

---

# **POSITIONIER- UND BAHNSTEUERUNG APCI-8001 UND APCI-8008**

## **Scanner-Interface**

Stand: 17.03.2020, ab CD-ROM V2.53VM  
Rev. 12/052022

[www.addi-data.de](http://www.addi-data.de)



<b>1 Einführung .....</b>	<b>5</b>
<b>2 Verwendung der Scanner-Funktionen .....</b>	<b>5</b>
2.1 Bereits implementierte Karten .....	5
2.2 Initialisierung des Scanners .....	5
2.3 Funktionen des Scanner-Moduls .....	6
2.4 Scan-Steuerung .....	7
2.5 Scan-Trigger-Output .....	7
2.6 Definition der Scan-Records .....	8
2.6.1 PCAP-Programmierung .....	8
2.6.2 SAP-Programmierung .....	9
2.7 Verwendung der Scanner-Funktionen .....	9
2.8 PCAP-Funktionen für Scanner-Zugriffe .....	10
2.8.1 rdScannerBuffer, read scanner buffer .....	10
2.8.2 rdScannerBufferSize, read scanner buffer size .....	10
2.8.3 rdScannerLsm, read scanner left spool memory .....	10
2.8.4 SendReqScannerBuffer, Send Request scanner buffer .....	11
2.8.5 rdScannerStatus, read scanner status .....	13
<b>3 Der Windows-Service rwPhysMemService .....</b>	<b>14</b>
3.1 Installation des Windows-Service rwPhysMemService .....	14
3.2 Deinstallation des Windows-Service rwPhysMemService .....	14



# 1 Einführung

Die Scanner-Funktionalität der APCI-8001 bzw. APCI-8008 kann verwendet werden, um Prozessdaten in Echtzeit zu scannen und zwischenspeichern. Hierbei werden die Prozessdaten zyklisch in einem Aufzeichnungsrecord gespeichert. Diese Records können als Record-Arrays ausgelesen und verarbeitet werden. Für die Anwendung dieser Funktionalität ist die Verwendung des Ressourcen-Interface erforderlich, welches im Dokument „Ressourcen-Interface“ beschrieben ist.

Diese Option kann nur verwendet werden, wenn eine Betriebssystem-Variante RWMos.ELF mit den Optionen *optionSCANNER* und *optionRESOURCE* zur Verfügung steht.

## 2 Verwendung der Scanner-Funktionen

### 2.1 Bereits implementierte Karten

APCI-3120: 16 analoge Eingänge, 8 analoge Ausgänge

APCI-3701: 16 induktive Messtaster

APCI-3003: 4 analoge Eingänge, simultane Erfassung

APCI-3501: 8 analoge Ausgänge

### 2.2 Initialisierung des Scanners

Folgende Werte für das Universelle Object-Interface sind für die Verwendung des Scanner-Moduls zu verwenden:

Tabelle 1: Object-Descriptor-Elemente

Object-Descriptor Element	Wert
<b>Handle</b>	Muss beim Start der Applikation oder nach Reboot der Steuerung mit 0 initialisiert sein und wird dann vom System geführt / verwendet. <b>Bei PCAP-Programmierung:</b> Nach einem Clean der Scanner-Funktionalität müssen die Handles der Elemente mit rdwr-Funktionalität erneut genullt werden.
<b>BusNumber</b>	1100
<b>DeviceNumber</b>	0
<b>Index</b>	0, 1, ... Funktionsnummer zum Konfigurieren / Bedienen des Scanners laut Tabelle 2
<b>SubIndex</b>	hier keine Bedeutung

Mit DeviceNumber > 0 und Index 1 werden die Scan-Objekte deklariert. DeviceNumber muss fortlaufend vergeben werden. Die zu schreibenden Parameter werden beim Aufruf von wrOptionInt (wrOptionDbl) als zweiter Parameter (value) übergeben bzw. bei der SAP-Programmierung direkt zugewiesen. Weitere Informationen zu den Object-Descriptor-Elementen finden Sie im Dokument „Universelles Objekt-Interface“.

## 2.3 Funktionen des Scanner-Moduls

Tabelle 2: Funktionen Scanner-Modul für Device Nr. 0

Nr. (Index)	Bezeichnung	Typ	Erläuterung	Übergabe- parameter (value)
1	CLEAN	integer w	Scanner zurücksetzen. Als Wert (value) muss 1 übergeben werden. <b>Bei PCAP-Programmierung:</b> Unmittelbar nach dem Rücksetzen müssen die Handles aller verwendeten Objekte mit rdwr-Funktionalität genullt werden.	1
2	INIT	integer w	Scanner vor dem Start initialisieren; so wird z.B. SizeOfRecord berechnet und der Datenpuffer wird geleert. <b>Vorsicht:</b> Durch diesen Aufruf wird auch die Variable HW_SCAN_STROBE zurückgesetzt.	1
3	STARTSTOP	integer r/w	Scanner starten oder stoppen bzw. Zustand abfragen	1 = Start 0 = Stop
4	STATUS	integer r	Status des Scanners lesen Der Rückgabewert dieser Funktion wird bei der Funktion rdScannerStatus beschrieben.	
5	SIZEBUFFER	integer r	Größe des gesamten Speichers des Scan-Buffers abfragen (in Byte) Default: 100.000 Byte Dieser Wert kann mit der Umgebungsvariable SZSCANBUFFER in fwsetup vorgegeben werden.	
6	TIMEFACTOR	integer r/w	Zeitraster in TA zur Aufzeichnung von Daten lesen / schreiben Default-Wert: 1	1, 2, ...
7	RECORDSTO SCAN	integer w	Anzahl der Datensätze (Records), die aufgezeichnet werden sollen lesen / schreiben. Die Anzahl wird im Übergabewert (value) übergeben. Falls hier 0 eingetragen wird, wird endlos gescannt. Falls mehr Records aufgezeichnet werden sollen als in den Datenpuffer passen, müssen die aufgezeichneten Daten während des Aufzeichnungsvorgangs ausgelesen werden. Dieser Wert wird beim Scan bis auf 0 abgezählt. Durch erneutes Beschreiben kann der Scan fortgesetzt werden. <b>Default-Wert: 1</b>	0
8	RECORDS SCANNED	integer r	Anzahl der aufgezeichneten Records abfragen. Durch Aufruf der Funktion ScannerInit wird dieser Wert auf 0 zurückgesetzt.	
9	SIZEOF RECORD	integer r	Größe des aufzuzeichnenden Records abfragen (in Byte). Dieser Wert steht erst nach Aufruf der Funktion INIT zur Verfügung.	

Nr. (Index)	Bezeichnung	Typ	Erläuterung	Übergabe- parameter (value)
10	CHECK BUFFER	integer r	Größe des freien Speichers im Scan-Buffer abfragen (in Byte).	
11	HW_SCAN_ STROBE	integer r/w	Durch Setzen von einem oder mehreren Bits in diesem Register können schnelle Hardware-Eingänge als Strobe-Eingänge für den Latchvorgang definiert werden. Damit diese Option genutzt werden kann, muss RWMOS.ELF mit der entsprechenden Option ausgerüstet sein. Achtung: Diese Variable wird durch einen Aufruf von INIT zurückgesetzt.	Bits 0..7
22	FREE BUFFER	integer w	Interne Funktion für Speicherverwaltung, nicht für den Anwender bestimmt.	freizugebender Speicher in Byte
64	SYNCPULSE OUT	integer r/w	Nur bei spezieller Hardware-Variante: Mit Hilfe dieses Registers kann eine scanner-synchrone Impulsausgabe zur Triggerung externer Komponenten erfolgen (siehe Kapitel 2.5)	bit-codierte Angabe der Achsen für Impulsausgabe

## 2.4 Scan-Steuerung

Standardmäßig wird der Scan zeitgesteuert und abtastsynchron durchgeführt. Es ist aber auch möglich, den Scan ereignisgesteuert durchzuführen. Hierzu wird als letztes Scan-Element die Ressource WTLSTRB (#101) für eine definierte Achse definiert. In diesem Fall wird der Scan jeweils dann aufgezeichnet, wenn ein Latch-Strobe-Signal der entsprechenden Achse erkannt wurde. Jedoch auch hier erfolgt der Scan abtastsynchron. Der Parameter TIMEFACTOR sollte hier immer auf 1 stehen.

Die Latchimpulse dürfen nicht schneller als die Abtastzeitpunkte anliegen. Das Latch-Strobe-Signal wird beim Aufzeichnen des Scan-Records jeweils zurückgesetzt.

## 2.5 Scan-Trigger-Output

In einer speziellen Hardware-Variante und mit RWMOS ab V2.5.3.78 ist es möglich, synchron zum Scan ein Hardware-Triggersignal auszugeben, welches verwendet werden kann, um z.B. externe Komponenten bei der Messwertaufnahme zu synchronisieren. Dieses Signal kann je nach Hardwarevariante ein RS422- oder ein 24 V-Digitalausgang sein.

Hierzu muss die Scanner-Variable SYNCPULSEOUT (#64) mit einem bit-codierten Wert beschrieben werden, in welchem die Achsen für die Impulsausgabe durch gesetzte Bits gekennzeichnet sind.

**Beispiel:** 3. Achse = Impulsausgang, dann muss der Wert 4 in die Variable SYNCPULSEOUT geschrieben werden.

Im Allgemeinen wird nur ein Ausgang für einen derartigen Zweck vorbereitet sein. Bei der entsprechenden Achse stehen kein Nullspursignal und kein Hardware-Latch zur Verfügung. Bei Verwendung eines 24 V-Digitalausgangs kann dieser auch auf herkömmliche Weise geschaltet werden. Der tatsächlich ausgegebene Pegel ergibt sich durch Veroderung der angegebenen Zustandsinformationen.

Mit der entsprechenden Hardwarevorbereitung ist es zusätzlich möglich, eine schnelle Impulsausgabe aus der Softwareumgebung über die Ressource #64 (siehe Handbuch „Ressourcen-Interface“) vorzunehmen, d.h., durch Schreiben auf die Ressource #64 kann ein RS422-Ausgang oder bei entsprechender Hardware-Vorbereitung ein Digitalausgang sofort betätigt werden.

**Hinweis:** Die Ausgabe der digitalen Ausgänge wird nur einmal pro Abtastintervall der Steuerung (normalerweise 1,28 ms) aktualisiert. Mit einer schnellen Impulsausgabe kann eine Ausgabe mehrfach im Abtastintervall erfolgen.

## 2.6 Definition der Scan-Records

### 2.6.1 PCAP-Programmierung

Die aufzuzeichnenden Daten werden durch Funktionsaufrufe mit nachfolgenden Daten definiert. Die aufzuzeichnenden Elemente müssen zuvor über das Ressourcen-Interface definiert werden. Der Aufzeichnungsrecord wird in der umgekehrten Reihenfolge aufgebaut wie die einzelnen Elemente definiert werden. Alle Elemente des Ressourcen-Interface können gescannt werden.

Tabelle 3: Definition des Aufzeichnungsrecords

Object-Descriptor Element	Wert
BusNumber	1100
AccessType	Input/Output
DataType	(unbedeutend)
DeviceNumber	1, 2,....
Index	0, 1, ... TIMEFACTOR für dieses Element
SubIndex	Gültiges Handle eines Object-Descriptors des Ressourcen-Interface

**Hinweis:**

Die DeviceNumber muss ungleich 0 sein und eindeutig vergeben werden.

Die Initialisierung der Objekte, die aufgezeichnet werden sollen, erfolgt mit der Zugriffsmethode `ATAccessInputOutput (= 3)`.

Der Inhalt des Parameters `DataType` bei den Object-Descriptors der aufzuzeichnenden Daten ist unbedeutend.

Die Variable `TIMEFACTOR`, die in `Index` eingetragen wird, ermöglicht es, die Datenaufzeichnung zu scannen; hierbei ist jedoch nur ein ganzzahliges Vielfaches der Aufzeichnungsintervalle erlaubt. Mit dem Wert 1 wird der Messwert in jedem Aufzeichnungsintervall aufgezeichnet. Mit dem Wert 0 wird der Messwert nie aufgezeichnet. Standardmäßig muss hier der Wert 1 eingetragen werden.



## 2.6.2 SAP-Programmierung

Für jedes aufzuzeichnende Datum wird ein AT-Spezifizierer deklariert:

### Beispiel:

```
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;
```

Dieser Variablen muss dann einmalig (pro Start des SAP-Programms) die zu scannende Ressource mit Hilfe des ptr-Operators zugewiesen werden. Diese Zuweisung muss in umgekehrter Reihenfolge angegeben werden als die Daten im Aufzeichnungsrecord abgelegt werden.

### Beispiel:

```
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);         // analog value channel 0
```

Hierbei ist darauf zu achten, dass die verwendeten Datentypen übereinstimmen.

## 2.7 Verwendung der Scanner-Funktionen

- Definition der benötigten ObjectDescriptor-Elemente
- aufzuzeichnende Ressourcen definieren, mindestens eine Leseoperation ausführen, damit ein gültiges Handle vorliegt.
- Scanner CLEAN aufrufen
- Liste der Scan-Objekte definieren
- Anzahl der aufzuzeichnenden Datensätze programmieren per Variable RECORDSTOSCAN
- Scanner INIT aufrufen
- Nun kann der Scan gestartet und auch wieder gestoppt werden mit Scanner STARTSTOP
- Das Auslesen der aufgezeichneten Daten erfolgt mit dem PCAP-Befehl rdScannerBuffer (siehe unten). Hierbei kann jederzeit der Zustand des Scanners abgefragt werden, z.B. mit rdScannerStatus.

### Bei PCAP-Programmierung:

Falls die Funktion „ResourceClean“ wiederholt aufgerufen wird, müssen alle Handles der Resource-Object-Descriptor-Elemente genullt werden.

Falls die Funktion „ScannerClean“ wiederholt aufgerufen wird, müssen alle Handles der Scanner-Object-Descriptor-Elemente mit rdwr-Funktionalität genullt werden.

## 2.8 PCAP-Funktionen für Scanner-Zugriffe

### 2.8.1 rdScannerBuffer, read scanner buffer

<b>BESCHREIBUNG:</b>	Diese Funktion kopiert den aktuellen Scanner-Buffer der xPCI-800x in einen Speicherbereich der aufrufenden Anwendung und gibt den entsprechenden Speicher auf der Steuerung wieder für den Scan frei.
<b>BORLAND DELPHI:</b>	function rdScannerBuffer (buffer: PChar; size: integer): integer;
<b>C:</b>	int rdScannerBuffer (char *buffer, int size);
<b>VISUAL BASIC:</b>	Function rdScannerBuffer (buffer As String, ByVal size As Long)
<b>PARAMETER:</b>	Der Parameter <i>buffer</i> ist ein Zeiger auf einen Speicherbereich der Anwendung, der mindestens <i>size</i> Bytes groß sein muss. Der Parameter <i>size</i> spezifiziert dabei die zu lesende Anzahl von Bytes.
<b>RÜCKGABEWERT:</b>	Anzahl von Bytes, die erfolgreich in den Speicherbereich <i>buffer</i> kopiert wurden 0 – wenn keine Daten im Scanner vorliegen -1 – Scanner-Buffer zu groß definiert -2 – Systemfehler im Scanner-Modul
<b>ANMERKUNG:</b>	Die maximale Anzahl auszulesender Bytes wird in dieser Funktion automatisch berechnet. Es werden also maximal <i>size</i> Bytes oder weniger ausgelesen. Für die nachfolgende Datenauswertung ist es sinnvoll, immer ein Vielfaches der vorgegebenen Record-Länge auszulesen. Der Aufbau der Datenstruktur, in welche die Daten geschrieben werden, muss mit der Auswahl der Scan-Objekte übereinstimmen. Die zurückgelieferten Daten sind nicht nach Wortgrenzen ausgerichtet. Zur Ausführung dieses Befehls ist eine spezielle <i>rwmos.elf</i> -Software erforderlich. Die Datenübertragungsraten können bei diesem Befehl je nach Hardware unter 2 MB/s liegen. Mit dem neueren Befehl SendReqScannerBuffer() (siehe Kap. 2.8.4) sind die Übertragungsraten deutlich höher.

### 2.8.2 rdScannerBufferSize, read scanner buffer size

<b>BESCHREIBUNG:</b>	Diese Funktion liefert die aktuelle Größe des Scanner-Buffers auf die APCI-8001 bzw. APCI-8008 zurück. Der Rückgabewert hat die Einheit Byte. Standardmäßig ist die Buffergröße auf 100.000 Bytes voreingestellt. Die Buffergröße kann mit einer Flash-Umgebungsvariablen auf bis zu maximal 13 MByte hochgesetzt werden.
<b>BORLAND DELPHI:</b>	function rdScannerBufferSize: integer;
<b>C:</b>	int rdScannerBufferSize(void);
<b>VISUAL BASIC:</b>	Function rdScannerBufferSize() As Long
<b>RÜCKGABEWERT:</b>	Buffergröße des Scanner-Buffers in Byte.
<b>ANMERKUNG:</b>	Zur Ausführung dieses Befehls ist eine spezielle <i>rwmos.elf</i> -Software erforderlich.

### 2.8.3 rdScannerLsm, read scanner left spool memory

<b>BESCHREIBUNG:</b>	Diese Funktion liefert den momentan frei verfügbaren Arbeitsspeicher des Scanner-Buffers zurück. Beim Eintrag im Scanner läuft dieser Wert gegen 0. Beim Auslesen des Scanners läuft der Wert gegen ScannerBufferSize.
<b>BORLAND DELPHI:</b>	function rdScannerLsm: integer;
<b>C:</b>	int rdScannerLsm(void);
<b>VISUAL BASIC:</b>	Function rdScannerLsm() As Long
<b>RÜCKGABEWERT:</b>	Frei verfügbarer Arbeitsspeicher des Scanner-Buffers in Byte.
<b>ANMERKUNG:</b>	Zur Ausführung dieses Befehls ist eine spezielle <i>rwmos.elf</i> -Software erforderlich.

### 2.8.4 SendReqScannerBuffer, Send Request scanner buffer

<b>BESCHREIBUNG:</b>	Diese Funktion kopiert den aktuellen Scanner-Buffer der xPCI-800x in einen Speicherbereich der aufrufenden Anwendung und gibt den entsprechenden Speicher auf der Steuerung wieder für den Scan frei.
<b>BORLAND DELPHI:</b>	function SendReqScannerBuffer (buffer: PChar; size: integer): integer;
<b>C:</b>	int SendReqScannerBuffer (char *buffer, int size);
<b>VISUAL BASIC:</b>	Function SendReqScannerBuffer (buffer As String, ByVal size As Long)
<b>PARAMETER:</b>	Der Parameter <i>buffer</i> ist ein Zeiger auf einen Speicherbereich der Anwendung, der mindestens <i>size</i> Bytes groß sein muss. Der Parameter <i>size</i> spezifiziert dabei die zu lesende Anzahl von Bytes.
<b>RÜCKGABEWERT:</b>	Anzahl von Bytes, die erfolgreich in den Speicherbereich <i>buffer</i> kopiert wurden 0 – wenn keine Daten im Scanner vorliegen und wenn kein Fehler vorliegt. Negative Werte zeigen Fehler oder Statusinformationen an. Eine Beschreibung der Bit-Codierung erfolgt in Kapitel 2.8.4.1.
<b>ANMERKUNG:</b>	Im Gegensatz zur Funktion <i>rdScannerBuffer</i> werden die Daten nicht von der PCI-Karte gelesen, sondern von der Steuerung in den Arbeitsspeicher des PCs geschrieben. Deshalb ist mit diesem Befehl die Datenübertragungsrate wesentlich höher. Die maximale Anzahl auszulesender Bytes wird in dieser Funktion ebenfalls automatisch berechnet. Es werden also maximal <i>size</i> Bytes oder weniger ausgelesen. Für die nachfolgende Datenauswertung ist es sinnvoll, immer ein Vielfaches der vorgegebenen Record-Länge auszulesen. Der Aufbau der Datenstruktur, in welche die Daten geschrieben werden, muss mit der Auswahl der Scan-Objekte übereinstimmen. Die zurückgelieferten Daten sind nicht nach Wortgrenzen ausgerichtet. Zur Ausführung dieses Befehls ist eine spezielle <i>rwmos.elf</i> -Software erforderlich. Dieser Befehl ist erst ab <i>rwmos.elf</i> V2.5.3.126 und mit <i>mcug3.dll</i> V2.5.3.107 verfügbar.

#### 2.8.4.1 Erläuterung der Rückgabe-Informationen bei negativem Wert

Wenn der Rückgabewert ein negatives Vorzeichen besitzt (Bit 31 gesetzt), dann hat dieser Wert nicht die Bedeutung von übertragenen Bytes, sondern zeigt bit-codierte Status- und gegebenenfalls Fehlerinformationen an. Wenn eines der unteren 16 Bits (0-15) gesetzt ist, handelt es sich um eine Fehlerinformation und eine Datenübertragung ist mit diesem Befehl nicht möglich. Die Bits 16-30 zeigen nur Statusinformationen an, aber die Verwendung des Befehls ist trotzdem möglich. Dennoch kann eine explizite Auswertung dieser Bits sinnvoll sein, um die Zuverlässigkeit des Systems beurteilen zu können. Die Fehler- und Statusinformation kann sicher ausgelesen werden, wenn *SendReqScannerBuffer* mit *size* = 0 aufgerufen wird. Wenn *size* > 0 angegeben wird und nur Statusinformationen, aber auch Daten im Scanner vorliegen, wird eine Anzahl von Bytes übertragen und im Rückgabewert angezeigt. In diesem Fall ist die Statusinformation nicht verfügbar.

Tabelle: Bit-Codierung des Rückgabewerts des DLL Befehls *SendReqScannerBuffer*

Bit	Wert [hex]	Beschreibung
0	0000 0001	Der Zugriff auf das Ressourcen-Interface ist fehlgeschlagen. In diesem Fall werden im Monitor-Screen von <i>fwsetup.exe</i> zusätzliche Informationen ausgegeben.
1-7	0000 00FE	derzeit nicht belegt
8	0000 0100	Der Versuch, physischen Speicher zu erhalten, ist fehlgeschlagen! <i>KS_allocPhysMem</i> lieferte den Fehler <code>KSERROR_NOT_ENOUGH_MEMORY</code>
9	0000 0200	Der Versuch, physischen Speicher zu erhalten, ist fehlgeschlagen! <i>KS_allocPhysMem</i> lieferte den Fehler <code>KSERROR_CANNOT_ALLOC_PHYSMEM</code>

Bit	Wert [hex]	Beschreibung
10	0000 0400	Der Versuch, physischen Speicher zu erhalten, ist fehlgeschlagen! <i>KS_allocPhysMem</i> lieferte den Fehler <code>KSERROR_CANNOT_MAP_PHYSMEM</code>
11	0000 0800	derzeit nicht belegt
12	0000 1000	Der Versuch, physischen Speicher zu erhalten, ist fehlgeschlagen! <i>KS_allocPhysMem</i> lieferte einen unbekannten Fehler.
13-16	0000 E000	derzeit nicht belegt
16	0001 0000	Der Zugriff auf den Dienst <i>rwPhysMemService</i> ist nicht möglich (Fehler <code>ERR_OpenFileMapping</code> ). In diesem Fall wird konventionell versucht, physischen Speicher vom Betriebssystem zu erhalten.
17	0002 0000	Der Zugriff auf den Dienst <i>rwPhysMemService</i> ist nicht möglich (Fehler <code>ERR_MapViewOfFile</code> ). In diesem Fall wird konventionell versucht, physischen Speicher vom Betriebssystem zu erhalten.
18	0004 0000	derzeit nicht belegt
19	0008 0000	Die hinterlegten Datenstrukturen in <i>mcug3.dll</i> und <i>rwPhysMemService</i> stimmen nicht überein (Versionsproblem). Ein Zugriff ist nicht möglich. In diesem Fall wird konventionell versucht, physischen Speicher vom Betriebssystem zu erhalten.
20	0010 0000	Die Speicherbereichsgröße des physischen Speichers ist falsch (Versionsproblem). In diesem Fall wird konventionell versucht, physischen Speicher vom Betriebssystem zu erhalten.
21	0020 0000	Kein physischer Speicher im Service vorhanden (Installationsproblem) In diesem Fall wird konventionell versucht, physischen Speicher vom Betriebssystem zu erhalten.
22	0040 0000	Die Daten-Kennungen in <i>mcug3.dll</i> und <i>rwPhysMemService</i> stimmen nicht überein (Versionsproblem). Ein Zugriff ist nicht möglich. In diesem Fall wird konventionell versucht, physischen Speicher vom Betriebssystem zu erhalten.
23	0080 0000	Der Speicher des Service <i>rwPhysMemService</i> ist bereits in Verwendung (markiert). In diesem Fall wird konventionell versucht, physischen Speicher vom Betriebssystem zu erhalten.
24-28	1F00 0000	derzeit nicht belegt
29	2000 0000	<i>allocPhysMem</i> mit <code>KSF_ALTERNATIVE</code> ist fehlgeschlagen (nur zur Info). Dieses Bit hat keine nennenswerte Bedeutung für den Anwender.
30	4000 0000	Das Anwenderprogramm hat keine Administratorrechte (werden zum Markieren der Speicherbelegung benötigt). Wenn der Speicher frei ist, wird er dennoch verwendet. Bei Mehrfachaufrufen ist jedoch eine Doppelbelegung und damit Datenverlust möglich.
31	8000 0000	Vorzeichen-Bit: Dieses Bit zeigt an, dass die Bedeutung dieser Tabelle wirksam ist. Wenn dieses Bit nicht gesetzt ist, zeigt der Zahlenwert die Anzahl der übertragenen Bytes an.

#### 2.8.4.2 Beschreibung und Hinweise zur Handhabung des Befehls

Vor der ersten Abholung von Daten sollte der Befehl mit `size = 0` aufgerufen werden. Hierbei wird versucht, physischen Speicher für die interne Verarbeitung des Befehls zu erhalten. Dieser Speicher steht dann für die gesamte Laufzeit des Programms zur Verfügung. Wenn kein Speicher verfügbar ist, kann dieser Befehl nicht verwendet werden. Deshalb sollte beim ersten Aufruf unbedingt die Auswertung des Rückgabewerts durchgeführt werden.

Falls der verwendete Speicher vom Betriebssystemdienst *rwPhysMemService* bereitgestellt wird, so wird der Speicher als „in Verwendung“ markiert, sofern das aufrufende Programm Administratorrechte besitzt. Somit wird eine doppelte Verwendung des gleichen Speicherbereichs verhindert. Beim Beenden der Applikation wird diese Markierung automatisch aufgehoben und der Speicher für den nächsten Programmaufruf wieder verfügbar gemacht. Wenn der Dienst *rwPhysMemService* nicht erreichbar ist oder der Speicher dieses Dienstes als „in Verwendung“ markiert ist, wird mit dem internen DLL-Befehl *KS\_allocPhysMem* versucht, Speicher vom Betriebssystem zu erhalten.

Falls Programme, die den Befehl *SendReqScannerBuffer* verwenden, gleichzeitig ausgeführt werden können, sollte darauf geachtet werden, dass die Programme mit Administratorberechtigung ausgeführt werden (bei Windows Vista / 7 / 8 / 10 und zukünftigen Versionen). Wenn der erste Aufruf ohne Rückgabe eines Fehlerbits erfolgte, kann der Befehl in dieser Session uneingeschränkt und in gleicher Weise wie *rdScannerBuffer* verwendet werden.

### 2.8.5 rdScannerStatus, read scanner status

BESCHREIBUNG:	Diese Funktion liefert den aktuellen Status des Scanners der APCI-8001 bzw. APCI-8008 zurück.																												
BORLAND DELPHI:	function rdScannerStatus: integer;																												
C:	int rdScannerStatus(void);																												
VISUAL BASIC:	Function rdScannerStatus() As Long																												
RÜCKGABEWERT:	Scanner-Status bit-codiert.																												
	Tabelle: Bit-codierter Aufbau des Scanner-Status-Wortes																												
	<table><tr><th>Bit-Nr.</th><th>Name</th><th>Funktion</th></tr><tr><td>0</td><td>empty</td><td>Status-Flag: Scanner ist komplett leer.</td></tr><tr><td>1</td><td>full</td><td>Status-Flag: Scanner ist voll. Die vorgegebene Anzahl von Records wurde eingetragen.</td></tr><tr><td>2</td><td>inprocess</td><td>Status-Flag: Der Scanner ist zurzeit in Arbeit.</td></tr><tr><td>3</td><td>endless</td><td>Status-Flag: Die Betriebsart Endlos-Scan ist angewählt.</td></tr><tr><td>8</td><td>norecords</td><td>Error-Flag: Es wurden keine Records definiert. Fehler bei Scan-Listen-Generierung.</td></tr><tr><td>9</td><td>overrun</td><td>Error-Flag: Scanner-Überlauf. Der Scanner-Buffer ist voll. Bei Endlos-Scan wird jeweils der zuletzt eingetragene Datensatz verworfen. Falls kein Endlos-Scan, wird der Scan gestoppt. und kann keine neuen Records mehr aufnehmen.</td></tr><tr><td>10</td><td>config error</td><td>Error-Flag: Es wurde ein Fehler bei der Konfiguration erkannt - unbekannter Datentyp</td></tr><tr><td>11</td><td>Scan Ressource Not Valid</td><td>Error-Flag: Eine Ressource in der Scan-Liste ist ungültig. Evtl. wurde der Inhalt des Ressourcen-Interface gelöscht.</td></tr></table>	Bit-Nr.	Name	Funktion	0	empty	Status-Flag: Scanner ist komplett leer.	1	full	Status-Flag: Scanner ist voll. Die vorgegebene Anzahl von Records wurde eingetragen.	2	inprocess	Status-Flag: Der Scanner ist zurzeit in Arbeit.	3	endless	Status-Flag: Die Betriebsart Endlos-Scan ist angewählt.	8	norecords	Error-Flag: Es wurden keine Records definiert. Fehler bei Scan-Listen-Generierung.	9	overrun	Error-Flag: Scanner-Überlauf. Der Scanner-Buffer ist voll. Bei Endlos-Scan wird jeweils der zuletzt eingetragene Datensatz verworfen. Falls kein Endlos-Scan, wird der Scan gestoppt. und kann keine neuen Records mehr aufnehmen.	10	config error	Error-Flag: Es wurde ein Fehler bei der Konfiguration erkannt - unbekannter Datentyp	11	Scan Ressource Not Valid	Error-Flag: Eine Ressource in der Scan-Liste ist ungültig. Evtl. wurde der Inhalt des Ressourcen-Interface gelöscht.	
Bit-Nr.	Name	Funktion																											
0	empty	Status-Flag: Scanner ist komplett leer.																											
1	full	Status-Flag: Scanner ist voll. Die vorgegebene Anzahl von Records wurde eingetragen.																											
2	inprocess	Status-Flag: Der Scanner ist zurzeit in Arbeit.																											
3	endless	Status-Flag: Die Betriebsart Endlos-Scan ist angewählt.																											
8	norecords	Error-Flag: Es wurden keine Records definiert. Fehler bei Scan-Listen-Generierung.																											
9	overrun	Error-Flag: Scanner-Überlauf. Der Scanner-Buffer ist voll. Bei Endlos-Scan wird jeweils der zuletzt eingetragene Datensatz verworfen. Falls kein Endlos-Scan, wird der Scan gestoppt. und kann keine neuen Records mehr aufnehmen.																											
10	config error	Error-Flag: Es wurde ein Fehler bei der Konfiguration erkannt - unbekannter Datentyp																											
11	Scan Ressource Not Valid	Error-Flag: Eine Ressource in der Scan-Liste ist ungültig. Evtl. wurde der Inhalt des Ressourcen-Interface gelöscht.																											
ANMERKUNG:	Zur Ausführung dieses Befehls ist eine spezielle <i>rwmos.elf</i> -Software erforderlich.																												

### 3 Der Windows-Service rwPhysMemService

Zur uneingeschränkten Nutzung der Funktion `SendReqScannerBuffer()`, welche zum beschleunigten Übertragen der Scanner-Daten von der Steuerung ins PC-Anwenderprogramm dient, sollte der Windows-Service `rwPhysMemService` installiert werden. Dieser Service fordert beim Booten des Windows-Betriebssystems physischen Speicher an und verwaltet diesen über die gesamte Laufzeit des Betriebssystems. Somit ist gewährleistet, dass die schnelle Datenübertragung jederzeit, d.h. auch nach mehreren Programmaufrufen oder bei starker Auslastung des Windows-Betriebssystems genutzt werden kann. Allerdings kann dieser Speicher nicht von verschiedenen Anwendungen gleichzeitig genutzt werden.

#### 3.1 Installation des Windows-Service rwPhysMemService

Zur Installation ist die Datei `rwMemMgnt.exe` in ein lokales Verzeichnis zu kopieren. In diesem Verzeichnis muss das Programm mit dem Parameter `/install` aufgerufen werden:

```
rwMemMgnt /install
```

Dies kann z.B. in einer Eingabeaufforderungsbox erfolgen. Hierbei ist zu beachten, dass je nach Windows-Version Administratorrechte zur Verfügung stehen müssen.

Durch diesen Aufruf wird der Dienst installiert, aber noch nicht gestartet. Der Start des Dienstes erfolgt durch einen Reboot des Windows-Betriebssystems oder manuell in `services.msc`.

#### 3.2 Deinstallation des Windows-Service rwPhysMemService

Zur Deinstallation muss die Datei `rwMemMgnt.exe` (siehe Kapitel 3.1) mit dem Parameter `/uninstall` aufgerufen werden:

```
rwMemMgnt /uninstall
```

Dies kann z.B. in einer Eingabeaufforderungsbox erfolgen. Hierbei ist zu beachten, dass je nach Windows-Version Administratorrechte zur Verfügung stehen müssen.

Durch diesen Aufruf wird dem Windows-Betriebssystem eine Deinstallationsaufforderung mitgeteilt, d.h., nach einem Reboot des Betriebssystems wird der Dienst deinstalliert und nicht mehr gestartet.

Der Stop des Dienstes kann auch manuell in `services.msc` erfolgen. Ohne Deinstallation wird der Dienst aber nach einem Reboot des Windows-Betriebssystems erneut gestartet.