structures
Num	Number of structures
Timeout	Timeout value which has to be used
UserVal	Value for free usage

Return value:

>= 0	Job number asynchronous call respectively AGL40_SUCCESS if synchronous call
< 0	Error number (s. AGL_Defines.H, AGL_Defines.BAS or AGLink40.PAS)

The DATA_RW40 structure has the following build up:

```
typedef struct tagDATA_RW40
{
  WORD  OpArea;    // Operand section which is accessed to
  WORD  OpType;    // Data type
  WORD  OpAnz;     // Only important when using ReadMixEx and WriteMixEx
  WORD  DBNr;      // Number of data module
  WORD  Offset;    // Offset for memory access (when using T+Z number)
```

```
  WORD  BitNr;     // Bit number (if necessary)
  int   Result;    // Result (Fehlercode) of the reading respectively
                   // writing operation
  union
  {
    DWORD Value;   // Operand value as  DWORD (ReadMix and WriteMix)
    WORD  W[2];    // For easier access (ReadMix and WriteMix)
    BYTE  B[4];    // For easier access (ReadMix and WriteMix)
    float fValue;  // Operand value as single (ReadMix and WriteMix)
    PVOID Buff;    // Indicator on buffer (ReadMixEx and WriteMixEx)
  };
} DATA_RW40, *LPDATA_RW40;
```

These fields have the following meaning:

OpArea	Operand type, valid values are AREA_IN (processing image inputs), AREA_OUT (processing image outputs), AREA_FLAG (marker), AREA_DATA (data files), AREA_TIMER (timers), AREA_COUNTER (counter), AREA_SFLAG_200 (special marker of series 200), AREA_VAR_200 (variable memory of series 200), AREA_TIMER_200 (timers of series 200), AREA_COUNTER_200 (counter of series 200) and AREA_PERIPHERIE (periphery inputs)
OpType	Operand type. Valid values are TYP_BIT, TYP _BYTE, TYP _WORD and TYP _DWORD. Timer must have TYP_TIMER there and counter must have TYP_COUNTER there (respectively the corresponding version of the 200 series). TYP _BIT is not valid when using AREA_PERIPHERIE and leads to an error return.
DBNr	The DB number, when it is about a data file access.
ByteNr	Offset for memory access
BitNr	Bit number, when it is about SIZE_BIT using the access size.
Result	Result of writing operation.
Value, W, B, fValue	Value which have to be read

8.4.12 Extended mixed writing job (AGL_WriteMixEX)

With this function you can write different data file types and data sizes in one job. For this the PLC must be already connected. The element Buff of the DATA_RW40 structure is used here. There is an indicator on the memory which should be written.

C/C++-Syntax:

```
int WINAPI AGL_WriteMixEx( int ConnNr, LPDATA_RW40 Buff, int Num,
int Timeout, int UserVal );
```

Visual Basic-Syntax:

```
Declare Function AGL_WriteMixEx Lib "AGLink40.DLL" (ByVal ConnNr As
Long, Buff As DATA_RW40, ByVal Num As Long, ByVal Timeout As Long,
ByVal UserVal As Long) As Long
```

Delphi-Syntax:

```
function AGL_WriteMixEx( ConnNr: Integer; var Buff: DATA_RW40; Num:
Integer; Timeout: Integer; UserVal: LongInt ): Integer; stdcall;
```

Parameter:

ConnNr	Variable for connection handle
Buff	Indicator on writing structures
Num	Number of structures
Timeout	Timeout value which has to be used
UserVal	Value for free usage

Return value:

>= 0	Job number asynchronous call respectively AGL40_SUCCESS if synchronous call
< 0	Error number (s. AGL_Defines.H, AGL_Defines.BAS or AGLink40.PAS)

The DATA_RW40 structure has the following build up:

```
typedef struct tagDATA_RW40
{
  WORD  OpArea;    // Operand section which is accessed to
  WORD  OpType;    // Data file type
  WORD  OpAnz;     // Only important when using ReadMixEx and WriteMixEx
  WORD  DBNr;      // Numnber of data module
```

```
  WORD  Offset;    // Offset for data access (when using T+Z number)
  WORD  BitNr;     // Bit number (if necessary)
  int   Result;    // Result (error code) of the reading and writing
operation
  union
  {
    DWORD Value;   // Operand value as DWORD (ReadMix and WriteMix)
    WORD  W[2];    // For easier access (ReadMix and WriteMix)
    BYTE  B[4];    // For easier access (ReadMix and WriteMix)
    float fValue;  // Operand value as single (ReadMix and WriteMix)
    PVOID Buff;    // Indicator on buffer (ReadMixEx and WriteMixEx)
  };
} DATA_RW40, *LPDATA_RW40;
```

These fields have the following meaning:

OpArea	Operand type, valid values are AREA_IN (processing image inputs), AREA_OUT (processing image outputs), AREA_FLAG (marker), AREA_DATA (data files), AREA_TIMER (timers), AREA_COUNTER (counter), AREA_SFLAG_200 (special marker of series 200), AREA_VAR_200 (variable memory of series 200), AREA_TIMER_200 (timers of series 200), AREA_COUNTER_200 (counter of series 200) and AREA_PERIPHERIE (periphery inputs)
OpType	Operand type. Valid values are TYP_BIT, TYP _BYTE, TYP _WORD and TYP _DWORD. Timer must have TYP_TIMER there and counter must have TYP_COUNTER there (respectively the corresponding version of the 200 series). TYP _BIT is not valid when using AREA_PERIPHERIE and leads to an error return.
DBNr	The DB number, when it is about a data file access.
ByteNr	Offset for memory access
BitNr	Bit number, when it is about SIZE_BIT using the access size.
Result	Result of writing operation.
Buff	Indicator on buffer with data which have to be written

8.4.13 Writing data block (AGL_BSend)

With this function you can send a data block which is received by the PLC with BReceive. For this, a connection has to be parametrized. At present only Industrial Ethernet with the ACCON-AGLink communication setting TYPE_S7CONN_IE is supported. For this the PLC must be already connected.

C/C++-Syntax:

```
int WINAPI AGL_BSend( int ConnNr, LPWSABUF pwsa, int Timeout, int
UserVal );
```

Visual Basic-Syntax:

```
Declare Function AGL_BSend Lib "AGLink40.DLL" (ByVal ConnNr As Long,
pwsa As WSABUFF, ByVal Timeout As Long, ByVal UserVal As Long) As
Long
```

Delphi-Syntax:

```
function AGL_BSend( ConnNr: Integer; var pwsa: TagWSABUFF; Timeout:
Integer; UserVal: LongInt ): Integer; stdcall;
```

Parameter:

ConnNr	Variable for connection handle
pwsa	Indicator on WSA buffer
Timeout	Timeout value which has to be used
UserVal	Value for free usage

Return value:

>= 0	Job number asynchronous call respectively AGL40_SUCCESS if synchronous call
< 0	Error number (s. AGL_Defines.H, AGL_Defines.BAS or AGLink40.PAS)

The WSABUF structure has the following build up:

```
typedef struct tagWSABUF
{
  ULONG  len;      // buffer length
  char  *buf;      // pointer to buffer
} WSABUF, *LPWSABUF;
```

8.4.14 Writing data block using R_ID (AGL_BSendEx)

With this function you can send a data block which is received by the PLC using BReceive. Compared with the function AGL_BSend, the R_ID will be indicated, too. Using AGL_BSend 1 is default for the R_ID. For this a connection has to be parametrized. At present only Industrial Ethernet with the AGLink communication setting TYPE_S7CONN_IE is supported. To this PLC a connection has to be already established to.

C/C++-Syntax:

```
int WINAPI AGL_BSendEx( int ConnNr, LPWSABUF pwsa, int R_ID, int
Timeout, int UserVal );
```

Visual Basic-Syntax:

```
Declare Function AGL_BSendEx Lib "AGLink40.DLL" (ByVal ConnNr As
Long, pwsa As WSABUFF, ByVal R_ID, ByVal Timeout As Long, ByVal
UserVal As Long) As Long
```

Delphi-Syntax:

```
function AGL_BSend( ConnNr: Integer; var pwsa: TagWSABUFF; R_ID:
Integer; Timeout: Integer; UserVal: LongInt ): Integer; stdcall;
```

Parameter:

ConnNr	Variable for connection handle
pwsa	Indicator on WSA buffer
R_ID	RemoteID for the communication
Timeout	Timeout value which has to be used
UserVal	Value for free usage

Return value:

>= 0	Job number asynchronous call respectively AGL40_SUCCESS if synchronous call
< 0	Error number (s. AGL_Defines.H, AGL_Defines.BAS or AGLink40.PAS)

9 Programming reference optimization and storage functions

With this function you can access your control in a optimized way. Depending on the query and currently used controler, the packages will be seperated so that they can be answered as fast as possible. The procedure is the following: you hand over the array including the DATA_RW40 structures to the optimization functions. Internally, ressources will be allocated and the strucutres will be packed, so that the PLC's query can be optimally processed then. Now you can read or write the data with the respective function as often as you like. If you do not need the optimized querry anymore, just release the ressources with a function.

Attention: Every Init function has to be closed with an Exit function.

9.1 Optimization functions

9.1.1 Optimizing ReadMix query (AGL_InitOptReadMix)

With this function a ReadMix query will be optimized. A connection to the PLC must be established.

C/C++-Syntax:

```
int WINAPI AGL_InitOptReadMix( int ConnNr, LPDATA_RW40 Buff, int
Num, int *Opt );
```

Visual Basic-Syntax:

```
Declare Function AGL_InitOptReadMix Lib "AGLink40.DLL" (ByVal ConnNr
As Long, ByRef Buff As DATA_RW40, ByVal Num As Long, ByRef Opt As
Long) As Long
```

Delphi-Syntax:

```
function AGL_InitOptReadMix( ConnNr: Integer; var Buff:
TagDATA_RW40; Num: Integer; var Opt: Integer ): Integer; stdcall;
```

Parameter:

ConnNr	Variable for connection handle
Buff	Indicator on DATA_RW40 structures
Num	Number of used structures
Opt	Indicator on variable for optimization handle

Return value:

= 0	AGL40_SUCCESS if successful
< 0	Error number (s. AGL_Defines.H, AGL_Defines.BAS or AGLink40.PAS)

9.1.2 Executing optimized ReadMix query (AGL_ReadOptReadMix)

With this function an optimized ReadMix query will be executed. A connection to the PLC must be established.

C/C++-Syntax:

```
int WINAPI AGL_ReadOptReadMix( int ConnNr, int Opt, int Timeout,
long UserVal );
```

Visual Basic-Syntax:

```
Declare Function AGL_ReadOptReadMix Lib "AGLink40.DLL" (ByVal ConnNr
As Long, ByVal Opt As Long, ByVal Timeout As Long, ByVal UserVal As
Long) As Long
```

Delphi-Syntax:

```
function AGL_ReadOptReadMix( ConnNr: Integer; Opt: Integer; Timeout:
Integer; UserVal: LongInt ): Integer; stdcall;
```

Parameter:

ConnNr	Variable for connection handle
Opt	Optimization handle
Timeout	Timeout value which has to be used
UserVal	Value for free usage

Return value:

= 0	AGL40_SUCCESS if successful
< 0	Error number (s. AGL_Defines.H, AGL_Defines.BAS or AGLink40.PAS)

9.1.3 Releasing optimized ReadMix query (AGL_EndOptReadMix)

With this function ressources of an optimized ReadMix query will be released again.

C/C++-Syntax:

```
int WINAPI AGL_EndOptReadMix( int Opt );
```

Visual Basic-Syntax:

```
Declare Function AGL_EndOptReadMix Lib "AGLink40.DLL" (ByVal Opt As
Long) As Long
```

Delphi-Syntax:

```
function AGL_EndOptReadMix( Opt: Integer ): Integer; stdcall;
```

Parameter:

Opt	Optimization handle

Return value:

= 0	AGL40_SUCCESS if successful
< 0	Error number (s. AGL_Defines.H, AGL_Defines.BAS or AGLink40.PAS)

9.1.4 Optimizing ReadMixExquery (AGL_InitOptReadMixEx)

With this function a ReadMixEx query will be optimized. A connection to the PLC must be established.

C/C++-Syntax:

```
int WINAPI AGL_InitOptReadMixEx( int ConnNr, LPDATA_RW40 Buff, int
Num, int *Opt );
```

Visual Basic-Syntax:

```
Declare Function AGL_InitOptReadMixEx Lib "AGLink40.DLL" (ByVal
ConnNr As Long, ByRef Buff As DATA_RW40, ByVal Num As Long, ByRef
Opt As Long) As Long
```

Delphi-Syntax:

```
function AGL_InitOptReadMixEx( ConnNr: Integer; var Buff:
TagDATA_RW40; Num: Integer; var Opt: Integer ): Integer; stdcall;
```

Parameter:

ConnNr	Variable for connection handle
Buff	Indicator on DATA_RW40 structures
Num	Number of used structures
Opt	Indicator on variable for optimization handle.

Return value:

= 0	AGL40_SUCCESS if successful
< 0	Error number (s. AGL_Defines.H, AGL_Defines.BAS or AGLink40.PAS)

9.1.5 Executing optimized ReadMixEx query (AGL_ReadOptReadMixEx)

With this function an optimized ReadMixEx query will be carried out. A connection to the PLC must be established.

C/C++-Syntax:

```
int WINAPI AGL_ReadOptReadMixEx( int ConnNr, int Opt, int Timeout,
long UserVal );
```

Visual Basic-Syntax:

```
Declare Function AGL_ReadOptReadMixEx Lib "AGLink40.DLL" (ByVal
ConnNr As Long, ByVal Opt As Long, ByVal Timeout As Long, ByVal
UserVal As Long) As Long
```

Delphi-Syntax:

```
function AGL_ReadOptReadMixEx( ConnNr: Integer; Opt: Integer;
Timeout: Integer; UserVal: LongInt ): Integer; stdcall;
```

Parameter:

ConnNr	Variable for connection handle
Opt	Optimization handle
Timeout	Timeout value which has to be used
UserVal	Value for free usage

Return value:

= 0	AGL40_SUCCESS if successful
< 0	Error number (s. AGL_Defines.H, AGL_Defines.BAS or AGLink40.PAS)

9.1.6 Releasing optimized ReadMixEx query (AGL_EndOptReadMixEx)

With this function the ressources of an optimized ReadMix query will be released again.

C/C++-Syntax:

```
int WINAPI AGL_EndOptReadMixEx( int Opt );
```

Visual Basic-Syntax:

```
Declare Function AGL_EndOptReadMixEx Lib "AGLink40.DLL" (ByVal Opt
As Long) As Long
```

Delphi-Syntax:

```
function AGL_EndOptReadMixEx( Opt: Integer ): Integer; stdcall;
```

Parameter:

Opt	Optimization handle

Return value:

= 0	AGL40_SUCCESS if successful
< 0	Error number (s. AGL_Defines.H, AGL_Defines.BAS or AGLink40.PAS)

9.1.7 Optimizing WriteMix query (AGL_InitOptWriteMix)

With this function a WriteMix query will be optimized. A connection to the PLC must be established.

C/C++-Syntax:

```
int WINAPI AGL_InitOptWriteMix( int ConnNr, LPDATA_RW40 Buff, int
Num, int *Opt );
```

Visual Basic-Syntax:

```
Declare Function AGL_InitOptWriteMix Lib "AGLink40.DLL" (ByVal
ConnNr As Long, ByRef Buff As DATA_RW40, ByVal Num As Long, ByRef
Opt As Long) As Long
```

Delphi-Syntax:

```
function AGL_InitOptWriteMix( ConnNr: Integer; var Buff:
TagDATA_RW40; Num: Integer; var Opt: Integer ): Integer; stdcall;
```

Parameter:

ConnNr	Variable for connection handle
Buff	Indicator on DATA_RW40 structures
Num	Number of used strucutres
Opt	Indicator on variable for optimization handle

Return value:

= 0	AGL40_SUCCESS if successful
< 0	Error number (s. AGL_Defines.H, AGL_Defines.BAS or AGLink40.PAS)

9.1.8 Executing optimized WriteMix query (AGL_WriteOptWriteMix)

With this function an optimized WriteMix query will be executed. A connection to the PLC must be established.

C/C++-Syntax:

```
int WINAPI AGL_WriteOptWriteMix( int ConnNr, int Opt, int Timeout,
long UserVal );
```

Visual Basic-Syntax:

```
Declare Function AGL_WriteOptWriteMix Lib "AGLink40.DLL" (ByVal
ConnNr As Long, ByVal Opt As Long, ByVal Timeout As Long, ByVal
UserVal As Long) As Long
```

Delphi-Syntax:

```
function AGL_WriteOptWriteMix( ConnNr: Integer; Opt: Integer;
Timeout: Integer; UserVal: LongInt ): Integer; stdcall;
```

Parameter:

ConnNr	Variable for connection handle
Opt	Optimization handle
Timeout	Timeout value which has to be used
UserVal	Value for free usage

Return value:

= 0	AGL40_SUCCESS if successful
< 0	Error number (s. AGL_Defines.H, AGL_Defines.BAS or AGLink40.PAS)

9.1.9 Releasing optimized WriteMix query (AGL_EndOptWriteMix)

With this function ressources of an optimized WriteMix query will be released again.

C/C++-Syntax:

```
int WINAPI AGL_EndOptWriteMix( int Opt );
```

Visual Basic-Syntax:

```
Declare Function AGL_EndOptWriteMix Lib "AGLink40.DLL" (ByVal Opt As
Long) As Long
```

Delphi-Syntax:

```
function AGL_EndOptWriteMix( Opt: Integer ): Integer; stdcall;
```

Parameter:

Opt	Optimization handle

Return value:

= 0	AGL40_SUCCESS if successful
< 0	Error number (s. AGL_Defines.H, AGL_Defines.BAS or AGLink40.PAS)

9.1.10 Optimizing WriteMixEx query (AGL_InitOptWriteMixEx)

With this function a WriteMix query will be optimized. A connection to the PLC must be established. A connection to the PLC must be established.

C/C++-Syntax:

```
int WINAPI AGL_InitOptWriteMixEx( int ConnNr, LPDATA_RW40 Buff, int
Num, int *Opt );
```

Visual Basic-Syntax:

```
Declare Function AGL_InitOptWriteMixEx Lib "AGLink40.DLL" (ByVal
ConnNr As Long, ByRef Buff As DATA_RW40, ByVal Num As Long, ByRef
Opt As Long) As Long
```

Delphi-Syntax:

```
function AGL_InitOptWriteMixEx( ConnNr: Integer; var Buff:
TagDATA_RW40; Num: Integer; var Opt: Integer ): Integer; stdcall;
```

Parameter:

ConnNr	Variable for connection handle
Buff	Indicator on DATA_RW40 structures
Num	Number of used structures
Opt	Indicator on variable for optimization handle

Return value:

= 0	AGL40_SUCCESS if successful
< 0	Error number (s. AGL_Defines.H, AGL_Defines.BAS or AGLink40.PAS)

9.1.11 Executing optimized WriteMixEx query (AGL_WriteOptWriteMixEx)

With this function an optimized WriteMixEx query will be executed. A connection to the PLC must be established.

C/C++-Syntax:

```
int WINAPI AGL_WriteOptWriteMixEx( int ConnNr, int Opt, int Timeout,
long UserVal );
```

Visual Basic-Syntax:

```
Declare Function AGL_WriteOptWriteMixEx Lib "AGLink40.DLL" (ByVal
ConnNr As Long, ByVal Opt As Long, ByVal Timeout As Long, ByVal
UserVal As Long) As Long
```

Delphi-Syntax:

```
function AGL_WriteOptWriteMixEx( ConnNr: Integer; Opt: Integer;
Timeout: Integer; UserVal: LongInt ): Integer; stdcall;
```

Parameter:

ConnNr	Variable for connection handle
Opt	Optimization handle
Timeout	Timeout value which has to be used
UserVal	Value for free usage

Return value:

= 0	AGL40_SUCCESS if successful
< 0	Error number (s. AGL_Defines.H, AGL_Defines.BAS or AGLink40.PAS)

9.1.12 Releasing optimized WriteMixEx query (AGL_EndOptWriteMixEx)

With this function the ressources of an optimized WriteMix query will be released again.

C/C++-Syntax:

```
int WINAPI AGL_EndOptWriteMixEx( int Opt );
```

Visual Basic-Syntax:

```
Declare Function AGL_EndOptWriteMixEx Lib "AGLink40.DLL" (ByVal Opt
As Long) As Long
```

Delphi-Syntax:

```
function AGL_EndOptWriteMixEx( Opt: Integer ): Integer; stdcall;
```

Parameter:

Opt	Optimization handle

Return value:

= 0	AGL40_SUCCESS if successful
< 0	Error number (s. AGL_Defines.H, AGL_Defines.BAS or AGLink40.PAS)

9.1.13 Entering notification for optimized query (AGL_SetOptNotification)

With this function a notification for the optimized query will be entered. You have to do this only once after the optimization.

C/C++-Syntax:

```
int WINAPI AGL_SetOptNotification( int Opt, LPNOTIFICATION pN );
```

Visual Basic-Syntax:

```
Declare Function AGL_SetOptNotification Lib "AGLink40.DLL" (ByVal
Opt As Long, ByRef pN As NOTIFICATION) As Long
```

Delphi-Syntax:

```
function AGL_SetOptNotification( Opt: Integer; var pN:
TagNOTIFICATION ): Integer; stdcall;
```

Parameter:

Opt	Optimization handle
pN	Indicator on notification structure

Return value:

= 0	AGL40_SUCCESS if successful
< 0	Error number (s. AGL_Defines.H, AGL_Defines.BAS or AGLink40.PAS)

9.2 Storage functions

This function is especially for programming languages in which the handling of indicators on memory in the DATA_RW40 structure is complex (e.g. Visual Basic or .net programming languages).

9.2.1 Allocating memory for DATA_RW40 structures (AGL_AllocRWBuffs)

With this function the needed memory will be allocated for the DATA_RW40 structures according to the entered values, and the indicators will be entered into the structures.

C/C++-Syntax:

```
int WINAPI AGL_AllocRWBuffs( LPDATA_RW40 Buff, int Num );
```

Visual Basic-Syntax:

```
Declare Function AGL_AllocRWBuffs Lib "AGLink40.DLL" (ByRef Buff As
DATA_RW40, ByVal Num As Long) As Long
```

Delphi-Syntax:

```
function AGL_AllocRWBuffs( var Buff: TagDATA_RW40; Num: Integer ):
Integer; stdcall;
```

Parameter:

Buff	Indicator on DATA_RW40 structures
Num	Number of structures

Return value:

0	If memory could not be allocated
!= 0	Handle for releasing memory

9.2.2 Releasing allocated memory (AGL_FreeRWBuffs)

This function releases the memory allocated by AGL_AllocRWBuffs again.

C/C++-Syntax:

```
int WINAPI AGL_FreeRWBuffs( int Handle );
```

Visual Basic-Syntax:

```
Declare Function AGL_FreeRWBuffs Lib "AGLink40.DLL" (ByVal Handle As
Long) As Long
```

Delphi-Syntax:

```
function AGL_FreeRWBuffs( Handle: Integer ): Integer; stdcall;
```

Parameter:

Handle	Return value from AGL_AllocRWBuffs

Return value::

= 0	Always AGL40_SUCCESS

9.2.3 Reading from memory (AGL_ReadRWBuff)

With this function you have reading access to data of a single structure. This function will be called by the PLC after reading the data.

C/C++-Syntax:

```
int WINAPI AGL_ReadRWBuff( LPDATA_RW40 Buff, int Index, PVOID pData
);
```

Visual Basic-Syntax:

```
Declare Function AGL_ReadRWBuff Lib "AGLink40.DLL" (ByRef Buff As
DATA_RW40, ByVal Index As Long, pData As Any) As Long
```

Delphi-Syntax:

```
function  AGL_ReadRWBuffs( var Buff: TagDATA_RW40; Index: Integer;
pData: Pointer ): Integer; stdcall;
```

Parameter:

Buff	Indicator on the DATA_RW40 structure array
Index	Index which should be accessed
pData	Indicator on memory into the values should be written in

Return value:

= 0	AGL40_SUCCESS if successful
< 0	Error number (s. AGL_Defines.H, AGL_Defines.BAS or AGLink40.PAS)

9.2.4 Writing into memory (AGL_WriteRWBuff)

With this function you have writing access to data of a single structure. This function will be called by the PLC after writing the data.

C/C++-Syntax:

```
int WINAPI AGL_WriteRWBuff( LPDATA_RW40 Buff, int Index, PVOID pData
);
```

Visual Basic-Syntax:

```
Declare Function AGL_WriteRWBuff Lib "AGLink40.DLL" (ByRef Buff As
DATA_RW40, ByVal Index As Long, pData As Any) As Long
```

Delphi-Syntax:

```
function  AGL_WriteRWBuff( var Buff: TagDATA_RW40; Index: Integer;
pData: Pointer ): Integer; stdcall;
```

Parameter:

Buff	Indicator on the DATA_RW40 structure array
Index	Index which should be accessed
pData	Indicator on memory into the values should be written in

Return value:

= 0	AGL40_SUCCESS if successful
< 0	Error number (s. AGL_Defines.H, AGL_Defines.BAS or AGLink40.PAS)

10 Programming reference RK512- respectively 3964R-functions

With RK 512 you can use the familiar reading and writing data functions. With the functions in this chapter you have the possibility to focus more on RK512 respectively 3964R. Furthermore, there are functions implemented which allow you to react on a RKSend or RKFetch telegram from the remote station. Through this you can develop an application which reacts in the same way as a real PLC.

To use the RK and 3964 related functions a connection to a PLC must be established. Because the connection handle is always passed as parameter.

All RK-related functions contain the following structures as parameters:

```
typedef struct tagDATA_RW40_RK
{
  WORD  OpArea;   // Operand section which is accessed to
  WORD  OpAnz;    // Number of elements (bytes or words)
  WORD  DBNr;     // Number of data module using DB or DX
  WORD  Offset;   // Offset for access
  BYTE  KMByte;   // Byte number of coordination marker (0-255)
  BYTE  KMBit;    // Bit number of coordination marker (0-7 or 15)
  BYTE  CPUNr;    // CPU number (1-4 or 15)
  BYTE  dummy;    // Filler byte for Alignment
  PVOID Buff;     // Indicator on buffer
  int   BuffLen;  // Buffer length (AGL_RKFetch, AGL_Recv_RKSend, etc.)
} DATA_RW40_RK, *LPDATA_RW40_RK;
```

The 3964-related functions only contain a byte buffer and the buffer length as parameter.

Now follows a detailed description of every function.

10.1 RK512-related functions

10.1.1 Reading data (AGL_RKFetch)

With this function you can read out data files from a PLC using the RK512 protocol. The communication parameters CPU number, the couple marker byte and couple marker bit will be accepted from the device settings. Only the ACCON-AGLink communication setting TYPE_RK is supported. Otherwise the error message AGL40_FUNC_NOT_SUPPORTED will be output. For this the PLC must be already connected.

C/C++-Syntax:

```
int WINAPI AGL_RKFetch( int ConnNr, LPDATA_RW40_RK Buff, int
Timeout, long UserVal );
```

Visual Basic-Syntax:

```
Declare Function AGL_RKFetch Lib "AGLink40.DLL" (ByVal ConnNr As
Long, ByRef Buff As DATA_RW40_RK, ByVal Timeout As Long, ByVal
UserVal As Long) As Long
```

Delphi-Syntax:

```
function AGL_RKFetch( ConnNr: Integer; var Buff: DATA_RW40_RK;
Timeout: Integer; UserVal: LongInt ): Integer; stdcall;
```

Parameter:

ConnNr	Variable for connection handle
Buff	Indicator on DATA_RW40_RK buffer
Timeout	Timeout value which has to be used
UserVal	Value for free usage

Return value:

>= 0	Job number asynchronous call respectively AGL40_SUCCESS if synchronous call
< 0	Error number (s. AGL_Defines.H, AGL_Defines.BAS or AGLink40.PAS)

10.1.2 Extended data reading (AGL_RKFetchEx)

With this function you can read out data files from a PLC using the RK512 protocol. The communication parameters CPU number, the couple marker byte and couple marker bit will be accepted from the DATA_RW40_RK structure. Only the ACCON-AGLink communication setting TYPE_RK is supported. Otherwise the error message AGL40_FUNC_NOT_SUPPORTED will be output. For this the PLC must be already connected.

C/C++-Syntax:

```
int WINAPI AGL_RKFetchEx( int ConnNr, LPDATA_RW40_RK Buff, int
Timeout, long UserVal );
```

Visual Basic-Syntax:

```
Declare Function AGL_RKFetchEx Lib "AGLink40.DLL" (ByVal ConnNr As
Long, ByRef Buff As DATA_RW40_RK, ByVal Timeout As Long, ByVal
UserVal As Long) As Long
```

Delphi-Syntax:

```
function AGL_RKFetchEx( ConnNr: Integer; var Buff: DATA_RW40_RK;
Timeout: Integer; UserVal: LongInt ): Integer; stdcall;
```

Parameter:

ConnNr	Variable for connection handle
Buff	Indicator on DATA_RW40_RK buffer
Timeout	Timeout value which has to be used
UserVal	Value for free usage

Return value:

>= 0	Job number asynchronous call respectively AGL40_SUCCESS if synchronous call
< 0	Error number (s. AGL_Defines.H, AGL_Defines.BAS or AGLink40.PAS)

10.1.3 Writing data (AGL_RKSend)

With this function you can write data files into the PLC using the RK512 protocol. The communication parameters CPU number, the couple marker byte and couple marker bit will be accepted from the device settings. Only the ACCON-AGLink communication setting TYPE_RK is supported. Otherwise the error message AGL40_FUNC_NOT_SUPPORTED will be output. For this the PLC must be already connected.

C/C++-Syntax:

```
int WINAPI AGL_RKSend( int ConnNr, LPDATA_RW40_RK Buff, int Timeout,
long UserVal );
```

Visual Basic-Syntax:

```
Declare Function AGL_RKSend Lib "AGLink40.DLL" (ByVal ConnNr As
Long, ByRef Buff As DATA_RW40_RK, ByVal Timeout As Long, ByVal
UserVal As Long) As Long
```

Delphi-Syntax:

```
function AGL_RKSend( ConnNr: Integer; var Buff: DATA_RW40_RK;
Timeout: Integer; UserVal: LongInt ): Integer; stdcall;
```

Parameter:

ConnNr	Variable for connection handle
Buff	Indicator on DATA_RW40_RK buffer
Timeout	Timeout value which has to be used
UserVal	Value for free usage

Return value:

>= 0	Job number asynchronous call respectively AGL40_SUCCESS if synchronous call
< 0	Error number (s. AGL_Defines.H, AGL_Defines.BAS or AGLink40.PAS)

10.1.4 Extended writing data (AGL_RKSendEx)

With this function you can write data files into the PLC using the RK512 protocol. The communication parameters CPU number, the couple marker byte and couple marker bit will be accepted from the device settings. Only the communication setting ACCON-AGLink TYPE_RK is supported. Otherwise the error message AGL40_FUNC_NOT_SUPPORTED will be output. For this the PLC must be already connected.

C/C++-Syntax:

```
int WINAPI AGL_RKSendEx( int ConnNr, LPDATA_RW40_RK Buff, int
Timeout, long UserVal );
```

Visual Basic-Syntax:

```
Declare Function AGL_RKSendEx Lib "AGLink40.DLL" (ByVal ConnNr As
Long, ByRef Buff As DATA_RW40_RK, ByVal Timeout As Long, ByVal
UserVal As Long) As Long
```

Delphi-Syntax:

```
function AGL_RKSendEx( ConnNr: Integer; var Buff: DATA_RW40_RK;
Timeout: Integer; UserVal: LongInt ): Integer; stdcall;
```

Parameter:

ConnNr	Variable for connection handle
Buff	Indicator on DATA_RW40_RK buffer
Timeout	Timeout value which has to be used
UserVal	Value for free usage

Return value:

>= 0	Job number asynchronous call respectively AGL40_SUCCESS if synchronous call
< 0	Error number (s. AGL_Defines.H, AGL_Defines.BAS or AGLink40.PAS)

10.1.5 Receive RKSend from the remote station (AGL_Recv_RKSend)

With this function you can receive data files which have been written by the remote station using RKSend. All communication parameters e.g. memory area, DB number, offset, etc. will be entered into the DATA_RW40_RK structure. Only the communication setting ACCON-AGLink TYPE_RK is supported. Otherwise the error message AGL40_FUNC_NOT_SUPPORTED will be output. For this the PLC must be already connected.

C/C++-Syntax:

```
int WINAPI AGL_Recv_RKSend( int ConnNr, LPDATA_RW40_RK Buff, int
Timeout, long UserVal );
```

Visual Basic-Syntax:

```
Declare Function AGL_Recv_RKSend Lib "AGLink40.DLL" (ByVal ConnNr As
Long, ByRef Buff As DATA_RW40_RK, ByVal Timeout As Long, ByVal
UserVal As Long) As Long
```

Delphi-Syntax:

```
function AGL_Recv_RKSend( ConnNr: Integer; var Buff: DATA_RW40_RK;
Timeout: Integer; UserVal: LongInt ): Integer; stdcall;
```

Parameter:

ConnNr	Variable for connection handle
Buff	Indicator on DATA_RW40_RK buffer
Timeout	Timeout value which has to be used
UserVal	Value for free usage

Return value:

>= 0	Job number asynchronous call respectively AGL40_SUCCESS if synchronous call
< 0	Error number (s. AGL_Defines.H, AGL_Defines.BAS or AGLink40.PAS)

10.1.6 Receive RKFetch from the remote station (AGL_Recv_RKFetch)

With this function you can receive a data file request written by the remote station using RKFetch. All communication parameters e.g. memory area, DB number, offset, etc. will be entered into the DATA_RW40_RK structure. After that you can send the desired data files using the AGL_Send_RKFetch function. Only the communication setting ACCON-AGLink TYPE_RK is supported. Otherwise the error message AGL40_FUNC_NOT_SUPPORTED will be output. For this the PLC must be already connected.

C/C++-Syntax:

```
int WINAPI AGL_Recv_RKFetch( int ConnNr, LPDATA_RW40_RK Buff, int
Timeout, long UserVal );
```

Visual Basic-Syntax:

```
Declare Function AGL_Recv_RKFetch Lib "AGLink40.DLL" (ByVal ConnNr
As Long, ByRef Buff As DATA_RW40_RK, ByVal Timeout As Long, ByVal
UserVal As Long) As Long
```

Delphi-Syntax:

```
function AGL_Recv_RKFetch( ConnNr: Integer; var Buff: DATA_RW40_RK;
Timeout: Integer; UserVal: LongInt ): Integer; stdcall;
```

Parameter:

ConnNr	Variable for connection handle
Buff	Indicator on DATA_RW40_RK buffer
Timeout	Timeout value which has to be used
UserVal	Value for free usage

Return value:

>= 0	Job number asynchronous call respectively AGL40_SUCCESS if synchronous call
< 0	Error number (s. AGL_Defines.H, AGL_Defines.BAS or AGLink40.PAS)

10.1.7 Replay RKFetch of the remote station (AGL_Send_RKFetch)

With this function you can send data files requested by the remote station using RKFetch. Only the communication setting ACCON-AGLink TYPE_RK is supported. Otherwise the error message AGL40_FUNC_NOT_SUPPORTED will be output. For this the PLC must be already connected.

C/C++-Syntax:

```
int WINAPI AGL_Send_RKFetch( int ConnNr, LPDATA_RW40_RK Buff, int
Timeout, long UserVal );
```

Visual Basic-Syntax:

```
Declare Function AGL_Send_RKFetch Lib "AGLink40.DLL" (ByVal ConnNr
As Long, ByRef Buff As DATA_RW40_RK, ByVal Timeout As Long, ByVal
UserVal As Long) As Long
```

Delphi-Syntax:

```
function AGL_Send_RKFetch( ConnNr: Integer; var Buff: DATA_RW40_RK;
Timeout: Integer; UserVal: LongInt ): Integer; stdcall;
```

Parameter:

ConnNr	Variable for connection handle
Buff	Indicator on DATA_RW40_RK buffer
Timeout	Timeout value which has to be used
UserVal	Value for free usage

Return value:

>= 0	Job number asynchronous call respectively AGL40_SUCCESS if synchronous call
< 0	Error number (s. AGL_Defines.H, AGL_Defines.BAS or AGLink40.PAS)

10.2 3964-related functions

10.2.1 Sending data via protocol 3964 (AGL_Send_3964)

With this function you can send data files via the 3964 or 3964R protocol. In the communication setting you can adjust which of these two protocols you want to use. Only the communication setting ACCON-AGLink TYPE_RK is supported. Otherwise the error message AGL40_FUNC_NOT_SUPPORTED will be output. For this the PLC must be already connected.

C/C++-Syntax:

```
int WINAPI AGL_Send_3964( int ConnNr, PBYTE Buff, int BuffLen, int
Timeout, long UserVal );
```

Visual Basic-Syntax:

```
Declare Function AGL_Send_3964 Lib "AGLink40.DLL" (ByVal ConnNr As
Long, ByRef Buff As Byte, ByVal BuffLen As Long, ByVal Timeout As
Long, ByVal UserVal As Long) As Long
```

Delphi-Syntax:

```
function AGL_Send_3964( ConnNr: Integer; var Buff: BYTE; BuffLen:
Integer; Timeout: Integer; UserVal: LongInt ): Integer; stdcall;
```

Parameter:

ConnNr	Variable for connection handle
Buff	Indicator on data files which have to be sent
BuffLen	Length of sending buffer
Timeout	Timeout value which has to be used
UserVal	Value for free usage

Return value:

>= 0	Job number asynchronous call respectively AGL40_SUCCESS if synchronous call
< 0	Error number (s. AGL_Defines.H, AGL_Defines.BAS or AGLink40.PAS)

10.2.2 Receiving data via protocol 3964 (AGL_Recv_3964)

With this function you can receive data files via the 3964 or 3964R protocol. In the communication setting you can adjust which of these two protocols you want to use. Only the communication setting ACCON-AGLink TYPE_RK is supported. Otherwise the error message AGL40_FUNC_NOT_SUPPORTED will be output. For this the PLC must be already connected.

C/C++-Syntax:

```
int WINAPI AGL_Recv_3964( int ConnNr, PBYTE Buff, int *BuffLen, int
Timeout, long UserVal );
```

Visual Basic-Syntax:

```
Declare Function AGL_Recv_3964 Lib "AGLink40.DLL" (ByVal ConnNr As
Long, ByRef Buff As Byte, ByRef BuffLen As Long, ByVal Timeout As
Long, ByVal UserVal As Long) As Long
```

Delphi-Syntax:

```
function AGL_Recv_3964( ConnNr: Integer; var Buff: BYTE; var
BuffLen: Integer; Timeout: Integer; UserVal: LongInt ): Integer;
stdcall;
```

Parameter:

ConnNr	Variable for connection handle
Buff	Indicator on data files which have to be sent
BuffLen	Length of sending buffer
Timeout	Timeout value which has to be used
UserVal	Value for free usage

Return value:

>= 0	Job number asynchronous call respectively AGL40_SUCCESS if synchronous call
< 0	Error number (s. AGL_Defines.H, AGL_Defines.BAS or AGLink40.PAS)

11 Programming reference additional functions

11.1 Function reference reading functions

11.1.1 Reading 16 bit integer from byte buffer (AGL_ReadInt16)

With this function you can read a signed 16 bit integer value from a byte buffer.

C/C++-Syntax:

```
short WINAPI AGL_ReadInt16( PBYTE Buff );
```

Visual Basic-Syntax:

```
Declare Function AGL_ReadInt16 Lib "AGLink40.DLL" (ByRef Buff As
Byte) As Integer
```

Delphi-Syntax:

```
function AGL_ReadInt16( var Buff: Byte ): SmallInt; stdcall;
```

Parameter:

Buff	Buffer with data bytes

Return value:

The 16 bit integer value at address Buff[0], Buff[1].

11.1.2 Reading 32 bit integer from byte buffer (AGL_ReadInt32)

With this function you can read a signed 32 bit integer value from a byte buffer.

C/C++-Syntax:

```
long WINAPI AGL_ReadInt32( PBYTE Buff );
```

Visual Basic-Syntax:

```
Declare Function AGL_ReadInt32 Lib "AGLink40.DLL" (ByRef Buff As
Byte) As Long
```

Delphi-Syntax:

```
function AGL_ReadInt32( var Buff: Byte ): LongInt; stdcall;
```

Parameter:

Buff	Buffer with data bytes

Return value:

The 32 bit integer value at address Buff[0], Buff[1], Buff[2], Buff[3].

11.1.3 Reading word from byte buffer (AGL_ReadWord)

With this function you can read a word from a byte buffer.

C/C++-Syntax:

```
WORD WINAPI AGL_ReadWord( PBYTE Buff );
```

Visual Basic-Syntax:

```
Declare Function AGL_ReadWord Lib "AGLink40.DLL" (ByRef Buff As
Byte) As Long
```

Delphi-Syntax:

```
function AGL_ReadWord( var Buff: Byte ): Word; stdcall;
```

Parameter:

Buff	Buffer with data bytes

Return value:

The word value at address Buff[0], Buff[1].

11.1.4 Reading double word from byte buffer (AGL_ReadDWord)

With this function you can read a double word from a byte buffer.

C/C++-Syntax:

```
DWORD WINAPI AGL_ReadDWord( PBYTE Buff );
```

Visual Basic-Syntax:

```
Declare Function AGL_ReadDWord Lib "AGLink40.DLL" (ByRef Buff As
Byte) As Long
```

Delphi-Syntax:

```
function AGL_ReadDWord( var Buff: Byte ): DWord; stdcall;
```

Parameter:

Buff	Buffer with data bytes

Return value:

The double word value at address Buff[0], Buff[1], Buff[2], Buff[3].

11.1.5 Reading real number from byte buffer (AGL_ReadReal)

With this function you can read a real number from a byte buffer.

C/C++-Syntax:

```
float WINAPI AGL_ReadReal( PBYTE Buff );
```

Visual Basic-Syntax:

```
Declare Function AGL_ReadReal Lib "AGLink40.DLL" (ByRef Buff As
Byte) As Single
```

Delphi-Syntax:

```
function AGL_ReadReal( var Buff: Byte ): Single; stdcall;
```

Parameter:

Buff	Buffer with data bytes

Return value:

The real number value at address Buff[0], Buff[1], Buff[2], Buff[3].

11.1.6 Reading S5 time from byte buffer (AGL_ReadS5Time)

With this function you can read a S5 time in BCD from a byte buffer.

C/C++-Syntax:

```
int WINAPI AGL_ReadS5Time( PBYTE Buff );
```

Visual Basic-Syntax:

```
Declare Function AGL_ReadS5Time Lib "AGLink40.DLL" (ByRef Buff As
Byte) As Long
```

Delphi-Syntax:

```
function AGL_ReadS5Time( var Buff: Byte ): Integer; stdcall;
```

Parameter:

Buff	Buffer with data bytes

Return value:

Value of S5 time (in miliseconds) at address Buff[0], Buff[1].

11.2 Function reference writing functions

11.2.1 Writing 16 bit integer into byte buffer (AGL_WriteInt16)

With this function you can write a signed 16 bit integer into a byte buffer.

C/C++-Syntax:

```
void WINAPI AGL_WriteInt16( PBYTE Buff, short Val );
```

Visual Basic-Syntax:

```
Declare Sub AGL_WriteInt16 Lib "AGLink40.DLL" (ByRef Buff As Byte,
ByVal Val As Integer)
```

Delphi-Syntax:

```
procedure AGL_WriteInt16( var Buff: Byte; Val: SmallInt ); stdcall;
```

Parameter:

Buff	Buffer with data bytes
Val	16 bit integer value which has to be written

Return value:

none

11.2.2 Writing 32 bit integer into byte buffer (AGL_WriteInt32)

With this function you can write a signed 32 bit integer into a byte buffer.

C/C++-Syntax:

```
void WINAPI AGL_WriteInt32( PBYTE Buff, long Val );
```

Visual Basic-Syntax:

```
Declare Sub AGL_WriteInt32 Lib "AGLink40.DLL" (ByRef Buff As Byte,
ByVal Val As Long)
```

Delphi-Syntax:

```
procedure AGL_WriteInt32( var Buff: Byte; Val: LongInt ); stdcall;
```

Parameter:

Buff	Buffer with data bytes
Val	32 bit integer value which has to be written

Return value:

none

11.2.3 Writing word into byte buffer (AGL_WriteWord)

With this function you can write a word into a byte buffer.

C/C++-Syntax:

```
void WINAPI AGL_WriteWord( PBYTE Buff, WORD Val );
```

Visual Basic-Syntax:

```
Declare Sub AGL_WriteWord Lib "AGLink40.DLL" (ByRef Buff As Byte,
ByVal Val As Long)
```

Delphi-Syntax:

```
procedure AGL_WriteWord( var Buff: Byte; Val: Word ); stdcall;
```

Parameter:

Buff	Buffer with data bytes
Val	Word value which has to be written

Return value:

none

11.2.4 Writing double word into byte buffer (AGL_WriteDWord)

With this function you can write a double word into a byte buffer.

Syntax:

```
void WINAPI AGL_WriteDWord( PBYTE Buff, DWORD Val );
```

Visual Basic-Syntax:

```
Declare Sub AGL_WriteDWord Lib "AGLink40.DLL" (ByRef Buff As Byte,
ByVal Val As Long)
```

Delphi-Syntax:

```
procedure AGL_WriteDWord( var Buff: Byte; Val: DWord ); stdcall;
```

Parameter:

Buff	Buffer with data bytes
Val	Double word value which has to be written

Return value:

none

11.2.5 Writing real number into byte buffer (AGL_WriteReal)

With this function you can write a real number into a byte buffer.

C/C++-Syntax:

```
void  WINAPI AGL_WriteReal( PBYTE Buff, float Val );
```

Visual Basic-Syntax:

```
Declare Sub AGL_WriteReal Lib "AGLink40.DLL" (ByRef Buff As Byte,
ByVal Val As Single)
```

Delphi-Syntax:

```
procedure AGL_WriteReal( var Buff: Byte; Val: Single ); stdcall;
```

Parameter:

Buff	Buffer with data bytes
Val	Real value which has to be written

Return value:

none

11.2.6 Writing S5 time into byte buffer (AGL_WriteS5Time)

With this function you can write a time in miliseconds as S5 time into a byte buffer. This time will be displayed using the highest possible accuracy. This time must be from 10 ms to 9990000 ms.

C/C++-Syntax:

```
void WINAPI AGL_WriteS5Time( PBYTE Buff, int Val );
```

Visual Basic-Syntax:

```
Declare Sub AGL_WriteS5Time Lib "AGLink40.DLL" (ByRef Buff As Byte,
ByVal Val As Long)
```

Delphi-Syntax:

```
procedure AGL_WriteS5Time( var Buff: Byte; Val: Integer ); stdcall;
```

Parameter:

Buff	Buffer with data bytes
Val	Time value which has to be written in miliseconds

Return value:

none

11.3 Function reference buffer converting functions

11.3.1 Converting byte buffer into word buffer (AGL_Byte2Word)

With this function you can convert a byte buffer into a word buffer.

C/C++-Syntax:

```
void WINAPI AGL_Byte2Word( PWORD OutBuff, PBYTE InBuff, int AnzWords
);
```

Visual Basic-Syntax:

```
Declare Sub AGL_Byte2Word Lib "AGLink40.DLL" (ByRef OutBuff As
Integer, ByRef InBuff As Byte, ByVal AnzWords As Long)
```

Delphi-Syntax:

```
procedure AGL_Byte2Word( var OutBuff: Word; var InBuff: Byte;
AnzWords: Integer ); stdcall;
```

Parameter:

OutBuff	Indicator on word buffer
InBuff	Indicator on byte buffer
AnzWords	Number of words which has to be converted

Return value:

none

11.3.2 Converting byte buffer into double word buffer (AGL_Byte2DWord)

With this function you can convert a byte buffer into a double word buffer.

C/C++-Syntax:

```
void  WINAPI AGL_Byte2DWord( PDWORD OutBuff, PBYTE InBuff, int
AnzDWords );
```

Visual Basic-Syntax:

```
Declare Sub AGL_Byte2DWord Lib "AGLink40.DLL" (ByRef OutBuff As
Long, ByRef InBuff As Byte, ByVal AnzDWords As Long)
```

Delphi-Syntax:

```
procedure AGL_Byte2DWord( var OutBuff: DWord; var InBuff: Byte;
AnzDWords: Integer ); stdcall;
```

Parameter:

OutBuff	Indicator on double word buffer
InBuff	Indicator on byte buffer
AnzDWords	Number of double words which has to be converted

Return value:

none

11.3.3 Converting byte buffer into real number buffer (AGL_Byte2Real)

With this function you can convert a byte buffer into a real number buffer.

C/C++-Syntax:

```
void  WINAPI AGL_Byte2Real( float *OutBuff, PBYTE InBuff, int
AnzReals );
```

Visual Basic-Syntax:

```
Declare Sub AGL_Byte2Real Lib "AGLink40.DLL" (ByRef OutBuff As
Single, ByRef InBuff As Byte, ByVal AnzReals As Long)
```

Delphi-Syntax:

```
procedure AGL_Byte2Real( var OutBuff: Single; var InBuff: Byte;
AnzReals: Integer ); stdcall;
```

Parameter:

OutBuff	Indicator on real number buffer
InBuff	Indicator on byte buffer
AnzReals	Number of real numbers which has to be converted

Return value:

none

11.3.4 Converting word buffer into byte buffer (AGL_Word2Byte)

With this function you can convert a word buffer into a byte buffer.

C/C++-Syntax:

```
void WINAPI AGL_Word2Byte( PBYTE OutBuff, PWORD InBuff, int AnzWords
);
```

Visual Basic-Syntax:

```
Declare Sub AGL_Word2Byte Lib "AGLink40.DLL" (ByRef OutBuff As Byte,
ByRef InBuff As Integer, ByVal AnzWords As Long)
```

Delphi-Syntax:

```
procedure AGL_Word2Byte( var OutBuff: Byte; var InBuff: Word;
AnzWords: Integer ); stdcall;
```

Parameter:

OutBuff	Indicator on byte buffer
InBuff	Indicator on word buffer
AnzReals	Number of words which has to be converted

Return value:

none

11.3.5 Converting double word buffer into byte buffer (AGL_DWord2Byte)

With this function you can convert a double word buffer into a byte buffer.

C/C++-Syntax:

```
void  WINAPI AGL_DWord2Byte( PBYTE OutBuff, PDWORD InBuff, int
AnzDWords );
```

Visual Basic-Syntax:

```
Declare Sub AGL_DWord2Byte Lib "AGLink40.DLL" (ByRef OutBuff As
Byte, ByRef InBuff As Long, ByVal AnzDWords As Long)
```

Delphi-Syntax:

```
procedure AGL_DWord2Byte( var OutBuff: Byte; var InBuff: DWord;
AnzDWords: Integer ); stdcall;
```

Parameter:

OutBuff	Indicator on byte buffer
InBuff	Indicator on double word buffer
AnzReals	Number of double words which has to be converted

Return value:

none

11.3.6 Converting real number buffer into byte buffer (AGL_Real2Byte)

With this function you can convert a real number buffer into a byte buffer.

C/C++-Syntax:

```
void WINAPI AGL_Real2Byte( PBYTE OutBuff, float *InBuff, int
AnzReals );
```

Visual Basic-Syntax:

```
Declare Sub AGL_Real2Byte Lib "AGLink40.DLL" (ByRef OutBuff As Byte,
ByRef InBuff As Single, ByVal AnzReals As Long)
```

Delphi-Syntax:

```
procedure AGL_Real2Byte( var OutBuff: Byte; var InBuff: Single;
AnzReals: Integer ); stdcall;
```

Parameter:

OutBuff	Indicator on byte buffer
InBuff	Indicator on real number buffer
AnzReals	Number of real numbers which has to be converted

Return value:

none

11.3.7 Converting byte buffer into string (AGL_Buff2String)

With this function you can convert a byte buffer into a string. This function was inplemented especially to ease the access when using VB respectively VBA.

Note:
Alternatively, VB programers should look for the function AGL_Buff2VBString in chapter »12 Additional functions for VB respectively VBA«.

C/C++-Syntax:

```
int WINAPI AGL_Buff2String( PBYTE Buff, LPSTR Text, int AnzBytes );
```

Visual Basic-Syntax:

```
Declare Function AGL_Buff2String Lib "AGLink40.DLL" (ByRef Buff As
Byte, ByVal Text As String, ByVal AnzBytes As Long) As Long
```

Delphi-Syntax:

```
function AGL_Buff2String( var Buff: Byte; Text: PChar; AnzBytes:
Integer ): Integer; stdcall;
```

Parameter:

Buff	Indicator on byte buffer
Text	Indicator on string
AnzBytes	Number of bytes which has to be copied

Return value

AGL40_SUCCESS	If all worked properly
< 0	Error number (s. AGL_Defines.H, AGL_Defines.BAS or AGLink40.PAS)

11.3.8 Converting string into byte buffer (AGL_String2Buff)

With this function you can convert a string into a byte buffer. This function was inplemented especially to ease the access when using VB respectively VBA.

Note:
Alternatively, VB programers should look for the function AGL_VBString2Buff in chapter »12 Additional functions for VB respectively VBA«.

C/C++-Syntax:

```
int WINAPI AGL_String2Buff( PBYTE Buff, LPSTR Text, int AnzBytes );
```

Visual Basic-Syntax:

```
Declare Function AGL_String2Buff Lib "AGLink40.DLL" (ByRef Buff As
Byte, ByVal Text As String, ByVal AnzBytes As Long) As Long
```

Delphi-Syntax:

```
function AGL_String2Buff( var Buff: Byte; Text: PChar; AnzBytes:
Integer ): Integer; stdcall;
```

Parameter:

Buff	Indicator on byte buffer
Text	Indicator on string
AnzBytes	Number of bytes which has to be copied

Return value:

AGL40_SUCCESS	If all worked properly
< 0	Error number (s. AGL_Defines.H, AGL_Defines.BAS or AGLink40.PAS)

11.3.9 Converting byte buffer into WideChar string (AGL_Buff2WString)

With this function you can convert a byte buffer into a WideChar string.

C/C++-Syntax:

```
int WINAPI AGL_Buff2WString( PBYTE Buff, LPWSTR Text, int AnzBytes
);
```

Visual Basic-Syntax:

```
Nicht implementiert
```

Delphi-Syntax:

```
function AGL_Buff2WString( var Buff: Byte; var Text: WideChar;
AnzBytes: Integer ): Integer; stdcall;
```

Parameter:

Buff	Indicator on byte buffer
Text	Indicator on WideChar string
AnzBytes	Number of bytes which has to be copied

Return value:

AGL40_SUCCESS	If all worked properly
< 0	Error number (s. AGL_Defines.H, AGL_Defines.BAS or AGLink40.PAS)

11.3.10 Converting WideChar string into byte buffer (AGL_WString2Buff)

With this function you can convert a WideCHar string into a byte buffer.

C/C++-Syntax:

```
int WINAPI AGL_WString2Buff( PBYTE Buff, LPWSTR Text, int AnzBytes
);
```

Visual Basic-Syntax:

```
Nicht implementiert
```

Delphi-Syntax:

```
function AGL_WString2Buff( var Buff: Byte; var Text: WideChar;
AnzBytes: Integer ): Integer; stdcall;
```

Parameter:

Buff	Indicator on byte buffer
Text	Indicator on WideChar string
AnzBytes	Number of bytes which has to be copied

Return value:

AGL40_SUCCESS	If all worked properly
< 0	Error number (s. AGL_Defines.H, AGL_Defines.BAS or AGLink40.PAS)

11.4 Function reference converting S7 types

11.4.1 Converting S7 string into string (AGL_S7String2String)

With this function you can convert a byte buffer, which content is a S7 string, into a string. The current length is considered here.

Note:
Alternatively, VB programers should look for the function AGL_S7String2VBString in chapter 12 Additional functions for VB respectively VBA.

C/C++-Syntax:

```
int WINAPI AGL_S7String2String( PBYTE S7String, LPSTR Text, int
MaxChars );
```

Visual Basic-Syntax:

```
Declare Function AGL_S7String2String Lib "AGLink40.DLL" (ByRef
S7String As Byte, ByVal Text As String, ByVal MaxChars As Long) As
Long
```

Delphi-Syntax:

```
function AGL_S7String2String( var S7String: Byte; Text: PChar;
MaxChars: Integer ): Integer; stdcall;
```

Parameter:

S7String	Indicator on byte buffer in S7 string format
Text	Indicator on string
MaxChars	Number of maximum characters which has to be copied

Return value:

AGL40_SUCCESS	If all worked properly
< 0	Error number (s. AGL_Defines.H, AGL_Defines.BAS or AGLink40.PAS)

11.4.2 Converting string int S7 string (AGL_String2S7String)

With this function you can convert a string into a S7 string. The maximum length will be consired and the actual length will be registered here.

Note:
Alternatively, VB programers should look for the function AGL_VBString2S7String in chapter 12 Additional functions for VB respectively VBA.

C/C++-Syntax:

```
int WINAPI AGL_String2S7String( PBYTE S7String, LPSTR Text, int
MaxChars );
```

Visual Basic-Syntax:

```
Declare Function AGL_String2S7String Lib "AGLink40.DLL" (ByRef
S7String As Byte, ByVal Text As String, ByVal MaxChars As Long) As
Long
```

Delphi-Syntax:

```
function AGL_String2S7String( var S7String: Byte; Text: PChar;
MaxChars: Integer ): Integer; stdcall;
```

Parameter:

S7String	Indicator on byte buffer in S7 string format
Text	Indicator on string
MaxBytes	Number of maximum bytes which has to be copied

Return value:

AGL40_SUCCESS	If all worked properly
< 0	Error number (s. AGL_Defines.H, AGL_Defines.BAS or AGLink40.PAS)

11.4.3 Converting DATE_AND_TIME in SysTime (AGL_S7DT2SysTime)

With this function you can convert a byte buffer, which content is a S7-DATE_AND_TIME structure, into a SystemTime structure.

C/C++-Syntax:

```
int WINAPI AGL_S7DT2SysTime( PBYTE Buff, LPSYSTEMTIME SysTime );
```

Visual Basic-Syntax:

```
Declare Function AGL_S7DT2SysTime Lib "AGLink40.DLL" (ByRef Buff As
Byte, ByRef SysTime as SYSTEMTIME) As Long
```

Delphi-Syntax:

```
function AGL_S7DT2SysTime( var Buff: Byte; var SysTime: SYSTEMTIME
): Integer; stdcall;
```

Parameter:

Buff	Indicator on byte buffer in S7 DATE_AND_TIME format
SysTime	Indicator on SysTime structure

Return value:

AGL40_SUCCESS	If all worked properly
< 0	Error number (s. AGL_Defines.H, AGL_Defines.BAS or AGLink40.PAS)

11.4.4 Converting SysTime in DATE_AND_TIME (AGL_SysTime2S7DT)

With this function you can convert a SysTime into a byte buffer, which content is a S7 DATE_AND_TIME structure.

C/C++-Syntax:

```
int WINAPI AGL_SysTime2S7DT( LPSYSTEMTIME SysTime, PBYTE Buff );
```

Visual Basic-Syntax:

```
Declare Function AGL_SysTime3S7DT Lib "AGLink40.DLL" (ByRef SysTime
as SYSTEMTIME, ByRef Buff As Byte) As Long
```

Delphi-Syntax:

```
function AGL_SysTime2S7DT( var SysTime: SYSTEMTIME; var Buff: Byte
): Integer; stdcall;
```

Parameter:

SysTime	Indicator on SysTime structure
Buff	Indicator on byte buffer in S7 DATE_AND_TIME format

Return value:

AGL40_SUCCESS	If all worked properly
< 0	Error number (s. AGL_Defines.H, AGL_Defines.BAS or AGLink40.PAS)

11.4.5 Converting S7 CPU time into SysTime (AGL_TOD2SysTime)

With this structure you can convert a TOD structure into a SystemTime structure. Thus a time read by AGL_GetPLCClock can process forward, directly.

C/C++-Syntax:

```
int WINAPI AGL_TOD2SysTime( LPTOD pTOD, LPSYSTEMTIME SysTime );
```

Visual Basic-Syntax:

```
Declare Function AGL_TOD2SysTime Lib "AGLink40.DLL" (ByRef pTOD As
TOD, ByRef SysTime as SYSTEMTIME) As Long
```

Delphi-Syntax:

```
function AGL_TOD2SysTime( var pTOD: TagTOD; var SysTime: SYSTEMTIME
): Integer; stdcall;
```

Parameter:

pTOD	Indicator on S7 CPU time
SysTime	Indicator on SysTime structure

Return value:

AGL40_SUCCESS	If all worked properly
< 0	Error number (s. AGL_Defines.H, AGL_Defines.BAS or AGLink40.PAS)

Ther TOD structure has the following build up:

```
typedef struct tagTOD
{
  WORD        ClockStatus;              // please look for PLC
                                        // description
  BYTE        Year;                     // Year in BCD (2-digit)
  BYTE        Month;                    // Month in BCD (2-digit)
  BYTE        Day;                      // Day in BCD (2-digit)
  BYTE        Hour;                     // Hour in BCD (2-digit)
  BYTE        Minute;                   // Minute in BCD (2-digit)
  BYTE        Second;                   // Second in BCD (2-digit)
  BYTE        Zero;                     // Should always be 0
  BYTE        Weekday;                  // Weekday, Sunday = 1
```

```
} TOD, *LPTOD;
```

Note:
On some PLCs this structure corresponds to the TIME_OF_DAY structure with a prefixed status word. That means the miliseconds are contained and will be transmitted, too. Thus ZERO contains hundreds and tenner, the High-Nibble of weekday the ones of the miliseconds. This is already considered when using the converting function.

11.4.6 Converting SysTime into S7 CPU time (AGL_SysTime2TOD)

With this function you can convert a SystemTime structure into a TOD structure. Thus you can directly prepare a SysTime for a AGL_SetPLCClock call.

C/C++-Syntax:

```
int WINAPI AGL_SysTime2TOD( LPSYSTEMTIME SysTime, LPTOD pTOD );
```

Visual Basic-Syntax:

```
Declare Function AGL_SysTime2TOD Lib "AGLink40.DLL" (ByRef SysTime
as SYSTEMTIME, ByRef pTOD As TOD) As Long
```

Delphi-Syntax:

```
function AGL_SysTime2TOD( var SysTime: SYSTEMTIME; var pTOD: TagTOD
): Integer; stdcall;
```

Parameter:

SysTime	Indicator on SysTime structure
pTOD	Indicator on S7 CPU timt

Return value:

AGL40_SUCCESS	If all worked properly
< 0	Error number (s. AGL_Defines.H, AGL_Defines.BAS or AGLink40.PAS)

The TOD has the following build up:

```
typedef struct tagTOD
{
  WORD        ClockStatus;              // please look for PLC
                                        // description
  BYTE        Year;                     // Year in BCD (2-digit)
  BYTE        Month;                    // Month in BCD (2-digit)
  BYTE        Day;                      // Day in BCD (2-digit)
  BYTE        Hour;                     // Hour in BCD (2-digit)
  BYTE        Minute;                   // Minute in BCD (2-digit)
  BYTE        Second;                   // Second in BCD (2-digit)
  BYTE        Zero;                     // Should always be 0
  BYTE        Weekday;                  // Weekday, Sunday = 1
```

```
} TOD, *LPTOD;
```

Note:
On some PLCs this structure corresponds to the TIME_OF_DAY structure with a prefixed status word. That means the miliseconds are contained and will be transmitted, too. Thus ZERO contains hundreds and tenner, the High-Nibble of weekday the ones of the miliseconds. This is already considered when using the converting function.

11.5 Function reference converting S5 types

11.5.1 Converting float number into KG format (AGL_Float2KG)

With this function you convert a float buffer into a buffer in the KG format. The output buffer has to be in the word format due to combatibility reasons to the function AGL_WriteDataWords.

C/C++-Syntax:

```
int WINAPI AGL_Float2KG( PWORD pKG, PFLOAT pFloat, int AnzFloats );
```

Visual Basic-Syntax:

```
Declare Function AGL_Float2KG Lib "AGLink40.DLL" (ByRef pKG As
Integer, ByRef pFloat As Single, ByVal AnzFloats As Long) As Long
```

Delphi-Syntax:

```
function AGL_Float2KG(var pKG: Word; var pFloat: Single; AnzFloats:
Integer ): Integer; stdcall;
```

Parameter:

pKG	Indicator on word buffer for numbers in the KG format
pFloat	Indicator on float bufer
AnzFloats	Amount of floats which have to be changed

Return value:

AGL40_SUCCESS	If all worked properly
< 0	Error number (s. AGL_Defines.H, AGL_Defines.BAS or AGLink40.PAS)

11.5.2 Converting KG number into Float number (AGL_KG2Float)

With this function you convert a buffer in the KG format into a Float buffer. The output buffer has to be in the word format due to combatibility reasons to the function AGL_ReadDataWords.

C/C++-Syntax:

```
int WINAPI AGL_KG2Float(PFLOAT pFloat, PWORD pKG, int AnzFloats );
```

Visual Basic-Syntax:

```
Declare Function AGL_KG2Float Lib "AGLink40.DLL" (ByRef pFloat As
Single, ByRef pKG As Integer, ByVal AnzFloats As Long) As Long
```

Delphi-Syntax:

```
function AGL_Float2KG(var pFloat: Single; var pKG: Word; AnzFloats:
Integer ): Integer; stdcall;
```

Parameter:

pFloat	Indicator on float buffer
pKG	Indicator on word buffer for numbers in the KG format
AnzFloats	Amount of floats which have to be changed

Return value:

AGL40_SUCCESS	If all worked properly
< 0	Error number (s. AGL_Defines.H, AGL_Defines.BAS or AGLink40.PAS)

11.6 Function reference special conversions

11.6.1 Converting BCD number into 16 bit integer (AGL_BCD2Int16)

With this function you can convert a BCD number into a signed 16 bit integer number.

C/C++-Syntax:

```
int WINAPI AGL_BCD2Int16( short BCD, PSHORT Dual );
```

Visual Basic-Syntax:

```
Declare Function AGL_BCD2Int16 Lib "AGLink40.DLL" (ByVal BCD As
Integer, ByRef Dual As Integer) As Long
```

Delphi-Syntax:

```
function AGL_BCD2Int16( BCD: SmallInt; var Dual: SmallInt ):
Integer; stdcall;
```

Parameter:

BCD	BCD number which has to be converted
Dual	Indicator on 16 bit integer number

Return value:

AGL40_SUCCESS	If all worked properly
< 0	Error number (s. AGL_Defines.H, AGL_Defines.BAS or AGLink40.PAS)

11.6.2 Converting 16 bit integer into BCD number (AGL_Int162BCD)

With this function you can convert a signed 16 bit integer number into a BCD number.

C/C++-Syntax:

```
int WINAPI AGL_Int162BCD( short Dual, PSHORT BCD );
```

Visual Basic-Syntax:

```
Declare Function AGL_Int162BCD Lib "AGLink40.DLL" (ByVal Dual As
Integer, ByRef BCD As Integer) As Long
```

Delphi-Syntax:

```
function AGL_Int162BCD( Dual: SmallInt; var BCD: SmallInt ):
Integer; stdcall;
```

Parameter:

Dual	16 bit integer number which has to be converted
BCD	Indicator onf BCD number

Return value:

AGL40_SUCCESS	If all worked properly
< 0	Error number (s. AGL_Defines.H, AGL_Defines.BAS or AGLink40.PAS)

11.6.3 Converting BCD number into 32 bit integer (AGL_BCD2Int32)

With this function you can convert a BCD number into a signed 32 bit integer numnber.

C/C++-Syntax:

```
int WINAPI AGL_BCD2Int32( long BCD, PLONG Dual );
```

Visual Basic-Syntax:

```
Declare Function AGL_BCD2Int32 Lib "AGLink40.DLL" (ByVal BCD As
Long, ByRef Dual As Long) As Long
```

Delphi-Syntax:

```
function AGL_BCD2Int32( BCD: LongInt; var Dual: LongInt ): Integer;
stdcall;
```

Parameter:

BCD	BCD number which has to be converted
Dual	Indicator on 32 bit integer number

Return value:

AGL40_SUCCESS	If all worked properly
< 0	Error number (s. AGL_Defines.H, AGL_Defines.BAS or AGLink40.PAS)

11.6.4 Converting 32 bit integer into BCD number (AGL_Int322BCD)

With this function you can convert a signed 32 bit integer number into a BCD number.

C/C++-Syntax:

```
int WINAPI AGL_Int322BCD( long Dual, PLONG BCD );
```

Visual Basic-Syntax:

```
Declare Function AGL_Int322BCD Lib "AGLink40.DLL" (ByVal Dual As
Long, ByRef BCD As Long) As Long
```

Delphi-Syntax:

```
function AGL_Int322BCD( Dual: LongInt; var BCD: LongInt ): Integer;
stdcall;
```

Parameter:

Dual	32 bit integer number which has to be converted
BCD	Indicator on BCD number

Return value:

AGL40_SUCCESS	If all worked properly
< 0	Error number (s. AGL_Defines.H, AGL_Defines.BAS or AGLink40.PAS)

11.6.5 Return 32 bit integer as float (AGL_LongAsFloat)

This function only performs a Typecast and returns the long number as float. This is especially useful when using type secure programming languages. Thus e.g. a result in a DATA_RW40 structure can be directly processed as a real number.

C/C++-Syntax:

```
float WINAPI AGL_LongAsFloat( long Var );
```

Visual Basic-Syntax:

```
Declare Function AGL_LongAsFloat Lib "AGLink40.DLL" (ByVal Var As
Long) As Single
```

Delphi-Syntax:

```
function AGL_LongAsFloat( Var: LongInt ): Single; stdcall;
```

Parameter:

Var	Number which has to be casted

Return value:

Number as real value

11.6.6 Return float as 32 bit integer (AGL_FloatAsLong)

This function only performs a Typecast and returns the float number as Long. This is especially useful when using type secure programming languages. Thus e.g. a real number can be written directly into a DATA_RW40 structure.

C/C++-Syntax:

```
long WINAPI AGL_FloatAsLong( float Var );
```

Visual Basic-Syntax:

```
Declare Function AGL_FloatAsLong Lib "AGLink40.DLL" (ByVal Var As
Single) As Long
```

Delphi-Syntax:

```
function AGL_FloatAsLong( Var: Single ): LongInt; stdcall;
```

Parameter:

Var	Number which has to be casted

Return value:

Number as long value

11.7 Function reference bit functions

11.7.1 Calling bit status (AGL_GetBit)

This function determines the status of the desired bit.

C/C++-Syntax:

```
int WINAPI AGL_GetBit( BYTE Wert, int BitNr );
```

Visual Basic-Syntax:

```
Declare Function AGL_GetBit Lib "AGLink40.DLL" (ByVal Wert As Byte,
ByVal BitNr As Long) As Long
```

Delphi-Syntax:

```
function AGL_GetBit ( Wert: Byte; BitNr: Integer ): Integer;
stdcall;
```

Parameter:

Wert	Byte which has to be checked
BitNr	Bit number in section 0 to 7

Return value:

Bit status

11.7.2 Setting bit to one (AGL_SetBit)

This function sets the desired bit to one.

C/C++-Syntax:

```
BYTE AGL_SetBit( PBYTE Buff, int BitNr );
```

Visual Basic-Syntax:

```
Declare Function AGL_SetBit Lib "AGLink40.DLL" (ByRef Buff As Byte,
ByVal BitNr As Long) As Byte
```

Delphi-Syntax:

```
Function AGL_SetBit ( var Buff: Byte; BitNr: Integer ): Byte;
stdcall;
```

Parameter:

Buff	Indicator on byte in which the bit has to be set
BitNr	Bit number in section 0 to 7

Return value:

Byte value

11.7.3 Setting bit to zero (AGL_ResetBit)

This function sets the desired bit to zero.

C/C++-Syntax:

```
BYTE AGL_ResetBit( PBYTE Buff, int BitNr );
```

Visual Basic-Syntax:

```
Declare Function AGL_ResetBit Lib "AGLink40.DLL" (ByRef Buff As
Byte, ByVal BitNr As Long) As Byte
```

Delphi-Syntax:

```
Function AGL_ResetBit ( var Buff: Byte; BitNr: Integer ): Byte;
stdcall;
```

Parameter:

Buff	Indicator on byte in which the bit has to be reset
BitNr	Bit number in section 0 to 7

Return value:

Byte value

11.7.4 Setting bit to value (AGL_SetBitVal)

This function sets the bit to the desired value.

C/C++-Syntax:

```
BYTE AGL_SetBitVal( PBYTE Buff, int BitNr, int Val );
```

Visual Basic-Syntax:

```
Declare Function AGL_SetBitVal Lib "AGLink40.DLL" (ByRef Buff As
Byte, ByVal BitNr As Long, ByVal Val As Long ) As Byte
```

Delphi-Syntax:

```
Function AGL_SetBitVal ( var Buff: Byte; BitNr: Integer; Val:
Integer ): Byte; stdcall;
```

Parameter:

Buff	Indicator on byte in which the bit has to be changed
BitNr	Bit number in section 0 to 7
Val	Bit value

Return value:

Byte value

11.8 Function reference format functions

11.8.1 Changing text into DATA_RW40 structure (AGL_Text2DataRW)

This function analyses the text and so it directly fills the structure elements of a DATA_RW40 structure.

C/C++-Syntax:

```
int AGL_Text2DataRW( LPSTR Text, LPDATA_RW40 RW );
```

Visual Basic-Syntax:

```
Declare Function AGL_Text2DataRW Lib "AGLink40.DLL" (ByVal Text As
String, ByRef RW As DATA_RW40 ) As Long
```

Delphi-Syntax:

```
Function AGL_Text2DataRW( Text: PChar; var RW: TagDATA_RW40 ):
Integer; stdcall;
```

Parameter:

Text	Indicator on text which has to be analysed
RW	Indicator on DATA_RW40 structure which has to be filled

Return value:

AGL40_SUCCESS	If all worked properly
< 0	Error number (s. AGL_Defines.H, AGL_Defines.BAS or AGLink40.PAS)

11.8.2 Changing DATA_RW40 structure into text (AGL_DataRW2Text)

This function analyses the structure elements of a DATA_RW40 structure and generates a text with readable operands out of it. The text buffer's minimum length is 32 characters.

C/C++-Syntax:

```
int AGL_DataRW2Text(LPDATA_RW40 RW, LPSTR Text );
```

Visual Basic-Syntax:

```
Declare Function AGL_DataRW2Text Lib "AGLink40.DLL" (ByRef RW As
DATA_RW40 ByVal Text As String) As Long
```

Delphi-Syntax:

```
Function AGL_DataRW2Text( var RW: TagDATA_RW40; Text: PChar ):
Integer; stdcall;
```

Parameter:

RW	Indicator on DATA_RW40 structure which has to be analysed
Text	Indicator on text

Return value:

AGL40_SUCCESS	If all worked properly
< 0	Error number (s. AGL_Defines.H, AGL_Defines.BAS or AGLink40.PAS)

11.9 Function reference DLL info functions

11.9.1 Determining the DLL's version numbers (AGL_GetDLLVersion)

This function outputs the AGLink40.DLL's version number. It is possible to output ZERO instead the single parameters, when these special information is not of interest.

C/C++-Syntax:

```
void WINAPI AGL_GetDLLVersion( int *Major, int *Minor );
```

Visual Basic-Syntax:

```
Declare Sub AGL_GetDLLVersion Lib "AGLink40.DLL" (Major As Long,
Minor As Long)
```

Delphi-Syntax:

```
procedure AGL_GetDLLVersion( var Major: Integer; var Minor: Integer
): Integer; stdcall;
```

Parameter:

Major	Variable for main version
Minor	Variable for minor version

Return value:

none

11.9.2 Determining the DLL's extended version number (AGL_GetDLLVersionEx)

This function outputs the AGLink40.DLL's extended version number. It is possible to ouput ZERO instead the single parameters, when these special information is not of interest.

C/C++-Syntax:

```
void WINAPI AGL_GetDLLVersionEx( int *Major, int *Minor, int *Build,
int *Revision, LPSTR Date );
```

Visual Basic-Syntax:

```
Declare Sub AGL_GetDLLVersionEx Lib "AGLink40.DLL" (Major As Long,
Minor As Long, Build As Long, Revision As Long, ByVal Datum As
String)
```

Delphi-Syntax:

```
procedure AGL_GetDLLVersionEx( var Major: Integer; var Minor:
Integer; var Build: Integer; var Revision: Integer; Date: PChar ):
Integer; stdcall;
```

Parameter:

Major	Variable for main version
Minor	Variable for minor version
Build	Variable for build version
Revision	Variable for revision version
Date	Variable for compiling date. There has to be a minimum space of 11 bytes. That means a date of this type »2002-07-08« including finsihing zero byte.

Return value:

none

11.9.3 Determining available DLL options (AGL_GetOptions)

This function outputs the AGLink40.DLL's available options.

C/C++-Syntax:

```
int WINAPI AGL_GetOptions( void );
```

Visual Basic-Syntax:

```
Declare Function AGL_GetOptions Lib "AGLink40.DLL" () As Long
```

Delphi-Syntax:

```
function AGL_GetOptions: Integer; stdcall;
```

Parameter:

none

Return value:

The available options as bitflag. A bit set means that the option is available. For possible bit masks please look for the respective header or module data file.

11.9.4 Determining the DLL's serial number (AGL_GetSerialNumber)

This function outputs the AGLink40.DLL's serial number.

C/C++-Syntax:

```
int WINAPI AGL_GetSerialNumber( void );
```

Visual Basic-Syntax:

```
Declare Function AGL_GetSerialNumber Lib "AGLink40.DLL" () As Long
```

Delphi-Syntax:

```
function AGL_GetSerialNumber: Integer; stdcall;
```

Parameter:

none

Return value:

The serial number

11.9.5 Determining the DLL's licensee (AGL_GetClientName)

This function outputs the AGLink40.DLL's licensee. The function awaits an indicator on a string buffer with a minimum space of 256 bytes.

C/C++-Syntax:

```
LPSTR WINAPI AGLGetClientName( LPSTR Name );
```

Visual Basic-Syntax:

```
Declare Sub AGLGetClientName Lib "AGLink40.DLL" (ByVal Name As
String)
```

Delphi-Syntax:

```
function AGLGetClientName( Name: PChar ): PChar; stdcall;
```

Parameter:

Name	Indicator on string

Return value:

An indicator on the passed string for a directly subsequent processing.

11.10 Function reference miscellaneous info functions

11.10.1 Inquiring valid access points of the application (AGL_GetPCCPConnNames)

This function determines the application's valid access points for the ACCON-AGLink S7-PC/CP interface. The list will be built up in the handed over text buffer. It has a field with the access point of the application, tab as seperator, the adjusted interface and ‚\0' as end for this entry, next entry…

C/C++-Syntax:

```
int WINAPI AGL_GetPCCPConnNames( LPSTR Names, int Len );
```

Visual Basic-Syntax:

```
Declare Function AGL_GetPCCPConnNames Lib "AGLink40.DLL" (ByVal
Names As String, ByVal AnzChars As Long) As Long
```

Delphi-Syntax:

```
function AGL_GetPCCPConnNames(Names: PChar; Len: Integer): Integer;
stdcall;
```

Parameter:

Names	Text buffer
Len	Length of text buffer

Return value:

>=0	Amount of taken signs in the text buffer
< 0	Error number (s. AGL_Defines.H, AGL_Defines.BAS or AGLink40.PAS)

11.10.2 Requesting the application's protocol of the access point (AGL_GetPCCPProtocol)

This function determines the application's protocol of the acess point for the interface type ACCON-AGLink S7-PC/CP.

C/C++-Syntax:

```
int WINAPI AGL_GetPCCPProtocol( LPSTR Name );
```

Visual Basic-Syntax:

```
Declare Function AGL_GetPCCPProtocol Lib "AGLink40.DLL" (ByVal Name
As String) As Long
```

Delphi-Syntax:

```
function AGL_GetPCCPProtocol(Name: PChar): Integer; stdcall;
```

Parameter:

Name	Desired access point of the application (e.g. »S7ONLINE«)

Return value:

PCCP_PROTO_UNSUPPORTED	If any error occurs
PCCP_PROTO_UNKNOWN	If not ascertainable
PCCP_PROTO_S7	For MPI/PB communikation
PCCP_PROTO_PPI	For PPI communikation
PCCP_PROTO_ISO	For ISO communikation
PCCP_PROTO_TCPIP	For TCP/IP communikation

11.10.3 Inquiring PLC type (AGL_GetPLCType)

This function determines the type of the connected PLC. The return value is a bit mask and can be evaluated corresponding to family and sub-group. The families are PLC_TYPE_S5 (including sub-group PLC_TYPE_S5_PG if communicating with S5-AS511 instead of S5-TCP/IP), PLC_TYPE_RK and PLC_TYPE_S7 (including sub-group PLC_TYPE_S7_200 or PLC_TYPE_S7_300_400).

C/C++-Syntax:

```
int WINAPI AGL_GetPLCType( int DevNr, int PlcNr );
```

Visual Basic-Syntax:

```
Declare Function AGL_GetPLCType Lib "AGLink40.DLL" (ByVal DevNr As
Long, ByVal PlcNr As Long) As Long
```

Delphi-Syntax:

```
function  AGL_GetPLCType(DevNr: Integer; PlcNr: Integer ): Integer;
stdcall;
```

Parameter:

DevNr	Device number
PlcNr	PLC number

Return value:

Bitmaske	From the constants PLC_TYPE_xxx (s. AGL_Defines.H, AGL_Defines.BAS or AGLink40.PAS)

11.10.4 Checkingt if PLC supports corresponding functions (AGL_HasFunc)

This function checks if the PLC even supports the desired function. This is to prevent the error message AGL40_FUNCTION_NOT_SUPPORTED. The corresponding function or function family will be requested via the constants HAS_FUNC_xxx (s. AGL_Defines.H, AGL_Defines.BAS or AGLink40.PAS).

C/C++-Syntax:

```
int WINAPI AGL_HasFunc( int DevNr, int PlcNr, int Func );
```

Visual Basic-Syntax:

```
Declare Function AGL_HasFunc Lib "AGLink40.DLL" (ByVal DevNr As
Long, ByVal PlcNr As Long, ByVal Func As Long) As Long
```

Delphi-Syntax:

```
function  AGL_HasFunc(DevNr: Integer; PlcNr: Integer; Func: Integer
): Integer; stdcall;
```

Parameter:

DevNr	Device number
PlcNr	PLC number
Func	Function or function family wich has to be requested

Return value:

0	Function not supported
!= 0	Function supported

12 Additional functions for VB respectively VBA

These functions will simplify the access to AGLink functions which are processing a string. They are in the AGL_VBFuncs.BAS' source code module.

12.1 Reading out error messages (AGL_GetVBErrorMsg)

This function directly outputs the AGLink's error messages as string. And is integrated as follows:

```
Public Function AGL_GetVBErrorMsg(ByVal ErrNr As Long) As String
  Dim Text As String

  Text = Space$(128)

  If AGL_GetErrorMsg(ErrNr, Text, 128) = 0 Then
    AGL_GetVBErrorMsg = "Unbekannter Fehler"
  Else
    AGL_GetVBErrorMsg = Left$(Text, lstrlen(Text))
  End If

End Function
```

12.2 Returning text of a DATA_RW40 structure (AGL_DataRW2VBString)

This function directly outputs the variable of a DATA_RW40 structure as string. If error messages appear an empty string will be returned. And is integrated as follows:

```
Public Function AGL_DataRW2VBString(RW As DATA_RW40) As String
  Dim Text As String

  Text = Space$(128)

  If AGL_DataRW2Text(RW, Text) = AGL40_SUCCESS Then
    AGL_DataRW2VBString = Left$(Text, lstrlen(Text))
  End If

End Function
```

12.3 Reading out the diagnostics buffer entry (AGL_GetVBDiagBufferEntry)

This function directly returns the diagnostics buffer entry of AGLink as string. And is integrated as follows:

```
Public Function AGL_GetVBDiagBufferEntry(ByVal Index As Long,
DiagBuff As Byte) As String
  Dim Text As String

  Text = Space$(128)
  If AGL_GetDiagBufferEntry(Index, DiagBuff, Text, 128) > 0 Then
    AGL_GetVBDiagBufferEntry = Left$(Text, lstrlen(Text))
  End If
End Function
```

12.4 Converting buffer into VB string (AGL_Buff2VBString)

This function directly converts a byte buffer content into a VB string. It is implemented as follows:

```
Public Function AGL_Buff2VBString(ByRef Buff As Byte, Text As
String, ByVal AnzChars As Long) As Long
  Dim Laenge As Long

  Laenge = Len(Text)
  If Laenge < AnzChars Then
    Text = Text & Space$(AnzChars - Laenge)
  ElseIf Laenge > AnzChars Then
    Text = Left$(Text, AnzChars)
  End If
  AGL_Buff2VBString = AGL_Buff2String(Buff, Text, AnzChars)
End Function
```

12.5 Converting VB string into buffer (AGL_VBString2Buff)

This function directly converts a VB string into a byte buffer content. It is implemented as follows:

```
Public Function AGL_VBString2Buff(ByRef Buff As Byte, Text As
String, ByVal AnzChars As Long) As Long
  Dim Laenge As Long

  Laenge = Len(Text)
  If Laenge < AnzChars Then
    Text = Text & Space$(AnzChars - Laenge)
  ElseIf Laenge > AnzChars Then
    Text = Left$(Text, AnzChars)
  End If
  AGL_VBString2Buff = AGL_String2Buff(Buff, Text, AnzChars)
End Function
```

12.6 Converting S7 string into VB string (AGL_S7String2VBString)

This function directly converts a S7 string into a VB string. It is implemented as follows:

```
Public Function AGL_S7String2VBString(ByRef S7String As Byte, Text
As String, ByVal MaxChars As Long) As Long
  Dim Laenge As Long
  Dim RetVal As Long

  Laenge = Len(Text)
  If Laenge < MaxChars Then
    Text = Text & Space$(MaxChars - Laenge)
  ElseIf Laenge > MaxChars Then
    Text = Left$(Text, MaxChars)
  End If
  AGL_S7String2VBString = AGL_S7String2String(S7String, Text,
MaxChars)
  If RetVal = AGL40_SUCCESS Then
    Text = Left$(Text, lstrlen(Text))
```

```
  End If
End Function
```

12.7 Converting VB string into S7 string (AGL_VBString2S7String)

This function directly converts a VB string into a S7 string. It is implemented as follows:

```
Public Function AGL_VBString2S7String(ByRef S7String As Byte, Text
As String, ByVal MaxChars As Long) As Long
  Dim Laenge As Long

  Laenge = Len(Text)
  If Laenge < MaxChars Then
    Text = Text & Space$(MaxChars - Laenge)
  ElseIf Laenge > MaxChars Then
    Text = Left$(Text, MaxChars)
  End If
  AGL_VBString2S7String = AGL_String2S7String(S7String, Text,
MaxChars)
End Function
```

12.8 Converting number into hex display (AGL_Hex)

This function converts a number into a hexadecimal display with leading zeros. It is implemented as follows:

```
Public Function AGL_Hex(ByVal Wert As Long, ByVal Digits As Long) As
String
  AGL_Hex = Right$("00000000" & Hex$(Wert), Digits)
End Function
```

13 Programming reference configuration functions

With this chapter's functions you can implement the same functionality as in AGLink40-Config.EXE. This can be reasonable e.g. for the administration of an own phone book.

The parameters will be read out from the XML data file when opening a device for the first time. They also can be written program-controlled back into a XML data file.

13.1 Reading parameters for type (AGL_GetParas)

With this function you can read a device's settings.

C/C++-Syntax:

```
int WINAPI AGL_GetParas( int DevNr, int ParaType, void *Para, int
Len );
```

Visual Basic-Syntax:

```
Declare Function AGL_GetParas Lib "AGLink40.DLL" (ByVal DevNr As
Long, ByVal ParaType As Long, ByRef Para As Any, ByVal Len As Long)
As Long
```

Delphi-Syntax:

```
function AGL_GetParas( DevNr: Integer; ParaType: Integer; Para:
Pointer; Len: Integer): Integer; stdcall;
```

Parameter:

DevNr	Device number
ParaTyp	Parameter type
Para	Indicator on the proper parameters
Len	Buffer length

Return value:

AGL40_SUCCESS	If all worked properly
< 0	Error number (s. AGL_Defines.H, AGL_Defines.BAS oder AGLink40.PAS)

13.2 Setting parameters for type (AGL_SetParas)

With this function you can set settings for a device and a communication path.

C/C++-Syntax:

```
int WINAPI AGL_SetParas( int DevNr, int ParaType, void *Para, int
Len );
```

Visual Basic-Syntax:

```
Declare Function AGL_SetParas Lib "AGLink40.DLL" (ByVal DevNr As
Long, ByVal ParaType As Long, ByRef Para As Any, ByVal Len As Long)
As Long
```

Delphi-Syntax:

```
function AGL_SetParas( DevNr: Integer; ParaType: Integer; Para:
Pointer; Len: Integer): Integer; stdcall;
```

Parameter:

DevNr	Device number
ParaTyp	Parameter type
Para	Indicator on proper parameters
Len	Buffer length

Return value:

AGL40_SUCCESS	If all worked properly
< 0	Error number (s. AGL_Defines.H, AGL_Defines.BAS oder AGLink40.PAS)

13.3 Reading parameter type of device (AGL_GetDevType)

With this function you can determine the actual adjusted device type.

C/C++-Syntax:

```
int WINAPI AGL_GetDevType( int DevNr );
```

Visual Basic-Syntax:

```
Declare Function AGL_GetDevType Lib "AGLink40.DLL" (ByVal DevNr As
Long) As Long
```

Delphi-Syntax:

```
function AGL_GetDevType( DevNr: Integer): Integer; stdcall;
```

Parameter:

DevNr	Device number

Return value

>=0	Actual adjusted device type
< 0	Error number (s. AGL_Defines.H, AGL_Defines.BAS oder AGLink40.PAS)

13.4 Setting parameter type of device (AGL_SetDevType)

With this function you can adjust the type of a device.

C/C++-Syntax:

```
int WINAPI AGL_SetDevType( int DevNr, int DevType );
```

Visual Basic-Syntax:

```
Declare Function AGL_SetDevType Lib "AGLink40.DLL" (ByVal DevNr As
Long, ByVal DevType) As Long
```

Delphi-Syntax:

```
function AGL_SetDevType( DevNr: Integer; DevType: Integer ):
Integer; stdcall;
```

Parameter:

DevNr	Device number
DevType	Type which has to be adjusted

Return value:

AGL40_SUCCESS	If all worked properly
< 0	Error number (s. AGL_Defines.H, AGL_Defines.BAS oder AGLink40.PAS)

13.5 Reading settings of device type (AGL_ReadParas)

With this function you can read out a device type's settings from the loaded XML data file. And they will be put into an internal memory area.

C/C++-Syntax:

```
void WINAPI AGL_ReadParas( int DevNr, in ParaType );
```

Visual Basic-Syntax:

```
Declare Sub AGL_ReadParas Lib "AGLink40.DLL" (ByVal DevNr As Long,
ByVal ParaType As Long) As Long
```

Delphi-Syntax:

```
procedure AGL_ReadParas( DevNr: Integer; ParaType: Integer ):
Integer; stdcall;
```

Parameter:

DevNr	Device number
ParaType	Type of parameters which has to be read

Return value

none

13.6 Writing settings for a device type (AGL_WriteParas)

With this function all settings of a device type read from the internal memory area will be written into the loaded XML data file.

C/C++-Syntax:

```
void WINAPI AGL_WriteParas( int DevNr, in ParaType );
```

Visual Basic-Syntax:

```
Declare Sub AGL_WriteParas Lib "AGLink40.DLL" (ByVal DevNr As Long,
ByVal ParaType As Long) As Long
```

Delphi-Syntax:

```
procedure AGL_WriteParas( DevNr: Integer; ParaType: Integer ):
Integer; stdcall;
```

Parameter:

DevNr	Device number
ParaType	Type of parameters which has to be read

Return value:

none

13.7 Reading complete device settings (AGL_ReadDevice)

With this function the complete settings, the parameters of all connection paths as well as the actual adjusted device type, will be read into the internal memory area from the XML data file.

C/C++-Syntax:

```
void WINAPI AGL_ReadDevice( int DevNr );
```

Visual Basic-Syntax:

```
Declare Sub AGL_ReadDevice Lib "AGLink40.DLL" (ByVal DevNr As Long)
```

Delphi-Syntax:

```
procedure AGL_ReadDevice( DevNr: Integer ); stdcall;
```

Parameter:

DevNr	Device number

Return value:

none

13.8 Writing complete device settings (AGL_WriteDevice)

With this function the complete settings, the parameters of all connection paths as well as the actual adjusted device type, will be completely written into the internal memory area from the XML data file.

C/C++-Syntax:

```
void WINAPI AGL_WriteDevice( int DevNr );
```

Visual Basic-Syntax:

```
Declare Sub AGL_WriteDevice Lib "AGLink40.DLL" (ByVal DevNr As Long)
```

Delphi-Syntax:

```
procedure AGL_WriteDevice( DevNr: Integer ); stdcall;
```

Parameter:

DevNr	Device number

Return value:

none

13.9 Reading complete device settings (AGL_ReadParasFromFile)

With this function the complete settings, the parameters of all connection paths as well as the actual adjusted device type, will be completely read into the internal memory area from the XML data file.

C/C++-Syntax:

```
int WINAPI AGL_ReadParasFromFile( int DevNr, LPSTR FileName );
```

Visual Basic-Syntax:

```
Declare Function AGL_ReadParasFromFile Lib "AGLink40.DLL" (ByVal
DevNr As Long, ByVal FileName As String)
```

Delphi-Syntax:

```
function AGL_ReadParasFromFile( DevNr: Integer; FileName: PChar ):
Integer; stdcall;
```

Parameter:

DevNr	Device number
FileName	Name of XML data file

Return value:

AGL40_SUCCESS	If all worked properly
< 0	Error number (s. AGL_Defines.H, AGL_Defines.BAS oder AGLink40.PAS)

13.10 Writing complete device settings (AGL_WriteParasToFile)

With this function the complete settings, the parameters of all connection paths as well as the actual adjusted device type, will be completely written from the internal memory area into the XML data file.

C/C++-Syntax:

```
int WINAPI AGL_WriteParasToFile( int DevNr, LPSTR FileName );
```

Visual Basic-Syntax:

```
Declare Function AGL_WriteParasToFile Lib "AGLink40.DLL" (ByVal
DevNr As Long, ByVal FileName As String)
```

Delphi-Syntax:

```
function AGL_WriteParasToFile( DevNr: Integer; FileName: PChar ):
Integer; stdcall;
```

Parameter:

DevNr	Device number
FileName	Name of XML data file

Return value:

AGL40_SUCCESS	If all worked properly
< 0	Error number (s. AGL_Defines.H, AGL_Defines.BAS oder AGLink40.PAS)

13.11 Requesting path for parameter data (AGL_GetParaPath)

This function returns the path from which the parameter data are read and written into.

C/C++-Syntax:

```
int WINAPI AGL_GetParaPath( LPTSTR DirName, int MaxLen );
```

Visual Basic-Syntax:

```
Declare Function AGL_GetParaPath Lib "AGLink40.DLL" (ByVal DirName
As String, ByVal MaxLen As String) As Long
```

Delphi-Syntax:

```
function AGL_GetParaPath(DirName: PChar; MaxLen: Integer ): Integer;
stdcall;
```

Parameter:

DirName	Space for the path name
MaxLen	Maximum path length

Return value:

AGL40_SUCCESS	If all worked properly
< 0	Error number (s. AGL_Defines.H, AGL_Defines.BAS or AGLink40.PAS)

13.12 Setting path for parameter data (AGL_SetParaPath)

This function sets the path from which the parameter data are read and written into.

Diese Funktion setzt den Pfad, aus dem die Parameterdateien gelesen und in den sie geschrieben werden.

C/C++-Syntax:

```
int WINAPI AGL_SetParaPath( LPTSTR DirName );
```

Visual Basic-Syntax:

```
Declare Function AGL_SetParaPath Lib "AGLink40.DLL" (ByVal DirName
As String) As Long
```

Delphi-Syntax:

```
function AGL_SetParaPath(DirName: PChar ): Integer; stdcall;
```

Parameter:

DirName	Space for path name

Return value:

AGL40_SUCCESS	If all worked properly
< 0	Error number (s. AGL_Defines.H, AGL_Defines.BAS or AGLink40.PAS)

14 The use of .Net

A special wrapper DLL, AGL4DotNet.dll, has been created for the use under .Net. The DLL is based on the original AGLink40.dll. It offers for most functions respective .Net functions. Functions either supported or not are listed in this chapter. If calls have changed there are suitable notes.

It is a precondition that all components of ACCON-AGLink can be loaded. That means the data files AGLink.dll, AGLink_Config.exe and AGLink_Error.txt are in the directory of the respective application or better in a directory which is located in the searching path. The easiest way is to copy the data files into the Windows system directory (C:\Windows\System32 using NT/2000/XP, C:\Windows\System using 9x/Me). Furthermore it is necessary that the adjustment layer AGL4DotNet.dll is located in the Global Assembly Cache (normally C:\Windows\assembly) of the .Net Framework as well as in the directory of the administrated assemblies of the Microsoft Visual Studio 2005 (normally C:\Program files\Microsoft Visual Studio 8\Common7\IDE\PublicAssemblies).

You cannot use the functions asynchronously, so a negative timeout value is not allowed. Parameters which have to be handed over as indicators, the key words »ref« for preinitialized variables and »out« for not preinitialized variables are used.

14.1 Special structures

Structures which have changed in comparison to C++ will be explained in this chapter.

14.1.1 DATA_RW40

The data type Union which was used in the structure of the C++ type, is implemented here with 5 different data types. Except for buff which represents an indicator, all others variables which display the Union are implemented as fields.

```
struct DATA_RW40
{
  OpArea;
  public UInt16 OpType;
  public UInt16 OpAnz;
  public UInt16 DBNr;
  public UInt16 Offset;
  public UInt16 BitNr;
  public Int32 Result;

  /// <summary>
  /// Value(s) of the operand as single.
  /// The field size has to be at least OpAnz elements.
  /// Usable for OpType TYP_DWORD.
  /// </summary>
```

```
    public Single[] fValue;
    /// <summary>
    /// Value of the operand as DWORD.
    /// The field size has to be at least OpAnz elements.
    /// Usable for OpType TYP_DWORD.
    /// </summary>
    public UInt32[] Value;
    /// <summary>
    /// Value of the operand as WORD
    /// The field size has to be at least OpAnz elements.
    /// Usable for OpType TYP_WORD, TYP_COUNTER, TYP_TIMER,
        TYP_COUNTER_200, TYP_TIMER_200 and TYP_HS_COUNTER_200.
    /// </summary>
    public UInt16[] W;
    /// <summary>
    /// Value of the operand as BYTE
    /// The field size has to be at least OpAnz elements.
    /// Usable for OpType TYP_BIT and TYP_BYTE.
    /// </summary>
    public Byte[] B;
    /// <summary>
    /// Indicator
    /// </summary>
    public IntPtr Buff;
  }
```

14.2 Supported functions – without changes

A list of functions which do not have any changes in parameters or return values compared to the C++ implementation.

All function names are identical to the C++ function names. For the .Net implementation, only the prefix »AGL_« has been removed.

14.2.1 Administration - General functions

.Net	C++
Activate	AGL_Activate
GetMaxDevices	AGL_GetMaxDevices
GetMaxPLCPerDevice	AGL_GetMaxPLCPerDevice
GetMaxQueues	AGL_GetMaxQueues
GetTickCount	AGL_GetTickCount

GetMicroSecs	AGL_GetMicroSecs
UseSystemTime	AGL_UseSystemTime
GetErrorMsg	AGL_GetErrorMsg
LoadErrorFile	AGL_LoadErrorFile
Config	AGL_Config
UnloadDyn	AGL_UnloadDyn
OpenDevice	AGL_OpenDevice
CloseDevice	AGL_CloseDevice

14.2.2 Communication functions - Adaptor

.Net	C++
DialUp	AGL_DialUp
HangUp	AGL_HangUp
InitAdapter	AGL_InitAdapter
ExitAdapter	AGL_ExitAdapter
GetLifeList	AGL_GetLifeList
GetDirectPLC	AGL_GetDirectPLC

14.2.3 Communication functions - PLC

.Net	C++
PLCConnect	AGL_PLCConnect
PLCConnectEx	AGL_PLCConnectEx
PLCDisconnect	AGL_PLCDisconnect
ReadPLCInfo	AGL_ReadPLCInfo
ReadDiagBufferEntrys	AGL_ReadDiagBufferEntrys
ReadDiagBuffer	AGL_ReadDiagBuffer
GetDiagBufferEntry	AGL_GetDiagBufferEntry
ReadOpState	AGL_ReadOpState
PLCStop	AGL_PLCStop
PLCStart	AGL_PLCStart
PLCResume	AGL_PLCResume
ReadCycleTime	AGL_ReadCycleTime
ReadProtLevel	AGL_ReadProtLevel
GetPLCClock	AGL_GetPLCClock
SetPLCClock	AGL_SetPLCClock
ReadSzl	AGL_ReadSzl
ReadDBCount	AGL_ReadDBCount

ReadDBList	AGL_ReadDBList
ReadDBLen	AGL_ReadDBLen
ReadMaxPacketSize	AGL_ReadMaxPacketSize

14.2.4 Communication functions - Reading data

.Net	C++
ReadInBytes	AGL_ReadInBytes
ReadPInBytes	AGL_ReadPInBytes
ReadOutBytes	AGL_ReadOutBytes
ReadFlagBytes	AGL_ReadFlagBytes
ReadSFlagBytes	AGL_ReadSFlagBytes
ReadVarBytes	AGL_ReadVarBytes
ReadDataBytes	AGL_ReadDataBytes
ReadDataWords	AGL_ReadDataWords
ReadTimerWords	AGL_ReadTimerWords
ReadCounterWords	AGL_ReadCounterWords
ReadMix	AGL_ReadMix
ReadMixEx	AGL_ReadMixEx
InitOptReadMix	AGL_InitOptReadMix
ReadOptReadMix	AGL_ReadOptReadMix
EndOptReadMix	AGL_EndOptReadMix
InitOptReadMixEx	AGL_InitOptReadMixEx
ReadOptReadMixEx	AGL_ReadOptReadMixEx
EndOptReadMixEx	AGL_EndOptReadMixEx

14.2.5 Communication functions - Writing data

.Net	C++
WriteInBytes	AGL_WriteInBytes
WriteOutBytes	AGL_WriteOutBytes
WritePOutBytes	AGL_WritePOutBytes
WriteFlagBytes	AGL_WriteFlagBytes
WriteSFlagBytes	AGL_WriteSFlagBytes
WriteVarBytes	AGL_WriteVarBytes
WriteDataBytes	AGL_WriteDataBytes
WriteDataWords	AGL_WriteDataWords
WriteTimerWords	AGL_WriteTimerWords
WriteCounterWords	AGL_WriteCounterWords

WriteMix	AGL_WriteMix
WriteMixEx	AGL_WriteMixEx
InitOptWriteMix	AGL_InitOptWriteMix
WriteOptWriteMix	AGL_WriteOptWriteMix
EndOptWriteMix	AGL_EndOptWriteMix
InitOptWriteMixEx	AGL_InitOptWriteMixEx
WriteOptWriteMixEx	AGL_WriteOptWriteMixEx
WriteRWBuff	AGL_WriteRWBuff

14.2.6 Additional functions - Reading functions

.Net	C++
ReadInt16	AGL_ReadInt16
ReadInt32	AGL_ReadInt32
ReadWord	AGL_ReadWord
ReadDWord	AGL_ReadDWord
ReadReal	AGL_ReadReal
ReadS5Time	AGL_ReadS5Time

14.2.7 Additional functions - Writing functions

.Net	C++
WriteInt16	AGL_WriteInt16
WriteInt32	AGL_WriteInt32
WriteWord	AGL_WriteWord
WriteDWord	AGL_WriteDWord
WriteReal	AGL_WriteReal
WriteS5Time	AGL_WriteS5Time

14.2.8 Additional functions - Buffer converting functions

.Net	C++
Byte2Word	AGL_Byte2Word
Byte2DWord	AGL_Byte2DWord
Byte2Real	AGL_Byte2Real
Word2Byte	AGL_Word2Byte
DWord2Byte	AGL_DWord2Byte
Real2Byte	AGL_Real2Byte
Buff2String	AGL_Buff2String

String2Buff	AGL_String2Buff

14.2.9 Additional functions - Converting S7 types

.Net	C++
S7String2String	AGL_S7String2String
String2S7String	AGL_String2S7String
S7DT2SysTime	AGL_S7DT2SysTime
SysTime2S7DT	AGL_SysTime2S7DT
TOD2SysTime	AGL_TOD2SysTime
SysTime2TOD	AGL_SysTime2TOD

14.2.10 Additional functions - Converting S5 types

.Net	C++
Float2KG	AGL_Float2KG
KG2Float	AGL_KG2Float

14.2.11 Additional functions ionen - Special conversions

.Net	C++
BCD2Int16	AGL_BCD2Int16
Int162BCD	AGL_Int162BCD
BCD2Int32	AGL_BCD2Int32
Int322BCD	AGL_Int322BCD
LongAsFloat	AGL_LongAsFloat
FloatAsLong	AGL_FloatAsLong
GetBit	AGL_GetBit
SetBit	AGL_SetBit
ResetBit	AGL_ResetBit
SetBitVal	AGL_SetBitVal
Text2DataRW	AGL_Text2DataRW
DataRW2Text	AGL_DataRW2Text

14.2.12 Additional functions – DLL info functions

.Net	C++
GetVersion	AGL_GetVersion
GetVersionEx	AGL_GetVersionEx
GetOptions	AGL_GetOptions

GetSerialNumber	AGL_GetSerialNumber
GetClientName	AGL_GetClientName

14.2.13 Additional functions – Miscellaneous functions

.Net	C++
GetPLCType	AGL_GetPLCType
HasFunc	AGL_HasFunc

14.2.14 Configuration functions

.Net	C++
GetDevType	AGL_GetDevType

14.2.15 Supported functions - with changes

A list of functions which differ in parameters or return values compared to the C++ implementation.

All function names are identical to the C++ function names. For the .Net implementation, only the prefix »AGL_« has been removed.

ReadMLFBNr

The parameter »MLFBNr« has to be a string which conforms to the type LPMLFB in C++. It has to be handed over as reference with the key word »out«. A pre initialization is not necessary.

ReadMLFBNrEx

The parameter »MLFBNr« has to be a string, the parameter »PLCVer« has to be a UInt16 and the parameter »PGASVer« has to be a UInt16. All three mentioned parameters will be handed over as reference with the key word »out«. These parameters conform to the structure LPMLFBEX in C++ and do not need a pre initialization.

15 ACCON-AGLink 4.0 special version for ACCON-NetLink-PRO/USB/S7

This special version for the ACCON-NetLink family has the same software interface as the »big« version. But it does not contain all functions. The missing funtions return error AGL40_FUNC_NOT_IMPLEMENTED. And the communication for each function call is limited to one MPI-/Profibus reference packet, too. The detailed limits are described in the following table. If these limits are being exceeded the function will not be executed and returns the error AGL40_DATA_TOO_LONG.

The following functions are available in this version:

Function	Limit
AGL_Activate	
AGL_GetMaxDevices	
AGL_GetMaxPLCPerDevice	
AGL_GetMaxQueues	
AGL_GetTickCount	
AGL_GetMicroSecs	
AGL_UseSystemTime	
AGL_GetErrorMsg	
AGL_LoadErrorFile	
AGL_Config	
AGL_UnloadDyn	
AGL_SetDevNotification	
AGL_SetConnNotification	
AGL_SetJobNotification	
AGL_WaitForJob	
AGL_WaitForJobEx	
AGL_DeleteJob	
AGL_GetJobResult	
AGL_OpenDevice	
AGL_CloseDevice	
AGL_DialUp	
AGL_HangUp	
AGL_InitAdapter	
AGL_ExitAdapter	
AGL_GetLifelist	
AGL_GetDirectPLC	

Function	Limit
AGL_PLCConnect	
AGL_PLCConnectEx	
AGL_PLCDisconnect	
AGL_ReadMLFBNr	
AGL_ReadPLCInfo	
AGL_ReadOpState	
AGL_ReadMaxPacketSize	
AGL_ReadDBCount	
AGL_ReadDBList	
AGL_ReadDBLen	
AGL_ReadInBytes	Maximum of 222 bytes for each call
AGL_ReadOutBytes	Maximum of 222 bytes for each call
AGL_ReadFlagBytes	Maximum of 222 bytes for each call
AGL_ReadSFlagBytes	Maximum of 222 bytes for each call
AGL_ReadVarBytes	Maximum of 222 bytes for each call
AGL_ReadDataBytes	Maximum of 222 bytes for each call
AGL_ReadTimerWords	Maximum of 111 timer for each call
AGL_ReadCounterWords	Maximum of 111 counter for each call
AGL_ReadMix	Maximum of 19 structures for each call
AGL_WriteInBytes	Maximum of 218 bytes for each call
AGL_WriteOutBytes	Maximum of 218 bytes for each call
AGL_WriteFlagBytes	Maximum of 218 bytes for each call
AGL_WriteSFlagBytes	Maximum of 218 bytes for each call
AGL_WriteVarBytes	Maximum of 218 bytes for each call
AGL_WriteDataBytes	Maximum of 218 bytes for each call
AGL_WriteTimerWords	Maximum of 109 timer for each call
AGL_WriteCounterWords	Maximum of 109 counter for each call
AGL_WriteMix	Maximum of 11 structures for each call
AGL_GetVersion	
AGL_GetVersionEx	
AGL_GetOptions	
AGL_GetSerialNumber	
AGL_GetClientName	

Function	Limit
AGL_GetParas	
AGL_SetParas	
AGL_GetDevType	
AGL_SetDevType	
AGL_ReadParas	
AGL_WriteParas	
AGL_ReadDevice	
AGL_WriteDevice	
AGL_ReadParasFromFile	
AGL_WriteParasToFile	
AGL_GetPLCType	
AGL_HasFunc	
AGL_GetParaPath	
AGL_SetParaPath	

Chart 10: Functions for ACCON-AGLink 4.0 special version

16 Appendix A FAQ list

16.1 Frequently asked questions

Q: How fast is ACCON-AGLink?

A:.It all depends on the used hardware or the transmission path. For this just use the supplied program AGLink_Performance. You will find a more detailed description in chapter »3.3 Performance Check Program (AGLink40_Performance.EXE)«.

Q: Do I have to change my porgram when an additional module is needed?

A: No. Due to the ACCON-AGLink's standard software interface it is possible to bring in a new DLL or shared object and you can adjust the parameters with the program AGLink40_Config.EXE. Then you can use the new hardware, immediately.

Q: When I need e.g. MPI serial and TCP/IP do I receive two DLL versions?

A: No, because all hardware support you have purchased is located in one DLL and always available. A version management is not necessary (which hardware DLL is on which customer PC).

Q: What advantages does ACCON-AGLink provide?

A: ACCON-AGLink is fast, compact and easy to use. All data files needed by the customer suit on one single disc. They can be easily copied to the target PC and a complex installation is not neccesary.
ACCON-AGLink is multi-thread secure and on the other hand it allows an asynchronous communication with all modules. In the front you can quickly react on user inputs and during this the communication will be handled unseen in the back. Furthermore, ACCON-AGLink provides a standard software interface for all modules. So, if customers need additional hardware support they do not have to change their programs. It is sufficient to bring in the new DLL or shared object and to configure the interface corresponding to the used hardware.

Q: What happens if there is new hardware which my version of ACCON-AGLink does not support?

A: The development of ACCON-AGLink will be constantly continued and adjusted to the new hardware. To be always up to date we offer a maintenance contract for ACCON-AGLink. With this you will receive all new DLL versions or shared objects via e-mail, automatically. That's comfort.

Q: Is there a demo version of ACCON-AGLink?

A: Of course. You can download it from our web site at www.deltalogic.de. Naturally, we can send you our acutal software CD-ROM. A call, fax or e-mail is sufficient.

Q: What functions does the demo version of ACCON-AGLink have?

A: With this demo version you can test all functions of ACCON-AGLink. Only all five minutes the communication stops and a information is displayed which says that this is a demo version of ACCON-AGLink.

Q: Do I need Siemens' TS-software when using the ACCON-AGLink S7 serial/TS?

A: No. All functions to manage telephone numbers, to establish a connection and for data transmission are handled by the ACCON-AGLink S7 seriell/TS.

Q: What happens when pulling the cable from the ACCON-AGLink S7-MPI serial?

A: You will receive a corresponding error message. In this case please completely deinitialize the interface with AGLPLCDisconnect, AGLExitAdapter and AGLHangUp. Now start the communication again using AGLDialUp and AGLInitAdapter. As soon as the adapter is plugged in you will not receive an error message with AGLInitAdapter. Then start AGLPLCConnect and continue the communication. You can also look for the example AGLink_ConnTest in chapter »3.4 Connection Check Program (AGLink40_ConnTest.EXE)«, it contains the complete source code.

Q: How big can the requested data file packages be?

A: In principle they can be of any size. The ACCON-AGLink automatically separates the data file packages for the PLC into suitable pieces. In doing so the transmission medium as well as the PLC family will be considered. You do not have to do anything, e.g. you can request 8192 data bytes in one piece. You will receive a corresponding error message from the PLC if the block size is too big for the PLC as the requested inputs are not available.

Q: Can I read double words with ACCON-AGLink, too?

A: As in S7 standard, ACCON-AGLink always reads byte for byte when requesting blocks. With AGL_ReadMix, AGL_WriteMix or AGL_ReadMixEx and AGL_WriteMixEx you have powerful functions which can process besides bytes words, double words and even bits. If there are words, double words, real numbers, etc. in a read data block, ACCON-AGLink contains corresponding converting functions for reading respecitvely writing access into the apropriate buffer.

Q: Do I have to change the S7 program when accessing a PLC?

A: No. You only have to know which control data files are important for you. With ACCON-AGLink you can access the data, input, output and marker bytes as well as the timers and counter of the PLC completely transparent.

Q: Can I access a control with ACCON-AGLink S7-MPI via Profibus, too?

A: Yes. There is no difference. But when accessing via Profibus it is important that you select the correct baud rate and the correct profile for the communication in AGLink40-Config.EXE (or program-controled).

Q: What can I do if the communication via PC adapter is too slow?

A: The easiest way is to use our MPI/Profibus card or an adapter from our ACCON-NetLink family. They immediately provide a higher data throughput in relation to moderate costs. PLC-sided there is no CP required.

Q: How can I access a S7 via PC adapter if the MPI interface is busy by a OP?

A: MPI is a bus system. So to connect the OP you just need a corresponding bus plug with lead-through. It is only important that the voltage feed is looped in otherwise the adapter remains off.

Q: How can I access several controls with ACCON-AGLink, simultaneously?

A: Very simple. On function call you indicate which PLC you want to access to. But you have to remember a few hardware restrictions when accessing parallely. Depending on the used hardware the number of connections at the same time are limited to 2 (MPI adapter), 4 (PC adapter and ACCON-NetLink), 12 (ACCON-NetLink-PRO or ACCON-NetLink –USB) and 16 (ACCON-AGlink). When exceeding this amount you can log off from one PLC and log into the next, by doing this you can prevent to go beyond the maximum amount.
Alternatively, you can split your bus system into smaller segments. Then use a communications adapter for each segment (valid for MPI/Profibus). ACCON-AGLink supports up to 256 devices at the same time with a maximum limit of 16 PLCs each. That means you can directly acces up to 4096 PLCs. When using TCP/IP it is sufficient to parametrize and use a further device. It is not necessary to use additional PC hardware for the 4096 PLCs.

Q: AGL-PLCConnect is successful when using ACCON-AGLink S7 TCP/IP. But reading data outpouts errors. What's wrong here?

A: Probably, when parametrizing via AGLink40_Config (or via programming code) you have entered the rack and slot number of the CP and not the CPU. Extract the correct values from the hardware configuration and enter them via AGLink40_Config into the parameters. The device test AGLink40-Config is the easiest way to try it out. When entering wrong rack and slot numbers this program shows you the correct numbers.

Q: Where do I have to place the prameter date file »AGLink40CfgDev0000.xml« for the program sequence?

A: At first ACCON-AGLink searches in the actual working directory on porgram start. If no parametrization data file is found the search will continue in the program directory (where the AGLink40.DLL is placed). If nothing is found there, too, the process will continue with the standard values.

Q: Which data files does the configuration program require?

A: The configuration program needs the following data files:

```
AGLink40_Config.exe          The proper configuration program
AGLink40_Config.xml          Parameters of the configuration program
AGLink40_Config_de.mo        German language dataS
AGLink40_Config_en.mo        English language data
```

If you want to perform a device test you have to add the following:

```
AGLink40.dll                 (Demo)version of ACCON-AGLink
AGLink40_Error.txt           Data with AGLink error messages in plain text
```