POSITIONING AND CONTOURING CONTROL SYSTEM

APCI-8001 and APCI-8008

Scanner Interface

Last updated: 17/03/2020, from CD-ROM V2.53VM
Rev. 12/052022

www.addi-data.com

1 Introduction

The scanner functionality of the APCI-8001 / APCI-8008 can be used to scan and temporarily store process data in real-time. In doing this, the process data is stored cyclically in a scan record. These records can be read out and processed as record arrays. The resource interface is necessary to use this functionality. It is described in the manual ”Resource Interface”.
This option can be used only if there is operating system version RwMos.ELF with the options *optionSCANNER* and *optionRESOURCE*.

2 Using the scanner functions

2.1 Boards already implemented

APCI-3120: 16 analog inputs, 8 analog outputs
APCI-3701: 16 inductive transducers
APCI-3003: 4 analog inputs, simultaneous acquisition
APCI-3501: 8 analog outputs

2.2 Initialising the scanner

The following values for the universal object interface must be used when using the scanner module:

Table 1: Object descriptor elements

Object descriptor element	Value
Handle	Must be initialised with 0 when starting the application or after rebooting the control system, and is then managed/used by the system. **For PCAP programming:** After the scanner functionality is cleaned, the handles for all elements must be reset to zero using the rdwr functionality.
BusNumber	1100
DeviceNumber	0
Index	0, 1, ... Function number for configuring/operating the scanner, according to table 2.
SubIndex	No function

When the DeviceNumber > 0 and the index is 1, the scan objects are declared. The DeviceNumber must be assigned consecutively.
The parameters to be written are returned as second parameters (value) by calling the function wrOptionInt or assigned directly at SAP programming.
For more information on the object descriptor elements, see the manual "Universal Object Interface".

2.3 Functions of the scanner module

Table 2: Functions of the scanner module for device No. 0

No. (index)	**Name**	**Type**	**Explanation**	**Return parameter (value)**
1	CLEAN	integer w	Reset scanner The value 1 must be returned. **For PCAP programming:** Immediately after the reset, the handles for all objects used must be reset to zero using the rdwr functionality.	1
2	INIT	integer w	Initialise scanner before start-up. This means, for example, that the SizeOfRecord is calculated and the data buffer is emptied. **Caution:** By this calling also the variable HW_SCAN_STROBE is reset.	1
3	STARTSTOP	integer r/w	Start or stop scanner, or request status.	1 = Start 0 = Stop
4	STATUS	integer r	Read the status of the scanner The return value of this function is described for the rdScannerStatus function.	
5	SIZEBUFFER	integer r	Request size of the total memory in the scan buffer (in bytes). Default: 100,000 bytes This value can be set using the SZSCANBUFFER environment variable in fwsetup.	
6	TIMEFACTOR	integer r/w	Read/write time factor in scanning time for scanning data Default value: 1	1, 2, ...
7	RECORDSTO SCAN	integer w	Read/write number of records that should be scanned. If 0 is entered here, scanning will be endless. If more records are to be scanned than fit into the data buffer, the scanned data must be read out during the scan process. Default value: 1	0,1, ...
8	RECORDS SCANNED	integer r	Request number of scanned records. By calling the function ScannerInit the value is reset to 0.	
9	SIZEOF RECORD	integer r	Request size of the record to be scanned (in bytes). This value is only available after calling the INIT function.	
10	CHECK BUFFER	integer r	Request size of the free memory in the scan buffer (in bytes).	

No. (index)	Name	Type	Explanation	Return parameter (value)
11	HW_SCAN_ STROBE	integer r/w	By setting of one or several bits in this register, fast hardware inputs can be defined as strobe inputs for the latch process. RWMOS.ELF must have the respecting options that this option can be used. Caution: This variable is reset by calling INIT.	bits 0..7
22	FREE BUFFER	integer w	Internal function for the memory administration, it is not considered for the user.	Memory to be released in bytes
64	SYNCPULSE OUT	integer r/w	Only available with special hardware version: This register allows for a scanner-synchronous pulse output to trigger external components (see Chapter 2.5)	bit-coded axis specification for pulse output

2.4 Scan control

By default, the scan is realised in a time-controlled and sampling-synchronous manner. However, it is also possible to realise the scan in an event-controlled way. For this, as the last scan element, the resource WTLSTRB (#101) is defined for a defined axis. In this case, the scan is recorded when a latch-strobe-signal of the respecting axis has been recognised. Also here, the scan is realised sampling-synchronously. The parameter TIMEFACTOR should always be set to 1.
The latch pulses must not be faster than the sampling times. The latch-strobe-signal is always reset during the recording of the scan record.

2.5 Scan trigger output

In a special hardware version and with RWMOS from V2.5.3.78, it is possible to output a hardware trigger signal synchronously to the scan. This signal can be used, for example, to synchronise external components during data acquisition. Depending on the hardware version, this signal may be an RS422 output or a digital 24 V output.
For this, the scanner variable SYNCPULSEOUT (#64) has to be written on by a bit-coded value in which the axes for the pulse output are flagged by set bits.

Example: 3rd axis = pulse output, the value 4 has to be written in the variable SYNCPULSEOUT

Normally, only one output will be prepared for such a purpose. In the corresponding axis, there is no zero-trace signal and no hardware latch available. When using a 24 V digital output, this can be connected in the usual way as well. The level actually output is the result of the disjunction (OR) of the indicated state information.
Through appropriate hardware preparation, it is also possible to have a fast pulse output from the software environment via the resource #64 (see manual “Resource Interface”), e.g. by writing on the resource #64, an RS422 output or, through appropriate hardware preparation, a digital output can be used at once.

Note: The output of the digital outputs is updated only once per sampling interval of the controller (usually 1.28 ms). A fast pulse output allows for a multiple output during the sampling interval.

2.6 Definition of the scan records

2.6.1 PCAP programming

The data to be scanned is defined by function calls with the following data. The elements to be scanned must first be defined via the G3 Resource Interface. The scan record is constructed in the reverse order to that in which the individual elements are defined. All the elements of the G3 resource interface can be scanned.

Table 3: Definition of the scan record

Object descriptor element	Value
BusNumber	1100
AccessType	Input/Output
DateType	(no function)
DeviceNumber	1, 2, ...
Index	0, 1, ... TIMEFACTOR for this element
SubIndex	Valid handle for an object descriptor from the G3 Resource Interface

Note:
The DeviceNumber must not be equal to 0 and must be assigned uniquely.
The initialisation of the objects that shall be recorded, is realised with the access method ATAccessInputOutput (= 3).
The contents of the parameter DataType at the object descriptors of the data to be recorded is of no importance.
The variable TIMEFACTOR, which is entered into the index, allows scanning the data record only to whole numbered multiples of the recording intervals. With the value 1 the measurement value is recorded in each recording interval. With the value 0, the measurement value is never recorded. As standard, here the value 1 must be entered.

2.6.2 SAP programming

An AT specifier is declared for each piece of data to be scanned.

Example:

```
var     ScanListItem_rp:            double AT %MRScannerBus.1.1.0;
var     ScanListItem_axst:          integer AT %MDScannerBus.2.1.0;
var     ScanListItem_digi:          integer AT %MDScannerBus.3.1.0;
```

The resource to be scanned must then be assigned once to this variable (each time the SAP programme is started) with the help of the ptr operator. This assignment must be specified in the reverse order to that in which the data is stored in the scan record.

Example:

```
        ScanListItem_rp         := ptr(G3R_rp_A1_r);        // real position of axis 1
        ScanListItem_digi       := ptr(G3R_digi_A1_r);      // digital inputs
        ScanListItem_ain_CH0 := ptr(G3R_ain_CH0_r);                 // analogue value channel 0
```

Please ensure that the data types used match.

2.7 Using the scanner functions

- Define the required ObjectDescriptor elements
- Define resources to be scanned,
  execute at least one read operation, so that there is a valid handle.
- Call scanner CLEAN
- Define a list of scan objects
- Program the number of records to be recorded using the variable RECORDSTOSCAN
- Call scanner INIT
- The scan can now be started and stopped again using
  scanner STARTSTOP
- The scanned data is read out using the PCAP command rdScannerBuffer (see below)
  Here, the scanner status can be requested at any time, using rdScannerStatus, for example.

For PCAP programming:
If the ResourceClean function is called repeatedly, all handles of the resource object descriptor elements must be reset.
If the ScannerClean function is called repeatedly, all handles of the scanner object descriptor elements must be reset using the rdwr functionality.

2.8 PCAP functions for scanner accesses

2.8.1 rdScannerBuffer, read scanner buffer

DESCRIPTION:	This function copies the current scanner buffer of the xPCI-800x into a memory area of the calling application and releases the corresponding memory on the control for the scan.
BORLAND DELPHI:	function rdScannerBuffer (buffer: PChar; size: integer): integer;
C:	int rdScannerBuffer (char *buffer, int size);
VISUAL BASIC:	Function rdScannerBuffer (buffer As String, ByVal size As Long)
PARAMETER:	The *buffer* parameter is a pointer to a memory area of the application, whose size must be at least *size* bytes. The parameter *size* specifies the number of bytes to be read.
RETURN VALUE:	Number of bytes that have been successfully copied into the *buffer* memory area 0 – No data available in the scanner -1 – ScannerBuffer is defined too large -2 – System error in the scanner module
NOTE:	The maximum number of bytes to be read out is computed in this function. Thus, a maximum number of *size* bytes or fewer are read out. For the subsequent data analysis, it is reasonable to always read out a multiple of the specified record length. The format of the data structure in which the data is written must correspond to the selection of the scan objects. The returned data is not aligned with word limits. To execute this command, specific *rwmos.elf* software is required. With this command, the data transfer rates may be under 2 MB/s depending on the hardware. With the newer command SendReqScannerBuffer() (see Chapter 2.8.4), the transfer rates are significantly higher.

2.8.2 rdScannerBufferSize, read scanner buffer size

DESCRIPTION:	This function returns the current size of the scanner buffer on the APCI-8001 / APCI-8008. The return value is given in bytes. By default, the buffer size is set to 100,000 bytes. The buffer size can be increased to a maximum of 13 MB, using a flash environment variable.
BORLAND DELPHI:	function rdScannerBufferSize: integer;
C:	int rdScannerBufferSize(void);
VISUAL BASIC:	Function rdScannerBufferSize() As Long
RETURN VALUE:	Buffer size of the scanner buffer, in bytes.
NOTE:	Specific *rwmos.elf* software is required to execute this command.

2.8.3 rdScannerLsm, read scanner left spool memory

DESCRIPTION:	This function returns the currently free available working memory of the scanner buffer. During entry in the scanner, this value counts down to 0. During the scanner read-out, the value returns to the ScannerBufferSize.
BORLAND DELPHI:	function rdScannerLsm: integer;
C:	int rdScannerLsm(void);
VISUAL BASIC:	Function rdScannerLsm() As Long
RETURN VALUE:	Free available working memory of the scanner buffer, in bytes.
NOTE:	Specific *rwmos.elf* software is required to execute this command.

2.8.4 SendReqScannerBuffer, Send Request scanner buffer

DESCRIPTION:	This function copies the current scanner buffer of the xPCI-800x into a memory area of the calling application and releases the corresponding memory on the control for the scan.
BORLAND DELPHI:	function SendReqScannerBuffer (buffer: PChar; size: integer): integer;
C:	int SendReqScannerBuffer (char *buffer, int size);
VISUAL BASIC:	Function SendReqScannerBuffer (buffer As String, ByVal size As Long)
PARAMETER:	The *buffer* parameter is a pointer to a memory area of the application, whose size must be at least *size* bytes. The parameter *size* specifies the number of bytes to be read.
RETURN VALUE:	Number of bytes that have been successfully copied into the *buffer* memory area 0 – No data available in the scanner and no error available Negative values indicate errors or status information. The bit coding is described in Chapter 2.8.4.1.
NOTE:	In contrast to the function *rdScannerBuffer*, the data is not read by the PCI board, but is written by the control into the PC's RAM. So with this command, the data transfer rate is significantly higher. The maximum number of bytes to be read out is computed in this function, too. Thus, a maximum number of *size* bytes or fewer are read out. For the subsequent data analysis, it is reasonable to always read out a multiple of the specified record length. The format of the data structure in which the data is written must correspond to the selection of the scan objects. The returned data is not aligned with word limits. To execute this command, specific *rwmos.elf* software is required. This command is only available from *rwmos.elf* V2.5.3.126 and with *mcug3.dll* V2.5.3.107.

2.8.4.1 Explanation of the return information for negative values

If the return value has a negative sign (bit 31 is set), this value does not stand for transferred bytes but indicates bit-coded status information and, if applicable, error information. If one of the lower 16 bits (0-15) is set, the value indicates error information and a data transfer is not possible with this command. Bits 16-30 indicate only status information, but it is still possible to use this command. However, an explicit analysis of these bits may make sense in order to evaluate the reliability of the system. The error and status information can be read out safely if *SendReqScannerBuffer* is called with *size* = 0. If *size* > 0 is indicated and only status information but also data is available in the scanner, a number of bytes is transferred and indicated in the return value. In this case, the status information is not available.

Table: Bit coding of the return value of the DLL command *SendReqScannerBuffer*

Bit	Value [hex]	Description
0	**0000 0001**	Access to the resource interface has failed. In this case, the monitor screen of *fwsetup.exe* outputs additional information.
1-7	**0000 00FE**	Currently not assigned
8	**0000 0100**	The attempt to get physical memory has failed! *KS_allocPhysMem* returned the error KSERROR_NOT_ENOUGH_MEMORY
9	**0000 0200**	The attempt to get physical memory has failed! *KS_allocPhysMem* returned the error KSERROR_CANNOT_ALLOC_PHYSMEM
10	**0000 0400**	The attempt to get physical memory has failed! *KS_allocPhysMem* returned the error KSERROR_CANNOT_MAP_PHYSMEM
11	**0000 0800**	Currently not assigned
12	**0000 1000**	The attempt to get physical memory has failed! *KS_allocPhysMem* has returned an unknown error.
13-16	**0000 E000**	Currently not assigned
16	**0001 0000**	Access to the service rwPhysMemService is not possible (ERR_OpenFileMapping error). In this case, a conventional attempt is made to get physical memory from the operating system.
17	**0002 0000**	Access to the service rwPhysMemService is not possible (ERR_ MapViewOfFile error). In this case, a conventional attempt is made to get physical memory from the operating system.
18	**0004 0000**	Currently not assigned
19	**0008 0000**	The stored data structures in mcug3.dll and rwPhysMemService do not match (problem of versions). Access is not possible. In this case, a conventional attempt is made to get physical memory from the operating system.
20	**0010 0000**	The memory area size of the physical memory is wrong (problem of versions). In this case, a conventional attempt is made to get physical memory from the operating system.
21	**0020 0000**	No physical memory available in the service (installation problem) In this case, a conventional attempt is made to get physical memory from the operating system.
22	**0040 0000**	The data identifiers in mcug3.dll and rwPhysMemService do not match (problem of versions). Access is not possible. In this case, a conventional attempt is made to get physical memory from the operating system.
23	**0080 0000**	The memory of the service rwPhysMemService is already used (marked). In this case, a conventional attempt is made to get physical memory from the operating system.
24-28	**1F00 0000**	Currently not assigned
29	**2000 0000**	allocPhysMem with KSF_ALTERNATIVE has failed (just for information). This bit has no significant meaning for the user.
30	**4000 0000**	The user program has no administrator rights (these are required to mark the assignment of memory space). If the memory is free, it is used anyway. However, multiple calls may result in double assignment and thus in data loss.
31	**8000 0000**	Sign bit: This bit indicates that the meaning of this table is effective. If this bit is not set, the numerical value indicates the number of transferred bytes.

2.8.4.2 Description and information on handling the command

Before the first retrieval of data, the command should be called with size = 0 in order to try to get physical memory for the internal processing of the command. This memory will then be available for the entire runtime of the program. If no memory is available, this command cannot be used. Therefore, it is absolutely necessary to analyse the return value with the first call.

In the event that the used memory is provided by the operating system service *rwPhysMemService*, the memory will be marked as “used” if the calling program has administrator rights. This prevents double use of the same memory area. When the application is quit, this mark is automatically removed and the memory is made available again for the next program call. If the service *rwPhysMemService* is unavailable or the memory of this service is marked as “used”, the internal DLL command *KS_allocPhysMem* tries to get some memory from the operating system.

If it is possible to simultaneously execute programs that use the command *SendReqScannerBuffer*, it should be ensured that these programs are executed with administrator rights (with Windows Vista / 7 / 8 / 10 and future versions). In case no error bit has been returned with the first call, the command may be used in this session without restrictions and in the same way as *rdScannerBuffer*.

2.8.5 rdScannerStatus, read scanner status

<table>
<tr><td><b>DESCRIPTION:</b></td><td>This function returns the current status of the scanner buffer on the APCI-8001 / APCI-8008.</td></tr>
<tr><td><b>BORLAND DELPHI:</b></td><td>Function rdScannerStatus: integer;</td></tr>
<tr><td><b>C:</b></td><td>int rdScannerStatus(void);</td></tr>
<tr><td><b>VISUAL BASIC:</b></td><td>Function rdScannerStatus() As Long</td></tr>
<tr><td><b>RETURN VALUE:</b></td><td>Bit-coded scanner status.

Table: Bit-coded construction of the scanner status word
<table>
<tr><th>Bit No.</th><th>Name</th><th>Function</th></tr>
<tr><td><b>0</b></td><td>empty</td><td>Status flag: scanner is completely empty</td></tr>
<tr><td><b>1</b></td><td>full</td><td>Status flag: scanner is full. The specified number of records has been entered.</td></tr>
<tr><td><b>2</b></td><td>inprocess</td><td>Status flag: the scanner is currently processing data.</td></tr>
<tr><td><b>3</b></td><td>endless</td><td>Status flag: the ‘endless’ scan operating mode has been selected.</td></tr>
<tr><td><b>8</b></td><td>norecords</td><td>Error flag: no records have been defined. Error during scan list generation.</td></tr>
<tr><td><b>9</b></td><td>overrun</td><td>Error flag: scanner overrun. The scanner buffer is full. For ‘endless’ scanning, the oldest, last entered, record is rejected. If ‘endless’ scanning has not been configured, the scan process is stopped and no new records can be recorded anymore</td></tr>
<tr><td><b>10</b></td><td>Config error</td><td>Error flag: An error was detected at configuration.
- unknown data type</td></tr>
<tr><td><b>11</b></td><td>Scan Ressource Not Valid</td><td>Error flag: A resource from the scan list is invalid. Possibly, the content of the resource interface was deleted.</td></tr>
</table>
</td></tr>
<tr><td><b>NOTE:</b></td><td>Specific <i>rwmos.elf</i> software is required to execute this command.</td></tr>
</table>

3 Windows service rwPhysMemService

The Windows service rwPhysMemService should be installed to use the function SendReqScannerBuffer() without restrictions. This function serves for transferring scanner data from the control to the PC user program in an accelerated way. When the Windows operating system boots up, this service requests physical memory and manages it during the entire runtime of the operating system. This ensures that the quick data transfer can be used any time, i.e. also after several program calls or with high usage of the Windows operating system. However, this memory cannot be used by different applications at the same time.

3.1 Installation of the Windows service rwPhysMemService

To install this service, the file rwMemMgnt.exe needs to be copied into a local directory. In this directory, the program must be called using the parameter /install:

```
rwMemMgnt /install
```

This can be made in a prompt box, for example. Here it should be noted that depending on the Windows version, administrator rights must be available.
By this call, the service is installed but not started yet. The service is started by a reboot of the Windows operating system or manually in services.msc.

3.2 Uninstallation of the Windows service rwPhysMemService

To uninstall this service, the file rwMemMgnt.exe (see Chapter 3.1) must be called using the parameter /uninstall:

```
rwMemMgnt /uninstall
```

This can be made in a prompt box, for example. Here it should be noted that depending on the Windows version, administrator rights must be available.
By this call, the Windows operating system receives an uninstallation prompt, i.e., after a reboot of the operating system, the service is uninstalled and not started anymore.
The service can also be stopped manually in services.msc. However, without uninstallation, the service will be restarted after a reboot of the Windows operating system.