POSITIONIER- UND BAHNSTEUERUNG APCI-8001 und APCI-8008

Universelles Objekt-Interface

Stand: 24.02.2015, CD-ROM V2.53VB
Rev. 11/112018

www.addi-data.de

1 Einführung

Das „Universelle Objekt-Interface“ [UOI] stellt eine universelle und flexible Software-Schnittstelle zur Programmierung der APCI-8001 / APCI-8008 dar. Es handelt sich hierbei um einen universellen Ansatz zur Integration verschiedenster Hard- und Software-Erweiterungen für die Positionier- und Bahnsteuerungen APCI-8001 und APCI-8008.
Der Vorteil für den Anwender liegt in der Universalität der Schnittstelle (SAP-Interface, DLL). Benutzerspezifische Erweiterungen benötigen nur die Aktualisierung des Applikationsprogramms und das Vorhandensein der Funktionalität in der Betriebssystem-Software RWMOS.ELF.

1.1 Die Software-Schnittstellen für das UOI

Das „Universelle Objekt-Interface“ wird durch die Programmiermethoden SAP und PCAP gleichermaßen unterstützt und stellt eine konsequente Erweiterung dieser bereits seit 10 Jahren bewährten Programmiermethoden dar. Anwender, denen die Begriffe SAP und PCAP noch neu sind, sollten zuerst das Programmierhandbuch (PHB) der APCI-8001 und APCI-8008 durcharbeiten.

1.2 Neue Funktionen mit Hilfe des UOI

Derzeit ermöglicht das „Universelle Objekt-Interface“ folgende Hard- und Software-Erweiterungen:

Tabelle 1-1: Mögliche Funktionserweiterungen der APCI-8001 / APCI-8008

Interface	Beschreibung	Dokument (PDF)
ELCAM	Electronic Cam: universelle Tabellen-Interpolation	ELCAM-Interface
CANOPEN	CANOPEN-Feldbusmaster (in Vorbereitung). Benötigt wird hierzu die Hardwareoption OPCAN.	CanBus-Interface
Ressourcen	Zugriff auf interne Hard- oder Softwareregister der APCI-8001-/APCI-8008-Controller.	Ressourcen-Interface
INTERBUS	INTERBUS-Feldbusmaster. Benötigt wird hierzu die Hardwareoption OPIBS.	Optionen-Handbuch
PCI I/O	PCI-Busmaster. Die APCI-8001 / APCI-8008 gestattet den direkten Zugriff auf andere PCI-Baugruppen im I/O Bereich ohne Zuhilfenahme des Betriebssystems. Diese Zugriffsmethode gestattet einen sehr schnellen Zugriff unter Einhaltung von harten Echtzeitbedingungen. Sie gestattet darüber hinaus eine flexible Erweiterung der CNC-I/O-Ebene.	PCI-Interface
PCI Memory	PCI-Busmaster. Die Zugriffe erfolgen hier über I/O- oder Memory-Bereiche anderer PCI-Baugruppen.	Ressourcen-Interface
Scanner	Verwalten von benutzerdefinierten Listen und Aufzeichnung von Daten im internen Arbeitsspeicher der APCI-8001 / APCI-8008 unter Einhaltung von hohen Echtzeit-bedingungen.	Scanner-Interface
TC	Werkzeugradius-Korrektur (Tool-Compensation)	TC-Interface

2 Aufbau des „Universellen Objekt-Interfaces“

2.1 PCAP-Programmierung

2.1.1 Der Funktionszugriff über *OptionDescriptorObject*

Der Zugriff auf das Universelle Objekt Interface erfolgt über die vordefinierten Datenstrukturen bzw. Records vom Typ ***OptionDescriptorObject.***
Für jede Funktion, die verwendet werden soll, muss ein OptionDescriptorObject angelegt und initialisiert werden, jeweils zum Lesen und zum Schreiben. Ganzzahlvariable werden dann durch Aufrufe der DLL-Funktionen wrOptionInt bzw. rdOptionInt behandelt, Gleitpunktzahlen durch Aufrufe von wrOptionDbl und rdOptionDbl.
Zur Übertragung von 16- und 8-Bit-Variablen wird ebenfalls wrOptionInt bzw. rdOptionInt verwendet. In diesem Fall wird nur der entsprechende Anteil des Parameters val (siehe unten) berücksichtigt.

Tabelle 1: Object-Descriptor-Elemente

Object-Descriptor-Element	Beschreibung
Handle	Muss beim Start der Applikation oder nach Reboot der Steuerung mit 0 initialisiert sein und wird dann vom System geführt / verwendet. **Bei PCAP-Programmierung:** Nach einem Clean der entsprechenden Funktionalität muss das Handle ggf. wieder genullt werden. Hierzu ist die Dokumentation des jeweiligen Moduls zu beachten.
AccessType	Zugriffsart: Hier muss vor der ersten Verwendung die Zugriffsart eingetragen werden. Die gültigen Zugriffsarten sind definiert in *ATAccessType* Mit dieser Variablen wird z.B. festgelegt, ob es sich um eine Lese- oder Schreiboperation handelt. Bei Operatoren, bei denen Lese- und Schreibzugriff möglich ist, muss für jede Zugriffsart ein eigener ObjectDescriptor mit dem entsprechenden AccessType angelegt werden.
DataType	Datentyp: Hier muss vor der ersten Verwendung der Datentyp der Variable eingetragen werden.
BusNumber	Hier wird die BusNummer des jeweiligen Moduls eingetragen, z.B. 1200 für das ELCAM-Modul
DeviceNumber	Modulspezifische Größe
Index	Modulspezifische Funktionen
SubIndex	Modulspezifische Unter-Funktionen

2.1.1.1 Handle

Das Element *Handle* des Objekt-Descriptor-Objects wird bei der erstmaligen Verwendung initialisiert. Der Wert ist eine laufzeitabhängige Variable der RWMOS-Betriebssystem-Software. Das heißt aber, dass dieser Wert ungültig ist, sobald die Steuerung neu gebootet wurde. Nach einem Reboot der Steuerung müssen also alle Handles von Object-Descriptor-Elementen abgenullt werden.

2.1.1.2 AccessType

Dieser Parameter beschreibt die Zugriffsart, mit welcher der Parameter verwendet wird.

Wert	Bezeichnung	Beschreibung
0	**ATAccessNone**	nicht verwendet
1	**ATAccessInput**	Lesezugriff
2	**ATAccessOutput**	Schreibzugriff
3	**ATAccess InputOutput**	Konfigurationswert, z.B. für die Definition des Scanner-Moduls

2.1.1.3 DataType

Dieser Parameter bestimmt das Datenformat des Zugriffsparameters.

Wert	Bezeichnung	Beschreibung
0	**ATDataNone**	nicht verwendet
1	**ATDataByte**	Byte (8-Bit)
2	**ATDataWord**	Ganzzahl-Datenwort mit 16 Bit
3	**ATData DoubleWord**	Ganzzahl-Datenwort mit 32 Bit (Integer)
4	**ATDataReal**	64-Bit-Gleitpunktzahl
5	**ATDataSingle**	32-Bit-Gleitpunktzahl
6	**ATDataBlock**	Benutzerspezifische Datenstruktur, z.B. beim Scanner-Modul
7	**ATDataBoolean**	Boolescher Wert (1 Byte)
8	**ATDataBuffer**	Benutzerspezifischer Datenbuffer (Größe in Byte wird stets in SubIndex angegeben)

2.1.1.4 BusNumber

Mit diesem Parameter wird das Funktionsmodul spezifiziert. Um auf ein Funktionsmodul zugreifen zu können, muss die entsprechende Option in RWMOS.ELF enthalten sein (siehe auch Tabelle 1-1).

Wert	Modul	Option in RWMOS	Beschreibung
100	**PciBusIO**	**optionPCI**	Bus-Master-Zugriffe auf den I/O-Bereich des PCI-Bus
200	**PciBusMem**	**optionPCI**	Bus-Master-Zugriffe auf den Memory-Bereich des PCI-Bus
400	**CanOpenBus**	optionMSM9225	Hardware-Option Can-Open
500	**Interbus**	optionIBSUART	Hardware-Option Interbus-S
1000	**Resourcenbus**	optionRESOURCE	Zugriff auf Systemvariable (Ressourcen)

1100	**Scannerbus**	optionSCANNER	Real-Time-Scanner-Modul
Wert	**Modul**	**Option in RWMOS**	**Beschreibung**
1200	**ElCamBus**	optionELCAM	ELCAM, Gear, Spindelsteigungsfehler- und Winkelfehlerkompensation
1300	**TcBus**	optionTC	Werkzeugradius- und Werkzeuglängen-korrektur

2.1.1.5 DeviceNumber, Index und SubIndex

In diesen Parametern sind die Zugriffsoptionen verschlüsselt. In der Dokumentation des jeweiligen Funktionsmoduls werden diese Parameter beschrieben.

2.1.2 Zugriffe über Object-Deskriptoren

Eine Auswertung der Rückgabewerte nachfolgender Funktionen wird dringend empfohlen. Rückgabewert 4 zeigt die erfolgreiche Ausführung der Funktion an. Bei Rückgabewert 2 muss der Funktionsaufruf wiederholt werden. Rückgabewerte ungleich 4, welche hier nicht aufgeführt sind, zeigen auf jeden Fall einen Fehler an. Folgende Rückgabewerte sind bei allen nachfolgend aufgeführten Funktionen möglich:

Rückgabe-Wert	**Beschreibung**
4	Die Funktion wurde erfolgreich ausgeführt.
-1	Die Option (Bus Number) wird nicht von RWMOS.ELF unterstützt.
1	Es wurde auf ein ungültiges Element zugegriffen oder es wurde eine ungültige Funktionsnummer verwendet.
2	Die Funktion ist nicht bereit (z.B. bei Lesen von WTLSTRB) – System Busy – der Funktionsaufruf muss unter Umständen wiederholt werden.
16 (10 hex)	ungültiger Übergabewert oder Parameter
32 (20 hex)	Das angesprochene Gerät ist nicht angeschlossen.
64 (40 hex)	Ein ungültiger Datentyp wurde verwendet (double).
128 (80 hex)	Der Befehl ist im aktuellen Betriebszustand nicht erlaubt.
256 (100 hex)	Timeout – Die Befehlsausführung wurde wegen Zeit-überschreitung abgebrochen; ein eventuell zurückgelieferter Funktionswert ist nicht verwendbar.
512 (200 hex)	Invalid Handle – Der Datensatz ist nicht mehr gültig!
1024 (400 hex)	Invalid Card – Es wurde auf eine nicht vorhandene Ressource zugegriffen.
2048 (800 hex)	ungültige Achse in den Zugriffsparametern
4096 (1000 hex)	Der benötigte Speicher kann in RWMOS.ELF nicht zur Verfügung gestellt werden.

2.1.2.1 rdOptionInt

BESCHREIBUNG:	Diese Funktion liest eine Integer-Variable vom Universellen Objekt-Interface
BORLAND DELPHI:	function rdOptionInt (var odesc: OptionDescriptorObject; var val: integer): integer;
C:	int rdOptionInt (struct OptionDescriptorObject *odesc, int *val);
VISUAL BASIC:	Function rdOptionInt (odesc As OptionDescriptorObject, val As Long)
RÜCKGABEWERTE:	siehe oben
ANMERKUNG:	Der zu lesende Parameter wird in val zurückgeliefert.

2.1.2.2 wrOptionInt

BESCHREIBUNG:	Diese Funktion schreibt eine Integervariable über das Universelle Objekt-Interface
BORLAND DELPHI:	function wrOptionInt (var odesc: OptionDescriptorObject; var val: integer): integer;
C:	int wrOptionInt (struct OptionDescriptorObject *odesc, int *val);
VISUAL BASIC:	Function wrOptionInt (odesc As OptionDescriptorObject, val As Long)
RÜCKGABEWERTE:	siehe oben
ANMERKUNG:	Der zu schreibende Wert wird in val (value) übergeben.

2.1.2.3 rdOptionDbl

BESCHREIBUNG:	Diese Funktion liest eine Gleitpunktzahl vom Universellen Objekt-Interface
BORLAND DELPHI:	function rdOptionDbl (var odesc: OptionDescriptorObject; var val: double): integer;
C:	int rdOptionDbl (struct OptionDescriptorObject *odesc, double *val);
VISUAL BASIC:	Function rdOptionDbl (odesc As OptionDescriptorObject, val As Double)
RÜCKGABEWERTE:	siehe oben
ANMERKUNG:	Der zu lesende Parameter wird in val zurückgeliefert.

2.1.2.4 wrOptionDbl

BESCHREIBUNG:	Diese Funktion schreibt eine Gleitpunktzahl über das Universelle Objekt-Interface
BORLAND DELPHI:	function wrOptionDbl (var odesc: OptionDescriptorObject; var val: double): integer;
C:	int wrOptionDbl (struct OptionDescriptorObject *odesc, double *val);
VISUAL BASIC:	Function wrOptionDbl (odesc As OptionDescriptorObject, val As Double)
RÜCKGABEWERTE:	siehe oben
ANMERKUNG:	Der zu schreibende Wert wird in val (value) übergeben.

2.1.2.5 rdOptionBuf

BESCHREIBUNG:	Diese Funktion liest ein Datenfeld vom Universellen Objekt-Interface
BORLAND DELPHI:	function rdOptionBuf (var odesc: OptionDescriptorObject; var val: buffer): integer;
C:	int rdOptionBuf (struct OptionDescriptorObject *odesc, void *buffer);
VISUAL BASIC:	Function rdOptionBuf (odesc As OptionDescriptorObject, buffer As Byte)
RÜCKGABEWERTE:	siehe oben
ANMERKUNG:	buffer zeigt auf einen Datenpuffer, der mindestens so groß sein muss wie in SubIndex (Element im Parameter odesc) angegeben. Der zu lesende Datenblock wird auf buffer zurückgeschrieben, d.h., der mit buffer referenzierte Datenbereich muss für die zu lesenden Daten groß genug sein.

2.1.2.6 wrOptionBuf

BESCHREIBUNG:	Diese Funktion schreibt ein Datenfeld über das Universelle Objekt-Interface
BORLAND DELPHI:	function wrOptionBuf (var odesc: OptionDescriptorObject; var val: buffer): integer;
C:	int wrOptionBuf (struct OptionDescriptorObject *odesc, double *buffer);
VISUAL BASIC:	Function wrOptionBuf (odesc As OptionDescriptorObject, val As buffer)
RÜCKGABEWERTE:	siehe oben
ANMERKUNG:	buffer zeigt auf einen Datenpuffer, der mindestens so groß sein muss wie in SubIndex (Element im Parameter odesc) angegeben. Das zu schreibende Datenfeld wird in buffer referenziert.

2.1.3 Informationen zur Handhabung per PCAP

Bei jedem erstmaligen Zugriff nach dem Start eines PCAP-Programmes auf ein Element des universellen Objekt-Interface wird ein Datenbereich für dieses Objekt im Speicher der Steuerung angelegt und ein Handle zurückgegeben. Wenn nun ein PCAP-Programm mehrfach oder sogar zyklisch gestartet wird, dann wird immer mehr Speicher verbraucht. Deshalb sollte sofort am Anfang entsprechender PCAP-Programme die Clean-Funktion für die entsprechende Busnummer aufgerufen werden. Mit dieser Anweisung werden die eventuell vorhandenen Objekte verworfen und der Speicher auf der Steuerung wieder freigegeben. Wenn in diesem Fall auf das entsprechende Objekt-Interface auch in anderen PCAP-Programmen zugegriffen wird, dann werden den anderen Programmen „die Objekte weggenommen". Ein gültiger Zugriff auf das Objekt ist dann nicht mehr möglich, weil das ***OptionDescriptorObject*** nun ein ungültiges Handle enthält. Weitere Zugriffe würden unweigerlich Fehlfunktionen, einen Programmabsturz oder aber eine Exception in Windows oder auf der Steuerung bewirken. Deshalb sollte im entsprechenden Fall darauf geachtet werden, dass Programme, welche das universelle Objekt-Interface verwenden, nicht mehrfach aufgerufen werden können. Dies kann z.B. durch ein Mutex-Handling verhindert werden.

2.2 SAP-Programmierung

2.2.1 Zugriffsvariable

Der Zugriff auf die entsprechenden Funktionen erfolgt über Variable, die per AT-Spezifizierer deklariert werden. Zu allen vorhandenen Modulen sind Include-Files verfügbar, die alle notwendigen Deklarationen beinhalten.

Der Aufbau dieser Deklaration sieht folgendermaßen aus:

var Name: DataType AT %XYBusNr.DeviceNumber.Index.SubIndex;

Die einzelnen Zeichen dieser Zeile haben folgende Bedeutung:

Tabelle: Parameter bei AT-Deklarationen

Zeichen	Beschreibung
Name	Variablenname, über den der Zugriff auf das Objekt erfolgt
DataType	Datentyp z.B. double, integer (wie in der jeweiligen Beschreibung angegeben)
X	Zugriffsart I = Input (lesen) Q = Output (schreiben) M = Input/Output (z.B. bei Scanner-Funktion)
Y	Datentyp der internen Größe B = Byte (Ganzzahl 8 Bit)

	D = Double Word (Ganzzahl 32 Bit) W = Word (Ganzzahl 16 Bit) R = Gleitpunktzahl (64 Bit) S = Gleitpunktzahl (32 Bit) M = DataBlock (Format abhängig vom Befehl, gesondert dokumentiert) Dieser Datentyp muss kompatibel sein zu DataType
BusNr	Bus-Nummer wie bei der Dokumentation des jeweiligen Moduls angegeben, z.B. 1200 für das ELCAM-Modul
DeviceNumber	wie bei der Dokumentation des jeweiligen Moduls angegeben
Index	wie bei der Dokumentation des jeweiligen Moduls angegeben
SubIndex	wie bei der Dokumentation des jeweiligen Moduls angegeben

Beispiel:

```
const G3ResourceBus  = 1000;

// Resource: rp (real position)
var G3R_rp_A1_r:        double  AT %IRG3ResourceBus.2.0.$0;
```

Falls weitere Deklarationen von AT-Spezifizierern notwendig sind, können diese vom Anwender im Quelltext deklariert werden. Wenn beim Zugriff auf einen AT-Spezifizierer ein Fehler auftaucht, wird die SAP-Task mit dem Laufzeitfehler 512 beendet!

2.2.2 Informationen zur Handhabung per SAP

Bei jedem erstmaligen Zugriff nach dem Start eines SAP-Programmes auf eine Variable des universellen Objekt-Interface wird ein Datenbereich für dieses Objekt angelegt. Wenn nun ein SAP-Programm mehrfach oder sogar zyklisch gestartet wird, dann wird immer mehr Speicher verbraucht. Deshalb sollte sofort am Anfang entsprechender SAP-Programme die Clean-Funktion für die entsprechende Busnummer aufgerufen werden. Mit dieser Anweisung werden die eventuell vorhandenen Objekte verworfen und der Speicher wieder freigegeben. Wenn in diesem Fall auf das entsprechende Objekt-Interface auch in anderen Programmen zugegriffen wird, dann werden den anderen Programmen „die Objekte weggenommen“. Diese werden bei Verwendung zwar wieder angelegt, doch wird dafür zusätzliche Ausführungszeit benötigt.