

AK Handbook



Author: Dr Dmitry A.Kazakov

The 8th edition, July 2009

AK Handbook

The eighth edition

Dmitry A. Kazakov

© Copyright [cbb software GmbH](http://www.cbb-software.com)

1. Preface

The *AK* is a 32 bit MS-Windows[®] multithreading application and dynamic link library supporting the [AK protocol](#). It provides an implementation of AK client to be used for communication with AK end devices. The software supports up to 16 simultaneous connections over RS-232 or TCP/IP. The client application is relieved of the burden of polling the AK devices. When necessary the software can be configured to poll required AK commands in background.

2. Installation and Components

The *AK* software consists of two parts [AK.exe](#), [AKdll.dll](#). They should be placed into the Windows home directory. The [AK.exe](#) provides dialogue based [user's interface](#). [AKdll.dll](#) is a dynamic link library that supports [programmer's interface](#). Its export symbol table is located in [AKdll.lib](#). This file is necessary in only rare cases when an application requires the symbol table (like LabWindows/CVI[®] by [National Instruments](#)). [AK.exe](#) may be started either manually by invoking [AK.exe](#) or automatically by starting an application that uses [AKdll.dll](#).

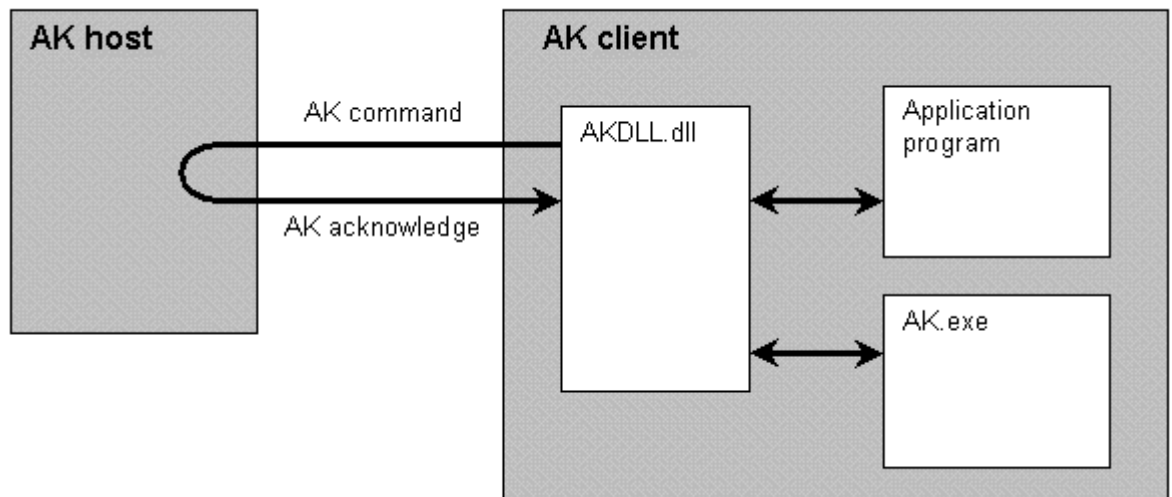


Figure 1. The software structure

As the transport layer for the AK communication may serve either a COM port or TCP/IP stream socket.

3. User's Interface

3.1 Main panel

[AK.exe](#) always starts minimized. If one opens its main panel it looks as shown on the fig.2.

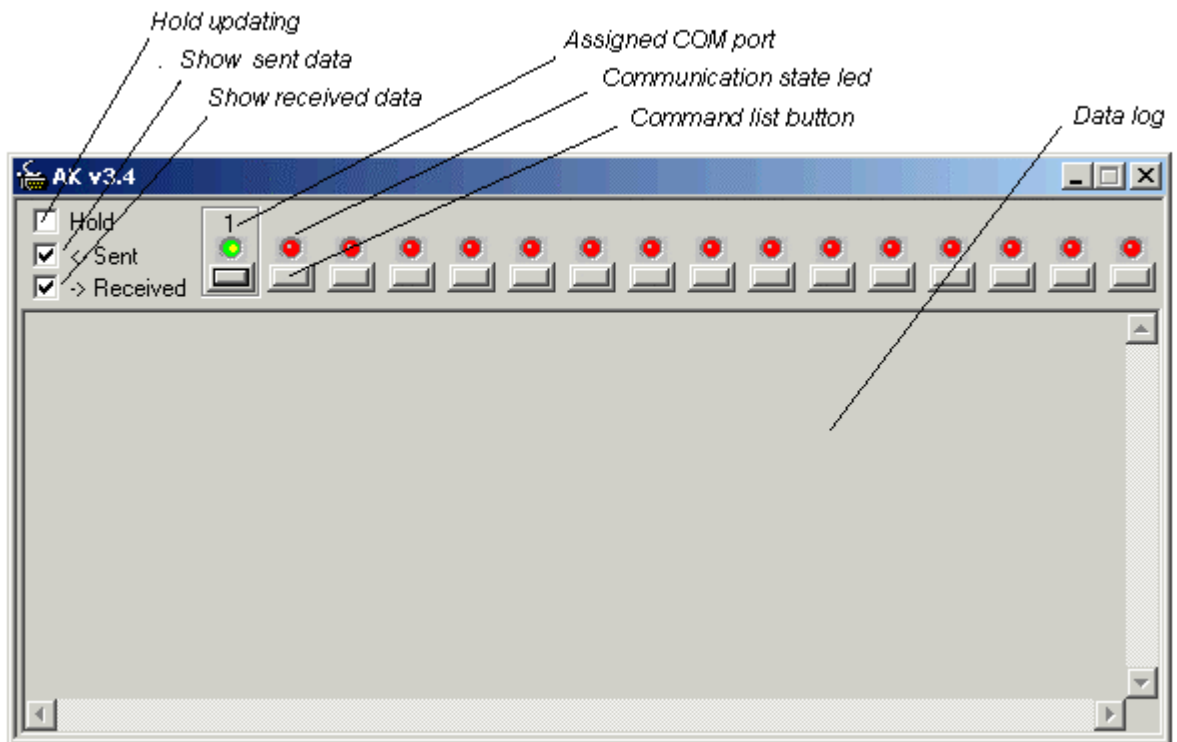


Figure 2. The main panel

The main panel shows the state of the currently selected communication port. Up to 8 ports are supported. To choose a communication port one can click on the corresponding "Command list button". The communication data log is shown in the "Data log" field. Sent data are shown using <- prefix. Received data have >- prefix. One can filter desired data flow direction using check buttons "Show sent data" and "Show received data". The "Hold updating" check button allows to stop actualizing the "Data log" field. The "Communication state led" indicates the integral state of the communication port. Green color indicates a normal state. Yellow means that it was impossible to reach the specified polling frequency for some of automatically sent commands (for more information see below). Red indicates an error.

3.2. Command list panel

The panel appears by pressing a "Command list button".

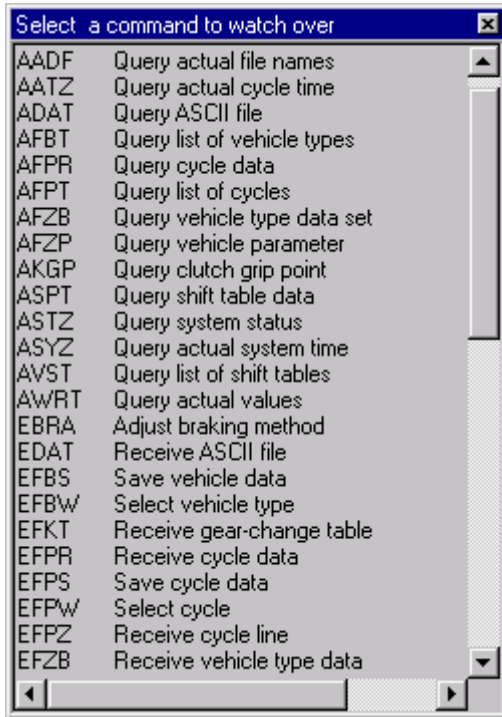


Figure 3. The command list panel (example)

The panel indicates the list of AK commands declared for the communication port. One can choose an AK command from the list by left mouse click. This panel is empty if no commands declared for the port.

3.3. Watch panel

The panel is opened when an AK command is selected. It is possible to have several watch panels to spy over more than one AK commands. The watch panel is shown on the fig. 4.

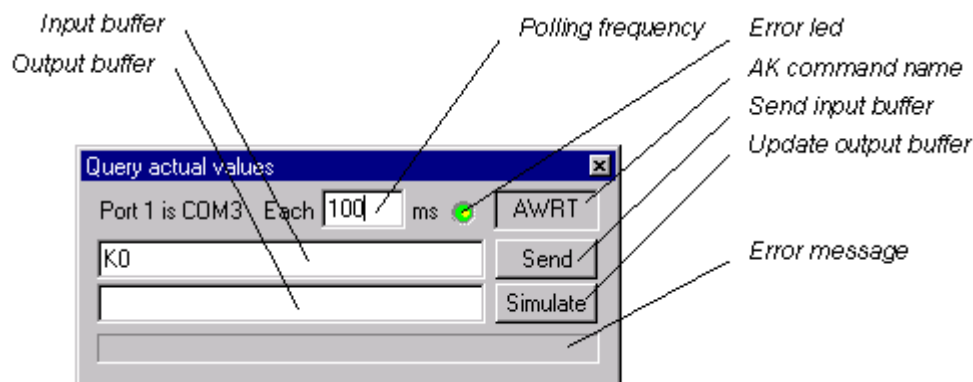


Figure 4. The watch panel

The "Input buffer" field contains the byte sequence to be sent over the communication port. The AK protocol prologue and epilogue fields are not shown. They are added automatically.

The "Send" button is used to begin a send operation. The "Output buffer" field contains the data field of the last received AK command. The buffer does not contain control fields of the AK telegram. One can simulate receiving of the AK command by editing the field and pressing "Simulate" button. The "Error led" shows the command execution state. Green color means successful completion, i.e. that the command was sent and the answer was received. Yellow indicates that I/O is in progress. Red indicates an error. The "Error message" field shows the error message. The panel's menu has the item "Undeclare" to delete the AK command.

3.4. AK command declaration

The AK command declaration panel is popped using the menu item "Declare" of the [main panel](#). It looks like:

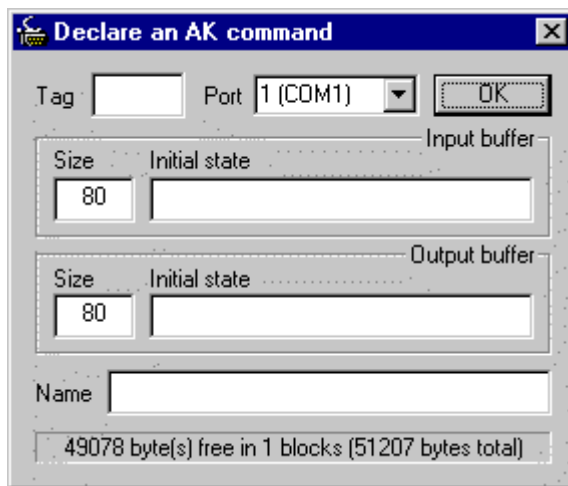


Figure 5. The AK command declaration panel

4. Programmer's interface

The *AK* software operates in full duplex mode. It means that it is able to read and write data from/to the AK host in parallel. Although it allows in most cases to boost performance and avoid deadlocks, it may also cause malfunction of some poorly developed AK devices. If you encounter a situation when an AK device function not as expected, ensure that your application do not send a new command before receiving an acknowledge to the previous one.

[AKdll.dll](#) exports the following functions:

AKCancel	AKGetOutputSize	AKSend
AKCanExit	AKGetPolicy	AKSetCallBack
AKDeclare	AKGetPortState	AKSetData
AKFirstIndex	AKGetRate	AKSetDefaultTimeOut
AKGetData	AKGetState	AKSetHandler
AKGetDefaultTimeOut	AKGetStatePanel	AKSetIndicator
AKGetInput	AKGetTag	AKSetInput
AKGetInputSize	AKGetTimeOut	AKSetOutput
AKGetName	AKIndex	AKSetPolicy
AKGetOutput	AKNextIndex	AKSetTimeOut

All of them have C calling convention. If other is not stated the return value is an unsigned integer containing the status bits set according the result of the function execution. See description of [AKGetState](#) for the bits definition. When used with:

- **Ada 95/2005.** Use **with** clause for the AK package. The package is defined in the *ak.ads* file (the package has no body). Make sure that your compiler has Win32[®] bindings installed and that the C interface package is compatible with Microsoft Visual C++ (i.e. supports C calling convention). The [AKdll.lib](#) must be used when the application is linked;
- **C/C++.** Include the *ak.h* header file, link the application with the [AKdll.lib](#). Make sure that you are using the multithreaded C run-time library in the form of a DLL;
- **Visual Basic.** Insert the *ak.bas* module in your project;

The following example highlights use of AK. Here we declare SMAN command ([AKDeclare](#)), then set timeout to 1 sec ([AKSetTimeOut](#)), then send the command to the AK host (see [AKSend](#)):

Ada 95/2005:

```
with AK, Interfaces, Interfaces.C;
use AK, Interfaces, Interfaces.C;

procedure Test is
  Result : unsigned;
begin
  Result := AKDeclare (1, "SMAN", 0, 10, 12, " " & Nul, 2,
"K0" & Nul, "Set manual mode" & Nul);
  Result := AKSetTimeOut (AKIndex (1, "SMAN"), 10000);
  Result := AKSend (AKIndex (1, "SMAN"));
end Test;
```

C/C++:

```
#include "ak.h"

int main (int, char * [])
{
    AKDeclare (1, "SMAN", 0, 10, 12, " ", 2, "K0", "Set manual
mode");
    AKSetTimeout (AKIndex (1, "SMAN"), 10000);
    AKSend (AKIndex (1, "SMAN"));
}
```

Visual Basic:

```
Sub Main()
    Call AKDeclare (1, "SMAN", 0, 10, 12, " ", 2, "K0", "Set
manual mode");
    Call AKSetTimeout (AKIndex (1, "SMAN"), 10000);
    Call AKSend (AKIndex (1, "SMAN"));
End Sub
```

4.1. AKCancel

```
unsigned AKCancel (int Index);
```

This function cancels the AK command specified by the parameter **Index**. If the command is not in progress nothing is made. Otherwise the command is aborted (with the 8D₁₆ [state code](#)).

4.2. AKCanExit

```
char AKCanExit (void);
```

This function checks if the [AK.exe](#) can be terminated. If the function returns 1, it means that there is no other applications using [AKdll.dll](#).

4.3. AKDeclare

```
unsigned AKDeclare
(
    unsigned      PortNo,
    const char    Tag [4],
    unsigned      Policy,
    unsigned      Timeout,
    unsigned      InBufSize,
    const char *  InBufInit,
    unsigned      OutBufSize,
    const char *  OutBufInit,
    const char *  Name
);
```

This function declares an AK command. Each AK command must be declared using this function. The function has the following parameters:

PortNo	The communication port used for the command. Value 1 corresponds to the first port shown on the main panel. Note that the port 1 can be assigned to any of COM hardware ports or a TCP/IP socket. See registry key configuration for more information.
Tag	A four character name of the AK command (SMAN, for instance). This name must be unique for the port.
Policy	It indicates how frequently (in milliseconds) the command must be sent. Zero value indicates that the command will be sent on request only. Otherwise, the AK.exe automatically sends the output buffer contents each Policy ms.
Timeout	Timeout in ms. If specified as 0, the default value is used.
InBufSize	The maximal size of the input buffer in bytes. The input buffer does not include AK protocol prologue and epilogue. The input buffer is enlarged automatically when necessary.
InBufInit	The pointer to a null terminated string to initialize the input buffer.
OutBufSize	The maximal size of the output buffer in bytes. The output buffer does not include AK protocol prologue and epilogue. The output buffer is enlarged automatically when necessary.
OutBufInit	The pointer to a null-terminated string used to initialize the output buffer.
Name	The pointer to a null-terminated string containing the AK command mnemonic name.

4.4. AKFirstIndex & AKNextIndex

```
int AKFirstIndex (unsigned PortNo);
int AKNextIndex (int Index);
```

[AKFirstIndex](#) returns the index of the first AK command declared for the port **PortNo**. The parameter **PortNo** is the communication port used for the command. Value 1 corresponds to the first port shown on the main panel. Note that the port 1 can be assigned to any of COM hardware ports or a TCP/IP socket. See [registry key configuration](#) for more information.

[AKNextIndex](#) delivers the [AK index](#) of the next command. Both functions return a positive value on success. They can be used to build an AK command iterator. For instance:

```
int Index; // AK index

for ( Index = AKFirstIndex (PortNo); //
Initialize
      Index >= 0; // End
condition
      Index = AKNextIndex (Index) // Get next
declared
      )
{ // For each declared AK command
  ...
}
```

Warning: An [AK index](#) is no longer valid if either [AKDeclare](#) or [AKUndeclare](#) was successfully called for the given port.

4.5. AKGetData

```
unsigned AKGetData (int Index, unsigned * Data);
```

One can associate a 32-bit data field to each declared AK command. The data is not interpreted by the DLL. The function copies the data field contents into the memory cell pointed by the parameter **Data**. The parameter **Index** specifies the AK command. See also [AKSetData](#) function that sets the data field.

4.6. AKGetDefaultTimeOut

```
unsigned AKGetDefaultTimeOut (unsigned * TimeOut);
```

This function stores the default time out latency (in ms) into the variable pointed by the parameter **TimeOut**. See also [AKSetDefaultTimeOut](#).

4.7. AKGetInput

```
unsigned AKGetInput  
( int Index,  
  char * String,  
  unsigned Size,  
  int Wait  
);
```

This function copies the input buffer contents. The parameter **Index** is the [AK index](#) of the command. The parameter **String** points to the buffer to receive the data field of acknowledge telegram (see [AK protocol](#)). Its size is specified by the parameter **Size**. The flag **Wait** indicates whenever the function should wait for the input buffer release. If the flag is not set [AKGetInput](#) immediately returns with the 8C₁₆ error code, if the input buffer is in use (because the AK command is being sent).

4.8. AKGetInputSize

```
unsigned AKGetInputSize (int Index, unsigned * Size);
```

This function gets the maximal size of the input buffer. The parameter **Index** is the [AK index](#) of the command. The parameter **Size** points to the variable where the result value will be stored.

4.9. AKGetName

```
unsigned AKGetName (int Index, char * Name, int Size);
```

This function gets the mnemonic name of the AK command. The parameter **Index** is the [AK index](#) of the command. The parameter **Name** is the buffer to accept the command name. The command name is stored as a NUL-terminated string. The parameter **Size** is the buffer size.

4.10. AKGetOutput

```
unsigned AKGetOutput
(
    int      Index,
    char *   String,
    unsigned Size,
    int      Wait
);
```

This function copies the output buffer contents. The parameter **Index** is the [AK index](#) of the command. The parameter **String** points to the buffer to receive the data. Its size is specified by the parameter **Size**. The flag **Wait** indicates whenever the function should wait for the output buffer release. If the flag is not set [AKGetOutput](#) immediately returns with the 8C₁₆ [error code](#), if the output buffer is in use (because the AK command is being received).

4.11. AKGetOutputSize

```
unsigned AKGetOutputSize (int Index, unsigned * Size);
```

This function gets the maximal size of the output buffer. The parameter **Index** is the [AK index](#) of the command. The parameter **Size** points to the variable where the result value will be stored.

4.12. AKGetPolicy

```
unsigned AKGetPolicy (int Index, unsigned * Policy);
```

This function gets the frequency used to poll the AK command. The parameter **Index** is the [AK index](#) of the command. The **Policy** points to the variable where the result value (in ms) will be stored.

4.13. AKGetPortState

```
char AKGetPortState (unsigned PortNo);
```

This command returns the integral state of the communication port associated with **PortNo**:

0	No error
1	Stalled. The polling frequency set for an AK command is unreachable
2	Error

4.14. AKGetRate

```
unsigned AKGetRate (int Index, unsigned * SendDelay, unsigned * ReceiveDelay);
```

This function returns the timings of the last answered command. The parameter **Index** is the [AK index](#) of the command. Two delays in ms are returned via parameters **SendDelay** and **ReceiveDelay**. The following figure shows how the delays are defined:

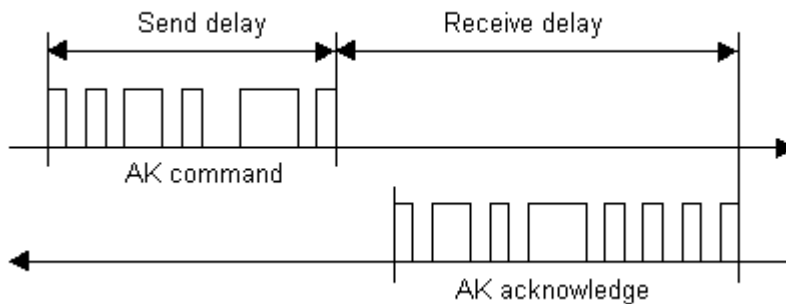


Figure 5. Timings

4.15. AKGetState

```
unsigned AKGetState
(
    int Index,
    unsigned * State,
    unsigned Mask
);
```

This function retrieves the state of an AK command. The parameter **Index** is the [AK index](#) of the command. The parameter **State** points to the variable where the status will be stored. The parameter **Mask** specifies which state bits must be cleaned after the function call. It can be a combination of 1000₁₆ and 2000₁₆ bits. The following table describes the defined state bits:

Bits	Description
000000FF ₁₆	The AK protocol error byte copied from the last incoming AK telegram. Values 128..255 are reserved for error codes generated by AK.exe
00000080 ₁₆	Buffer overflow. This error code is set when input or output buffer is too small to accept the data and the buffer cannot be enlarged
00000081 ₁₆	Timed out
00000082 ₁₆	Not declared. An unknown AK command tag specified for the AKIndex command
00000083 ₁₆	Re-declaration. The AK command is already declared
00000084 ₁₆	Out of memory. Not enough memory to declare an AK command
00000085 ₁₆	Table overflow. Too many commands declared
00000086 ₁₆	Wrong AK index

00000087 ₁₆	Wrong communication port number
00000088 ₁₆	Wrong handler. The entry point is not found in the executable or library file, or the file does not exist
00000089 ₁₆	Syntax error
0000008A ₁₆	Function failed
0000008B ₁₆	Busy
0000008C ₁₆	Not ready. A buffer is currently in use
0000008D ₁₆	Canceled
00000100 ₁₆	The input buffer is currently locked
00000200 ₁₆	The output buffer is currently locked
00000400 ₁₆	The I/O is in progress
00000800 ₁₆	Error. The error byte indicates the reason
00001000 ₁₆	State change flag (used by AK.exe)
00002000 ₁₆	State change flag (free). This bit is set each time an answer comes from the AK host or the command is timed out. AKGetState can reset the flag by specifying it in the parameter Mask . This way one can poll the state of an AK command
00004000 ₁₆	Cancel is in progress (for send)
00008000 ₁₆	Cancel is in progress (for receive)

4.16. AKGetStatePanel

HWND [AKGetStatePanel](#) (unsigned **PortNo**);

This command returns the Windows handle of the state panel associated **PortNo**. The state panel handle is one returned by the handler function set by [AKSetIndicator](#) function. Zero is returned if no panel exists.

4.17. AKGetTag

unsigned [AKGetTag](#) (int **Index**, char **Tag** [4]);

This function gets the name of an AK command. The parameter **Index** is the [AK index](#) of the command. The parameter **Tag** is the four byte character array where the name is stored.

4.18. AKGetTimeOut

unsigned [AKGetTimeOut](#) (int **Index**, unsigned * **Timeout**);

This function gets the time out (in ms) used for the AK command. The parameter **Index** is the [AK index](#) of the command. The parameter **Timeout** points to the variable where the result must be stored.

4.19. AKIndex

```
int AKIndex (unsigned PortNo, char Tag [4]);
```

This function returns the index of an AK command. All commands are referenced by their port number and name. It requires a table search each time one needs to specify a command. To speed up this procedure, the [AK index](#) is provided. It allows to avoid the table search. [AKIndex](#) constructs the [AK index](#) from the virtual port number provided by the parameter **PortNo** and the command name specified by the parameter **Tag**. The result may be used in any function that requires [AK index](#). A correct index is always positive.

4.20. AKSend

```
int AKSend (int Index);
```

This function sends an AK command. The parameter **Index** is the [AK index](#) of the command. The current contents of the output buffer associated with the command is sent to the AK host. See also [AKSetOutput](#).

4.21. AKSetCallback

```
unsigned AKSetCallback  
(  
    unsigned PortNo,  
    AKCallbackEntryPoint * Callback  
);
```

This function sets a callback function (the parameter **Callback**) for all incoming messages for the specified virtual port (the parameter **PortNo**). Each time when an AK command is accepted and recognized, the function is called. The function has the AKCallbackEntryPoint type. It receives the single parameter, that is the [AK index](#) of the accepted AK command. The callback function is called on the context of the current process. As a C function, it should have the following parameter profile:

```
void AKCallbackEntryPoint (int Index);
```

See also [AKSetHandler](#).

4.22. AKSetData

```
unsigned AKSetData (int Index, unsigned Data);
```

This function associates a 32-bit data field with the specified AK command. The parameter **Index** is the [AK index](#) of the command. The parameter **Data** contains the data. The data are not interpreted; they are saved as-is. The [AKGetData](#) function can be used to retrieve data associated with a command.

4.23. AKSetDefaultTimeOut

```
unsigned AKSetDefaultTimeOut (unsigned TimeOut);
```

This function sets the default maximal command execution latency. The value is provided by the parameter **TimeOut** in milliseconds. The value is used for all AK commands except for ones having explicitly set value (see [AKSetTimeOut](#)).

4.24. AKSetHandler

```
unsigned AKSetHandler
(
    unsigned      PortNo,
    const char * File,
    const char * Name
);
```

This function sets a handler for all incoming messages for the specified port. The parameter **PortNo** is the virtual port used for the command. Value 1 corresponds to the first port shown on the main panel. Note that the port 1 can be assigned to any of COM hardware ports or a TCP/IP socket. See [registry key configuration](#) for more information. The parameter **File** points to the name (as a null-terminated string) of a file that contains the handler code. It can be the name of a binary (*exe*) or dynamic library (*dll*) file. The parameter **Name** points to the external name of the handler code. The handler must conform *stdcall* call convention. As a C function it should have the following parameter profile:

```
void HandlerName
(
    unsigned      PortNo,
    char          Tag [4],
    char *       Buffer,
    int          MaxSize,
    unsigned *   State
);
```

The handler is called by the [AK.exe](#) each time an AK telegram comes from the AK port. The parameter **PortNo** is the virtual port number. The **Tag** is the AK command name. The parameter **Buffer** points to the input buffer associated with the command. The handler can observe the buffer and change its contents. The parameter **MaxSize** is the size input buffer. If the handler changes the input buffer contents its total size including the terminating null shall not exceed this value. The parameter **State** points to the state field associated with the AK command. The handler can set or clean desired state bits (see [AKGetState](#)).

Notes:

- The handler is called on the context of the [AK.exe](#).
- Use of any other functions of [AKdll.dll](#) from the handler is prohibited (it will lead to hanging the [AK.exe](#) and [AKdll.dll](#)).
- The communication over the port is completely blocked till return from the handler. It means that the handler code should be as short as possible.
- If it is necessary to open some additional windows to indicate data, specific to the port, it is better to do it using the handler set by the [AKSetIndicator](#) function.
- The 88₁₆ [status](#) bit is set when the handler cannot be mapped.

- The parameter **File** can be 0 to indicate that the currently mapped handler should be unmapped without setting a new one. The parameter **Name** is ignored in this case.
- See also [AKSetCallBack](#).

4.25. AKSetIndicator

```
unsigned AKSetIndicator
(
    unsigned    PortNo,
    const char * File,
    const char * Name
);
```

This function sets an indicator handler for the specified port. The parameter **PortNo** is the virtual port number. [AK.exe](#) periodically calls the indicator handler function to highlight the communication state. Value 1 corresponds to the first port shown on the main panel. Note that the port 1 can be assigned to any of COM hardware ports. See [registry key configuration](#) for more information. The parameter **File** points to the name (as a null-terminated string) of a file that contains the handler. It can be the name of a binary (*exe*) or dynamic library (*dll*) file. The parameter **Name** points to the external name of the handler. The handler must conform *stdcall* call convention. As a C function it would have the following parameter profile:

```
HWND HandlerName
(
    unsigned    PortNo,
    HINSTANCE   Instance,
    HWND        Parent
);
```

The handler is called by the [AK.exe](#) each 0.5 second. The parameter **PortNo** is the number of the virtual port number used for communication with the host. The parameter **Instance** is the handle of the instance of the file where handler code belongs to. One can use it to load necessary Windows resources, like dialogue templates and etc. The parameter **Parent** is the handle of the [AK.exe](#) window. One can use it as the parent window for the window created by the handler. The handle of the window created by the handler must be returned as the result.

Note:

- The handler is called on the context of the [AK.exe](#).
- One can use any other functions from [AKdll.dll](#).
- The 88₁₆ [status](#) bit is set when the handler cannot be mapped.
- The parameter **File** can be 0 to indicate that the currently mapped handler should be unmapped without setting a new one. The parameter **Name** is ignored in this case.

4.26. AKSetInput

```
unsigned AKSetInput
(
    int    Index,
    char * String,
    int    Wait
);
```

This function copies the input buffer contents. It works as if the an AK command would be received with the data field containing the buffer contents. The parameter **Index** is the [AK index](#) of the command. The parameter **String** points to the buffer to containing the data as a null-terminated string. The flag **Wait** indicates whenever the function should wait for the input buffer release. If the flag is not set [AKSetInput](#) immediately returns with the 8C₁₆ [error code](#), if the input buffer is in use (because the AK command is being received).

4.27. AKSetOutput

```
unsigned AKSetOutput
(
    int    Index,
    char * String,
    int    Wait
);
```

This function copies the output buffer contents and sends AK command containing the new buffer as the channel number and data field (in the most short form it is "K0" - a command without parameters for the first channel, see [AK protocol](#) for more information). The parameter **Index** is the [AK index](#) of the command. The parameter **String** points to the buffer to containing the data as a null-terminated string. The flag **Wait** indicates whenever the function should wait for the input buffer release. If the flag is not set [AKSetOutput](#) immediately returns with the 8C₁₆ [error code](#), if the output buffer is in use (because the AK command is being sent).

4.28. AKSetPolicy

```
unsigned AKSetPolicy (int Index, unsigned Policy);
```

This function sets the frequency used to poll the AK command. The parameter **Index** is the [AK index](#) of the command. The parameter **Policy** is the frequency in milliseconds. Zero value stops polling.

4.29. AKSetTimeOut

```
unsigned AKSetTimeOut (int Index, unsigned TimeOut);
```

This function sets the maximal command execution latency. The parameter **Index** is the [AK index](#) of the command. The value is provided by the parameter **TimeOut** in milliseconds. Zero value indicates that the value set by [AKSetDefaultTimeOut](#) must be used.

4.30. AKUndeclare

```
unsigned AKUndeclare (int Index);
```

This function undeclares an AK command. The parameter **Index** is the [AK index](#) of the command. This function also cancels all I/O induced by the removed AK command.

5. Configuration

The configuration data for the are stored in the windows registry under *HKEY_LOCAL_MACHINE / Software / Cbb-automation / AK*. The following table describes the most important keys:

Key name	Value	Description
DefaultTimeout	15000	The default value in ms
PortN	string	<ul style="list-style-type: none"> When the value is a number, it is treated as the hardware COM port number associated with the port <i>N</i>. For instance, Port1=3 means that the first port shown on the main panel is COM3. Each specified by this manner COM port is configured according to the settings of the Windows <i>ControlPanel / Ports</i> under Windows NT or <i>ControlPanel / System / Device Manager / Ports</i> under Windows 95. These settings Windows NT saves in the registry under <i>HKEY_LOCAL_MACHINE / Software / Microsoft / WindowsNT / CurrentVersion / Ports</i>. Windows 95 saves port settings in the win.ini file under section [Ports]. Otherwise the value must have the following syntax: <Name> : <Port>. Here <Name> is the IP address or name of the AK host. <Port> is the TCP port used listened by the host.
PortN.Add CR/LF	0	When non-zero, CR/LF is added to the end of each telegram being sent.
PortN.Begin with CR	0	When non-zero, CR is added in front of each telegram being sent
PortN.Start	2	The first character of a telegram. The default is STX (ASCII).
PortN.DontCare	32	The second character of a telegram. The default is space (ASCII).
PortN.Stop	3	The last character of a telegram. The default is ETX (ASCII).
PortN.Ignore error code	0	When non-zero, the generic error code in AK acknowledge telegram is ignored (treated as if it were '0'). Usually the generic error code is used by AK devices to indicate severe errors. Although ome devices use it rather as an additional information field. For such devices ignoring the error code might useful.
AutoExit	0	Being set to 1 this parameter causes AK.exe to exit if no application communicates with the AKdll.dll . By default AK.exe remains active since there is no necessity to restart it when an application starts.
Enable	0	This parameter if specified as 1 allows to unlock the main panel. Unlike other parameters this one is read each time a

		user tries to unlock the panel.
StartTimeout	6000	The maximal time (ms) to wait for AK.exe start. Before a prompt appears.
TCP.RecoveryTimeout	2000	The delay (ms) before a next attempt to restore a TCP/IP connection is made.

6. AK protocol

[AK protocol](#) is asymmetric. AK host works as a server that answers to clients. It accepts *command* telegrams and answers with *acknowledge* telegrams. For each command telegram AK host answers with exactly one acknowledge telegram. It does not send anything to the client without a request. AK telegrams have the following format:

<i>AK command telegram format</i>		
Byte	Contains	Description
1	STX (ASCII 02)	This byte indicates start of a new telegram
2	Ignored	This byte is ignored, but it must be a printable character
3..6	Function code	The function code is a four byte identifier. It must contain only printable characters
7	SP (space, ASCII 32)	Separator
8..9	Channel number	The channel number. Usually this field contains Kn , where n is the number of the channel in ASCII format. It may contain more than one digit. If only one channel is supported it is K0 (ASCII 75,48). Note that some AK devices do not support the channel number, though it is not AK protocol conform.
10	SP (space, ASCII 32)	Separator. It can be omitted if no data field present
11..n	Data field	It is AK host specific. It can be any sequence of printable characters. The data field may be omitted
n+1	ETX (ASCII 03)	End byte. After the end byte CR (ASCII 13) may follow

<i>AK acknowledge telegram format</i>		
Byte	Contains	Description
1	STX (ASCII 02)	This byte indicates start of a new telegram
2	Ignored	This byte is ignored, but it must be a printable character
3..6	Function code	The function code (echoed from the command telegram)
7	SP (space, ASCII 32)	Separator
8	General error code	The error code is a digit 0..9 (in ASCII format). Zero (ASCII 48) indicates no error
9	SP (space, ASCII 32)	Separator. It can be omitted if no data field present
10..n	Data field	It is AK host specific. It can be any sequence of printable characters.

		The data field may be empty
<i>n</i> +1	ETX (ASCII 03)	End byte. After the end byte CR (ASCII 13) may follow.

Example. Note that any example is hypothetical, because the function codes and data fields are host specific.

Command: **STX SP 'S' 'M' 'A' 'N' SP 'K' '0' ETX**

Acknowledge: **STX SP 'S' 'M' 'A' 'N' SP '0' ETX**