POSITIONING AND CONTOURING CONTROL SYSTEM

APCI-8001 and APCI-8008

Universal Object Interface

Rev. 11/112018

www.addi-data.com

1 Introduction

The “Universal Object Interface” [UOI] represents a universal and flexible software interface for programming the APCI-8001 / APCI-8008. This is a universal approach to the integration of various hardware and software extensions for the APCI-8001 and APCI-8008 positioning and contouring control systems.
The advantage for the user lies in the universal nature of the interface (SAP interface, DLL). User-specific extensions simply require the update of the application programme, and the availability of the functionality in the RWMOS.ELF operating system software.

1.1 The software interfaces for the UOI

The “Universal Object Interface” is supported equally by SAP and PCAP programming methods, and represents a consistent extension of these programming methods that have been tried and tested for 10 years. Users to whom the terms SAP and PCAP are still new, should initially read through the programming manual [PM] for the APCI-8001 and APCI-8008.

1.2 New functions using the UOI

At the moment, the “Universal Object Interface” enables you to make the following hardware and software extensions:

Table 1-1: Possible function extensions of the APCI-8001 / APCI-8008

Interface	Description	Document (PDF)
ELCAM	Electronic Cam: Universal table interpolation	ELCAM Interface
CANOPEN	CANOPEN field bus master (in progress). The OPCAN hardware option is required here.	CanBus Interface
Resources	Access to the internal hardware or software register of the APCI-8001 / APCI-8008 controllers.	Resource Interface
INTERBUS	INTERBUS field bus master. The OPIBS hardware option is required here.	Options Manual
PCI I/O	PCI bus master. The APCI-8001 / APCI-8008 boards provide direct access to other PCI modules in the I/O area, without requiring help from the operating system. This access method provides very fast access, in compliance with strict real-time conditions. Furthermore, it provides a flexible extension to the CNC-I/O level.	PCI Interface
PCI Memory	PCI bus master. Accesses take place via memory areas of other PCI modules.	Resource Interface
Scanner	Management of user-defined lists and scanning of data in the internal memory of the APCI-8001 / APCI-8008 boards, in compliance with strict real-time conditions.	Scanner Interface
TC	Tool radius correction (Tool Compensation)	TC Interface

2 Construction of the “Universal Object Interface”

2.1 PCAP programming

2.1.1 Function access via OptionDescriptorObject

The Universal Object Interface is accessed via the pre-defined data structures or ***OptionDescriptorObject*** records.
For each function that should be used, an OptionDescriptorObject must be created and initialised, both for read and write access. Then integer variables are handled by calling the wrOptionInt or rdOptionInt DLL functions, and floating point numbers are handled by calling the wrOptionDbl and rdOptionDbl functions. For transferring 16-bit and 8-bit variables, also wrOptionInt or rdOptionInt is used. In this case, only the respective part of the parameter val (see below) is considered.

Table 1: Object descriptor elements

Object descriptor element	Description
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 cleaning the respective functionality, the handle must be reset to zero if necessary. Please refer to the documentation for the respective module.
AccessType	Access type: The access type must be entered here before the first use. The valid access types are defined in *ATAccessType* This variable defines whether this is a read or a write operation, for example. For operators where both read and write access are allowed, a separate ObjectDescriptor must be created for each, with the appropriate AccessType.
DataType	Data type: The data type of the variable must be entered here before the first use.
BusNumber	The BusNumber of the respective module is entered here, e.g. 1200 for the ELCAM module.
DeviceNumber	Module-specific variable
Index	Module-specific functions
SubIndex	Module-specific sub-functions

2.1.1.1 Handle

The element *Handle* of the object descriptor objects is initialised at the first use. The value is a variable of the RWMOS operating system software that depends on the runtime. This means, that this value become invalid, as soon as the control is rebooted. After the reboot of the control, all handles and object descriptor elements must be nulled.

2.1.1.2 AccessType

This parameter describes the type of access with which the parameter is used.

Value	Name	Description
0	**ATAccessNone**	Not used
1	**ATAccessInput**	Read access
2	**ATAccessOutput**	Write access
3	**ATAccess InputOutput**	Configuration value, e.g. for the definition of the scanner module

2.1.1.3 DataType

This parameter defines the data format of the access parameter.

Value	Name	Description
0	**ATDataNone**	Not used
1	**ATDataByte**	Byte (8-bit)
2	**ATDataWord**	Integer data word with 16-bit
3	**ATData DoubleWord**	Integer data word with 32-bit (Integer)
4	**ATDataReal**	64-bit floating-point number
5	**ATDataSingle**	32-bit floating-point number
6	**ATDataBlock**	User specific data structure, e.g. at the scanner module
7	**ATDataBoolean**	Boolean value (1 byte)
8	**ATDataBuffer**	User-specific data buffer (size in bytes is always indicated in SubIndex)

2.1.1.4 BusNumber

With this parameter the function module is specified. To be able to access to a function module the corresponding option must be contained in RWMOS.ELF (see also Table 1-1).

Value	Module	Option in RWMOS	Description
100	**PciBusIO**	**optionPCI**	Busmaster accesses to the I/O range of the PCI bus
200	**PciBusMem**	**optionPCI**	Busmaster accesses to the memory range of the PCI bus
400	**CanOpenBus**	optionMSM9225	Hardware option Can-Open
500	**Interbus**	optionIBSUART	Hardware option Interbus-S
1000	**Resourcenbus**	optionRESOURCE	Access to system variable (resources)
1100	**Scannerbus**	optionSCANNER	Real-Time scanner module
1200	**ElCamBus**	optionELCAM	ELCAM, Gear, spindle inclination error and angle error compensation
1300	**TcBus**	optionTC	Tool radius and tool length correction

2.1.1.5 DeviceNumber, Index and SubIndex

In these parameters, the access options are encoded. The documentation of the corresponding function module describes these parameters.

2.1.2 Accesses via Object Descriptors

It is strongly recommended to analyse the return values of the following functions. Return value 4 indicates the successful execution of the function. If return value 2 is indicated, the function call must be repeated. Return values not equal to 4, which are not listed here, indicate an error in any case. The following return values are possible with all of the functions specified below:

Return value	Description
4	The function has been successfully executed.
-1	The option (Bus Number) is not supported by RWMOS.ELF.
1	An invalid element has been accessed or an invalid function number has been used.
2	The function is not ready (e.g. when reading WTLSTRB) – System Busy – it may be necessary to repeat the function call.
16 (10 hex)	Invalid transfer value or parameter
32 (20 hex)	The accessed device is not connected.
64 (40 hex)	An invalid data type has been used (double).
128 (80 hex)	The command is not allowed in the current operating state.
256 (100 hex)	Timeout – The execution of the command has been cancelled due to a timeout; a possibly returned function value is not usable.
512 (200 hex)	Invalid Handle – The data record is no longer valid!
1024 (400 hex)	Invalid Card – An unavailable resource has been accessed.
2048 (800 hex)	Invalid axis in the access parameters
4096 (1000 hex)	The required memory cannot be allocated in RWMOS.ELF.

2.1.2.1 rdOptionInt

DESCRIPTION:	This function reads an integer variable from the Universal Object 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)
RETURN VALUE:	see above
NOTE:	The parameter to be read is returned in val.

2.1.2.2 wrOptionInt

DESCRIPTION:	This function writes an integer variable via the Universal Object 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)
RETURN VALUE:	see above
NOTE:	The value to be written is returned in val (value).

2.1.2.3 rdOptionDbl

DESCRIPTION:	This function reads a floating point number from the Universal Object 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)
RETURN VALUE:	see above
NOTE:	The parameter to be read is returned in val.

2.1.2.4 wrOptionDbl

DESCRIPTION:	This function writes a floating point number via the Universal Object 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)
RETURN VALUE:	see above
NOTE:	The parameter to be written is returned in val (value).

2.1.2.5 rdOptionBuf

DESCRIPTION:	This function reads a data field from the Universal Object 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)
RETURN VALUE:	see above
NOTE:	buffer points to a data buffer that must be at least as big as indicated in SubIndex (element in the odesc parameter). The data block to be read is written back to buffer, i.e. the data range referenced with buffer must be big enough for the data to be read.

2.1.2.6 wrOptionBuf

DESCRIPTION:	This function writes a data field via the Universal Object 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)
RETURN VALUE:	see above
NOTE:	buffer points to a data buffer that must be at least as big as indicated in SubIndex (element in the odesc parameter). The data field to be written is referenced in buffer.

2.1.3 Information about the handling per PCAP

At each first access to an element of the universal object interface, after starting a PCAP program, a data range is determined for this object in the memory of the control and a handle is returned. If a PCAP program is started various times or even cyclically, more and more memory is used. Thus, right at the beginning of the respecting PCAP programs, the clean function shall be called for the adequate bus number. With this command, possibly present objects are taken out and the memory of the control is released. If in this case, the respecting object interface is also access via other PCAP programs, then the “objects are taken away” from other programs. Then a valid access to the object is not possible anymore because now the ***optionDescriptorObject*** contains an invalid handle.
Further access would lead to error functions, a program crash or to a Exception in Windows or in the control. Therefore, in this case, it must be observed, that the programs, which use the universal object interface, are not called various times. This can be avoided e.g. via a mutex-handling.

2.2 SAP programming

2.2.1 Access variable

The corresponding functions are accessed via variables that are declared via AT specifiers. Include files, which contain all necessary declarations, are available for all existing modules.

The declarations are constructed as follows:

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

The individual characters of this line have the following meaning:

Table: Parameters in AT

Characters	Description
Name	Name of the variable, via which the object is accessed
DataType	Data type, e.g. double, integer (as specified in the respective description)

X	Access type
	I = Input (read)
	Q = Output (write)
	M = Input/Output (e.g. for scanner function)

Y	Data type of the internal variable
	B = Byte (integer 8-bit)
	D = Double Word (integer 32-bit)
	W = Word (integer 16-bit)
	R = Floating point number (64-bit)
	S = Floating point number (32-bit)
	M = Data block (format depends on the command, separately documented)
	This data type must be compatible with DataType

BusNr	BusNumber, as specified in the documentation for the respective module (e.g. 1200 for the ELCAM module).
DeviceNumber	As specified in the documentation for the respective module
Index	As specified in the documentation for the respective module
SubIndex	As specified in the documentation for the respective module

Example:

```
const G3ResourceBus  = 1000;

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

Include files, which contain all necessary declarations, are available for all existing modules. If an error occurs when an AT specifier is being accessed, the SAP task is ended with runtime error 512.

2.2.2 Information about the handling with SAP

At each first access after the start of a SAP program to a variable of the universal object interface, a data range will be constructed for this object. If a SAP program is started more fold or even cyclically, more and more memory is needed. Thus, right at the beginning of the calling of respecting SAP programs, the Clean function for the respecting bus number shall be called. With this command, the possibly present objects will be ignored and the memory will be released. If, in this case, the respecting object interface is also accessed in other programs, then the “objects are taken away” from the other programs. At the application they will be constructed newly, but this requires additional execution time.