WinAC AccessDB V2

Operator manual

V 2.01 • February 2012

Applikationen & Tools

Answers for industry.

SIEMENS

Industry Automation and Drives Technologies Service & Support Portal

This article is taken from the Service Portal of Siemens AG, Industry Automation and Drives Technologies. The following link takes you directly to the download page of this document.

http://support.automation.siemens.com/WW/view/en/59061340

If you have any questions concerning this document please e-mail us to the following address:

online-support.automation@siemens.com

applications.aud.koe.nrh.rd@siemens.com

S

SIMATIC
WinAC AccessDB V2

Operator manual

Warranty and Liability

Note

The Application Examples are not binding and do not claim to be complete regarding the circuits shown, equipping and any eventuality. The Application Examples do not represent customer-specific solutions. They are only intended to provide support for typical applications. You are responsible for ensuring that the described products are used correctly. These application examples do not relieve you of the responsibility to use safe practices in application, installation, operation and maintenance. When using these Application Examples, you recognize that we cannot be made liable for any damage/claims beyond the liability clause described. We reserve the right to make changes to these Application Examples at any time without prior notice.
If there are any deviations between the recommendations provided in these application examples and other Siemens publications – e.g. Catalogs – the contents of the other documents have priority.

We do not accept any liability for the information contained in this document.

Any claims against us – based on whatever legal reason – resulting from the use of the examples, information, programs, engineering and performance data etc., described in this Application Example shall be excluded. Such an exclusion shall not apply in the case of mandatory liability, e.g. under the German Product Liability Act ("Produkthaftungsgesetz"), in case of intent, gross negligence, or injury of life, body or health, guarantee for the quality of a product, fraudulent concealment of a deficiency or breach of a condition which goes to the root of the contract ("wesentliche Vertragspflichten"). The damages for a breach of a substantial contractual obligation are, however, limited to the foreseeable damage, typical for the type of contract, except in the event of intent or gross negligence or injury to life, body or health. The above provisions do not imply a change of the burden of proof to your detriment.

Table of Contents

1 Automation Task

1.1 Overview

Introduction

The existing driver **WinAC Access DB** enables external high-level language applications to get direct access to WinAC's variables (flags, data blocks, inputs, outputs).

Because of the internal functionality of the driver it was not possible to migrate existing applications to Windows 7.

That's why the driver was completely rewritten – **WinAC AccessDB V2**. The Application Programming Interface (API) for the user stays exactly the same. Existing applications using WinAC AccessDB V1.x should work without any code changes with WinAC AccessDB V2.x, too.

There are various reasons to redesign the complete driver:

- No direct memory access any longer (e.g. problems when downloading data blocks in RUN – addresses are changed)
- Reduce the number of involved components
- No utilization of kernel drivers (Windows 7 security concept)
- Usage of functions of the WinAC ODK V4.2 (ODK_ReadData / ODK_WriteData)
- No mapping of physical memory (resources limited – long term stability)
- Reduce of function block / operation block calls in the WinAC program

The advantages of the WinAC AccessDB against alternative options for WinAC variable access (e.g. OPC. SAPI-S7, etc.) are characteristic for the new version, too:

- High performant access to WinAC data
- Easy integration in the high-level language application
- No connection projecting needed in Step7 project

1.2 Needed knowledge

To understand this document the following knowledge needed:

- SIMATIC WinAC RTX 2010
- SIMATIC Manager STEP7 V5.5

  or
- SIAMTIC STEP7 V11 SP2 (TIA Portal)

1.3 Required Hardware and Software Components

The application was generated with the following components:

Hardware components

- Simatic Microbox IPC 427C (1,2 GHz, 4 GB RAM, 4 GB CF-Card) with Windows Embedded Standard 7

Standard software components

- SIMATIC WinAC RTX 2010
- SIMATIC Manager STEP 7 V5.5

  or
- SIAMTIC STEP7 V11 SP2 (TIA Portal)

Sample files and projects

The following list includes all files and projects that are used in this example.

Table 1-1 Included files

Component	Note
Documentation \ WinAC-AccessDB_Doku_v2xx_DE.pdf Documentation \ WinAC-AccessDB_Doku_v2xx_EN.pdf	This documentation in German and English
Driver \ WinAcAccDbSrv.rtdll	The WinAC driver
Driver \ setup.bat	Setup batch for the driver
S7ClassicExample \ AccessDBv2.zip	Step7 example project including all WinAC function blocks of the driver
TiaExample \ AccessDBv2.zip	Example for TIA portal project including all WinAC function blocks of the driver
Examples \	Directory with example applications in various high-level languages

2 Overview

2.1 Functional range

The WinAC driver **WinAC AccessDB V2** enables the easy and high performant access to any data of the running WinAC RTX from an external PC application. The application as well as the WinAC has to run on one PC.

The driver can be used in all common development environments and high level programm languages (MS Visual Basic 6, MS Visual C++ 6, Delphi, etc.)

The Application Programming Interface (API) is compatible to the Version V1.x.

NOTE Only **one** client application can use the WinAC AccessDB V2 functionality at same time!

The new version does not access physical memory, any longer. Instead supported functions of the WinAC Open Development Kit (ODK) are used. The speed should be comparable with version V1.x when “Safe Mode” is activated.

2.2 Version of the driver

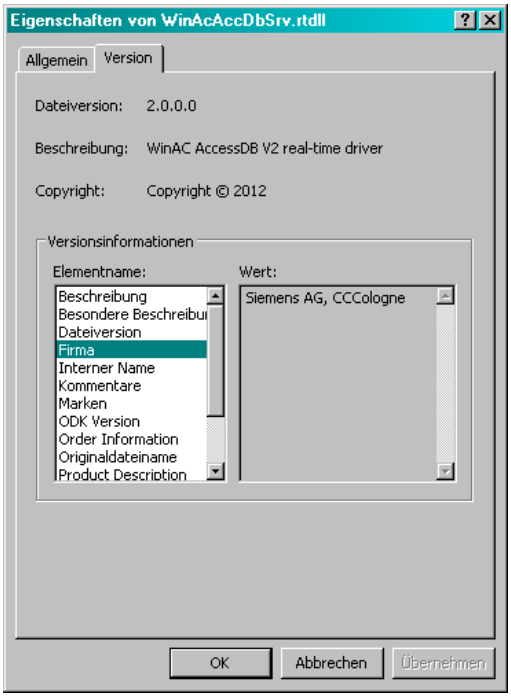
Determine driver under Windows OS

The registered reeal-time driver (RT)DLL is located in the directory

C:\Windows\RTSS\rtdll**WinAcAccDbSrv.rtdll**

You can identify the version of the driver RTDLL in the file properties (Windows explorer → right click → properties)

Figure 2-1 Properties of the driver RTDLL (Windows XP)



Check driver version in Step7 project

In the instance DB of FB_INIT_ACCESS_DB the version of the RTDLL is stored, too:

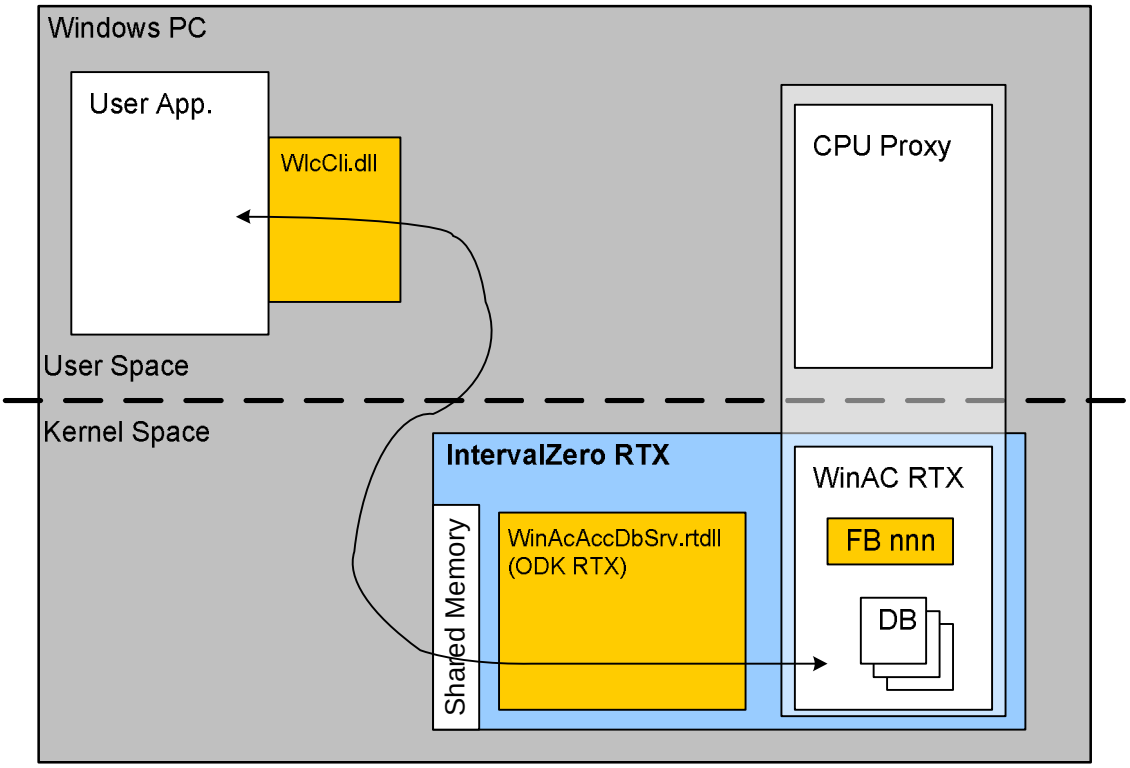
tOdkIf.dwDllVersion version of the driver DLL

3 Driver supported functionality

3.1 General overview

For the new version of AccessDB V2 the number of involved components is significantly reduced.

Figure 3-1 Structural overview of WinAC AccessDB V2



The user application includes (uses) the DLL **WlcCli.dll**. This DLL communicates via Inter Process Communication (IPC) with the WinAC ODK real-time driver **WinAcAccDbSrc.rtdll**.

In the WinAC program only one single function block is needed. It has to be called once. This function block loads the driver ODK-DLL. During runtime no additional FB or OB call is needed.

Example – How the access “Read DB10 DBB5” is executed?

- WlcCli.dll requests data from WinAcAccDbSrv.rtdll:
  Access: “Read”
  Type: “Byte”
  Address: “DB10, Byte 5”
- WinAcAccDbSrv.rtdll utilizes the ODK function ODK_ReadData to retrive the data and returns the data to the WlcCli.dll

Because ODK functions are used to access data instead of direct memory access, the operation is a little bit slower. It is much safer, because the download of data

blocks in state RUN can not generate errors. The performance is comparable with the version V1.x when “Safe Mode” is activated.

3.2 Application DLL (Client)

The DLL **WIcCli.dll** has to be utilized by the high-level application.

From the application’s point of view the DLL provides a number of funtions to access WinAC data, like ReadBOOL, WriteBOOL, ReadBYTE, WriteBYTE, etc.

Before using any read/write function the function InitAccessDB() has to be called. This function initializes the driver including the inter-process communication.

When a read/write function is called, the IPC (shared memory, events, etc.) is used to send a request to the ODK driver. The answer (or a time-out error) is returned to the caller.

The function SetSafeMode() is not needed any longer. Because of compatibility reasons it is still available (but without any function).

3.3 Step 7 functions

The Step7 part contains only one single function block: FB_INIT_ACCESS_DB. This function has to be called once (e.g. in OB100 COMPLETE RESTART). It loads the driver. By loading the driver, a separate thread is started. This thread processes all request from the client side. Additional calls of function blocks or operation blocks are not needed.

This is an important simplification in comparison to version V1.x

3.4 ODK driver (server)

NOTE The following remarks are not necessarily needed for the high-level programmer. It shows the internal structure of the driver.

The ODK driver is realized as real-time driver (IntervalZero RTX).

The ODK application contains only one thread (“ODK monitor thread”). This thread is waiting for request from the client side. If a request arrives, it is processed and the answer (or error code) is returned to the client.

NOTICE **The server processes only one request at same time. There is no queue available. Multiple clients are not supported.**

The limitation to one single request was a conscious decision. In that way the interface is easy to implement, less error prone and faster.

The WinAC driver consists in one single RTDLL, only (WinAcAccDbSrv.rtdll). This is an important simplification in comparison to version V1.x

3.5 IPC between client application and server (ODK side)

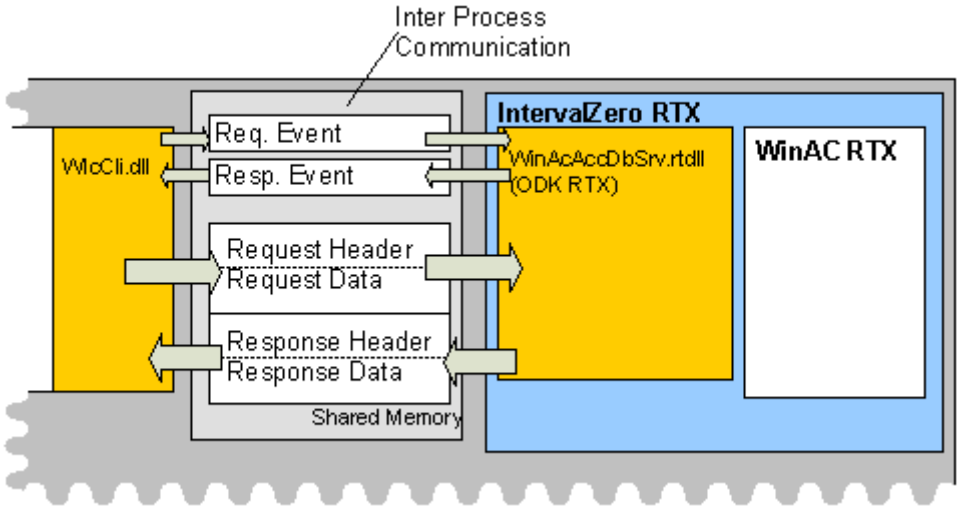
NOTE The following remarks are not necessarily needed for the high-level programmer. It shows the internal structure of the driver.

The communication between user application (client DLL) and the ODK RTDLL (server) is realized via Inter Process Communication (IPC).

Events and shared memory is needed. For every item it is defined which side has write or read access. Either the client is writing and the server is reading or vice versa.

The following image shows the IPC between client and server.

Figure 3-2 IPC between client application and ODK RTDLL



In the shared memory there are four areas:

- Request header
- Request data
- Response header
- Response data

Request Block

The request block is only written by the client. When the request block is filled, the “request event” is set.

If it is a write request, the data to be written is stored in the “request data” area.

One data block can be maximum 65 Kbyte. That’s why this area is 65 Kbyte, too.

Response Block

The response block is only written by the server. If the server recognized the “request event” it reads the data from the “request block” and writes the data to the “response block”. Finally it sets the “response event”.

It the request was a read task, the data is stored in the “response data”.

One data block can be maximum 65 Kbyte. That’s why this area is 65 Kbyte, too.

3.6 Performance

The performance of the WinAC AccessDB V2 driver was examined with the reference system:

Microbox IPC427C

Intel Core2Duo U9300 @ 1,20 GHz

2 GB RAM

Windows Embedded Standard 2010 (Windows 7 based) - Microbox Image

WinAC RTX 2010

Tuning-Panel: Min Sleep Time: 0.0 ms

Min Cycle Time: 10.0 ms

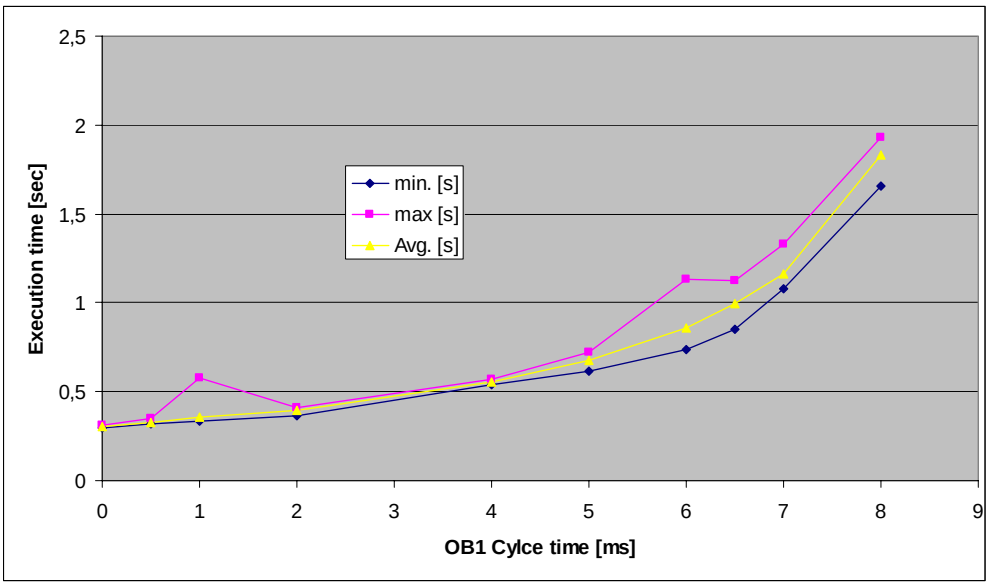
A CPU load is generated by a loop with numerous float multiplications. The OB1 execution time can be adjusted by changing the loop count.

An external high-level language application executes 5.000 write commands with AccessDB V2. The time is measured in the high-level language program: how long takes the execution of the 5.000 commands. This measurement is repeated 100 times.

Table 3-1 Performance of 5.000 write commands with AccessDB V2

OB1 Exec time [ms]	min. [s]	max [s]	Avg. [s]
0	0,2969	0,3129	0,3049
0,5	0,3157	0,3509	0,3277
1	0,3322	0,5779	0,3544
2	0,3678	0,4092	0,3927
4	0,5359	0,5704	0,5542
5	0,616	0,7248	0,6777
6	0,7389	1,1292	0,8589
6,5	0,8539	1,128	0,9925
7	1,0755	1,3317	1,1612
8	1,6583	1,9268	1,8289

Figure 3-3 Diagram of performance of the AccessDB V2 driver



The speed for writing variables with AccessDB V2 depends on the CPU load of the WinAC.

The external PC application is a Windows application. That's why the execution time of the test varies.

Read commands are executed with the same speed like reading.

4 The Step7 Interface (WinAC FBs)

To use the WinAC AccessDB V2 driver only one single function block is needed:

FB10.001 – FB_INIT_ACCESS_DB

The number of the function block and the instance data block may be changed by the user.

4.1 FB_INIT_ACCESS_DB

This block initializes the driver. It has to be called once.

Table 4-1 Parameter of FB FB_INIT_ACCESS_DB

Parameter	In/ Out	Typ	Description
ERROR	Out	Bool	Error
STATUS	Out	WORD	Status information

Additional information in instance DB of FB_INIT_ACCESS_DB

Additional to the output parameters some information is stored in the instance data block of the function block:

Table 4-2 Information in instance data block of FB_INIT_ACCESS_DB

Name	In/ Out	Description
CIF.DLL_VERSION	In	Version of driver DLL

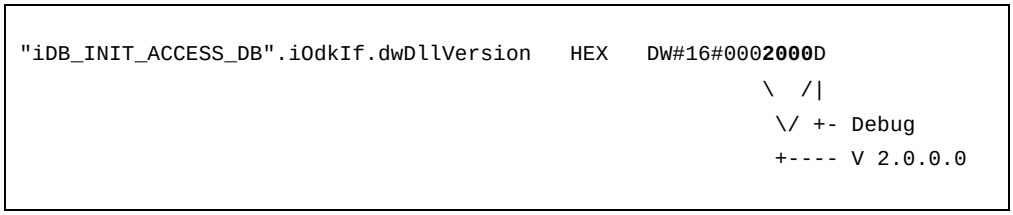
Coding of DLL version

The DLL version is coded hexadecimal. The last sign of the DWORD is used for mark Debug and Release version:

D – Debug-Version

A – Release-Version

Figure 4-1 Examples for DLL versions in the instance DB



```
"iDB_INIT_ACCESS_DB".iOdkIf.dwDllVersion   HEX   DW#16#000 2100 A
                                                          \  /|
                                                           \/ +- Release
                                                           +---- V 2.1.0.0
```

Note The data in the instance data block is valid after the first call of INIT function block!

5 The Application Programming Interface (API)

5.1 Initialization

Before the first usage of the driver, it has to be initialized.

DWORD WINAPI InitAccessDB( )		Initialize WinAC AccessDB interface.

NOTICE **This function has to be called before any other call of functions from the WinAC AccessDB V2 API!**

5.2 Activate / Deactivate “Safe Mode”

The so called “safe mode” was introduced in driver version V1.x. It was needed for right operation when data blocks are loaded during run-time. This download changes addresses.

NOTE

This function is not needed any longer in version V2.x!

It is still part of the API because of compatibility.

DWORD WINAPI SetSafeMode( )		*Without function (Compatibility)*

5.3 Functions for Reading (WinAC → application)

NOTE

For all functions for the data block number either a data block is provided or a special code:

66000 – Input process image
66001 – Output process image
66002 – Flags (“Merker”)

DWORD WINAPI **ReadBOOL**( )		Read a BOOL value from WinAC *Parameter:* DWORD **DBNummer** – data block to read DWORD **ByteOffset** – byte address DWORD **BitOffset** – bit address BIT16 **pValue** – pointer to variable for read value
DWORD WINAPI **ReadBYTE**( )		Read a BYTE value from WinAC *Parameter:* DWORD **DBNummer** – data block to read DWORD **ByteOffset** – byte address BIT8 **pValue** – pointer to variable for read value
DWORD WINAPI **ReadWORD**( )		Read a WORD value from WinAC *Parameter:* DWORD **DBNummer** – data block to read DWORD **ByteOffset** – byte address BIT16 **pValue** – pointer to variable for read value
DWORD WINAPI **ReadDWORD**( )		Read a DWORD value from WinAC *Parameter:* DWORD **DBNummer** – data block to read DWORD **ByteOffset** – byte address BIT32 **pValue** – pointer to variable for read value
DWORD WINAPI **ReadINT**( )		Read a INT value from WinAC *Parameter:* DWORD **DBNummer** – data block to read DWORD **ByteOffset** – byte address SINT16 **pValue** – pointer to variable for read value
DWORD WINAPI **ReadDINT**( )		Read a DINT value from WinAC *Parameter:* DWORD **DBNummer** – data block to read DWORD **ByteOffset** – byte address SINT32 **pValue** – pointer to variable for read value

DWORD WINAPI **ReadREAL**( )		Read a REAL-Wert from WinAC *Parameter:* DWORD **DBNummer** – data block to read DWORD **ByteOffset** – byte address FLOAT **pValue** – pointer to variable for read value
DWORD WINAPI **ReadDATE**( )		Read a DATE value from WinAC *Parameter:* DWORD **DBNummer** – data block to read DWORD **ByteOffset** – byte address UINT16 **pValue** – pointer to variable for read value **Attention!** This function does not execute any conversion. It only provides 2 bytes ot the S7 variable with type DATE.
DWORD WINAPI **ReadS5TIME**( )		Read a S5TIME value from WinAC *Parameter:* DWORD **DBNummer** – data block to read DWORD **ByteOffset** – byte address BIT16 **pValue** – pointer to variable for read value **Attention!** This function does not execute any conversion. It only provides 2 bytes ot the S7 variable with type S5TIME.
DWORD WINAPI **ReadS7TIME**( )		Read a S7TIME value from WinAC *Parameter:* DWORD **DBNummer** – data block to read DWORD **ByteOffset** – byte address SINT32 **pValue** – pointer to variable for read value **Attention!** This function does not execute any conversion. It only provides 2 bytes ot the S7 variable with type TIME.
DWORD WINAPI **ReadS7TIME_OF_DAY**		Read a S7TIME_OF_DAY value from WinAC *Parameter:* DWORD **DBNummer** – data block to read DWORD **ByteOffset** – byte address UINT32 **pValue** – pointer to variable for read value **Attention!** This function does not execute any conversion. It only provides 4 bytes ot the S7 variable with type TIME_OF_DAY.
DWORD WINAPI **ReadS7STRING( )**		Read a S7STRING from WinAC *Parameter:* DWORD **DBNummer** – data block to read DWORD **ByteOffset** – byte address BIT8 **Writemax** – Lenght of string in charachters

		CHAR **pValue** – address of string to read **Attention!** This function determines the number of bytes to transfer from the header of the S7 string on the WinAC side. Only the number of bytes is read, which is stored in the current lenght of the S7 string.
DWORD WINAPI **ReadS7STRING_LEN**		Read the length parameters of a S7-STRING from the WinAC (max. Length / current length) *Parameter:* DWORD **DBNummer** – data block to read DWORD **ByteOffset** – byte address BIT8 ***pMaxLen** – max. length of S7-STRING BIT8 ***pCurLen** – current length of S7-STRING
DWORD WINAPI **ReadS7BLOCK( )**		Read a S7BLOCK (array of byte) from WinAC *Parameter:* DWORD **DBNummer** – data block to read DWORD **ByteOffset** – byte address DWORD **Blocklength** – lenght of block to read in bytes BIT8 **pBlock** – address of the buffer for read data

5.4 Functions for Writing (application → WinAC)

NOTE

For all functions for the data block number either a data block is provided or a special code:

66000 – Input process image
66001 – Output process image
66002 – Flags (“Merker”)

DWORD WINAPI **WriteBOOL**( )		Write a BOOL value to WinAC *Parameter:* DWORD **DBNummer** – data block to write DWORD **ByteOffset** – byte address DWORD **BitOffset** – bit address BIT16 **Value** –variable with value to write
DWORD WINAPI **WriteBYTE**( )		Write a BYTE value to WinAC *Parameter:* DWORD **DBNummer** – data block to read DWORD **ByteOffset** – byte address BIT8 **Value** –variable to write
DWORD WINAPI **WriteWORD**( )		Write a WORD value to WinAC *Parameter:* DWORD **DBNummer** – data block to Write DWORD **ByteOffset** – byte address BIT16 **Value** –variable to write
DWORD WINAPI **WriteDWORD**( )		Write a DWORD value to WinAC *Parameter:* DWORD **DBNummer** – data block to Write DWORD **ByteOffset** – byte address BIT32 **Value** –variable to write
DWORD WINAPI **WriteINT**( )		Write a INT value to WinAC *Parameter:* DWORD **DBNummer** – data block to Write DWORD **ByteOffset** – byte address SINT16 **pValue** –variable to write
DWORD WINAPI **WriteDINT**( )		Write a DINT value to WinAC *Parameter:* DWORD **DBNummer** – data block to Write DWORD **ByteOffset** – byte address SINT32 **pValue** –variable to write

DWORD WINAPI **WriteREAL**( )		Write a REAL-Wert to the WinAC *Parameter:* DWORD **DBNummer** – data block to Write DWORD **ByteOffset** – byte address FLOAT **Value** –variable to write
DWORD WINAPI **WriteDATE**( )		Write a DATE value to the WinAC *Parameter:* DWORD **DBNummer** – data block to Write DWORD **ByteOffset** – byte address UINT16 **Value** –variable for write **Attention!** This function does not execute any conversion. It only copies 2 bytes to the S7 variable with type DATE.
DWORD WINAPI **WriteS5TIME**( )		Write a S5TIME value to theWinAC *Parameter:* DWORD **DBNummer** – data block to Write DWORD **ByteOffset** – byte address BIT16 **Value** –variable to write **Attention!** This function does not execute any conversion. It only copies 2 bytes to the S7 variable with type S5TIME.
DWORD WINAPI **WriteS7TIME**( )		Write a S7TIME value to the WinAC *Parameter:* DWORD **DBNummer** – data block to Write DWORD **ByteOffset** – byte address SINT32 **Value** –variable to write **Attention!** This function does not execute any conversion. It only copies 2 bytes to the S7 variable with type TIME.
DWORD WINAPI **WriteS7TIME_OF_DAY**		Write a S7TIME_OF_DAY value to the WinAC *Parameter:* DWORD **DBNummer** – data block to Write DWORD **ByteOffset** – byte address UINT32 **Value** –variable to write **Attention!** This function does not execute any conversion. It only copies 4 bytes to the S7 variable with type TIME_OF_DAY.
DWORD WINAPI **WriteS7STRING( )**		Write a S7STRING to the WinAC *Parameter:* DWORD **DBNummer** – data block to Write DWORD **ByteOffset** – byte address BIT8 **Writemax** – Lenght of string in charachters

		CHAR **pValue** – address of string to write **Attention!** This function first determines the maximum possible lenght of the S7-string out of the S7 string header. I.e. the value of the „maximum lenght“ in the S7 string header has to be set right before using this function! **Attention!** If the provided string is longer than the maximum length of the S7 string, only the maximum possible number of characters is transferred to the WinAC.
DWORD WINAPI **WriteS7BLOCK( )**		Write a S7BLOCK (array of byte) to the WinAC *Parameter:* DWORD **DBNummer** – data block to Write DWORD **ByteOffset** – byte address DWORD **Blocklength** – lenght of block to Write in bytes BIT8 **pBlock** – address of the buffer with write data

6 Installation

6.1 Quick-start

Run-time system

- Install the driver on the run-time system with the **setup.bat** (real-time DLL will be registered)
- Copy the application DLL (**WIcCli.dll**) to a location, the customer application will find it (e.g. same path as application or \system32\).

Engineering system (Step7 V5.5)

- Retrive and open the archived Step7 project with the Simatic Manager
- Copy the driver function block to your own project (FB10001, DB10001, SFB65001, SFB65002).
- Call the function FB_ACCESS_DB_INIT once (e.g. in OB 100)

Engineering system (TIA Portal)

- Unpack the archive with the TIA project
- Open the TIA project with separate instance of the TIA portal
- Copy the driver function block to your own project (FB10001, DB10001).
- Call the function FB_ACCESS_DB_INIT once (e.g. in OB 100)

6.2 Installation WinAC driver on runtime system

To install the WinAC driver only the registration of the real-time DLL is needed and the copying of the application DLL **WlcCli.dll** the System32 path of the Windows operating system. This is done by the BATCH file **setup.bat**.

6.2.1 Installation under Windows XP

When running under Windows XP(e) the batch file can be executed. This works from USB stick, too.

NOTE Please check the output of the batch file for possible errors.

6.2.2 Installation under Windows 7

The setup needs Administrator rights for correct operation.

Figure 6-1 Error when the setup is executed without Administrator rights

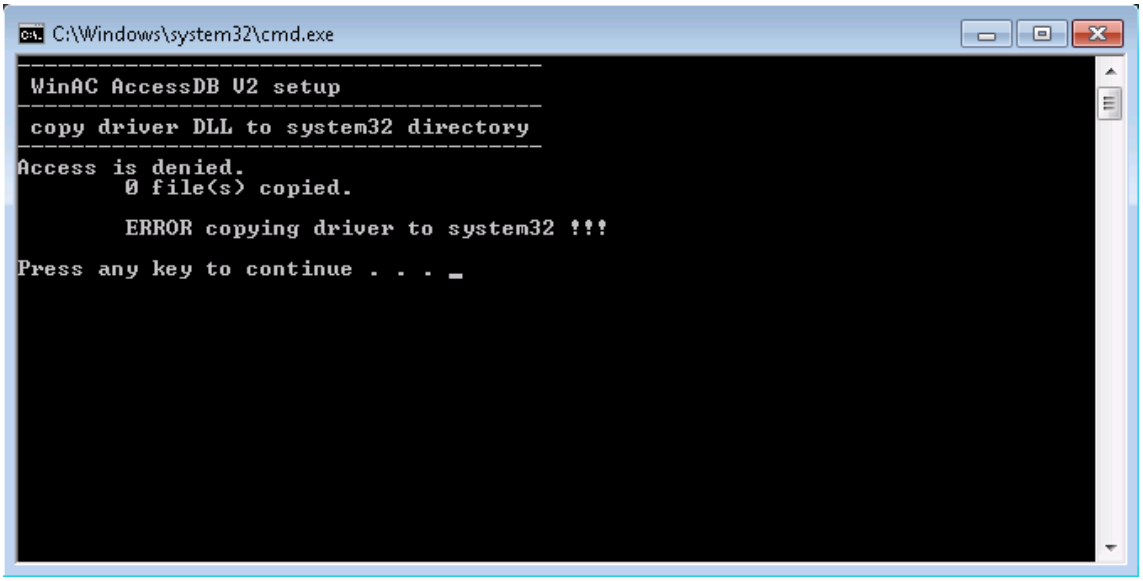
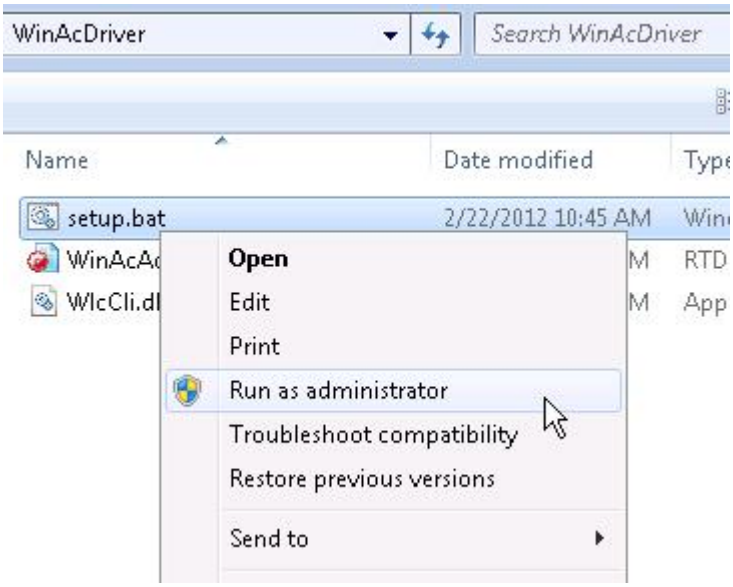


Figure 6-2 Execute setup.bat with Administrator rights



6.3 Installation WinAC driver on engineering system

On the engineering system these components are needed:

- Documentation
- Example project Step7

NOTE There is **no** installation needed of the WinAC FileServer driver on the engineering station.

6.3.1 Usage with Step7 Classic (V5.5)

Please copy the driver function block and the matching instance DB to your own project. Because the driver is based on the WinAC ODK, the ODK SFBs SFB65001 CREA_COM and SFB65002 EXEC_COM have to be copied to the Step7 project.

6.3.2 Usage with TIA Portal V11 SP2

When using the TIA portal only the FB FB_ACCESS_DB_INIT as well as the corresponding instance DB has to be copied.

NOTE The used SIMATIC CPU has to be a WinAC. In other case (e.g. unspecific CPU) there will be an error in the driver function block.

7 Use Cases of the Application

The driver package includes a Step7 example project plus example applications in different high level languages / engineering environments.

The functionality is the same for Step7 Classic (V5.5) and TIA Portal V11 SP2.

7.1 Provided Step7 example project

The provided Step7 program shows how to utilize the WinAC AccessDB V2 driver.

OB100 Complete Restart

In the restart function block the driver is initialized (**FB_INIT_ACCESS_DB**).

OB1 CYCL_EXC

No special action is needed for the driver in the main task. That's why the OB is empty in the example project.

FB10.001 – FB_INIT_ACCESS_DB

This is the only function block of the WinAC AccessDB Vs driver.

DB10.001 – iDB_INIT_ACCESS_DB

This is the instance data blocks of the driver's FB.

DB6 TEST_DATA

This data block contains various variables used by this demo project. These data is used by the high-level language demo programs

7.2 VB6 Example

The VB6 example implements a simple dialog based application. It shows how to use the data access to a WinAC by utilize the WinAC AccessDB V2 driver.

The module **ImportFunctions.bas** contains all functions of the driver DLL (WlcCli.dll). This module has to be included into the VB6 project to be able to use the functionality of the driver.

Initialization

The initialization of the driver is done by calling **InitAccessDB()** once. This is done in the function **Form_Load()** which is executed automatically when the application is started.

Access to WinAC data

After successful call of InitAccessDB() all read and write functions of the driver may be used. In this way content of data blocks, flags and process image may be accessed or modified.

Safe Mode

The functions for the “Safe Mode” are without any functionality. They are available in version V2 because of compatibility, only.

8 Error Codes

The driver **WinAC AccessDB V2** can provide different classes of error messages:

- Code in the FB-output **STATUS** according to WinAC-ODK (see chapter 8.1 in this document)
- Special error codes of the ODK driver (see chapter 8.2 on page 32 in this document)

8.1 Error codes of WinAC ODK

The driver had been developed with the WinAC ODK (Open Development Kit). The ODK can generate error codes, which are returned from the **STATUS** of the FBs.

Table 8-1 WinAC ODK error messages

ODK Code (HEX)	Description
0	Success
8001	An exception occurred.
8002	Input: the ANY pointer is invalid.
8003	Input: the ANY pointer range is invalid.
8004	Output: the ANY pointer is invalid.
8005	Output: the ANY pointer range is invalid.
8006	More bytes were written into the output buffer by the extension object than were allocated.
8007	ODK system has not been initialized: no previous call to SFB65001 (CREA_COM).
8008	The supplied handle value does not correspond to a valid extension object.
8009	More bytes were written into the input buffer by the extension object than were allocated.
807F	An internal error occurred.
80C3	Maximum number (32) of parallel jobs/instances exceeded.
8102	The call to CLSIDFromProgID failed.
8103	The call to CoInitializeEx failed.
8104	The call to CoCreateInstance failed.
8105	The library failed to load.
8106	A Windows response timeout occurred.
8107	Controller is in an invalid state for scheduling an OB.
8108	Schedule information for OB is invalid.
8109	Instance ID for SFB65001 call is invalid.
810A	Controller could not load proxy DLL.
810B	The WinAC controller could not create or initialize shared memory area.
810C	Attempt to access unavailable option ocurred.
8201	The Execute command index could not be found
8250	No more available positions in the job list

ODK Code (HEX)	Description
8252	The count is invalid
8253	A data type of an item in the list is invalid
8254	The count specified is invalid
8255	A memory area of an item in the list is invalid
8256	A DB number of an item in the list is invalid
8257	A bit number of an item in the list is invalid
8258	A pBuff of an item in the list is invalid
8259	A data quantity is invalid
825A	The area offset parameter is invalid for this type
825B	The frequency value is invalid
825C	The callback pointer is invalid
825D	The job ID pointer is invalid
825E	The job ID is invalid
825F	Job could not be completed because address is incorrect
8260	Job could not be completed because of protection level
8261	Job could not be completed because of hardware issues
8301	Invalid Thread Execution Priority
8401	Invalid Asynchronous Event
8402	Asynchronous Processor Queue is empty
8403	Asynchronous Processor Queue is full

8.2 Special error codes of the WinAC File Server

Among the general error bit of the driver FBs there is a special error code in the value of **STATUS** to describe the reason of the problem.

Table 8-2 Error codes of WinAC AccessDB V2

0 - no error

Errors with WinAC Handling
0x8501 - error using ODK_Read.. function
0x8502 - error using ODK_Write.. function
0x8503 - can't read init parameter (FB TINIT_AK)
0x8504 - false FB version (does not match with RTDLL version)

Reading / writing S7 strings
0x8531 - error reading STRING - invalid pointer to string
0x8532 - error reading STRING - error reading string len
0x8533 - error reading STRING - string is too large for the STEP 7 string
0x8534 - error reading STRING - string is too large for the output data buffer
0x8535 - error reading STRING - error writing current string len
0x8536 - error reading STRING - error writing max. string len
0x8537 - error reading STRING - error writing string to WinAC

IPC sync errors
0x8841 - error creating shared memory for IPC sync objects
0x8842 - error creating event to real-time side for IPC sync objects
0x8843 - error creating event to Windows side for IPC sync objects
0x8844 - IPC sync object memory undefined
0x8845 - error set event to real-time side
0x8846 - error waiting for command event from Windows side (abandoned)
0x8847 - error waiting for command event from Windows side
0x8848 - error set event to Windows side
0x8849 - error waiting for answer event from real-time side
0x884a - error waiting for answer event from real-time side (abandoned)
0x884b - timeout answer from real-time side - WinAC (driver) is in RUN
0x884c - timeout answer from real-time side - driver never started
0x884d - timeout answer from real-time side - driver still initialising
0x884e - timeout answer from real-time side - WinAC (driver) in STOP
0x884f - timeout answer from real-time side - WinAC (driver) shuts down
0x8850 - timeout waiting for answer event from real-time side - unkown driver state

Handling errors
0x8901 - got no data for WRITE command
0x8902 - data length 0 for WRITE command
0x8903 - got no return variable for READ command
0x8904 - no buffer for return data of READ
0x8905 - data length 0 for READ command
0x8906 - given buffer to small for read data
0x8907 - error reading S7 string - could not determine string length
0x8908 - received command count does not match expected
0x8909 - given DB number not valid
0x890a - IPC version Windows / real-time side does not match!

9 Related Literature

9.1 Bibliography

This list is not complete and only represents a selection of relevant literature.

Table 9-1

	Subject	Title
/1/	STEP7	Automation with STEP7 in STL and SCL Hans Berger Publisher: Vch Pub ISBN-10 3895783412 ISBN-13 9783895783418
/2/	WinAC	Windows Automation Center RTX – WinAC RTX 2010 Operating Instructions
/3/		

9.2 Internet Link Specifications

This list is not complete and only represents a selection of relevant information.

Table 9-2

	Subject	Title
\1\	Reference to the entry	http://support.automation.siemens.com/WW/view/en/EntryID
\2\	Siemens I IA/DT Customer Support	http://support.automation.siemens.com
\3\	WinAC Operating Instructions	http://support.automation.siemens.com/WW/view/en/43715176
\4\		

10 History

Table 10-1 Version History

Version	Date	Modifications
V2.01	29.02.11	First version of the English documentation