POSITIONING AND CONTOURING CONTROL SYSTEM

APCI-8001 AND APCI-8008

PROGRAMMING AND REFERENCE MANUAL / PM (PART 1)

Last updated: 31/03/2021, from disk V2.53VP
Rev. 15/052022

www.addi-data.com

1 Introduction

What is the content of this manual?

This manual contains all the details you will need for programming the APCI-800x controllers. The complete documentation is divided into 3 parts: OM (Operating Manual), PM (Programming and Reference Manual) and CM (Commissioning Manual).

Which boards belong to the APCI-800x family?

The APCI-800x family includes positioning and contouring control systems of the third generation, i.e. the boards APCI-8001, APCI-8008, APCIe-8008 and APCIe-8008-EC. Other boards are being planned.

Further remarks

If the functions described in this manual do not apply to all devices of the APCI-800x family, they are specially marked. In this case, the respective function only applies to the marked device!
Before the various programming methods and operating modes can be presented, we must first describe various functions provided by the *rw_MOS* operating system software. You will find further information on *rw_MOS* in the Operating Manual, Chapter 4.1.

2 Internal details of the *rw_MOS* operating system software

As already mentioned in the Operating Manual, one of the main factors in the performance capabilities of the APCI-800x controllers is the *rw_MOS* operating system software. The following chapters will describe the functions implemented in *rw_MOS,* like profile generation or limit switch handling.

2.1 The APCI-800x position controller

The basic operating mode of the APCI-800x controllers is the position control mode. In this operating mode, the board attempts to keep the motor position in the setpoint position. The control loop usually consists of the following components: digital controller - digital/analog converter - power section - motor - encoder - pulse acquisition. The encoder is in most cases attached directly to the motor, i.e. rigidly connected to the motor axis.
If this is not the case, the transmission elements between motor axis and encoder axis are also incorporated in the control loop. The load is also connected to the motor axis. The response of the control system is determined by all the elements contained in the control loop and by the load. In any given system, the control response can be influenced only by the filter parameters of the digital filter. Remember that all possible operating cases (e.g. changes in load) have to be allowed for.

2.1.1 Control loop opened/closed

After power-up, the control loop is at first open. The value 0 is outputted on the manipulated variable output (Motor-Command-Port). The connected axis can be traversed in uncontrolled mode by outputting a value. The PCAP command *cl()* (close loop) is used to close the control loop. Note that the current position is adopted as the setpoint position, in order to prevent the motor axis being traversed unintentionally. Traversing profiles cannot be carried out until the position control has been activated. This also applies for stepping motors.

2.1.2 PIDF filter

The digital filter has the structure of a real PIDF filter. Almost all controlled systems encountered in practice can be stably adjusted with this type of controller.

2.1.2.1 The filter parameters K_D, K_I, K_P

The setting procedure utilizes the filter parameters K_D , K_I and K_P. The significance of these parameters can be very simply understood in terms of the common parameters encountered in the literature: proportional amplification K_P , derivative-action time K_D and integral-action time K_I.

- K_P - Proportional amplification
- K_I - Integral-action coefficient
- K_D - Derivative-action coefficient
- T_V - Derivative-action time
- T_N - Integral-action time

$K_I = K_P / T_N$
$K_D = K_P * T_V$

If a controller with a different structure is to be implemented, the individual components involved can be simply de-activated by setting them to zero.

2.1.2.2 Additional phase element

The digital PIDF filter provided is in the standard version cascaded with a first-order time-delay element with a time constant of $T_A/2$ (half the scan time). This is why it is referred to as a real PIDF filter. The filter parameter K_{PL} can now be used to reduce this time-delay still further, thus making a harder controller setting possible. The K_{PL} parameter may in theory assume any value between 0 and 1. In practice, however, a value greater than approx. 0.95 is no longer expedient.

The connection between K_{PL} and the time delay can be simply represented as:

K_{PL} - Filter parameter
T_{DELAY} - real time-delay of the PIDF filter
T_A - Scan time

$$T_{DELAY} = (1 - K_{PL}) * T_A / 2$$

2.1.2.3 Scan time

In the paragraph above, the scan time T_A was used: this is a characteristic variable for the digital controller. The scan time is the time after which setpoint and actual values are each scanned and the command value is computed using the control algorithm. If the scan time is small compared to the system time constants involved, the controller can be dimensioned like a continuous controller. This means that no special knowledge of digital control engineering is required for adjustment purposes.

Note: In the APCI-800x standard version of the controller boards, the scan time has been set to **1.28 ms**.

2.2 The APCI-800x profile generator

When traversing with the individual axes, the specified paths are approached with a trapezoidal speed profile. For a trapezoidal speed profile of this kind, the determinant variables are initial velocity, initial position, acceleration, maximum velocity, target position and target velocity. The profile generation feature under discussion here generates the appropriate setpoint values for the position controller [chapter 2.1] synchronously with the scans, so that starting from the current position the axis accelerates from the current velocity up to the maximum velocity. The initial velocity and initial position are instantaneous values and are not specified as parameters for a motion profile. Before the target position is reached, the profile generator decelerates in good time with the specified deceleration, so that the target velocity is reached in the specified target point.

2.2.1 Profile generation for JOG commands

There are certain special cases possible when running a trapezoidal speed profile with JOG traversing commands (single-axis movements):

- The initial velocity is negative in relation to the traversing direction. This means that the axis is initially traversing in the wrong direction, but decelerates, reverses and now accelerates in the right direction.
- The final velocity is negative in relation to the traversing direction. The axis initially moves beyond the target point, decelerates, reverses its direction and has the target velocity when it reaches the target point again.
- The initial velocity is equal to the maximum velocity.
- The initial velocity is higher than the maximum velocity. In this case, the axis is automatically decelerated to the maximum velocity.
- The final velocity is equal to the maximum velocity.

- The maximum velocity is not reached, because the axis has to be decelerated beforehand in order to reach the target velocity by the time it gets to the target position. In this case, a triangular speed profile is run.

All these cases will be correctly handled if the distance to be traversed is sufficient. In addition, a positive maximum velocity and acceleration must always be specified and the final velocity must be smaller than or equal to the maximum velocity. When a negative acceleration is stated, this will be utilized for the profile's braking ramp.
With JOG traversing commands, it is thus possible to program acceleration ramps and braking ramps with differing degrees of steepness.

In the cases listed below, velocity jumps occur (undesirably high accelerations). If these cannot be implemented by the system, a position error will occur, which will, however, generally be corrected after a limited time period. When stepping motors are used, these cases cannot usually be permitted.

- The traverse distance specified is not sufficient for deceleration.
- The target velocity is higher than the maximum velocity. In this case, the traversing velocity is set to target velocity at the end of the profile.

2.2.2 Profile generation for MOVE commands

When running a trapezoidal speed profile with MOVE traversing commands (multiple-axis movements with interpolation) with one or more than one axis, the following special cases are possible:

- The final velocity is negative in relation to the traversing direction. The system first traverses beyond the target point, decelerates, reverses direction and when it reaches the target point again possesses the target velocity.
- The initial velocity is equal to the maximum velocity.
- The initial velocity is higher than the maximum velocity. With direct MOVE commands, the system automatically decelerates down to maximum velocity in this case. With spooler commands, the initial velocity is set to maximum velocity. This corresponds to a velocity jump.
- The final velocity is equal to the maximum velocity.
- The maximum velocity is not reached, since the system must decelerate beforehand in order to reach the target velocity by the time the target position is reached. In this case, a triangular speed profile is run.

All these cases are handled correctly if the traverse distance is sufficient in each case. Furthermore, a positive maximum velocity and acceleration must always be stated and the final velocity must be smaller than or equal to the maximum velocity. If a negative acceleration or negative maximum velocity is stated, the profile will be discarded.

With MOVE traversing commands, it is not possible to program acceleration ramps and braking ramps with differing degrees of steepness in one traversing command. Should this be required, you can, however, program several MOVE commands consecutively.

In the cases listed below, velocity jumps are involved, i.e. unwantedly high accelerations. If these cannot be implemented by the system, a position error will occur, which will, however, generally be corrected again after a limited time period. These cases must not as a rule be permitted in conjunction with stepping motors.

- The traverse distance stated is not sufficient for accelerating up to target velocity. In this case, the target velocity is set to a value which can actually be reached within the profile stated. In this case, there will however be no velocity jump.
- The traverse distance stated is not sufficient for deceleration. In this case, the profile's initial velocity is set to a value which permits deceleration down to final velocity within the profile stated.
- The target velocity is higher than the maximum velocity. In this case, the traversing velocity is set to target velocity at the end of the profile.

- The traversing profile's direction is altered. In this case, the amount of the velocity vector is taken from the previous direction and placed in the direction now to be traversed. In this case, there will be velocity jumps of varying magnitude at the axes involved. Special caution is required here when stepping motor systems are used.

This type of profile generation is not only executed when linear MOVE commands are being run. This pattern is also used for generating the trajectory velocity when running circular movements with two axes.

2.2.3 Acceleration

If an acceleration smaller than zero is stated, then the data record is discarded with MOVE commands. With JOG commands, a negative acceleration specifies the steepness of the braking ramp. As a default, the braking ramp and the acceleration ramp are of identical steepness. The units for the acceleration can be axis-specifically stated in the *mcfg.exe* utility program. For the interpolation commands (MOVE commands) there are various options for selecting the units. The value for acceleration is specified as a floating-point number, meaning that the value range is almost unlimited. If you specify an acceleration higher than the system can implement, an enlarged position error will be produced during the acceleration phase.

2.2.4 Maximum velocity

The maximum velocity must always be specified as greater than zero, otherwise the data record will be rejected (MOVE commands) or an endless profile will be run in the wrong direction (JOG). The units for the maximum velocity can be axis-specifically specified in the *mcfg.exe* utility program. For the interpolation commands there are various options for selecting the units. The value for the maximum velocity is specified as a floating-point number, meaning that the value range is almost unlimited. If you specify a velocity higher than the system can implement, an enlarged position error will be produced during traversing. If the maximum velocity specified is smaller than the initial velocity, the conditions mentioned above shall apply, depending on the command type involved.

2.2.5 Target velocity

The target velocity can be specified as positive, negative or set to 0. The direction of the target velocity is always referenced to the direction of traversing. If traversing is in a negative direction and the target velocity is positive, this means the system will continue to move in a negative direction. The target velocity has the same unit as the maximum velocity. The value is specified as a floating-point number, meaning that the value range is almost unlimited. If you specify a velocity higher than the system can implement, an enlarged position error will be produced during traversing. If the target velocity specified is greater than the maximum velocity, the traversing profile will be concluded with a velocity jump. The current velocity will in this case be set to the target velocity at the end of the profile.

2.2.6 Velocity correction

In certain cases, you may want to alter the axis or trajectory velocity during execution of a trapezoidal speed profile. A typical example of this is manual velocity correction (override). You have various SAP and PCAP commands available for this purpose.
The velocity correction factor, whose default value is 1.0, acts on velocities and accelerations alike.
According to the operating mode it is important to differentiate the way the override is used by programming: For one-axis traversing commands (JOG commands) the JOG override can be separately programmed for each axis. Yet it must not be made by interpolation travel. For interpolation commands (MOV commands) the trajectory override is to be set and taken over with the command utrovr synchronously for all axes, which take part to the interolation travel. The synchronisation of the axes to be interpolated can only be ensured this way. When the trajectory override is taken over, the value is automatically accepted as a JOG override by the working axes. (can be switched off). To avoid velocity jumps during the programming of the override, an adjustment time can be programmed for the trajectory-override.

2.2.7 Target position / Traverse distance

The target can be specified as a relative or absolute value. If you specify a relative value, traversing will be by the distance specified, i.e. you have programmed a traverse distance. If you specify an absolute value, the system will traverse to the position specified, i.e. you have programmed a target position. The reference point for absolute target positions is the machine zero.

2.2.8 Operating modes for command processing

Traversing commands and other commands can be executed in two different operating modes, the "direct mode" and the "spool mode". The operating mode being implemented at any time is automatically specified by the syntax of the command involved.

Note: The command abbreviations for the *spool* commands are distinguished from the direct commands by the character 's' as the first letter in the command word. There are identical *spool* commands available for both programming methods, SAP and PCAP programming alike.

2.2.8.1 Direct mode

Direct mode is activated automatically by calling special *move* and *jog* commands. When you program a traversing command in direct mode, the program begins to execute the specified command after a system-entailed time-delay (approx. 2 - 3 scan intervals). A profile which is already running will not be run till its end: the instantaneous values for velocity and position will be accepted as initial values for the current traversing command. If the profile data and the initial values are consistent, i.e. comply with the above requirements, a currently running profile will be seamlessly continued.
It is thus possible, for example, to alter the target point of a running profile, to increase the velocity again, to subsequently alter the deceleration of the braking ramp, or even alter the acceleration during an acceleration ramp. If different profiles are to be run in succession, you have to wait for the end of the profile concerned in each case.

Note: Any data present in the spooler will be rejected when commands are executed in direct mode.

2.2.8.2 Spool mode

In spool mode, a large number of traverse or other commands can be entered in a queue (spooler). Each axis has its own spooler. Once an interpolation command has occurred, the respective spoolers are synchronously loaded and processed. Processing of the commands entered in the spooler is started by the PCAP command *ssms(),* for example. During processing, you can write further commands into the spooler. Commands from the spooler are processed one after another without any time-delay. The free spooler area becomes smaller each time a command is entered, but becomes larger again every time a command is executed. When all commands in the spooler have been processed, the system automatically switches back to direct mode, i.e. after more *spool* commands have been entered, their processing has to be started anew.

Note: For the spooler entries to be processed correctly, the following conditions are to be satisfied:

- All axes for which commands are to be spooled must be in position control at the first spooler entry.
- The velocity of these axes must be zero before the first spool command is executed, which is why the Start Spooled Motions Synchronized ssms() command may be executed only when all axes involved are at rest.
- Traverse profiles in the spooler need an execution time that is higher than the scan time of the control (default: 1.28 ms). The execution time of a traverse profile is calculated (approx.) by way / velocity. Shorter traverse profiles must be suppressed by the application program.

2.2.8.3 Additional notes on spooler operation

In order that a contour programmed with spooler commands can be run on an accurate path, within a command sequence, all axes must always be programmed and started synchronously. Furthermore, any override value must always be taken over synchronously for all interpolation axes (utrovr command). As soon as the spooler operation is interrupted by an asynchronous operation, it can be expected that the programmed contour is not complied with. Automatic spooler synchronisation monitoring can be used to detect this type of error. This is available from RWMOS V2.5.3.88.
If an asynchronous spooler operation is detected by the operating system, the SAF (#19) bit is set. If the JSatSAF (#28) bit is set in the MODEREG register, in this case, all axes are stopped via Jog-Stop using the programmed stop deceleration. This error event is indicated by the corresponding saf Flag in axst and bit 19 in ErrorReg (see also Chapter 4.4.63.1).
By calling a Jog or direct Move command, SAF flags potentially set are deleted again.

2.3 Interpolation with APCI-800x

Individual axes are moved with the board APCI-800x using the *jog* commands. The *move* commands are available for moving more than one axis in interpolated mode. The APCI-800x boards enable you to perform circular, linear and helical interpolations. It can process several interpolation profiles simultaneously, with any initial and final velocities you want. All interpolation computations are synchronized with the scan function (1.28 ms).

2.3.1 Linear interpolation

With linear interpolation, any desired number of axes are moved on a line of space (n-dimensional) from the starting point to the target point (absolute positioning) or by a space vector (relative positioning). Parameters used in linear interpolation are the axes involved, the traverse distance or the target position, the trajectory acceleration, the maximum trajectory velocity and the trajectory target velocity. When interpolating with an initial velocity, you should make sure that the direction vectors for the initial velocity and for the interpolation profile coincide. Otherwise the direction of the velocity vector will be altered and this may lead to velocity jumps at the axes involved. If the interpolation direction has to be altered from one profile to the next, an intermediate stop should be made. For direction reversal, there is an option for ending the first profile with negative target velocity.

2.3.1.1 Formal linear interpolation

When running contours, one axis can remain in the instantaneous motor position, while the other axes are run in interpolated mode. This stationary axis can, however, participate formally in this interpolation for the other axes and thus remains synchronized with them. This formal interpolation is particularly important in the spool operating mode and is selected automatically for all axes at which a traverse distance of 0 is programmed.

2.3.2 Circular interpolation

Circular interpolation is performed with any two axes. Parameters used for circular interpolation are the axes involved, the coordinates of the circle's centre, the traverse angle (positive or negative), the trajectory acceleration, the trajectory maximum velocity and the trajectory target velocity. The coordinates of the circle's centre can be specified in absolute or relative coordinates.
When interpolating a circle with an initial velocity, you must always make sure that the initial velocity has the direction you want, i.e. the direction of the tangent in the circle's starting point. Otherwise the direction of the velocity vector will be altered and this may lead to velocity jumps at the axes involved. If the interpolation direction has to be altered from one profile to the next, an intermediate stop should be made. For direction reversal, there is an option for ending the first profile with negative target velocity.

2.3.3 Helical interpolation

Helical interpolation is executed for any two axes as a circular interpolation and with any third axis as a linear interpolation.

2.3.4 Surface area processing

For interpolation commands the trajectory parameters speed and acceleration are defined with the system parameters PositionUnit (PU) and TimeUnit (TU). The conditiotion is that axes of the same type (translatory or rotatory axes) always take part to the interpolation travel. This is to ensure that the PositionUnit is processed appropriately. When only rotatory axes are involved in translatory interpolation travel as for example by processing cylinder surface. the effective radius must be defined. It will then enable the conversion of the rotatory axes values in translatory interpolation values.
For this, the axis-specific value *effradius* is available. The trajectory speed and acceleration can be set correctly. The radius is given in the unit set in the linear interpolation. This setting is possible for linear, circular as well as helical interpolation.

Example:

The surface of a pipe must be welded. This pipe has a diameter of 200mm and is turned with an axis C defined as rotaroryt.

```
PU := 0;                // Position Unit = mm
C.effradius := 100;     // Enter radius
```

The axis C kann nows be used in the linear interpolation:

```
mlr (X := 25, C := 60);
```

The traverse distance of the rotatory axis is given in the translatory position unit (here mm). In case the traverse distance of the rotary axis must be given in the axis-specific rotatory unit (e.g. deg), the Bit 10 must set in the register MODEREG (see chapter 6.3.1.5). The conversion can be made simultaneously for all axes.

2.3.5 Synchronous and asynchronous interpolations

One of the options provided by the APCI-800x board is to process several different interpolations at the same time. It is possible, for example, to execute two circular interpolations with two different axis channels each. The interpolations concerned can be executed synchronously or asynchronously with each other. The synchronous operating mode is supported particularly well by the spooler mechanism. Of course, besides an interpolation, any other axis you want that is not used in an interpolation context can be run independently.

2.4 APCI-800x limit switch handling

The APCI-800x board offers a wide range of options for limit switch handling and traversing range limitation. You have options, for example, for configuring any one or more digital inputs as left or right hardware limit switches. During configuration, a TOM, SMA or SMD function is additionally assigned to the limit switch input. What's more, you can additionally define a software limit switch (left and right) for each axis channel. You can select any limit switch positions you want. Here too, you can choose between the TOM, SMA and SMD functions. The state of the limit switches can be taken from the *axst* status flag.
A particular limit switch state is erased if the setpoint position is below the limit switch position.

Note: All limit switch states are erased when the control loop is closed [chapter 4.4.6 - *cl()*].

2.4.1 TOM limit switch function (Turn-Off-Motor)

With this limit switch function, the motor is turned off in the limit switch direction, i.e. the axis comes to rest in uncontrolled mode when the limit switch is tripped and cannot be moved further into the limit switch zone, only against the limit switch direction. The setpoint position can, however, continue to run into the limit switch zone, e.g. due to a profile currently being run. When it exits from the limit switch zone, uncontrolled velocity jumps may occur.

2.4.2 SMA limit switch function (Stop-Motor-Abruptly)

With this limit switch function, the setpoint position is retained when the limit switch position is exceeded. The position controller halts the axis in this position. The setpoint position computed by the profile generator will, however, be correctly continued internally. When the setpoint value position leaves the limit switch zone, uncontrolled velocity jumps may occur.

2.4.3 SMD limit switch function (Stop-Motor-Decelerate)

With this limit switch function, the axis concerned is decelerated with the stop deceleration {*sdec*} specified down to zero velocity. The axis is switched to direct mode and any spooler entries are discarded. It is no longer possible to perform further controlled traversing into the limit switch area. The axis can be moved out of the limit switch area with all traversing commands. This is the multi-purpose limit switch function.

2.5 Other function groups

2.5.1 Application-specific system variables

In the RWMOS system software, system-specific variables may be available. These variables are adapted and documented specifically to requirements and may be available in any number. For such variables, there are access mechanisms for integer and floating point values:

For this, in the PCAP programming environment, there are the read functions rdSysVarInt and rdSysVarDbl, and the write functions wrSysVarInt and wrSysVarDbl.
In the stand-alone programming environment, access to the system variable SysVarInt[] or SysVarDbl[] is indexed.

```
CI0 := SysVarInt[1];
```

2.5.2 Application-specific axis variables

In the RWMOS system software, axis-specific variables may be available. These variables are adapted and documented specifically to requirements and may be available in any number. For such variables, there are access mechanisms for integer and floating point values:

For this, in the PCAP programming environment, there are the read functions rdAxVarInt and rdAxVarDbl, and the write functions wrAxVarInt and wrAxVarDbl.

In the stand-alone programming environment, access is indexed by means of the axis qualifiers varint[] or vardbl[].

```
CI0 := A3.varint[1];
```

3 APCI-800x Programming methods

One of the important features of the APCI-800x positioning and contouring control system is the real-time multi-task operating system *rw_MOS* (*M*ips *O*perating *S*ystem).
This is contained in the *rwmos.elf* file and is loaded, once per PC boot, into the local main memory of the APCI-800x board within a few seconds, using the *mcfg.exe* boot menu or a user program.
The *rw_MOS* operating system software is divided up into various tasks, which basically provide for two different kinds of user programming.

Note: *rwmos.elf* and *mcfg.exe* form part of the APCI-800x TOOLSET software. You will find further information in the Operating Manual.

3.1 PC application programming (PCAP programming, or direct programming)

The APCI-800x application programming (PCAP) is handled with a user program running on the PC. Programs are written using a higher-level programming language like *Borland C*, *Microsoft C, Borland Delphi* or *Microsoft Visual Basic*. By using the function libraries included in the scope of delivery for these programming languages, you can draw on a powerful reservoir of commands, enabling you to create your programs quickly and effectively. The commands available include traversing commands, for example, with and without interpolation, input/output commands, interrogation commands, *spool* commands, etc.
A typical application program transmits one or more of these commands to the APCI-800x board and then waits for these orders to be processed. After the commands concerned have been autonomously executed by the *PC-Task* in the *rw_MOS* operating system, new command orders can be transferred to the *PC-Task*. The time between command order and command processing can be utilized by the application program to perform other application-specific tasks.
Since programming is performed by directly accessing a PC application program, this programming method is also referred to as "PC direct programming".

Note: In the following chapters, you will occasionally find the term "PCAP command". This type of command is based on the programming method outlined above.

3.2 Stand-alone application programming (SAP programming)

In contrast to PC application programming, stand-alone application programming permits a program to be processed entirely without the aid of a PC application program. An application program written in the *rw_SymPas* programming language is compiled using the *NCC* compiler integrated in the development environment *mcfg.exe* or the commando line compiler *ncc.exe and generates* an operating program which the APCI-800x board can understand.
This operating program can be loaded onto the APCI-800x board and is executed autonomously using the *CNC-Task* (CNC = Computerized Numerical Control) in *rw_MOS*. If synchronization is required between a PC application program and the APCI-800x board stand-alone program, this can be carried out using predefined system variables, which both system partners (PC and APCI-800x board) can access.

Note: In the following chapters, you will often encounter the term "SAP command". This type of command is based on the programming method outlined above.

3.2.1 SAP-Multitasking

The operating system software *rw_MOS* can process up to 4 SAP programs simultaneously. All tasks executed simultaneously have the same priority. The different tasks are addressed by means of numbers. The smallest task number has the value of 0 and the largest thus the value of 3.
Multitasking programming enables a complex task to be divided up into small, easy-to-handle subtasks. For example, one task could be used for reference travel, another for monitoring the drive with appropriate EVENT handlers and yet another for PLC control pure and simple, with appropriate accessing of digital I/O or PC communication with predefined registers.
The various SAP programs can autonomously stop, start or continue by means of various task control commands.
The CNC tasks are synchronized with each other, synchronization with any parallel-running PCAP application program and exchange of data between these, can be carried out using predefined registers, what are referred to as COMMON variables. 1,000 common integer and 1,000 common floating-point registers are available to all CNC tasks for this purpose.
Each CNC task can also utilize a local memory area of 1,000 bytes (COMMON BUFFER), which the PC and the CNC task involved can access in both read and write modes. This can be used to build up a user-specific command set, for example.

4 PC application programming

4.1 Introduction

The APCI-800x TOOLSET Software includes library functions for the programming languages *Borland Delphi,*
C (e.g. *Borland C++Builder, Microsoft Visual C++*) and *Microsoft Visual Basic*. These are programming tools for the Windows platforms Windows 95, 98, Me, Windows NT 4.0, Windows NT Embedded 4.0, Windows 2000, XP, Vista and Windows 7. The individual functions of the high-level language libraries are executed by using the system driver *mcug3.dll*. The meaning of the individual function parameters and their data types is identical for all programming languages listed above.
Integration of the function libraries into the programming language involved is explained below:

Programming language	Use description
Borland Delphi	The name of the function library is *mcug3.pas*. These functions are used to establish the link between the PC application program and the system driver *mcug3.dll*. This file is declared as a *unit* and is linked to the application program by means of the *uses* statement. Important: Various system parameters possess the data type *double*. This means that the user program has to be compiled with the *{$N+}* option!
C* (Borland *C, Microsoft C or others)	The function library's name is *mcug3*.lib. These functions are used to establish the link between the PC application program and the system driver *mcug3.dll*. The Lib files are available for various C programming tools and are to be linked with the application program. The file *mcug3.h* contains the function declarations. It should be incorporated in the application program by using the *#include*-instruction.
Microsoft Visual Basic	The name of the function library is *mcug3.bas*. The link between PC application program and the system driver *mcug3.dll* is created by the functions declared in *mcug3.bas*. This file is available as a basic module and can be inserted in the project environment of the application program.

4.2 Example programs for using the function libraries

The example programs included in the APCI-800x TOOLSET software show simple applications for the functions described below. The source texts for the example programs are provided with comments to render them self-explanatory. So there is no need for a detailed description of these example programs at this point. The individual example programs for the two programming languages can be found in the subdirectories specified here and have the following names:

Programming language	Sub-directory	Files
Borland Delphi	**Delphi**	*mcug3.pas*, *ld.pas, move.pas* etc.
Borland C++ Builder	**C** **C/Borland**	*mcug3.h*, *ld.c, move.c* etc. *mcug3.lib*
Microsoft Visual C++	**C** **C/mvc**	*mcug3.h*, *ld.c, move.c* etc. *mcug3.lib*
Microsoft Visual Basic	**Vb**	*mcug3*.bas, *ld.bas, move*.bas etc.

4.3 Definitions, structures and records

Before the individual functions are explained, certain definitions, structures and records will be described, some of which are required as parameters for these functions. The structure/record data fields required are always declared in the application program. The advantage of this is that the system driver does not take up too much PC RAM memory and that several PC applications can access the APCI-800x controllers at the same time.
All the structure/record types and system constants listed below have been defined in the *mcug3.h, mcug3.pas or mcug3.bas* files using the programming languages mentioned above.

4.3.1 Definitions

Table 1**: System constants**

Name	Type	Function
MAXAXIS	integer	Maximum number of possible axes. Currently, the TOOLSET software supports up to 18 axes. **Warning:** This value must not be modified!
LONGINT	integer int long	Synonym for the data type int or integer in the *C* or *DELPHI Pascal* programming language and longint in the *Microsoft Visual Basic programming language*.

4.3.2 Structures, records and types

Depending on the programming language involved, we speak either of structures *(C),* records *(Pascal)* or types (*Visual Basic*). The composition and the functioning of these data types is identical in all programming languages. In the description below the term structure or record type is used. For easier comprehension, all structure or record types are written in capitals and their components in lower-case characters.

4.3.2.1 Structure/record type AS

Table 2**: Structure/record type AS**

Element	Type	(Abbreviation meaning), Function
unoa	LONGINT	(used number of axis) Number of axes to be selected at various function calls.
san	Field with MAXAXIS LONGINT	(selected axis number) Field of the axes to be selected. This field must be initialized beginning with Index 0, depending on the number of axes used.

Note: Counting for axis channels begins with the value 0.

Example: Selecting the first and third axes

```
as.unoa = 2;        // number of axes
as.san[0] = 0;      // first axis
as.san[1] = 2;      // third axis
```

4.3.2.2 Structure/record type TSRP

A structure/record type *TSRP* has to be declared for each axis to work with the individual axis systems. Using the structure/record elements contained in TSRP, data are exchanged with the APCI-800x board at various PCAP commands. For example, axis-specific system variables like accelerations, velocities and positions can be interrogated or set using special read and write commands.
Important: The individual elements of the TSRP structure are not initialized automatically, i.e. you have to update them by setting them directly and reading them in beforehand.
Note: You have to make sure that, when more than one axis channel are used, the *TSRP* structures/records are located directly behind each other in memory, since the system driver *mcug3.dll* sometimes accesses the various axis parameters using address computations. Therefore the data alignment has to be defined on 4 bytes if necessary. Correct arrangement in the PC's main memory is reliably achieved by declaring TSRP as a field variable.
The size of the field is to be defined for the MAXAXIS axes.

Before use, this data structure must have been initialised. The initialisation is done, e.g. with the commands InitMcuSystem, InitMcuSystem2 or InitMcuSystem3. In the most cases an instance of this data structure for each control in the system is defined globally and is initalised when calling the program or after booting of the control. A use of locally declared instances without previous initialisation is not allowed and can lead to unexpected error functions.

Table 3: Structure/record type TSRP (axis-specific parameters)

Element	Type	(Abbreviation meaning), Function
an	LONGINT	(axis number)
kp	double	(PIDF filter parameter kp)
ki	double	(PIDF filter parameter ki)
kd	double	(PIDF filter parameter kd)
kpl	double	(PIDF filter parameter kpl)
kfca	double	(PIDF forward compensation acceleration)
kfcv	double	(PIDF forward compensation velocity)
jac	double	(jog acceleration)
jvl	double	(jog velocity)
jtvl	double	(jog target velocity)
jovr	double	(jog override)
hac	double	(home acceleration)
hvl	double	(home velocity)
rp	double	(real position)
dp	double	(desired position)
tp	double	(target position)
sll	double	(software limit left)
slr	double	(software limit right)
ipw	double	(in position window)
mpe	double	(maximum position error)
gf	double	(gear factor)
mcp	LONGINT	(motor command port)
axst	LONGINT	(axis status)
lsm	LONGINT	(left spool memory)
epc	LONGINT	(eeprom programming cycle)
digi	LONGINT	(digital inputs)
digo	LONGINT	(digital outputs)
ifs	LONGINT	(interface status)

Element	Type	(Abbreviation meaning), Function
scratch	Field with 4 times LONGINT	(scratch field) wildcard for next TSRP record

4.3.2.3 Structure/record type TRU (Trajectory Units)

This structure or record type is a parameter for the PCAP command *ctru()*.

Table 4**: Structure/record type TRU**

Element	Type	Abbreviation meaning/Function
pu	LONGINT	position unit
tu	LONGINT	time unit

4.3.2.4 Structure/record type LMP (Linear Motion Parameters)

This structure or record type is a parameter with all linear interpolation commands.

Table 5: Structure/record type LMP

Element	Type	(Abbreviation meaning), Function
ac	double	(acceleration) trajectory acceleration
vl	double	(velocity) trajectory velocity
tvl	double	(target velocity) trajectory target velocity
dtm	Field with MAXAXIS double	(distance to move) This field must be initialized in accordance with the index of the axes used. Index counting begins from 0. The traverse distances desired are entered into the individual elements to suit the positioning mode involved (absolute or relative). The entries in this data field must correspond with the selected axes in the *AS* structure/record type. E.g. the traverse distance of the 5th axis (axis index 4) always must be entered in the element dtm[4]

4.3.2.5 Structure/record type CMP (Circular Motion Parameters)

This structure or record type is a parameter with all circular interpolation commands.

Table 6**: Structure/record type CMP**

Element	Type	(Abbreviation meaning), Function
ac	double	(acceleration) trajectory acceleration
vl	double	(velocity) trajectory velocity
tvl	double	(target velocity) trajectory target velocity
phi	double	traverse angle in degrees
dtca1	double	(distance to center x-axis)
dtca2	double	(distance to center y-axis) The assignment of dtca1 and dtca2 to the desired axis channels is established with the structure/record type *AS*. The axis channel entered there in Field 0 is the x-axis. The y-axis is correspondingly entered in Field 1.

4.3.2.6 Structure/record type HMP (Helical Motion Parameters)

This structure or record type is a parameter with all helical interpolation commands.

Table 7: Structure/record type HMP

Element	Type	(Abbreviation meaning), Function
ac	double	(acceleration) trajectory acceleration
vl	double	(velocity) trajectory velocity
tvl	double	(target velocity) trajectory target velocity
phi	double	traverse angle in degrees The sign determines the circular direction. If the traverse angle ≤ 1e-100 a circle is run according to information of the target point.
dtca1	double	(distance to center x-axis)
dtca2	double	(distance to center y-axis)
dtm	Field with MAXAXIS double	(distance to move z-axis and higher) This field must be initialized in accordance with the index of the axes used. Index counting begins from 0. (see also LMP). The traverse distances desired are entered into the individual elements to suit the positioning mode involved (absolute or relative). By running a circle specified by the traverse angle, the traverse direction amd target points of the axes to be linearly interpolated are entered from Index 2. By running a circle specified by the target point, the target points of the circular axes are entered as well.

4.3.2.7 Structure/record type HMP 3D (Helical Motion Parameters 3-Dimensional)

This structure or record type is a parameter with all 3D interpolation commands.

Table 8: Structure/record type HMP3D

Element	Type	(Abbreviation meaning), Function
ac	double	(acceleration) trajectory acceleration
vl	double	(velocity) trajectory velocity
tvl	double	(target velocity) trajectory target velocity
phi	double	traverse angle in degrees The sign determines the circular direction. Running with target point instructions is not possible here.
dtca1	double	(distance to center x-axis)
dtca2	double	(distance to center y-axis)
dtca3	double	(distance to center z-axis)
pn1	double	Surface normal X-vector
pn2	double	Surface normal Y-vector
pn3	double	Surface normal Z-vector
dtm	Field with MAXAXIS double	reserved for future program extensions

4.3.2.8 Structure/record type ROSI (Risc Operating System Information)

This structure or record type is a parameter for the PCAP initialization command *mcuinit()*. After successful initialization of the APCI-800x board, the following *rw_MOS* data (*rwmos.elf*) are entered in the ROSI structure:

Table 9: Structure/record type ROSI

Element	Type	(Abbreviation meaning), Function
revision	Field with SIZE_STRREV characters	Current software revision of the *rw_MOS* operating system software.
number_axis	LONGINT	Number of axis channels available
sysfile_loaded	LONGINT	This status variable indicates with the value 1 whether the system file has already been transferred to the APCI-800x board.

Note: You can use the PCAP load command *InitMcuSystem2()* or *InitMcuSystem3()* to transfer the system.dat system file (which is altered mainly by means of the TOOLSET program mcfg.exe) to the APCI-800x board, where it will trigger initialization of intra-system parameters like accelerations, velocities, filter coefficients, limit values, etc. This load operation must be run once per system boot.

4.3.2.9 Structure/record type CBCNT (Common Buffer CNC-Task)

Each CNC task is provided with a local memory area with a size of 1,000 bytes (COMMON BUFFER), which both the PC and the CNC task involved can access in both read and write modes. This buffer can be used, for example, to build up a user-specific command set.

The structure/record type *CBCNCT* is a parameter for the PCAP commands *rdcbcnct()* and *wrcbcnct()*, which can be used to read or write the COMMON BUFFERs.

Table 10: Structure/record type CBCNCT

Element	Type	(Abbreviation meaning), Function
Task number	LONGINT	Task number (0..3)
Size	LONGINT	Size of buffer [Bytes]
Buffer	Pointer	Pointer to a buffer which is to be transferred to the APCI-800x board, or read in from the APCI-800x board. The buffer must be at least *size* bytes in size!

4.3.2.10 Structure/record type CNCTS (Computerized Numerical Control Task Status)

This structure/record type is a parameter for the PCAP status interrogation command *rdcncts()*.

Table 11: Structure/record type CNCTS

Element	Type	(Abbreviation meaning), Function
errnum	LONGINT	Internal CNC task error number. If no error has occurred, then *errnum* has the value 0. Information about runtime errors can be found in section 6.8
errline	LONGINT	In connection with *errnum*, this element is used to display the error-causing source text line of the CNC stand-alone application program.
stackfree	LONGINT	Currently free stack areas [bytes] for the CNC task.
running	LONGINT	This status word shows in Bit 0 whether the CNC task is currently processing a program. Bit 1 shows that the task is in single step operating mode, the system waits for a step (stepcnct) or continuation command (contcnct). If in the Halt-mode Bit 2 is set, it is indicated that the Task-stopp was caused by the SAP-command writelin. (see also notes to the register MODEREG Bit26 in section 6.3.1.5). Bit 3 indicates that the task is currently in wait state (wt or wait for “End of Profile”).
csrcline	LONGINT	Line number in the source text, which is being executed

4.4 PCAP high-level language function reference list

4.4.1 Structure of the reference list

The function and command reference list is sorted alphabetically. The descriptions for the individual commands and functions are structured as follows:

Element	Description
FUNCTION NAME:	This is the name which is used to call the function subsequently described.
ABBREVIATION MEANING:	Here you will find a detailed description of the function name concerned.
BORLAND DELPHI :	Here you will find the prototype definitions for the *Borland Delphi* programming language (*Pascal programming language)*. The parameters necessary to call up the function are listed.
C:	Prototype definition for the *C* programming language, e.g. *Microsoft Visual C++* or *Borland C++Builder* otherwise for *Borland Delphi*.
VISUAL BASIC:	Prototype definition for the *Microsoft Visual Basic* or *Borland Delphi* programming language.
TSRP COMPONENTS:	Various functions require components of the structure or record TSRP as parameters. They are listed here.
DESCRIPTION:	Plaintext description of the command.
RETURN VALUE:	If the function returns a value, you will find here a description.
NOTE:	For recurrent notes and explanations, you will find a cross-reference to the corresponding chapters here.
EXAMPLE:	Occasionally, examples are given for the function calls involved.

4.4.2 General information

All commands and functions, except the *spool* commands, are executed immediately after being called. For all *move* and *jog* commands, you must make sure before they are executed that the axes involved have been switched into position control beforehand (PCAP command *cl()*). In addition, some of the motion functions require differentiation between absolute and relative traversing commands. The absolute traversing commands are executed in the absolute measurement system, i.e. are referenced to the machine zero. The relative traversing commands are executed incrementally, i.e. starting from the current motor position.
The end of profile processing is indicated both in direct mode and in spool mode by the *pe* flag in the *axst* register of the structure/record TSRP [chapter 4.4.45 - *rdaxst()*].
In the case of the axis-specific motion commands, (*jog* commands), all system parameters like positions, traverse distances, accelerations and velocities are specified in the axis-specific units stated in the TOOLSET program *mcfg.exe*. For the interpolation commands (*move* commands), the units selected in the TRU structure (record) are utilised. This means that a PCAP function is to be called up before executing the *move* commands.
Conversion between application-specific and intra-system units is made automatically, using the factors specified in *mcfg.exe*. Conversion is determined by the encoder resolution or step number, the gear factor and the distance and time units selected.

4.4.2.1 Function values and function return values

The function value is the value which is to be read by the control system in the event of a read command. The function return value is the value returned through a command call. In many cases, this is not the function value but a value indicating success or an error information concerning a DLL function call. Write commands may also send a function return value; but it is not sent by all functions.
If the operating system of the motion control board is stopped due to an unexpected event on the board, such as an exception, or through user intervention in a program running in parallel, the following happens in a user program that calls DLL functions:
In the call during or after the event, the function is stopped without success after a time-out of several seconds. Success of the call can only be determined with functions which return status information on success.
Once the communication via DLL has been interrupted, it can only be re-established through reinitialisation, for example via InitMcuSystem3(). This is possible, for instance, when the board has been rebooted by a program or by itself. But mostly in this case, a complete reinitialisation of the application is required anyway.

4.4.3 azo, activate zero offsets

DESCRIPTION:	Each axis channel can be assigned five different zero offsets. You can use the *azo()* command to activate the axis-specific offset parameters you want. In the *set* (or *set_*) parameter, you specify which set of zero offsets is to be activated. This variable, with the value 0 .. 4, is used to select the set of zero offsets you want. But if the variable has a value greater than 4, no zero offsets will be taken into account any more.
BORLAND DELPHI:	procedure azo(set_: integer);
C:	void azo(int set);
VISUAL BASIC:	Sub azo(ByVal set_ As Long)
NOTE:	Zero offsets are used to specify a new system of coordinates, without having to influence (new setting) the actual machine zero. The currently set position value of the zero offset can be read with the command rdZeroOffset (Chapter 4.4.110).
RETURN VALUE:	None

4.4.4 BootErrorReport, initialisation error report

DESCRIPTION:	This functions explains in plaintext the error return values of the function BootFile() described below. A message box displays it on the screen. The user has then to close it.
BORLAND DELPHI:	procedure BootErrorReport(filename:PChar; error:integer);
C:	void BootErrorReport(char *filename, int error);
VISUAL BASIC:	Sub BootErrorReport (ByVal filename As String, ByVal error As Long)
NOTE:	PCAP command BootFile()
EXAMPLE:	*booterror = BootFile(...); // execute boot sequence* *BootErrorReport(..., booterror); // In case of error, display error return value*
RETURN VALUE:	None

4.4.5 BootFile, boot operating system file

DESCRIPTION:	The function transfers the operating system software (rwmos.elf) to the control process. The system is reset first. Afterwards the file specified in *BootFileName* (usually rwmos.elf) is loaded for the control.
BORLAND DELPHI:	function BootFile(var BootFileName:string; TpuBaseAddress: integer):integer;
C:	int BootFile(char* BootFileName, int TpuBaseAddress);
VISUAL BASIC:	Function BootFile(ByVal filename As String, ByVal TpuBaseAddress As Long) As Long
NOTE:	After successful booting the function InitMcuSystem2() or InitMcuSystem3() is to be called up in order to initialise the control completely. *TpuBaseAddress* is available to be compliant with the PA 8000 controller and is to be initialised with the value 0.
RETURN VALUE:	The function delivers return values as follows:

Return value	Error description
0	No error, boot process is completed successfully
10	The file name specified in *BootFileName* is not correct.
11	The file specified in *BootFileName* cannot be opened.
12	Unknown file format. At the moment only files with ELF file format are allowed
13	Incorrect ELF file format or transfer error.
14	An incorrect start address in RWMOS.ELF has been detected. RWMOS.ELF may be incorrect.
15	Incorrect platform for RWMOS.ELF The used RWMOS.ELF is not suited for the available hardware platform.
16	Verify has failed while transferring the boot file, the file has been incorrectly transferred.
200	Flash memory system of the target system cannot be accessed.
201	Incorrect flash memory size in the target system
202	The required device addresses of the target system cannot be accessed.

4.4.6 CardSelect

DESCRIPTION:	With this function you can select an APCI-800x controller if several should be installed in the PC. The selection is active until the function CardSelect is called for another device or until the application is terminated. After the selection all commands of the mcug3.dll, which are called within the application, refer to to selected device.
BORLAND DELPHI:	functionCardSelect (CardNum: integer): integer;
C:	int CardSelect (int CardNumber);
VISUAL BASIC:	Function CardSelect (ByVal CardNr As Long) As Long
PARAMETER:	Index of the board in the PC (0, 1, ...)
RETURN VALUE:	Index of the board that has been selected successfully. –1 if the selected device is not in the PC (in this case, the device is selected with Index 0). **Before using this command, one of the InitMcuSystem commands must be called up** so that the internal list of available devices is up-to-date.
NOTE:	See also CM, Chapter 5.3

4.4.7 ClearCI99

DESCRIPTION:	With this function, the common integer variable CI99 is reset synchronously to the operating system software RWMOS.ELF.
BORLAND DELPHI:	procedure ClearCI99 ();
C:	void ClearCI99 (void);
VISUAL BASIC:	Sub ClearCI99 ()
PARAMETER:	none
RETURN VALUE:	none, the variable CI99 is set to 0
NOTE:	This function has to be used when the ssf functions 1005 – 1025 for the synchronisation of spooler commands are used.

4.4.8 cl, close loop

DESCRIPTION:	All axis channels specified in *AS* are brought into position control with this command. Note that the actual positions of the axes involved are accepted as setpoint positions, in order to avoid large system deviations. In addition, all digital outputs planned with PAE are set. These outputs can, for example, be used for controlling relays, which in turn can be used to enable power amplifier units. Depending on the selected axis channel, the release relays of the assigned axis channel are switched on (CM / Chapter 5.2.10).
BORLAND DELPHI:	procedure cl(var as:AS);
C:	void cl(struct AS far *as);
VISUAL BASIC:	Sub cl(DASEL As ASEL) 'close loop
NOTE:	The position control causes the PIDF filter to be processed with the appropriately set filter coefficients. When the position control loop is closed, all spooler data for the axis channels specified will be rejected! See also PCAP command clv().

4.4.9 clv, close loop velocity

DESCRIPTION:	All axis channels specified in AS are brought in position control with the command. The actual positions of the axes involved are accepted as setpoint positions and the actual speeds as setpoint speeds in order to avoid large system deviations. In addition, all digital outputs planned with PAE are set. This command is to be used when the axes are moving before the control loop is closed. The corresponding axes obtain the current speed when the control loop is closed and are running further with this command. They can now be decelerated e.g. through js() to prevent a hard stop of the axes when the control loop is closed. Depending on the selected axis channel, the release relays of the assigned axis channel are switched on (CM / Chapter 5.2.9).
BORLAND DELPHI:	procedure clv(var as:AS);
C:	void clv(struct AS far *as);
VISUAL BASIC:	Sub clv(DASEL As ASEL) 'close loop velocity
NOTE:	See also PCAP command cl()

4.4.10 contcnct, continue numeric controller task

DESCRIPTION:	You can use this command to continue a SAP program which has previously been halted with the SAP command *STOP*, *STOPCNCT()* or with the PCAP command *stopcnct()*. The task selected in *TaskNr* (values 0..3) will be continued.
BORLAND DELPHI:	procedure contcnct(TaskNr:integer);
C:	void contcnct(int TaskNr);
VISUAL BASIC:	Sub contcnct(ByVal TaskNr As Long)
NOTE:	A SAP program which has been halted with the SAP command *ABORT*, can only be restarted (i.e. not continued) with the SAP command *STARTCNCT()* or the PCAP command *startcnct()*.

4.4.11 ctru, change trajectory units

DESCRIPTION: This command can be used to switch over the units for the velocity, acceleration and position parameters of all interpolation commands (*move* commands). The parameters are specified in the units selected. The following values are permitted for the TRU structure component *pu* (position unit).

BORLAND DELPHI: procedure ctru(var tru:TRU);

C: void ctru(struct TRU far *tru);

VISUAL BASIC: Sub ctru(DTRU As tru)

ALL LANGUAGES: The following values are permitted for the TRU structure component *pu* (position unit):

Index	Unit	Description
0	mm	Millimeter
1	inch	Inch
2	m	Meter
3	rev	Revolution
4	deg	Degree
5	rad	Radiant
6	counts	Counts
7	steps	Steps

The following values are permitted for the TRU structure component *tu* (time unit):

Index	Unit	Description
0	sec	Seconds
1	min	Minutes
2	tsample	Sampling time

NOTE: The default value for *pu* and *tu* is 0. This means that for all distance particulars the unit [mm] is assumed, for velocities the unit [mm/s] and for accelerations the unit [mm/s²]. The units selected are utilized only for interpolation commands (all *move* commands)! If the commands involved are axis-specific motion commands (all *jog* commands), the axis units specified in *mcfg.exe* are taken into account. The units selected are also decisive for any SAP program running in parallel. In the rw_SymPas programming environment, these parameters are accessed via the system parameters PU and TU (Table 32).

4.4.12 getEnvStr, get Environment String

DESCRIPTION:	With this command the environment variable, which is specified in the string or sign parameter, is read out from the control and the value is entered into the calling parameter.
BORLAND DELPHI:	function getEnvStr (var EnvStr:CppString):integer;
C:	int getEnvStr (char far * EnvStr);
VISUAL BASIC:	Function getEnvStr (ByVal EnvStr As String) As Long
RETURN VALUE:	The function can return the following values: **Return value** — **Errror description** **-1** — Error: RWMOS does not supply the function, for example. **-4** — Error: time-out, reason unknown, communication with the motion control board is interrupted **0** — The parameter was not found or is an empty string **> 0** — Indicates the string lenth of the found string (without concluding zero byte). So this means that a return value >= 0 indicates the successful execution of the command.
NOTE:	With this function an application program can check the availabilty of environment variables that are significantly important for the application. In this way an application can react even then controlled, if for example because of a hardware change important characteristics of the control are not available anymore. Th writing of environment variables is only possible with unbooted system in fwsetup This function firstly is available in RWMOS.ELF from V2.5.3.37 on and in mcug3.dll from V2.5.3.25 on. In mcug3.pas, the data type CppString for Delphi is defined according to the version.
DELPHI SAMPLE:	EnvString : CppString; ..... EnvString := allocmem (1024); StrPCopy (EnvString, 'SerialNumber'); getEnvStr (EnvString);

4.4.13 gettskinfo, Get Task Informations

DESCRIPTION:	With this command a task can be asked if there is still a string that is not already read out.
BORLAND DELPHI:	function gettskinfo (TaskNr: integer; var tskinfo: integer): integer;
C:	int gettskinfo (int TaskNr, int *tskinfo);
VISUAL BASIC:	Function gettskinfo (ByVal tasknr As Long, tskinfo As Long) As Long
Parameter:	TaskNr: Task number (0..3) tskinfo: In this variable, the function value is returned.
Return value:	0 for success -1 Command is not available in RWMOS version. -4 Time-out, reason unknown, communication with the motion control board is interrupted Other value <> 0 Unknown error at command execution
NOTE:	This function is returned in tskinfo. Bit 0 indicates that there is a not already completed string (write). Bit 1 indicates that there is a completed string (writeln). The respecting bits are reset automatically by reading the string by gettskstr(). Task message strings can be generated in the programming environment of the stand-alone tasks by WRITE or WRITELN (chapter 6.6.78 and 6.6.79).

4.4.14 gettskstr, Get Task Message String

DESCRIPTION:	With this command the task specific output string can be read.
BORLAND DELPHI:	function gettskstr (TaskNr: integer; buffer: PChar, szbuffer: integer): integer;
C:	int gettskstr (int TaskNr, char * buffer, int szbuffer);
VISUAL BASIC:	Function gettskstr (ByVal tasknr As Long, ByVal buffer As String, ByVal szbuffer) As Long
Parameter:	TaskNr: Task number (0..3) buffer: In this variable, the read string is returned. szbuffer: Max. size of the string to be read.
Return value:	Number of the read signs
NOTE:	This call resets the respecting status bits in tskinfo. The storage section of TskStr must be sufficient in order to store the returned string. Max. 512 bytes will be returned. Task Message Strings can be generated in the programming environment of the stand-alone tasks by WRITE or WRITELN (chapters 6.6.78 and 6.6.79).

4.4.15 InitMcuErrorReport, initialisation error report

DESCRIPTION:	This functions explains in plaintext the error return values of the functions InitMcuSystem(), InitMcuSystem2() and InitMcuSystem3() described below. A message box displays it on the screen. The user has then to close it.
BORLAND DELPHI:	procedure InitMcuErrorReport(error:integer);
C:	void InitMcuErrorReport (int error);
VISUAL BASIC:	Sub InitMcuErrorReport (ByVal error As Long)
NOTE:	PCAP command InitMcuSystem(), InitMcuSystem2() and InitMcuSystem3()
EXAMPLE:	*initerror = InitMcuSystem3(...); // Start initialisation* *InitMcuErrorReport(initerror); // In case of error, display error return value*

4.4.16 InitMcuSystem, initialise mcu system

DESCRIPTION:	This function performs the complete software initialization for the drive system. The function call should be executed at the beginning of every PCAP application program at any case before any other PCAP calls. Inside this function, various PCAP basic functions are called. This includes initialization of the axis numbers {an} in the *tsrp* structure. If the *system.dat* system file has not yet been transferred onto the APCI-800x board, this will be done here. At the end of the function, the axis parameters of all axes are read into the *tsrp* structure.
BORLAND DELPHI:	function InitMcuSystem(var tsrp:TSRP):integer;
C:	int InitMcuSystem(var TSRP far *tsrp);
VISUAL BASIC:	Function InitMcuSystem(DTSRP As TSRP) As Long
NOTE:	PCAP commands *txbf2()*, *mcuinit()*, structure/record type ROSI **Important:** This function has been written to be compliant with the PA 8000. You should use instead the functions InitMcuSystem2() or rather InitMcuSystem3().
RETURN VALUE:	The function can return the following values:

Return value	Error description
0	No error
31	No APCI-800x controller found
32	The rw_MOS operating software has not been loaded or has been stopped. See PCAP command *BootFile()* or service program *mcfg.exe*
33	Wrong operating system software. The file versions of the *mcug3.dll* and *rwmos.elf files* have incompliant revision states and do not match.
34	The device driver rnwmc.sys (Windows NT 4.0, 2000) or rnwmc.vxd (Windows 95/98/Me) cannot be opened.
35	Error while mapping the physical APCI-800x board memory.
36	Error while mapping in the physical APCI-800x board memory.
37	Error while mapping out the physical APCI-800x board memory.
38	APCI-800x board cannot be accessed
39	APCI-800x board mail-box-interface cannot be accessed
Iderr	Error return value from PCAP command txbf()

4.4.17 InitMcuSystem2, initialise mcu system (2nd method)

DESCRIPTION:	This function is identical to the *InitMcuSystem()*, except that the parameters *SystemFileName* and *TpuBaseAddress* are specified. *SystemFileName* contains the file name of the system file (usually system.dat) as well as path and drive information.
BORLAND DELPHI:	function InitMcuSystem2(var tsrp:TSRP; TpuBaseAddress: integer, var SystemFileName: string):integer;
C:	int InitMcuSystem2(struct TSRP *tsrp, int TpuBaseAddress, char *SystemFileName)
VISUAL BASIC:	Function InitMcuSystem2(DTSRP As TSRP, ByVal TpuBaseAddress As Long, ByVal filename As String) As Long
RETURN VALUE:	The function has the same return values as the function *InitMcuSystem() txbf2* implicitly called up.
NOTE:	See *InitMcuSystem(),* *TpuBaseAddress has no meaning and is to be transferred with the value 0.*

4.4.18 InitMcuSystem3, initialise mcu system (3rd method)

DESCRIPTION:	This function is identical to *InitMcuSystem()*, except that the parameters *SystemFileName, rosi, TpuBaseAddress* and *BoardType* are specified. *SystemFileName* contains the file name of the system file (usually system.dat) as well as path and drive information.
BORLAND DELPHI:	function InitMcuSystem3(var tsrp:TSRP; var rosi:ROSI, TpuBaseAddress: integer, var SystemFileName: string; var BoardType: integer):integer;
C:	int InitMcuSystem3(struct TSRP *tsrp, struct ROSI *rosi, int TpuBaseAddress, char *SystemFileName, int *BoardType)
VISUAL BASIC:	Function InitMcuSystem3(DTSRP As TSRP, DROSI As ROSI, ByVal TpuBaseAddress As Long, ByVal filename As String, BoardType As Long) As Long
RETURN VALUE:	The function has the same return values as the function *InitMcuSystem()*. Further return values can returned by the function txbf2 implicitly called up. The structure *rosi* is updated according to the system information returned by the control. The value *BoardType* informs about the control type. *BoardType* can contain the following values: 1 = PA 8000 (ISA board) 2 = PS 840 (ISA board) 4 = APCI-8001 32 (20 hex) = APCI-8008 0 = unknown board or old RWMOS other values = more recent products
NOTE:	See *InitMcuSystem()* *TpuBaseAddress has no meaning and is to be transferred with the value 0.* As the initialisation function has currently the highest priority, the use of this function is recommended.

4.4.19 ja, jog absolute

DESCRIPTION:	The axis channels selected in *AS* are moved absolutely to the target positions specified in *TSRP[n].tp* using a trapezoidal speed profile. The profile is generated using the axis-specific system parameters *jac* (*jog* acceleration), *jvl* (*jog*-velocity) and *jtvl* (*jog* target velocity). You can set and interrogate these parameters at any time using write and read commands. The default values are specified in the *mcfg.exe* utility program. The trajectory parameters are stated in the axis-specific units (distance, time) specified in *mcfg.exe*.
BORLAND DELPHI:	procedure ja(var as:AS; var tsrp:TSRP);
C:	void ja(struct AS far *as, struct TSRP far *tsrp);
VISUAL BASIC:	Sub ja(DASEL As ASEL, DTSRP As TSRP)
TSRP COMPONENTS:	TSRP[n].tp n = 0 .. Number of axes present-1
NOTE:	If this command is executed simultaneously for more than one axis, these may (due to the axis-specific system parameters) reach the target positions at different points in time (see chapter 2.2.7) You can set and interrogate the axis-specific parameters like accelerations and velocities at any time using write and read commands. They are not transferred automatically with ja. **Important:** By calling out the function ja the element 0 of the global data structure TSRP must be entered, as ja() takes the index of the used TSRP structure elements from the AS structure entnimmt.

4.4.20 jhi, jog home index

DESCRIPTION:	This command starts the index search run for all the axis channels selected in *AS*. The search run is terminated either when the index (zero track) signal of the incremental encoder is activated or after the distance or angle particular specified in *tp* has been exceeded. The search run is carried out using a trapezoidal speed profile. The parameters for the profile generator are the system data *hac* and *hvl*, which can be set using *mcfg.exe* or the appropriate write commands. When the index signal (zero track) is detected, the motor is decelerated with the deceleration *hac* to velocity 0. The *tp* parameter is stated as a relative traverse distance in the axis-specific position unit. The search direction is determined by the sign of *tp*. Generally, the axis system is first run in relation to a reference switch (cam). To eliminate the mechanical inaccuracy of this cam, the obvious solution is to perform the index search run afterwards. The command can be executed with the aid of the profile end flag (PE) in the *axst* register and the state of the index signal interrogated with the *digi* register (Chapter 4.4.52.1). The profile end flag remains set to 0 until the end of the search run.
BORLAND DELPHI:	procedure jhi(var as:AS; var tsrp:TSRP);
C:	void jhi(struct AS far *as, struct TSRP far *tsrp);
VISUAL BASIC:	Sub jhi(DASEL As ASEL, TSRP As TSRP)
TSRP COMPONENTS:	TSRP[n].tp n = 0 .. Number of axes present -1
NOTE:	To maximize the accuracy of index positioning, the search run should be executed with as small a traversing velocity as possible. You do, however, also have an option for performing the search run in two steps. In the first of these steps, the search run can be started in a positive traversing direction, for example, at a relatively high search speed. In the second step, the search run is then concluded in the negative direction at a low search speed. The search speed can be read and written with the PCAP commands *rdhvl()* and *wrhvl()*. **Important:** By calling out the function jhi() the element 0 of TSRP must be entered, as jhi() takes the index of the used TSRP structure elements from the AS structure entnimmt.

4.4.21 jhl, jog home left

DESCRIPTION:	This command starts the reference search run for all axis channels specified in *AS,* in a negative traversing direction. The search run is executed with the aid of an endless trapezoidal speed profile. The axis-specific system data *hac* and *hvl* here serve as parameters for profile generation. If a digital input of the APCI-800x board planned with REF function is activated at the axis channel selected, the search run will be terminated by decelerating (with *hac*) the axis to a velocity of 0. This state can be interrogated in the *axst* register with the aid of the *pe* profile flag. The profile flag remains set to 0 until the end of the search run.
BORLAND DELPHI:	procedure jhl(var as:AS);
C:	void jhl(struct AS far *as);
VISUAL BASIC:	Sub jhl(DASEL As ASEL)

4.4.22 jhr, jog home right

DESCRIPTION:	This command functions in an identical way to the PCAP command *jhl()*, except that the search run is started in the positive traversing direction.
BORLAND DELPHI:	procedure jhr(var as:AS);
C:	void jhr(struct AS far *as);
VISUAL BASIC:	Sub jhr(DASEL As ASEL)

4.4.23 jr, jog relative

DESCRIPTION:	This command is identical to the PCAP command *ja()*, except that the distance particular *tp* is a relative (incremental) traverse distance. Starting from the instantaneous position, the motor is moved by the specified distance (or angle) to the left (negative values) or the right (positive values).
BORLAND DELPHI:	procedure jr(var as: AS; var tsrp:TSRP);
C:	void jr(struct AS far *as, struct TSRP far *tsrp);
VISUAL BASIC:	Sub jr(DASEL As ASEL, DTSRP As TSRP)
TSRP COMPONENTS:	TSRP[n].tp n = 0 .. number of existing axes-1
NOTE:	By calling out the function jr the element 0 of TSRP must be entered, as jr() takes the index of the used TSRP structure elements from the AS structure entnimmt.

4.4.24 js, jog stop

DESCRIPTION:	The axis channels - selected in AS - are decelerated with the axis-specific time-delay *sdec* to velocity 0 and hold in position control. Until the end of deceleration the *pe* flag is reset in the *axst* register. You can set and interrogate the time-delay *sdec* at any time using write and read commands. The default value is specified in the *mcfg.exe* utility program.
BORLAND DELPHI:	procedure js(var as: AS);
C:	void js(struct AS far *as);
VISUAL BASIC:	Sub js(DASEL As ASEL)
NOTE:	If this command is executed simultaneously for more than one axis, these may (due to the axis-specific system parameters) reach the target positions at different points in time [Chapter 2.2.7]. The value sdec = 0 forces an immediate axis stop without braking ramp.

4.4.25 lpr – Latch Position Registers

DESCRIPTION:	This command can start the recording of the graphical system analysis for one axis.
BORLAND DELPHI:	procedure lpr (var latch_infos: LATCH_INFOS);
C:	void lpr (struct LATCH_INFOS *latch_infos);
VISUAL BASIC:	Sub lpr (DLATCH_INFOS As LATCH_INFOS)
RETURN VALUE:	None
EFFECT:	After the command lprs has been executed the recording of the graphcal system analyse is started. The recording parameters are given in latch_infos.
NOTE:	See also command lprs and grafical system analyse in mcfg **Important:** The data structure latch_infos must be aligned in 4 bytes.

4.4.26 lprs – Latch Position Registers Synchronous

DESCRIPTION:	This command can start the recording of the graphical system analysis synchronously for one or several axes.
BORLAND DELPHI:	procedure lprs (var as: AS; var latch_infos: LATCH_INFOS);
C:	void lprs (struct AS *as, struct LATCH_INFOS *latch_infos);
VISUAL BASIC:	Sub lprs (DASEL As ASEL, DLATCH_INFOS As LATCH_INFOS)
RETURN VALUE:	None
EFFECT:	After the command lprs has been executed the recording of the graphcal system analyse is started. The recording parameters are given in latch_infos. The element *san of* the data structure *latch_infos* has no significance with this command as the axes are specified in *as*.
NOTE:	See also command lpr and grafical system analyse in mcfg **Important:** The data structure latch_infos must be aligned in 4 bytes.

4.4.27 lps, latch position synchronous

DESCRIPTION:	This command can be used to initiate a latch routine synchronized with the scan cycle of the axis channel selected in *an.* After call-up, the actual position *{rp}* is put into intermediate storage after every *mst* scan intervals. If a latch procedure has taken place, this will be displayed in the *axst* register in the *lpsf* flag (Bit No. 16). The PCAP read command *rdlp()* or the *lp* SAP axis qualifier can be used to read out the position from intermediate storage. Readout will also erase the *lpsf* flag in the *axst* register.
BORLAND DELPHI:	procedure lps(an: integer; mst: integer);
C:	void lps(int an, int mst);
VISUAL BASIC:	Sub lps(ByVal an As Long, ByVal mst As Long)
NOTE:	The command is primarily used when recording contours and teach-in applications, since it enables position data in real time to be recorded from one or more axes. Typical values for *mst* are 10 ... 100 scan intervals (-> 12.8 ms ... 128.0 ms). The precise value will, however, depend on the processing speed of the application concerned.

4.4.28 mca, move circular absolute - smca, spool motion circular absolute

DESCRIPTION:	This command causes circular interpolation of the first two axis channels specified in *AS*. There are no restrictions regarding axis selection. Circular interpolation is carried out on the basis of a trapezoidal speed profile, i.e. taking into account maximum acceleration and maximum velocity. The structure/record components specified in CMP are utilized as interpolation parameters. These are the trajectory acceleration *ac,* the trajectory velocity *vl* and the trajectory target velocity *tvl*. The coordinates entered in *dtca1* and *dtca2* specify the circle's centre in an absolute system of units. Note that *dtca1* is assigned to the first axis programmed in AS and *dtca2* to the second axis specified in AS. The units for the trajectory parameters are selected with the PCAP command *ctru()*. The angle *phi* specifies the traverse angle to be run with the unit degrees. The sense of rotation is specified by the sign of the angle variable. Positive values signify anti-clockwise rotation and negative values signify clockwise rotation. The traverse angle range is not fixed to defined limits, i.e. part or multiple circles can be run as well.
BORLAND DELPHI:	procedure mca(var as: AS; var cmp: CMP); procedure smca(var as: AS; var cmp: CMP);
C:	void mca(struct AS far *as, struct CMP far *cmp); void smca(struct AS far *as, struct CMP far *cmp);
VISUAL BASIC:	Sub mca(DASEL As ASEL, CMP As CMP) Sub smca(DASEL As ASEL, CMP As CMP)
NOTE:	Chapter 2.3 Interpolation with the APCI-800x.

4.4.29 mcr, move circular relative - smcr, spool motion circular relative

DESCRIPTION:	This command is identical to the PCAP command *mca(),* except that the coordinates specified in *dtca1* and *dtca2* are incrementally (or relatively) referenced to the current motor position.
BORLAND DELPHI:	procedure mcr(var as: AS; var cmp: CMP); procedure smcr(var as: AS; var cmp: CMP);
C:	void mcr(struct AS far *as, struct CMP far *cmp); void smcr(struct AS far *as, struct CMP far *cmp);
VISUAL BASIC:	Sub mcr(DASEL As ASEL, CMP As CMP)
NOTE:	Chapter 2.3 Interpolation with the APCI-800x.

4.4.30 mca3d, move circular absolute three dimensional - smca3d, spool motion circular absolute three dimensional

DESCRIPTION:	This function is used to carry out the circular interpolation of the 3 specified axis channels. There are not restrictions regarding axis selection. Circular interpolation is carried out on the basis of a trapezoidal speed profile, i.e. considering the maximum acceleration and maximum velocity. The trajectory acceleration *ac*, the trajectory velocity *vl* and the trajectory target velocity *tvl* are used as interpolation parameters *in hmp3d*. The coordinates entered in *dtca1, dtca2 and dtca3* specify the circle's center in absolute measurement system. Note that *dtca1* is assigned to the first axis programmed in AS, dtca2 to the second axis and dtca3 to the third axis. The units of the trajectory parameters are selected with PCAP command *ctru()*. The circle can be traversed in any wished level, which is specified in the surface normal in PN1, PN2 and PN3. The current start coordinates always remain in the given level. The angle *phi* specifies the traverse angle to be run with the unit Degree. The sense of rotation is determined by the sign of the angle variable. Positive values signify anti-clockwise rotation and negative values clockwise rotation. The traverse angle range is not fixed to defined values, i.e. part or multiple circle can be runs as well. The data field dtm[] is not used here.
BORLAND DELPHI:	procedure mca3d(var as: AS; var hmp3d: HMP3D); procedure smca3d(var as: AS; var hmp3d: HMP3D);
C:	void mca3d(struct AS far *as, struct HMP3D far *hmp3d); void smca3d(struct AS far *as, struct HMP3D far *hmp3d);
VISUAL BASIC:	Sub mca3d(DASEL As ASEL, HMP3D As HMP3D) Sub smca3d(DASEL As ASEL, HMP3D As HMP3D)
NOTE:	Chapter 2.3 Interpolation with the APCI-800x.

4.4.31 mcr3d, move circular relative three dimensional - smcr3d, spool motion circular relative three dimensional

DESCRIPTION:	This function is identical to the PCAP command *mca3d()* except that the coordinates specified in *dtca1, dtca2* and *dtca3* are incrementally or relatively referenced to the instantaneous motor positions.
BORLAND DELPHI:	procedure mcr3d(var as: AS; var hmp3d: HMP3D); procedure smcr3d(var as: AS; var hmp3d: HMP3D);
C:	void mcr3d(struct AS far *as, struct HMP3D far *hmp3d); void smcr3d(struct AS far *as, struct HMP3D far *hmp3d);
VISUAL BASIC:	Sub mcr3d(DASEL As ASEL, HMP3D As HMP3D) Sub smcr3d(DASEL As ASEL, HMP3D As HMP3D)
NOTE:	Chapter 2.3 Interpolation with the APCI-800x.

4.4.32 mcuinit, motion control unit initialisation

DESCRIPTION:	This function is used to carry out various initialization routines inside the system driver *mcug3.dll*. It checks whether communication is possible between PC and APCI-800x board. If this is the case, the *rw_MOS*-specific system data returned by the APCI-800x board are entered in the structure/record ROSI, which can then be used to check the *rw_MOS*-specific system information for validity. If it has not proved possible to establish communication to the APCI-800x, the entire TOSI structure will have the value 0
BORLAND DELPHI:	procedure mcuinit(var rosi:ROSI);
C:	void mcuinit(struct ROSI far *rosi);
VISUAL BASIC:	Sub mcuinit(DROSI As ROSI)
NOTE:	This command does not trigger a reset on the APCI-800x board. This must be carried out with the PCAP commands *ra()* or *rs()*. You can use the ROSI.sysfile_loaded return value to ascertain whether the *system.dat* system file has already been transferred to the APCI-800x board with the aid of the PCAP load command *txbf2()*. If this value is 0, then after a successful *mcuinit()* PCAP command the PCAP command *txbf2()* must be executed, so that you can work with the APCI-800x board. The PCAP example programs provided include this command in the *InitMcuSystem(), InitMcuSystem2() and InitMcuSystem3()* functions, where the monitoring mechanism for system initialization is once more illustrated. **Important:** mcuinit() is compatible with the PA 8000 and PS840 controllers and should be replaced by the InitMcuSystem3() command. You can use this command only to check if the motion control board is still online.

4.4.33 MCUG3_SetBoardIntRoutine

DESCRIPTION:	With this function a user specific interrupt processing routine can be installed and activated.
BORLAND DELPHI:	function MCUG3_SetBoardIntRoutine (func : Pointer): integer;
C:	int MCUG3_SetBoardIntRoutine(InterruptRoutine func);
VISUAL BASIC:	Function MCUG3_SetBoardIntRoutine (ByVal func As Long) As Long
PARAMETER:	func is a function pointer onto the interrupt processing routine that was written by the user. It is declared (in C++) e.g. in the following manner: void CALLBACK EventHandler(int IRQLineBits) {}
RETURN VALUE:	No meaning
NOTE:	Within the interrrupt processing routine the programming conventions of the Window operating system have to be observed. So, it is not allowed to generate window objects in a callback-handler. For Visual Basic 6.0 the additional module „MCUG3Interrupt.BAS“ is contained in the scope of delivery for the use of this function

4.4.34 MCUG3_ResetBoardIntRoutine

DESCRIPTION:	With this function a previously enabled user specific interrupt processing routine can be disabled.
BORLAND DELPHI:	function MCUG3_ResetBoardIntRoutine (): integer;
C:	int MCUG3_ResetBoardIntRoutine(void);
VISUAL BASIC:	Function MCUG3_ResetBoardIntRoutine () As Long
NOTE:	Before quitting the application the currently installed interrupt service routine must be desinstalled.

4.4.35 mha, move helical absolute - smha, spool motion helical absolute

DESCRIPTION:	This command is used to perform a helical interpolation; it is an extension of circular interpolation. This is why the particulars given for the PCAP command *mca()* also apply to this command, except that the trajectory parameters are entered in the structure/record HMP. For additional axes specified in AS, the *dtm* parameter can be programmed as well. These are the absolute target positions for additional axes. While the first two axes perform a circular interpolation, the other ones execute a linear movement. All axes reach their target positions at the same moment. Unlike the circular interpolation the circle target point can be defined per target position instead through the circle angle. This case must be displayed by the user with a traverse angle value ≤ 1e-100. The angle sign indicates the traverse direction. The required circle target point are defined in this case in dtm [0] and dtm[1] of HMP. In case the given target point is not located on the circle which results from the start point and the middle point, the target position is corrected.
BORLAND DELPHI:	procedure mha(var as: AS; var hmp: HMP); procedure smca(var as: AS; var hmp: HMP);
C:	void mha(struct AS far *as, struct HMP far *hmp); void smha(struct AS far *as, struct HMP far *hmp);
VISUAL BASIC:	Sub mha(DASEL As ASEL, HMP As HMP) Sub smha(DASEL As ASEL, HMP As HMP)

4.4.36 mhr, move helical relative - smhr, spool motion helical relative

DESCRIPTION:	This command is identical to the PCAP command *mha(),* except that the distance particulars programmed in *dtca1*, *dtca2* and *dtma3* are referenced to the instantaneous motor position incrementally (or relatively).
BORLAND DELPHI:	procedure mhr(var as: AS; var hmp: HMP); procedure smhr(var as: AS; var hmp: HMP);
C:	void mhr(struct AS far *as, struct HMP far *hmp); void smhr(struct AS far *as, struct HMP far *hmp);
VISUAL BASIC:	Sub mhr(DASEL As ASEL, HMP As HMP) Sub smhr(DASEL As ASEL, HMP As HMP)

4.4.37 mla, move linear absolute - smla, spool motion linear absolute

DESCRIPTION:	This command is used to carry out a linear interpolation with absolute target particulars. All axes in n-dimensional space are permitted for interpolation. You specify in AS which axes you want to participate in interpolation. You use LMP to specify the trajectory acceleration *ac*, the trajectory velocity *vl* and the trajectory target velocity *tvl* for linear interpolation. The units for the trajectory parameters are selected with the *ctru()* command. Depending on the number of axes involved (*unoa)*, you enter the axes you want in the *san* field and the corresponding traverse distances in the *dtm* field. Note that the traverse distance in the *dtm[n]* field is assigned to the axis number *n + 1*. The interpolation is referenced to the axes entered in AS. The traverse distances are interpreted as absolute distance or angle information, i.e. referenced to the machine zero.
BORLAND DELPHI:	procedure mla(var as: AS; var lmp: LMP); procedure smla(var as: AS; var lmp: LMP);
C:	void mla(struct AS far *as, struct LMP far *lmp); void smla(struct AS far *as, struct LMP far *lmp);
VISUAL BASIC:	Sub mla(DASEL As ASEL, lmp As lmp) Sub smla(DASEL As ASEL, lmp As lmp)
NOTE:	Chapter 2.3 Interpolation with the APCI-800x.

4.4.38 mlr, move linear relative - smlr, spool motion linear relative

DESCRIPTION:	This command is identical to the PCAP command *mla()*, except that the traverse distances specified in the *dtm* field are interpreted incrementally or relatively to the instantaneous motor position.
BORLAND DELPHI:	procedure mlr(var as: AS; var lmp: LMP); procedure smlr(var as: AS; var lmp: LMP);
C:	void mlr(struct AS far *as, struct LMP far *lmp); void smlr(struct AS far *as, struct LMP far *lmp);
VISUAL BASIC:	Sub mlr(DASEL As ASEL, lmp As lmp) Sub smlr(DASEL As ASEL, lmp As lmp)
NOTE:	Chapter 2.3 Interpolation with the APCI-800x.

4.4.39 ms, motion stop

DESCRIPTION:	The axis channels selected in *AS* are decelerated with the trajectory acceleration or axis deceleration currently valid down to zero velocity and kept in position control mode. The *pe* flag in the *axst* register is reset by the time the deceleration procedure has been completed. The direction vector of a perhaps currently ongoing interpolation function is not altered by this command. If the axes selected are currently running a circle, deceleration will be performed on the circular trajectory with the trajectory acceleration specified. Axes which traverse with one final velocity are decelerated down to zero velocity with the axis-specific deceleration *sdec*.
BORLAND DELPHI:	procedure ms(var as: AS);
C:	void ms(struct AS far *as);
VISUAL BASIC:	Sub ms(DASEL As ASEL)
NOTE:	Axes which are not interpolating jointly may reach the target point at different points in time.

4.4.40 MsgToScreen, message to screen

DESCRIPTION:	This command disables or enables the screen messages of the DDL driver. If the parameter Enable = 0 the screen messages are disabled.
BORLAND DELPHI:	procedure MsgToScreen (Enable: integer);
C:	void MsgToScreen (long Enable);
VISUAL BASIC:	Sub MsgToScreen (ByVal Enable As Long)
NOTE:	This option is important for systems without user interface. If screen messages are enabled the system can otherwise wait for an entry which cannot be used. This command is available from the version 3.5.2.10.

4.4.41 ol, open loop

DESCRIPTION:	This command opens the position control loop of all axes selected in *AS*. On each of the Motor-Command-Ports, 0 V output voltage is outputted in the case of servo axes and 0 Hz stepping frequency in the case of stepping motor axes. All APCI-800x digital outputs planned with PAE function are de-activated for the axis channels programmed. Depending on the axis channels selected, the relays K2 (axis channel 1), K3 (axis channel 2) and K4 (axis channel 3) are switched off. [CM / Chapter 5.2.10]
BORLAND DELPHI:	Procedure ol(var as: AS);
C:	void ol(struct AS far *as);
VISUAL BASIC:	Sub ol(DASEL As ASEL)
NOTE:	This command is used mainly in exceptional situations, like limit switch limitation, position error violation, etc.

4.4.42 ra, reset axis

DESCRIPTION:	This command can be used to carry out an axis-specific reset operation. This means that any profile running will be aborted, the position control loop will be opened, the setpoint value will be switched off, any spooler data will be rejected and the position registers set to zero. The digital outputs are set to the default values planned. The axis-specific override factors (PCAP commands *wrjovr()* and *wrtrovr()*) are set to the value 1.0. Any software limits planned will no longer be monitored for the axis channels selected in *ra()*.
BORLAND DELPHI:	Procedure ra(var as: AS);
C:	void ra(struct AS far *as);
VISUAL BASIC:	Sub ra(DASEL As ASEL)
NOTE:	All system data, like accelerations, velocities, filter parameters, etc. remain stored in memory and therefore need not be loaded anew. This command is mainly used at system initialization or in exceptional situations. **Warning:** PAE outputs of other axes in the same output group which could have been set, are reset with this command.

4.4.43 rdap, read axis parameters

DESCRIPTION:	This command can be used to read in all axis-specific input and output variables of the structure and/or the TSRP record with one read command.
BORLAND DELPHI:	procedure rdap(var tsrp:TSRP);
C:	void rdap(struct TSRP far *tsrp);
VISUAL BASIC:	Sub rdap(DTSRP As TSRP)
TSRP COMPONENTS:	all, i.e. TSRP[n].an .. TSRP[n].ifs
RETURN VALUE:	Once the command has been executed, the input and output variables will be located in the structure or record components concerned in each case, or in the TSRP record.
NOTE:	The individual structure or record components can also be interrogated, using special read commands. Normally, these read commands are preferred due to the shorter access time involved.

4.4.44 rdaux, read auxiliary register

DESCRIPTION:	The function returns the axis-specific auxiliary register. [Chapter 6.3.3]
BORLAND DELPHI:	procedure rdaux (var tsrp:TSRP);
C:	void rdaux (struct TSRP *tsrp);
VISUAL BASIC:	Sub rdaux(DTSRP As TSRP)
TSRP COMPONENTS:	TSRP[n].aux
NOTE:	See also chapter 4.4.133

4.4.45 rdaxst, read axis status

DESCRIPTION:	This command can be used to interrogate various axis-specific status and error flags of the ramp and interpolation task. Normally this command is repeated cyclically in the PCAP program, in order to check by means of the *pe* flag described below whether the traversing commands of the axes involved have been completely processed. In addition, this command causes a series of error flags in the *axst* register to be updated. These should likewise be evaluated cyclically, to guarantee reliable operating behaviour of the PCAP program.
BORLAND DELPHI:	procedure rdaxst(var tsrp:TSRP);
C:	void rdaxst(struct TSRP far *tsrp);
VISUAL BASIC:	Sub rdaxst(DTSRP As TSRP)
TSRP COMPONENTS:	TSRP[n].axst
RETURN VALUE:	After this command has been executed, the bit-coded return value is located in the structure/record component *axst*, with the structure described in the table below.

Table 12: Bit-decoded structure of the axst word

Bit No.	Name	Function
0 **0000 0001**	***-***	Not assigned, this flag has an undefined value.
1 **0000 0002**	***eo***	Emergency-Out Error-Flag: Has the value 1, when a digital input as EO planned is active.
2 **0000 0004**	***dnr***	Drive-Not-Ready error-flag: Has the value 1, when a digital input (as DR-planned) is inactive.
3 **0000 0008**	***lslh***	Limit-Switch Left Hardware error-flag: Has the value 1, when a digital input (as LSL_SMD, LSL_TOM or LSL_SMA planned) is active.
4 **0000 0010**	***lsrh***	Limit-Switch Right Hardware error-flag: Has the value 1, when a digital input (as LSL_SMD, LSR_TOM or LSR_SMA planned) is active.
5 **0000 0020**	***lsls***	Limit-Switch left software error-flag: Has the value 1, when the left software limit is exceeded. The left software limit is filed in the axis-specific system parameter {sll}. For this flag to become active, two additional conditions must be satisfied: the software limit must be planned with one of the functions TOM or SMA and the shp() command must heave been executed beforehand.
6 **0000 0040**	***lsrs***	Limit-Switch right software error-flag: has the value 1, when the right software limit is exceeded. The right software limit is filed in the axis-specific system parameter {slr}. For this flag to become active, two additional conditions must be satisfied: The software limit must be planned with one of the functions TOM or SMA and the shp() command must heave been executed beforehand.
7 **0000 0080**	***mpe***	Maximum Position error-flag: Has the value 1, when the permissible position error has been exceeded. The maximum permitted position error is specified in system parameter {mpe}. The PCAP commands wrmpe() and rdmpe() can be used to alter the parameter even during run time.
8 **0000 0100**	***dhef***	Data Handling error-flag: has the value 1, when a data error (e.g. inconsistent profile data) is detected by the rw_MOS operating system. In certain cases, when this bit occurs, the control loops of the axis concerned in each case are opened. The resetting of this bit is only possible by a system restart (BootFile) or by the execution of the ra() [chapter 4.4.42] or rs() [chapter 4.4.112] commands. If necessary, also the system variable ErrorReg must be taken into consideration.
9 **0000 0200**	***cef***	Data Configuration error-flag. The cef flag is set when the information for operating modes, signal processing or CPU number on the APCI-800x do not agree with the system data (system.dat). The configuration-check is carried out automatically after the following events: • after every reset statement (e.g. PCAP command *rs()*) • after every transfer of the *system.dat* system file with the PCAP command *txbf2()*. The cause of the error can be eliminated by saving the system data in the [Save Changes] menu.
10..11		Not assigned, these flags always have an non-definied value .
12 **0000 1000**	***pe***	Profile-End status-flag: Has the value 1, when the end of the profile has been reached.
13 **0000 2000**	***cl***	Closed-Loop status-flag: Has the value 1, when the axis channel is in position control.
14 **0000 4000**	***ip***	In-Position Status-flag: Has the value 1, when the profile end has been reached and in addition the difference of setpoint and actual position of the axis channel is smaller then the position differential contained in the axis-specific system parameter {ipw}.
15 **0000 8000**	***ui***	User Input status-flag: Has the value 1, when a digital input (as UI-planned) is active.

Bit No.	Name	Function
16 0001 0000	***lpsf***	The Latch Position Synchronous Flag indicates that latching has occurred synchronously to the sampling cycle [chapter 4.4.25], or that a digital input (planned with the LP function) has been activated (MCFG / Chapter 1.7.2.5). The flag is reset by reading the latched position LP, e.g. by the command rdlp.
17 0002 0000	***referenced***	This flag indicated that the respecting axis is reduced with the command shp. The flag is reset at booting with the commands rs(), ra() or by writing on rp. At stepper motor axes the flag is also reset at opening and closing the control loop. This flag is only available from RWMOS version V2.5.3.16.
18 0004 0000	***refh***	Ref-hardware input flag: Has value 1 if a digital input projected as REF is enabled. This flag is only available from RWMOS version V2.5.3.47.
19 0008 0000	***saf***	Spooler-Asynchronous-Flag – indicates that the spooler of this axis in the interpolation compound is asynchronous. The flag is reset by ResetAxis (ra) or when the control loop is closed (cl). This flag is only available from RWMOS version V2.5.3.88.
18..31		Not assigned, these flags always have a non-defined value and are reserved for future use.

4.4.46 rdaxstb, read axis status bit

DESCRIPTION:	This function can be used to interrogate <u>one</u> piece of the axis status-information of the APCI-800x board. The axis number must be specified in the *an* parameter (0, 1, ... *MAXAXIS*).
BORLAND DELPHI:	function rdaxstb(an:integer; bitnr:integer):integer;
C:	int rdaxstb(int an, int bitnr);
VISUAL BASIC:	Function rdaxstb(ByVal an As Long, ByVal bitnr As Long) As Long
RETURN VALUE:	The function returns the value 1 or if the corresponding input of *bitnr* is active. Assignment of *bitnr* to the axis status information involved is described in Table 12, but in the case of *bitnr* counting starts with the value 1, so that to interrogate *pe*, for example, *bitnr* must have the value 13!
NOTE:	See also PCAP command *rdaxst()*

4.4.47 rdcbcnct, read common buffer CNC-Task

DESCRIPTION:	Each CNC task has a local memory area, referred to as the "Common Buffer", which can be read and written both by the CNC task involved and by a PCAP program. This function can be used to read in the complete CNC-task-specific buffer (or part of it). The function parameter *cbcnct* is used to select the CNC task buffer, the read-in size in bytes and the memory address where this block is to be read in.
BORLAND DELPHI:	function rdcbcnct(var cbcnct:CBCNCT):integer;
C:	int rdcbcnct(struct CBCNCT far *cbcnct);
VISUAL BASIC:	Sub rdcbcnct(DCBCNCT As CBCNCT)
RETURN VALUE:	The function rdcbcnct() has the following bit-coded return value:

Bit number	Return value	Error description
0	0	No error
0	1	Invalid task number
1	0	No error
1	1	Maximum permitted buffer size exceeded. This means that normally, the function returns the value 0.
2	0	No error
2	1	Address error / Memory error

NOTE:	The CNC-task-specific buffer size is 1,000 bytes. The record structure of CBCNCT is to be found in chapter 4.3.2.9. PCAP command *wrcbcnct()*, SAP commands *RDCBx()* and *WRCBx()*

4.4.48 rdcd, read common double

DESCRIPTION:	This function can be used to read in predefined variables of the CNC task. The variables concerned are the *rw_SymPas v*ariables CD0 to CD99. The first parameter here specifies the number -*index*- of the variable you want to have read in. The value range of *index* here is 0 to 999. The second parameter is a pointer to a field with 1,000 double variables.
BORLAND DELPHI:	procedure rdcd(ndx: integer; var cdbuf:CDBUF);
C:	void rdcd(int ndx, struct CDBUF far *cdbuf);
VISUAL BASIC:	Sub rdcd(ByVal ndx As Long, CDBUF As CDBUF)
RETURN VALUE:	The *rdcd()* command enters the current value of the relevant *CD* variable in the field specified with *index*.
NOTE:	The content of all common variables remains stored in memory even after a system reset operation, executed by the *rs()* command, for example. If you do not want this, you should set the variables concerned to the value you want when starting the program. **Special note:** With Index 100, variables 0 to 99 are read together. The variable with Index 100 cannot be read with rdcd. With Index 1000, variables 0 to 999 are read together.

4.4.49 rdci, read common integer

DESCRIPTION:	This command is identical to the PCAP command *rdcd()*, except that here it is not values of the double type that are read in, but of the LONGINT type. The values concerned are the *rw_SymPas* variables CI0 to CI999.
BORLAND DELPHI:	procedure rdci(ndx: integer; var cibuf:CIBUF);
C:	void rdci(int ndx, struct CIBUF far *cibuf);
VISUAL BASIC:	Sub rdci(ByVal ndx As Long, CIBUF As CIBUF)
NOTE:	**Special note:** With Index 100, variables 0 to 99 are read together. The variable with Index 100 cannot be read with rdci. With Index 1000, variables 0 to 999 are read together.

4.4.50 rdcncts, read computerized numeric controller task status

DESCRIPTION:	This command can be used to interrogate the current status of the CNC task selected in *TaskNr* (values 0..3). After this command has been executed, the results can be found in the structure/record *CNCTS*.
BORLAND DELPHI:	procedure rdcncts(TaskNr:integer; var cncts:CNCTS):integer;
C:	void rdcncts(int TaskNr, struct CNCTS far *cncts);
VISUAL BASIC:	Sub rdcncts(ByVal TaskNr As Long, CNCTS As CNCTS)
RETURN VALUE:	The return values obtained in CNCTS after *rdcncts()* has been executed are described in chapter 4.3.2.10.

4.4.51 rdControllerFlags, read Controller Flag register

DESCRIPTION:	This command is used to read the axis-specific bit-coded ControllerFlags register of the RWMOS operating system software.
BORLAND DELPHI:	procedure rdControllerFlags (an: integer; var value: integer);
C:	void rdControllerFlags (long an, long *value);
VISUAL BASIC:	Sub rdControllerFlags (ByVal an As Long, value As Long)
PARAMETER:	With *an*, the axis channel that has to be accessed is indicated (0, 1, ...). In *value*, the bit-coded value of the ControllerFlags register that has to be read is transferred.
NOTE:	With the aid of flags (bits) in the axis-specific ControllerFlags register, different options in the RWMOS.ELF control algorithm can be activated or controlled (see also Chapters 4.4.137 and 6.3.1.4).

4.4.52 rddigi, read digital inputs

DESCRIPTION:	This function you can be used to interrogate the following signal states: The current status of the 16 APCI-800x digital inputs The current status of the zero-track (index) signal from the incremental coder An error of the measured-value-acquisition system put into intermediate storage An edge of the zero-track (index) signal from the incremental coder put into intermediate storage • An edge of the hardware latch signal (strobe) put into intermediate storage. If an input is active, this will be indicated by the bit concerned having the value 1. As an optional extra, all digital inputs in the *mcfg.exe* TOOLSET program can be planned with inversion. It is likewise possible to plan the polarity you want when an incremental coder with index signal is used.
BORLAND DELPHI:	procedure rddigi(var tsrp:TSRP);
C:	void rddigi(struct TSRP far *tsrp);
VISUAL BASIC:	Sub rddigi(DTSRP As TSRP)
TSRP COMPONENTS:	TSRP[n].digi n = 0 .. Number of axes -1
RETURN VALUE:	The bit-encoded return value is located in the *digi* structure or record component and is structured as described in the table printed below.
NOTE:	There is no specified axis assignment for the digital inputs. Bits 16 ... 19 can be reset by means of the *rdigi()* command [chapter 4.4.72]. (MCFG / Chapters 1.7.2.5 and 1.7.2.5.1).

4.4.52.1 Axis-qualifier digi

The register *digi* can be used to check the state of the APCI-800x digital inputs. Active inputs have the value 1 at the concerned bit position.

Table 13: Bit-coded structure of the digi word

Bit No.	Function	X1/Pin APCI-8001 APCI-8008
0	Input 1	9
1	Input 2	10
2	Input 3	11
3	Input 4	12
4	Input 5	13
5	Input 6	14
6	Input 7	15
7	Input 8	16
8	Input 9	42
9	Input 10	43
10	Input 11	44
11	Input 12	45
12	Input 13	46
13	Input 14 **APCI-8001/APCI-8008:** hardware strobe signal for latching the actual position (axis channel 1)	47

Bit No.	Function	X1/Pin APCI-8001 APCI-8008
14	Input 15 **APCI-8001/APCI-8008:** hardware strobe signal for latching the actual position (axis channel 2)	48
15	Input 16 **APCI-8001/APCI-8008:** hardware strobe signal for latching the actual position (axis channel 3)	49
16	Zero track of incremental encoder, axis-specific	--
17	Error of the encoder data acquisition system, axis-specific	--
18	Value of the zero-track signal from the incremental coder (axis-specific) put into intermediate storage	--
19	Value of the latch signal (hardware strobe) (axis-specific) put into intermediate storage	--
20	**APCI-8008:** AEA alarm error encoder channel A	--
21	**APCI-8008:** AEB alarm error encoder channel B	--
22	**APCI-8008:** AEN alarm error encoder channel Index	--
23	**APCI-8008:** AES alarm error encoder group error	--

4.4.53 rddigib, read digital input bit

DESCRIPTION:	This function can be used to interrogate the current state of one APCI-800x digital input and other logic signals. The axis number must be specified in the *an* parameter (0, 1, ... *MAXAXIS*).
BORLAND DELPHI:	function rddigib(an:integer; bitnr:integer):integer;
C:	int rddigib(int an, int bitnr);
VISUAL BASIC:	Function rddigib(ByVal an As Long, ByVal bitnr As Long) As Long
RETURN VALUE:	The function returns the value 1 or TRUE, if the corresponding input of *bitnr* is active.
NOTE:	Bit numbers 17..20 can be reset via the *rdigi()* command [Chapter 4.4.72], (MCFG / Chapters 1.7.2.5 and 1.7.2.5.1) and PCAP command *rddigi()* **Caution:** The bit number counting begins at 1.

Table 14: Assignment of bitnr to the various APCI-800x digital inputs

'bitnr'	Function	X1/Pin APCI-8001 APCI-8008
1	Input 1	9
2	Input 2	10
3	Input 3	11
4	Input 4	12
5	Input 5	13
6	Input 6	14
7	Input 7	15
8	Input 8	16
9	Input 9	42
10	Input 10	43
11	Input 11	44
12	Input 12	45
13	Input 13	46
14	Input 14	47
15	Input 15	48

'bitnr'	Function	X1/Pin APCI-8001 APCI-8008
16	Input 16	49
17	Zero track of incremental encoder, axis-specific	--
18	Error of the encoder data acquisition system, axis-specific	--
19	Value of the zero-track signal from the incremental coder (axis-specific) put into intermediate storage	--
20	Value of the latch signal (hardware strobe) (axis-specific) put into intermediate storage Strobe), axis-specific	--
21	**APCI-8008:** AEA alarm error encoder channel A	--
22	**APCI-8008:** AEB alarm error encoder channel B	--
23	**APCI-8008:** AEN alarm error encoder channel Index	--
24	**APCI-8008:** AES alarm error encoder group error	--
21..32	The flags that are not assigned depending on the control type have an undefined value and are reserved for future use.	--

4.4.54 rddigo, read digital outputs

DESCRIPTION:	This command is used to read the current output status of the APCI-800x digital outputs into the axis-specific structure/record component *digo*. The bits set there represent outputs set.
BORLAND DELPHI:	procedure rddigo(var tsrp:TSRP);
C:	void rddigo(struct TSRP far *tsrp);
VISUAL BASIC:	TSRP[n].digo
TSRP COMPONENTS:	Sub rddigo(DTSRP As TSRP)
RETURN VALUE:	After this command has been executed the bit-coded return values are located in the structure/record component *digo*. This component has the structure/record defined in the PCAP-command wrdigo().

4.4.55 rddigob, read digital output bit

DESCRIPTION:	This function can be used to interrogate the current state of one APCI-800x digital output. The axis number must be specified in parameter *an* (0, 1, ... *MAXAXIS-1*).
BORLAND DELPHI:	function rddigob(an:integer; bitnr:integer):integer;
C:	int rddigob(int an, int bitnr);
VISUAL BASIC:	Function rddigob(ByVal an As Long, ByVal bitnr As Long) As Long
RETURN VALUE:	This function returns the value 1 or TRUE, if the corresponding output of *bitnr* is active. Assignment of *bitnr* to the outputs involved is shown in the PCAP command *wrdigob()*.

4.4.56 rddp, read desired position

DESCRIPTION:	The APCI-800x profile generator computes an internal reference variable, referred to as the "setpoint position" (= desired position). This can be read in with this command. Normally, in the position control operating mode, the actual position [[chapter 4.4.95 - *rdrp()*]] and this setpoint position must be identical, apart from tolerable deviations.
BORLAND DELPHI:	procedure rddp(var tsrp:TSRP);
C:	void rddp(struct TSRP far *tsrp);
VISUAL BASIC:	Sub rddp(DTSRP As TSRP)
TSRP COMPONENTS:	TSRP[n].dp
RETURN VALUE:	After the command has been executed, the setpoint position is available in the *dp* field. The value is returned in the axis-specific position unit.
NOTE:	This setpoint position is also utilized for setpoint/actual-differential formation, for the automatic position error monitoring function.

4.4.57 rddpoffset, read desired position offset

DESCRIPTION:	With this function the currently programmed value of the axis qualifier dpoffset can be read.
BORLAND DELPHI:	function rddpoffset (an: integer; var value: double): integer;
C:	int rddpoffset(int an, double *value);
VISUAL BASIC:	Function rddpoffset (ByVal an As Long, value As Double) As Long
PARAMETER:	With *an*, the axis channel which has to be read out is indicated (0, 1, ...). In *value*, the positon offset which has to be written is returned in the axis-specific position unit.
RETURN VALUE:	0 for success -1 Command is not available in RWMOS version. -4 Time-out, reason unknown, communication with the motion control board is interrupted Other value <> 0 Unknown error at command execution
NOTE	See also chapter 4.4.141

4.4.58 rddpd – read desired position in display unit

DESCRIPTION:	The APCI-800x profile generator computes an internal reference variable, referred to as the “setpoint position" (= desired position). This can be read with this command in the axis-specific display unit.
BORLAND DELPHI:	procedure rddpd(var tsrp:TSRP);
C:	void rddpd(struct TSRP far *tsrp);
VISUAL BASIC:	Sub rddpd(DTSRP As TSRP)
TSRP COMPONENTS:	TSRP[n].dp
RETURN VALUE:	None
EFFECT:	After the command has been executed, the setpoint position is available in the *dp* field. The value is returned in the axis-specific position unit.
NOTE:	See also commands rddp, rdrp, rdrpd

4.4.59 rddv, read desired velocity

DESCRIPTION:	This function returns the axis-specific setpoint velocity of the APCI-800x profile generator. In best case the value read in corresponds to the real axis velocity (actual velocity).
BORLAND DELPHI:	procedure rddv(var tsrp:TSRP);
C:	void rddv(struct TSRP far *tsrp);
VISUAL BASIC:	Sub rddv(DTSRP As TSRP)
TSRP COMPONENTS:	TSRP[n].dv
RETURN VALUE:	After the command has been executed, the setpoint velocity is available in the *dv* register with the axis-specific velocity unit.
NOTE:	The setpoint velocity can only be influenced by corresponding traversing commands.

4.4.60 rddvoffset, read desired velocity offset

DESCRIPTION:	With this function, the currently programmed value of the axis qualifier dvoffset can be read.
BORLAND DELPHI:	function rddvoffset (an: integer; var value: double): integer;
C:	int rddvoffset(int an, double *value);
VISUAL BASIC:	Function rddvoffset (ByVal an As Long, value As Double) As Long
PARAMETER:	With *an*, the axis channel which has to be read out is indicated (0, 1, ...). In *value*, the currently set velocity value is returned in the axis-specific position unit.
RETURN VALUE:	0 for success -1 Command is not available in RWMOS version. -4 Time-out, reason unknown, communication with the motion control board is interrupted Other value <> 0 Unknown error at command execution
NOTE:	For this, also see Chapter 4.4.142

4.4.61 rdEffRadius – Read Effective Radius

DESCRIPTION:	The effective radius can be read with this command for a rotatory axis (see chapter 0).
BORLAND DELPHI:	rdEffRadius (an: integer; var value: double);
C:	void rdEffRadius (long an, double *value);
VISUAL BASIC:	Sub rdEffRadius (an As Long, ByVal value As Double)
PARAMETER:	The axis number is indicated in *an.* The effective radius is returned in *value* in the unit defined through PU.
RETURN VALUE:	None
NOTE:	see chapter 6.3.3

4.4.62 rdepc, read EEPROM programming cycle

DESCRIPTION:	This function can be used to read the instantaneous number of APCI-800x EEPROM programming cycles. The cycle number is increased by one in the EEPROM for every save operation in the TOOLSET program *mcfg.exe*. The EEPROM can be written at least 10,000 times.
BORLAND DELPHI:	Procedure rdepc(var tsrp:TSRP);
C:	void rdepc(struct TSRP far *tsrp);
VISUAL BASIC:	Sub rdepc(DTSRP As TSRP)
TSRP COMPONENTS:	TSRP[n].epc
RETURN VALUE:	After this command has been executed, the current programming cycle number is in the structure/record component *epc*.

4.4.63 rdErrorReg, read Error Register

DESCRIPTION:	This function can be used ro read the Error Register for the RWMOS operating system software.
BORLAND DELPHI:	procedure rdErrorReg(var ErrorReg: integer);
C:	void rdErrorReg (long *ErrorReg);
VISUAL BASIC:	Sub rdErrorReg (ErrorReg As Integer)
RETURN VALUE:	The bit-coded value of the Error Register is returned in ErrorReg. The function has no return value.
NOTE:	For the layout of the Error Register, see next chapter.

4.4.63.1 Register ErrorReg

The *ErrorReg* register displays various error states of the RWMOS operating system software. The register is bit-coded.

Table 15: Bit-coded construction of the ErroReg word

Bit No.	Name	Function	Hex
0	errAxDef	Axis in AS was selected more than once in a positioning command	1
1	errTargetVel	Target velocity <> 0 at spooler end, although ForbidTargetVel set: System has been reset	2
2	errUnit	An invalid unit was used	4
3	errCenterPoint	Invalid center point programmed for circle or a circle with a radius = 0 has been programmed	8
4	errSpooler Overrun	Spooler overrun detected for an axis	10
5	ProfileToSmall	In spooler operation, at least two traverse profiles whose execution time is shorter than the scan time are executed consecutively. This may cause errors in the program flow and is not allowed.	20
6	SplineSizeErr	Too many spline sets loaded	40
7	RotationFail	Error in axis rotation	80
8	PciBusError	Error detected in Interrupt Cause Register of PCI bridge	100
9	CheckMonitor Screen	Incorrect output to Monitor Screen generated	200
10	SsfWait Refused	At least one SSF wait command was ignored, because the target velocity of the previous traversing command did not equal 0. This suggests a programming error in the user software!	400
11	SpoolerLoad Error	Error while writing on the spooler, as at the same time, a positioning profile was generated by the system. This may happen if, for example, a limit switch switches during the call of an interpolation command.	800

Bit No.	Name	Function	Hex
12	VelocityZero	This bit indicates that an interpolation command with a traversing velocity = 0 was detected. Depending on the bit InhibitProfileRefuse (register MODEREG Chapter 6.3.1.5), the profile is automatically rejected. This suggests a programming error in the user software or a configuration problem of the user!	1000
13	AccelZero	This bit indicates that an interpolation command with an acceleration = 0 was detected. Depending on the bit InhibitProfileRefuse (register MODEREG Chapter 6.3.1.5), the profile is automatically rejected. This suggests a programming error in the user software or a configuration problem of the user!	2000
14	LimitDefError	An incorrect limit value has been detected in mcpmax, mcpmin, mcpcp or mcpcn (incorrect numerical value).	4000
15	ZeroProfile	An interpolation command has been rejected because the indicated traverse distance is almost or equal 0.	8000
16	RadiusError	A circle or helix command has been rejected because the circle radius to be implemented is almost or equal 0.	0001 0000
17	SpoolerDeep ToLess	The traversing profiles entered in the spooler are not sufficient for the Look-ahead calculation. This causes unnecessary velocity limitations. In worse cases, unauthorised accelerations are possible.	0002 0000
18	IO ResetHandled	In the I/O area of the control system, there has been an exceptional reset. This may suggest a hardware problem.	0004 0000
19	JSatSAFdone	The spooler synchronicity monitoring system has detected an error state and has stopped at least one axis unexpectedly and taken it out of the interpolation group.	0008 0000
20	reserved	This bit is reserved for the handling of an option.	0010 0000
21..31		Reserved for future use; these flags have an undefined value	

4.4.64 rdf, read filter

DESCRIPTION:	This command can be used to read in the current axis-specific PIDF filter coefficients of the APCI-800x board. The default values of these coefficients are specified using the TOOLSET program *mcfg.exe*.
BORLAND DELPHI:	Procedure rdf(var tsrp:TSRP);
C:	void rdf(struct TSRP far *tsrp);
VISUAL BASIC:	Sub rdf(DTSRP As TSRP)
TSRP COMPONENTS:	TSRP[n].kp, TSRP[n].ki, TSRP[n].kd, TSRP[n].kpl, TSRP[n].kfca, TSRP[n].kfcv n = 0 .. Number of axis present -1
RETURN VALUE:	After the command has been executed, the return values are in the TSRP structure/record components listed above.
NOTE:	You will find further details on the PIDF filter in chapter 2.1.2, OM / Chapter 4.1.1, CM / Chapter 6.2 and PCAP command *uf()*

4.4.65 rdGCR, read gear configuration register

DESCRIPTION:	With this function, the axis-specific Gear Configuration Register can be read. [Chapter 6.3.3]
BORLAND DELPHI:	procedure rdGCR (an: integer; var value: integer);
C:	void rdGCR (long an, long *value);
VISUAL BASIC:	Sub rdGCR (ByVal an As Long, value As Long)
PARAMETER:	With *an*, the axis channel which has to be read out is indicated (0, 1, ...). In *value*, the contents of the GCR register is returned.
RETURN VALUE:	None
NOTE:	See also document on the resource interface - GEAR

4.4.66 rdgf, read gear factor

DESCRIPTION:	This function returns the axis-specific gear factor {gf}. The default value is specified using the TOOLSET program *mcfg.exe*.
BORLAND DELPHI:	Procedure rdgf(var tsrp:TSRP);
C:	void rdgf(struct TSRP far *tsrp);
VISUAL BASIC:	Sub rdgf(DTSRP As TSRP)
TSRP COMPONENTS:	TSRP[n].gf
RETURN VALUE:	After the command has been executed, the factor is available in the *gf* field with the axis-specific unit.
NOTE:	The gear factor can be set at any time with the PCAP command *wrgf()*.

4.4.67 rdgfaux, read gear factor auxiliary channel

DESCRIPTION:	This function returns the axis-specific ratio of stepper motor resolution to encoder channel in stepper systems with encoder verification. The default value is 1.0; the value can only be changed at runtime.
BORLAND DELPHI:	function rdgfaux (an: integer; var value: double) : integer;
C:	int rdgfaux(int an, double *value)
VISUAL BASIC:	Function rdgfaux (ByVal an As Long, value As Double) As Long
RETURN VALUE:	After successful execution, the function returns 0. In this case, the axis-specific value of gfaux is available in *value*. With a return value ≠ 0, the value could not be read, because e.g. RWMOS.ELF does not support the command. 0 for success -1 Command is not available in RWMOS version. -4 Time-out, reason unknown, communication with the motion control board is interrupted Other value <> 0 Unknown error at command execution
NOTE:	The factor can be set at any time with the PCAP command *wrgfaux()*. See also Chapter 6.3.3

4.4.68 rdhac, read home acceleration

DESCRIPTION:	This command can be used to read in the axis-specific reference travel acceleration *hac*. The default value is specified using the TOOLSET program *mcfg.exe*.
BORLAND DELPHI:	Procedure rdhac(var tsrp:TSRP);
C:	void rdhac(struct TSRP far *tsrp);
VISUAL BASIC:	Sub rdhac(DTSRP As TSRP)
TSRP COMPONENTS:	TSRP[n].hac
RETURN VALUE:	After the command has been executed, the reference travel acceleration is available in the *hac* field. The value is returned in the axis-specific acceleration unit.
NOTE:	The reference travel acceleration can be set at any time with the PCAP command *wrhac()*.

4.4.69 rdhvl, read home velocity

DESCRIPTION:	This command can be used to read in the axis-specific reference travel velocity *hvl*. The default value is specified using the TOOLSET program *mcfg.exe*.
BORLAND DELPHI:	procedure rdhvl(var tsrp:TSRP);
C:	void rdhvl(struct TSRP far *tsrp);
VISUAL BASIC:	Sub rdhvl(DTSRP As TSRP)
TSRP COMPONENTS:	TSRP[n].hvl
RETURN VALUE:	After the command has been executed, the reference travel velocity is available in the *hvl* field. The value is returned in the axis-specific velocity unit.
NOTE:	The reference travel velocity can be set at any time with the PCAP command *wrhvl()*.

4.4.70 rdifs, read interface status

DESCRIPTION:	This command can be used to read in status information of the APCI-800x.
BORLAND DELPHI:	function rdifs(var tsrp:TSRP): integer;
C:	int rdifs(struct TSRP far *tsrp);
VISUAL BASIC:	Function rdifs(DTSRP As TSRP) As Long
TSRP COMPONENTS:	TSRP[n].ifs
RETURN VALUE:	The bit-coded function value is located in the structure/record component *ifs* and has the structure described in the table below. **Function return value:** 0 for success -1 Command is not available in RWMOS version. -4 Time-out, reason unknown, communication with the motion control board is interrupted

4.4.70.1 Axis qualifier ifs

This register can be used to interrogate various pieces of status information for the APCI-800x. If the status or error information concerned is valid, this is indicated by the value 1 at the bit position involved. The bits represent important internal status information for the APCI-8001. Possible causes of errors can be problems at the voltage supply, EMC or hardware problems and should not actually occur. In case such an error occurs the controlling internal I/0 interface is reset. A normal working process is then ensured once the controller is booted anew.
The status information must be controlled cyclically by an application program.

Table 16: Bit-coded structure of the ifs word

Bit-No.	Function
0	*edv* the system information and data filed in the EPROM are valid.
1	*cncrdy*: The CNC ready to operate relay is active (closed).
16	*pfe*: The Power Fail Error flag is set to "1" whenever the operating voltage at the APCI-800x falls below a threshold voltage of 2.85V. After the module is switched on, the flag is likewise set to "1".
17	*wdog*: The Watchdog flag is set to "1" if the watchdog logic on the APCI-800x has been tripped.
18	*iae*: The Invalid Access Error flag is set to "1" if an invalid access operation has taken place within the *rw_MOS* operating system software.
19	*scwdog*: The watchdog flag is set to „1“, if the watchdog safety logic (Secondary circle) has tripped the APCI-800x.
20	*scpfe*: The Power Fail Error flag is always set to „1“, when the operating voltage at the APCI-800x falls below a threshold of 4.75V. After the module is switched on, the flag is likewise set to „1“.
21	Bus Error flag: indicates an error in communication, e.g. with ENDAT, SSI or EtherCAT
22	EpmBaseResetFlag: An unexpected hardware reset in the I/O area of the motherboard has been detected. This indicates EMC problems or a hardware error.
23	EpmOpmfResetFlag: An unexpected hardware reset in the I/O area of the option print has been detected. This indicates EMC problems or a hardware error.
24..31	Not assigned, these flags have an undefined value.

Note: In an initialisation routine of the *rw_MOS* firmware, error flags 16 ... 20 are copied from an internal logic register into the *ifs* register. The logic register is then erased, i.e. the flags are no longer available after a second booting routine (*BootFile*). The flags can also be reset by the *rifs()* command [chapter 0].

4.4.71 rdifsb, read interface status bit

DESCRIPTION:	This function can be used to interrogate one piece of APCI-8001 interface status information. The axis number must be specified in the *an* parameter (0, 1, ... *MAXAXIS-1*)
BORLAND DELPHI:	function rdifsb(an:integer; bitnr:integer):integer;
C:	int rdifssb(int an, int bitnr);
VISUAL BASIC:	Function rdifsb(ByVal an As Long, ByVal bitnr As Long) As Long
RETURN VALUE:	This function returns the value 1 or TRUE, if the corresponding input of *bitnr* is active. Assignment of *bitnr* to the status information concerned is described in Table 16. However, for *bitnr*, counting starts with the value 1. This means that *bitnr* must have the value 1 to retrieve *edv*, for example!
NOTE:	See also PCAP command *rdifs()*

4.4.72 rdigi, reset digital inputs

DESCRIPTION:	This function can be used to clear axis-specific status information(s) filed in *digi*.
BORLAND DELPHI:	procedure rdigi(var tsrp:TSRP);
C:	void rdigi(struct TSRP far *tsrp);
VISUAL BASIC:	Sub rdigi(DTSRP As TSRP)
TSRP COMPONENTS:	TSRP[n].digi n = 0 .. number of axes -1
NOTE:	*rddigi()* [Chapter 4.4.52]

4.4.73 rdipw, read in position window

DESCRIPTION:	This function returns the axis-specific In-Position Window.
BORLAND DELPHI:	procedure rdipw(var tsrp:TSRP);
C:	void rdipw(struct TSRP far *tsrp);
VISUAL BASIC:	Sub rdipw(DTSRP As TSRP)
TSRP COMPONENTS:	TSRP[n].ipw
NOTE:	After the command has been executed, the In-Position Window is available in the *ipw* register in the axis-specific position unit. PCAP command *wripw()*

4.4.74 rdirqpc, read interrupt request PC

DESCRIPTION:	This command can be used to interrogate the instantaneous status of the interrupt source generated on the APCI-800x board. If the interrupt is active, the function returns the value 1, otherwise the value 0.
BORLAND DELPHI:	function rdirqpc: integer;
C:	int rdirqpc(void);
VISUAL BASIC:	Function rdirqpc() As Long
NOTE:	The interrupt can be set or reset by the system variable *IRQPC* using an SAP program [chapter 6.3.1.1 - PC interrupt generation].

4.4.75 rdjac, read jog acceleration

DESCRIPTION:	This command can be used to read in the axis-specific *jog* acceleration *jac*. The default value is specified using the TOOLSET program *mcfg.exe*.
BORLAND DELPHI:	procedure rdjac(var tsrp:TSRP);
C:	void rdjac(struct TSRP far *tsrp);
VISUAL BASIC:	Sub rdjac(DTSRP As TSRP)
TSRP COMPONENTS:	TSRP[n].jac
RETURN VALUE:	After the command has been executed, the *jog* acceleration is available in the *jac* field. The value is returned in the axis-specific acceleration unit.
NOTE:	The *jog* acceleration can be set at any time with the PCAP command *wrjac()*.

4.4.76 rdJerkRel, read jerkrel

DESCRIPTION:	This command can be used to read in the axis-specific paramter *jerkrel* in *value*.
BORLAND DELPHI:	procedure rdJerkRel (an: integer; var value: double);
C:	void rdJerkRel (long an, double *value);
VISUAL BASIC:	Sub rdJerkRel (an As Long, ByVal value As Double)
PARAMETER:	an = axis number (0..n) Double = free variable for function value
RETURN VALUE:	None
NOTE:	*jerkrel* has always a value from 0..1. See also chapter 6.3.3.

4.4.76.1 Axis qualifier *jerkrel*

This variable can parameterise the acceleration characteristics for S-form speed profiles (jerk limitation). This factor is only effective when an S profile is selected (see register MODEREG chapter 6.3.1.5) and has the following meaning:
The acceleration defined for S profiles is constantly the medium acceleration above the whole acceleration/deceleration process. The maximum acceleration in the acceleration/braking ramp is calculated as follows:

amax = a * (1 + jerkrel)

The value of jerkrel has the following consequence on the acceleration course.

0 = rectangular acceleration course
1 = triangular acceleration course
inbetween = trapezoidal acceleration course

Example: The value 0.2 is allocated to jerkrel.

The acceleration has now a trapezoidal course for all profiles.
The maximum acceleration in the middle of the trapez is 1.2 times faster as the set acceleration.

The medium acceleration above the whole acceleration/deceleration process is the programmed acceleration (jac at JOG commands or trac at MOVE commands).
Values between 0 and 1 are possible for jerkrel. The default value is 1. Values out of the range 0..1 are limited either to 0 or 1.

4.4.77 rdjtvl, read jog target velocity

DESCRIPTION:	This command can be used to read in the axis-specific *jog* target velocity *jtvl*. The default value is specified using the TOOLSET program *mcfg.exe*.
BORLAND DELPHI:	procedure rdjtvl(var tsrp:TSRP);
C:	void rdjtvl(struct TSRP far *tsrp);
VISUAL BASIC:	Sub rdjtvl(DTSRP As TSRP)
TSRP COMPONENTS:	TSRP[n].jtvl
RETURN VALUE:	After the command has been executed, the *jog* target velocity is available in the *jtvl* field. The value is returned in the axis-specific velocity unit.
NOTE:	The jog target velocity can be set at any time using the PCAP command *wrjtvl()*.

4.4.78 rdjvl, read jog velocity

DESCRIPTION:	This command can be used to read in the axis-specific *jog velocity jvl*. The default value is specified using the TOOLSET program *mcfg.exe*.
BORLAND DELPHI:	Procedure rdjvl(var tsrp:TSRP);
C:	void rdjvl(struct TSRP far *tsrp);
VISUAL BASIC:	Sub rdjvl(DTSRP As TSRP)
TSRP COMPONENTS:	TSRP[n].jvl
RETURN VALUE:	After the command has been executed, the *jog* velocity is available in the *jvl* field. The value is returned in the axis-specific velocity unit.
NOTE:	The jog velocity can also be set at any time using the PCAP command *wrjvl()*.

4.4.79 rdledgn, read led green

DESCRIPTION: **APCI-8001:** **APCI-8008:**	This function can be used to read in the current state of LED D29 (green). This function can be used to read in the current state of LED D53 (green).
BORLAND DELPHI:	function rdledgn: integer;
C:	int rdledgn(void);
VISUAL BASIC:	Function rdledgn() As Long
RETURN VALUE:	The function's return value is 1, provided the LED is switched on, otherwise it is 0.
NOTE:	PCAP command *wrledgn()*, system variable *LEDGN*

4.4.80 rdledrd, read led red

DESCRIPTION: **APCI-8001:** **APCI-8008:**	This function can be used to read in the current state of LED D31 (red). This function can be used to read in the current state of LED D56 (red).
BORLAND DELPHI:	function rdledrd: integer;
C:	int rdledrd(void);
VISUAL BASIC:	Function rdledrd() As Long
NOTE:	PCAP command *wrledrd()*, system variable *LEDRD*

4.4.81 rdledyl, read led yellow

DESCRIPTION: **APCI-8001:** **APCI-8008:**	This function can be used to read in the current state of LED D30 (yellow). This function can be used to read in the current state of LED D55 (yellow).
BORLAND DELPHI:	function rdledyl: integer;
C:	int rdledyl(void);
VISUAL BASIC:	Function rdledyl() As Long
NOTE:	PCAP command *wrledyl()*, system variable *LED*YL

4.4.82 rdlp, read latched position

DESCRIPTION:	This function returns the axis-specific latch position. The latching procedure can be triggered by various mechanisms: 1. When an input planned with LP function is activated. Here, the maximum time delay is two scan intervals (2.56 ms). A new latching procedure will only be enabled after the latching input has been de-activated. This method should only be used if 3 is not possible. 2. If an *lps()* PCAP command [chapter 4.4.25] has previously been executed and the time delay specified there in the *mst* parameter has elapsed. This method should only be used if 3 is not possible. 3. In real time (max. 1 μs time delay) by means of default-setting APCI-800x digital inputs. A new latching procedure will only be enabled after the latching input has been de-activated. This is the preferred method for the acquisition of latched position values. In all these methods, the actual position {*rp}* of the motor axis is put into intermediate storage. In stepper motor systems or analog feedback with encoder verification, also the auxiliary channel AUX can be latched if the option "*Use Encoder for position feedback"* is activated (mcfg system data).
BORLAND DELPHI:	procedure rdlp(var tsrp:TSRP);
C:	void rdlp(struct TSRP far *tsrp);
VISUAL BASIC:	Sub rdlp(DTSRP As TSRP)
TSRP COMPONENTS:	TSRP[n].lp
RETURN VALUE:	After the function has been executed, the latch position is available in the *lp* register in the axis-specific position unit. The priority of the three methods is the same as the order of their listing, i.e. real-time latching has top priority
NOTE:	PCAP command *wrlp()*

4.4.83 rdlpndx, read latched position index

DESCRIPTION:	This function returns the axis-specific latch position of the index signal (zero track). When the incremental coder's zero track is activated, the actual position {*rp}* of the motor axis in real time is put into intermediate storage.
BORLAND DELPHI:	procedure rdlpndx(var tsrp:TSRP);
C:	void rdlpndx(struct TSRP far *tsrp);
VISUAL BASIC:	Sub rdlpndx(DTSRP As TSRP)
TSRP COMPONENTS:	TSRP[n].lp
RETURN VALUE:	After the function has been executed, the latch position is available in the *lp* register in the axis-specific position unit.
NOTE:	Latching of the incremental coder's zero track is helpful in the coder verification routine and for reference travel programming. PCAP command *wrlpndx()*

4.4.84 rdlsm, read left spool memory

DESCRIPTION:	This command returns the free spool area in bytes. By means of a PCAP or SAP-application program, the freely available spool area can be interrogated at any time you want and reloaded if necessary. This enables you to load very large traversing profiles without interrupting profile generation. The spool area is loaded with *spool* commands, using both programming methods (PCAP and SAP). All *spool* commands cause the freely available spool area to decrease and all commands executed from the spool area cause it to grow again.
BORLAND DELPHI:	procedure rdlsm(var tsrp:TSRP);
C:	void rdlsm(struct TSRP far *tsrp);
VISUAL BASIC:	Sub rdlsm(DTSRP As TSRP)
TSRP COMPONENTS:	TSRP[n].lsm
NOTE:	The spooler size is axis-specific, i.e. the free spool area of the individual axis channels may vary significantly. Approx. 145 kByte of spool area are available for each axis channel. The required spool memory per command can be modified by the future operating system versions. It should not be used to determine the traverse profiles present in the spooler.

4.4.85 rdMaxAcc – Read Maximum Acceleration Check

DESCRIPTION:	With this command the maximum axis-specific acceleration value (*MaxAcc*) can be read. This value can be used by RWMOS operating system software in order to limit the trajectory acceleration so that no axis involved in a linear interpolation exceeds the maximum acceleration accepted. If required, the trajectory acceleration can be reduced.
BORLAND DELPHI:	rdMaxAcc (an: integer; var value: double);
C:	void rdMaxAcc (long an, double *value);
VISUAL BASIC:	Sub rdMaxAcc (an As Long, ByVal value As Double)
PARAMETER:	The axis number is indicated in *an*, the maximum acceleration accepted is returned in *value*. This value is always interpreted in the interpolation unit.
RETURN VALUE:	None
NOTE:	See chapter 4.4.160 and 6.3.3

4.4.86 rdMaxVel – Read Maximum Velocity Check

DESCRIPTION:	With this command the maximum axis-specific velocity value (*MaxVel*) can be read for linear interpolation commands. The value can be used by RWMOS operating system software in order to limit the trajectory velocity so that no axis involved in a linear interpolation exceeds the maximum velocity accepted. If required, the trajectory velocity can be reduced.
BORLAND DELPHI:	rdMaxVel (an: integer; var value: double);
C:	void rdMaxVel (long an, double *value);
VISUAL BASIC:	Sub rdMaxVel (an As Long, ByVal value As Double)
PARAMETER:	The axis number is indicated in *an*, the maximum velocity accepted is returned in *value*. This value is always interpreted in the interpolation unit.
RETURN VALUE:	None
NOTE:	See chapter 4.4.161 and 6.3.3

4.4.87 rdMCiS – Read Move Commands in Spooler

DESCRIPTION:	With this function the number of motion commands in the spooler of an axis.
BORLAND DELPHI:	procedure rdMCiS (an: integer; var value: integer);
C:	void rdMCiS (long an, long *value);
VISUAL BASIC:	Sub rdMCiS (an As Long, ByVal value As Long)
PARAMETER:	an = axis number
RETURN VALUE:	The number of motion commands in spooler of the correspondung axis is returned in value.
COMMENT:	This functionality is only available in versions from May 2002 and later.
NOTE:	This commands gives the current process state in Spooler.

4.4.88 rdmcp, read motor command port

DESCRIPTION:	This command can be used to read in the current command values of the Motor-Command-Ports.
BORLAND DELPHI:	procedure rdmcp(var tsrp:TSRP);
C:	void rdmcp(struct TSRP far *tsrp);
VISUAL BASIC:	Sub rdmcp(DTSRP As TSRP)
TSRP COMPONENTS:	TSRP[n].mcp
RETURN VALUE:	The return value is available in the *mcp* field after the command has been executed. In the case of servo axes, a value in the range -32,767 .. 32,767 is returned. This corresponds to a setpoint output voltage of approx. -10V .. +10V. In the case of stepping motor axes, this value is a time-delay, which is determinant for the stepping frequency outputted. The time-delay can be converted into the unit [s] as follows: tver = (mcp+1) * 2 / CLOCK; *Example: with mcp = 12,499 and CLOCK = 70MHz* *tver = 0.333ms and f = 3kHz* Each time the time-delay *tver* elapses, the pulse signal is switched over, i.e. after *2*tver* a stepping signal with f = 1 / (2*tver) [Hz] is outputted. The value returned in *mcp* lies within the range of -1,048,575 .. +1,048,575. The sign determines the current sense of rotation, i.e. for computing *tver* only the absolute value of *mcp* must be utilized. If the value 0 is returned in *mcp*, this means that no stepping signal is being output, i.e. the motor is at a standstill.

4.4.89 rdMDVel – Read Maximum Velocity Skip

DESCRIPTION:	The maximum axis-specific velocity jump (*MDVEL*) can be read with this command. The value is used by the Look-ahead functionality of the operating system software RWMOS to reduce the trajectory velocity so that no axis involved in an interpolation exceeds the maximum velocity jump accepted.
BORLAND DELPHI:	rdMDVel (an: integer; var value: double);
C:	void rdMDVel (long an, double *value);
VISUAL BASIC:	Sub rdMDVel (an As Long, ByVal value As Double)
PARAMETER:	The axis number is indicated in *an*, the maximum velocity jump accepted is returned in *value*. This value is always interpreted in the interpolation-specific velocity unit.
RETURN VALUE:	None
NOTE:	See Chapter 0 and 6.3.3

4.4.90 rdModeReg – Read MODEREG

DESCRIPTION:	With this command the register MODEREG of the operating system software RWMOS can be read.
BORLAND DELPHI:	rdModeReg (var value: integer);
C:	void rdModeReg(long *value);
VISUAL BASIC:	rdModeReg (value As Long)
PARAMETER:	The ModeReg is returned in value.
NOTE:	See also chapters 6.3.1.5 and 4.4.164.

4.4.91 rdmpe, read maximum position error

DESCRIPTION:	This function returns the axis-specific position error limit value.
BORLAND DELPHI:	procedure rdmpe(var tsrp:TSRP);
C:	void rdmpe(struct TSRP far *tsrp);
VISUAL BASIC:	Sub rdmpe(DTSRP As TSRP)
TSRP COMPONENTS:	TSRP[n].mpe
NOTE:	After the function has been executed, the maximum permitted position error is available in the *mpe* register in the axis-specific position unit. PCAP command *wrmpe()*

4.4.92 rdnfrax – read No-Feed-Rate-Axis

DESCRIPTION:	With this command, the register NFRAX of the RWMOS operating system software is read.
BORLAND DELPHI:	Rdnfrax (var value: integer);
C:	void rdnfrax (long *value);
VISUAL BASIC:	Sub rdnfrax (ByVal value As Long)
PARAMETER:	In value NFRAX is returned
NOTE:	See also chapters 6.3.1 and 4.4.166

4.4.93 rdPosErr, read Position Error

DESCRIPTION:	This function returns the axis-specific position error of the APCI-800x controller actual value channel.
BORLAND DELPHI:	procedure rdPosErr (var an: integer; var value: double);
C:	void rdPosErr (long an, double *value)
VISUAL BASIC:	Sub rdPosErr (an As Long, value As Double)
PARAMETER:	*an* = Number of axes (0..n) *value* = read position error
RETURN VALUE:	After this function has been executed the position error of the axis *an* is available in the variables *value* in the axis-specific poition unit.
NOTE:	The position error cannot be written in [Chapter 6.3.3]

4.4.94 rdPcapIndex

DESCRIPTION:	This command can be used to read the axis-specific variable PcapIndex.
BORLAND DELPHI:	function rdPcapIndex (an: integer; var PcapIndex: integer): integer;
C:	int rdPcapIndex (int an, int * PcapIndex);
VISUAL BASIC:	Function rdPcapIndex (ByVal an As Long, PcapIndex As Long) As Long
PARAMETER:	The axis channel to be read out is indicated by an (0, 1, ...). In PcapIndex, the index value to be read is returned.
RETURN VALUE:	0 for success -1 Command is not available in RWMOS version. -4 Time-out, reason unknown, communication with the motion control board is interrupted Other value <> 0 Unknown error at command execution
NOTE:	See also smlai, smlri, spda, spdr, ssfi, ssfni.

4.4.95 rdrp, read real position

DESCRIPTION:	This function returns the axis-specific current position (= actual position or real position). The position can be read out at any time you want, even while the axis is being moved. A new actual value is available in each scan cycle (1.28 ms).
BORLAND DELPHI:	procedure rdrp (var tsrp:TSRP);
C:	void rdrp(struct TSRP far *tsrp);
VISUAL BASIC:	Sub rdrp(DTSRP As TSRP)
TSRP COMPONENTS:	TSRP[n].rp
NOTE:	After the function has been executed, the current position is available in the *rp* register, in the axis-specific position unit.

4.4.96 rdrpd – read real position in display unit

DESCRIPTION:	The function returns the current axis-specific position (= real position) in axis-specific display unit. The position can be read at any time also during the running of an axis. Per scan cycle (1.28 ms) a new actual value is available.
BORLAND DELPHI:	procedure rdrpd(var tsrp:TSRP);
C:	void rdrpd(struct TSRP far *tsrp);
VISUAL BASIC:	Sub rdrpd(DTSRP As TSRP)
TSRP COMPONENTS:	TSRP[n].rp
RETURN VALUE:	None
EFFECT:	After the execution of the function, the real position is available in the *rp* field, in the axis-specific display unit.
NOTE:	See also the commands rddp, rdrp, rddpd

4.4.97 rdrv, read real velocity

DESCRIPTION:	The function returns the axi-specific actual velocity of the APCI-800x controller actual valu channel.
BORLAND DELPHI:	Procedure rdrv (var an: integer; var value: double);
C:	void rddv (int *an, double *value);
VISUAL BASIC:	Sub rddv (an As Long, value As Double)
PARAMETER:	*an* = Number of axes (0..n) *value* = Read velocity value
RETURN VALUE:	After the execution of the function, the actual velocity is available in the variable *value* in the axis-specific velocity unit.
NOTE:	The actual velocity cannot be written in, yet can be read even if the control loop is open.

4.4.98 rdSampleTime – Read Sample Time

DESCRIPTION:	This command can be used to determine the sample time of the control.
BORLAND DELPHI:	function rdSampleTime (var value: integer) as integer;
C:	void rdSampleTime(long *value);
VISUAL BASIC:	Function rdSampleTime (ByVal value As Long) As Long
RETURN VALUE:	1 for success, 0 for failure, where e.g. RWMOS.ELF does not yet support this function
PARAMETERS:	Sample time returned in µs
NOTE:	The sample time is displayed as a whole number of µs. The default value is 1280.

4.4.99 rdsdec, read stop deceleration

DESCRIPTION:	This command returns the axis-specific stop deceleration *sdec*. The default value is specified using the TOOLSET program *mcfg.exe*.
BORLAND DELPHI:	procedure rdsdec(var tsrp:TSRP);
C:	void rdsdec(struct TSRP far *tsrp);
VISUAL BASIC:	Sub rdsdec(DTSRP As TSRP)
TSRP COMPONENTS:	TSRP[n].sdec
RETURN VALUE:	After the command has been executed the stop deceleration is available in the *sdec* field. The value is returned in the axis-specific acceleration unit.
NOTE:	The stop deceleration can be set at any time using the PCAP-command *wrsdec()*.

4.4.100 rdsll, read software limit left

DESCRIPTION:	This function returns the axis-specific left software limit position.
BORLAND DELPHI:	procedure rdsll(var tsrp:TSRP);
C:	void rdsll(struct TSRP far *tsrp);
VISUAL BASIC:	TSRP[n].sll
TSRP COMPONENTS:	Sub rdsll(DTSRP As TSRP)
NOTE:	After the function has been executed, the left software limit position is available in the *sll* register in the axis-specific position unit. PCAP command *wrsll()*

4.4.101 rdslr, read software limit right

DESCRIPTION:	This function returns the axis-specific right software limit position.
BORLAND DELPHI:	procedure rdslr(var tsrp:TSRP);
C:	void rdslr(struct TSRP far *tsrp);
VISUAL BASIC:	Sub rdslr(DTSRP As TSRP)
TSRP COMPONENTS:	TSRP[n].slr
NOTE:	After the function has been executed, the right software limit position is available in the *slr* register in the axis-specific position unit. PCAP command *wrslr()*

4.4.102 rdslsp, read Slits / Stepperpulses

DESCRIPTION:	This function determines the axis-specific resolution per motor turn {slsp} at encoder or stepper motor system. The default value is determined with the TOOLSET program *mcfg.exe*.
BORLAND DELPHI:	functionrdslsp (an: integer; var value: double): integer;
C:	int rdslsp (long an, double *value);
VISUAL BASIC:	Function rdslsp (ByVal an As Long, value As Double) As Long
TSRP-COMPONENTS:	None
RETRUN VALUE:	1 when succesful, 0 when not successful, e.g.if the function RWMOS.ELF is not yet supported. After successful execution of the function, the factor slsp in value is available in units, which were selected in mcfg.
NOTE:	slsp can be set with the PCAP command *wrslsp().* See also axis qualifier slsp.

4.4.103 rdtp, read target position

DESCRIPTION:	This function can be used to interrogate the axis-specific target position. The target position is always returned as an absolute distance or angle quantity.
BORLAND DELPHI:	Procedure rdtp(var tsrp:TSRP);
C:	void rdtp(struct TSRP far *tsrp);
VISUAL BASIC:	Sub rdtp(DTSRP As TSRP)
TSRP COMPONENTS:	TSRP[n].tp
NOTE:	After the function has been executed, the target position of the last traversing command is available in the *tp* register in the axis-specific position unit. This command is used for monitoring purposes only.

4.4.104 rdtpd – read target position in display unit

DESCRIPTION:	This function can be used to interrogate the target position) in the axis-specific display unit. The target position is always returned as an absolute position value.
BORLAND DELPHI:	Procedure rdtpd(var tsrp:TSRP);
C:	void rdtpd(struct TSRP far *tsrp);
VISUAL BASIC:	Sub rdtpd(DTSRP As TSRP)
TSRP COMPONENTS:	TSRP[n].tp
RETURN VALUE:	none
EFFECT:	After the function has been executed, the target position of the last traversing command is available in the *tp* register in the axis-specific position unit. This command is used for monitoring purposes only.
NOTE:	See also commands rdtp, rddp, rdrp, rdrpd, rddpd

4.4.105 rdtrac, read trajectory acceleration

DESCRIPTION:	Read the RWMOS system variable TRAC
BORLAND DELPHI:	function rdtrac (var value:double): integer;
C:	int rdtrac (double *value);
VISUAL BASIC:	Function rdtrac (value As Double) as long
RETURN VALUE:	0 for success -1 Command is not available in RWMOS version. -4 Time-out, reason unknown, communication with the motion control board is interrupted Other value <> 0 Unknown error at command execution
NOTE:	TRAC is the interpolation trajectory acceleration which is used with interpolation commands that are called from the rw_SymPas programming environment (see also *rw_SymPas* system parameter in Table 32).

4.4.106 rdtrovr, read trajectory override

DESCRIPTION:	This command reads a state variable of the currently set trajectory velocity correction value, which is taken into account for all interpolation commands (*move* commands) and the correspondingly selected axes (PCAP command *utrovr()*).
BORLAND DELPHI:	procedure rdtrovr(var value:double);
C:	void rdtrovr(double *value);
VISUAL BASIC:	Sub rdtrovr(value As Double)
RETURN VALUE:	After the command has been executed, the trajectory velocity correction value will be in the *value* variable.
NOTE:	PCAP commands *utrvr()*, *wrtrovr()*, *wrjovr()*, *rdtrovr()* and *rdjovr()*

4.4.107 rdtrovrst, read trajectory override settling time

DESCRIPTION:	With this command the programmed override-settling-time (see wrtrovrst chapter 4.4.175) can be read out.
BORLAND DELPHI:	function rdtrovr(var value:double) : integer;
C:	int rdtrovr(double *value);
VISUAL BASIC:	Function rdtrovr(value As Double) as long
RETURN VALUE:	0 for success -1: Command is not available in RWMOS version -4: Time-out, reason unknown, communication with the motion control board is interrupted The set override settling time is returned in *value*.
NOTE:	PCAP command wrtrovrst

4.4.108 rdtrvl, read trajectory velocity

DESCRIPTION:	Read the RWMOS system variable TRVL
BORLAND DELPHI:	function rdtrvl (var value:double): integer;
C:	int rdtrvl (double *value);
VISUAL BASIC:	Function rdtrvl (value As Double) as long
RETURN VALUE:	0 for success -1 Command is not available in RWMOS version. -4 Time-out, reason unknown, communication with the motion control board is interrupted Other value <> 0 Unknown error at command execution
NOTE:	TRVL is the interpolation trajectory acceleration which is used with interpolation commands that are called from the rw_SymPas programming environment (see also *rw_SymPas* system parameter in Table 32).

4.4.109 rdtrtvl, read trajectory target velocity

DESCRIPTION:	Read the RWMOS system variable TRTVL
BORLAND DELPHI:	function rdtrtvl (var value:double): integer;
C:	int rdtrtvl (double *value);
VISUAL BASIC:	Function rdtrtvl (value As Double) as long
RETURN VALUE:	0 for success -1 Command is not available in RWMOS version. -4 Time-out, reason unknown, communication with the motion control board is interrupted Other value <> 0 Unknown error at command execution
NOTE:	TRTVL is the interpolation trajectory acceleration which is used with interpolation commands that are called from the rw_SymPas programming environment (see also *rw_SymPas* system parameter in Table 32).

4.4.110 rdzeroOffset, read zero offset

DESCRIPTION:	With this command the currently set axis specific zero offset can be read. The absolute value of th currently set axis specific zero offset is returned in *value* in the axis specific position unit. With the parameter *an* the axis index of the axis channel to be read (0..n) is indicated.
BORLAND DELPHI:	Function rdZeroOffset (an: integer; var value: double) : integer;
C:	Int rdZeroOffset (integer an, double *value);
VISUAL BASIC:	Function rdZeroOffset (ByVal an As Long, value As Double) As Long
RETURN VALUE:	0 for success -1 Command is not available in RWMOS version. -4 Time-out, reason unknown, communication with the motion control board is interrupted Other value <> 0 Unknown error at command execution
NOTE:	The zero offset can be set for example with the PCAP commands szpa (see chapter 4.4.127) or szpr (see chapter 4.4.128).

4.4.111 rifs, reset interface status register

DESCRIPTION:	This command causes various error flags in the APCI-800x interface status register *ifs* (error bits 16, 17, 18 - *pfe*, *wdog* and *iae)* to be reset. Resetting should be performed only in exceptional situations, e.g. in an error monitoring routine.
BORLAND DELPHI:	procedure rifs(var tsrp:TSRP);
C:	void rifs(struct TSRP far *tsrp);
VISUAL BASIC:	Sub rifs(DTSRP As TSRP)
TSRP COMPONENTS:	TSRP[n].ifs
NOTE:	[chapter 0 - *rdifs()*]

4.4.112 RPtoDP, Real-Position to Desired-Position

DESCRIPTION:	This command enables the setpoint position {dp} of an axis to adopt the current position {rp}. The command is executed without delay. It only takes effect though if the relevant axes are not in a positioning profile, as otherwise the setpoint position will be immediately replaced by the computed value of the profile generation. However, it is possible to correct an axis traversing with a target velocity ≠ 0. The relevant axes serve as parameters.
BORLAND DELPHI:	procedure RPtoDP(var as: AS);
C:	void RPtoDP (struct AS far *as);
VISUAL BASIC:	Sub RPtoDP (DASEL As ASEL)
RETURN VALUE:	none
NOTE:	This command can be used when one or more axes are no longer regulated due to a position error caused, for example, by axis blocking. Once the error has been cleared, regulation can be continued from the previous position, even with traversing axes. This command is available from RWMOS.ELF V2.5.3.100 and mcug3.dll V2.5.3.80.

4.4.113 rs, reset system

DESCRIPTION:	This command causes the complete axis system to be reset. The digital outputs are set to the default values planned with the aid of the TOOLSET program *mcfg.exe*. On the setpoint value channels 0 V output voltage is outputted in the case of servo axes and 0 Hz stepping frequency in the case of stepping motor axes. The position control loop is opened for all axes. The spooler data are rejected in their entirety. All CNC task are halted. All software limits planned will no longer be monitored. All override factors (PCAP commands *wrjovr()* and *wrtrovr()*) are set to the value 1.0.
BORLAND DELPHI:	procedure rs;
C:	void rs(void);
VISUAL BASIC:	Sub rs()
NOTE:	All system data, like accelerations, velocities, filter parameters, etc. remain stored in memory and therefore need not be loaded again. The status flags in register *ifs* are not influenced by this command. The contents of all common integers and double variables are retained

4.4.114 scp – set controller params

DESCRIPTION:	This function is used for customised extensions and serves for transferring a parameter field of 15 x 15 floating points (predefined data structure CTRLRPARAMS) axis-specifically to the control process.
BORLAND DELPHI:	procedure scp (an: integer; var ctrlrparams: CTRLRPARAMS);
C:	void scp (long an, struct CTRLRPARAMS *ctrlrparams);
VISUAL BASIC:	Sub scp (ByVal an As Long, DCTRLRPARAMS As CTRLRPARAMS)
RETURN VALUE:	None
EFFECT:	The values in CTRLRPARAMS are transferred to the control system.
NOTE:	Use and significance of the data to be transferred are described according to the application.

4.4.115 sdels, spooler delete synchronous

DESCRIPTION:	All commands entered in the spooler will be rejected. The entire spooler area is again freely available. Spooler data rejection takes place for the axes specified in *AS*.
BORLAND DELPHI:	procedure sdels(var as:AS);
C:	Void sdels(struct AS far *as);
VISUAL BASIC:	Sub sdels(DASEL As ASEL)
NOTE:	The ongoing operation, like a traversing command, will be concluded.

4.4.116 shp, set home position

DESCRIPTION:	This command can be used to set the axis-specific zero (home position). The *tp* parameter is stated in the axis-specific position unit. The command is generally used after a reference search run for setting the machine zero. It can be executed in both operating modes: control loop open and control loop closed. In order to prevent jerky motor movements, however, it should not be used while the selected axis channel is being moved.
BORLAND DELPHI:	procedure shp(var tsrp:TSRP);
C:	Void shp(struct TSRP far *tsrp);
VISUAL BASIC:	Sub shp(DTSRP As TSRP)
TSRP COMPONENTS:	TSRP[n].tp n = 0 .. Number of axes present-1
NOTE:	Until the first time this command is executed, the software limits planned are not being monitored. This means that before execution of the *shp()* command a reference travel can be carried out using all *move* and *jog* commands. After the *shp()* command has been executed, the software limits are monitored until the next *ra()* or *RA()* or *rs()* or *RS* command. Readiness for software limit monitoring is indicated by the ref bit in the axst register. The shp command sets the actual position rp to the indicated value. Here, a possible shift by a “backlash” value persists. A possible position offset due to a value in “dpoffset” remains unconsidered in the actual position, but affects the new value of the setpoint position (dp). shp must not be called when traverse commands are entered in the spooler; especially not in case of commands that are tool radius-corrected or with spline commands.

4.4.117 spd, Spool Position Data

DESCRIPTION:	The spd command can be used to spool position values, which are each taken sampling-synchronously as setpoint position values. By calling this command for several axes with an equal number of sets, an interpolated traverse movement is possible.
BORLAND DELPHI:	procedure spda (an:integer; size:integer; var spdbuf:SPDBUF);
C:	void spda (int an, int size, struct SPDBUF *spdbuf);
VISUAL BASIC:	Sub spda (ByVal an As Long, ByVal size As Long, SPDBUF As SPDBUF)
PARAMETER:	an is the index of the axis to be accessed. In size, the number of calibration points is specified. size may take values between 1 and 1000. spdbuf is an array, in which the position calibration points are passed.
NOTE:	• In this context, the *sstvl* command has to be taken into account if a transition from a trajectory into a profile consisting of *spd* commands is to take place without intermediate stop. • This command is basically the same as spda; only here, no PcapIndex is passed.

4.4.118 spda, Spool Position Data Absolute

DESCRIPTION:	The spda command can be used to spool position values, which are each taken sampling-synchronously as setpoint position values. By calling this command for several axes with an equal number of sets, an interpolated traverse movement is possible. The execution state can be determined using an index.
BORLAND DELPHI:	procedure spda (an:integer; size:integer; var spdbuf:SPDBUF; PcapIndex: integer);
C:	void spda (int an, int size, struct SPDBUF *spdbuf, long PcapIndex);
VISUAL BASIC:	Sub spda (ByVal an As Long, ByVal size As Long, SPDBUF As SPDBUF, ByVal PcapIndex As Long)
PARAMETER:	*an* is the index of the axis to be accessed. In *size*, the number of calibration points is indicated. *size* values may be between 1 and 1000. *spdbuf* is an array in which the position calibration points are transferred. In *PcapIndex*, an index for identifying the current state of the command execution is transferred. For each calibration point value, this index increments in the display. The index can be read using the PcapIndex (# 32) resource.
NOTE:	In this context, the *sstvl* command has to be taken into account if a transition from a trajectory into a profile consisting of *spda* commands is to take place without intermediate stop.

4.4.119 spdr, Spool Position Data Relative

Basically, this command is like *spda*; only in *spdbuf*, the position values are relative coordinates.

4.4.120 ssms, start spooled motions synchronous

DESCRIPTION:	*Spool* commands can be used to transfer commands to the individual axis channels of the APCI-800x: They are entered in a queue. The PCAP command *ssms()* causes a synchronous start for spooler command processing for all axes specified in *AS*.
BORLAND DELPHI:	procedure ssms(var as:AS);
C:	void ssms(struct AS far *as);
VISUAL BASIC:	Sub ssms(DASEL As ASEL)
NOTE:	chapter 2.2.8.2 - Spool-Mode

4.4.121 sstps, spooler stop synchronous

DESCRIPTION:	This command is used to interrupt command processing from the spooler for all axis channels selected in *AS*.
BORLAND DELPHI:	procedure sstps(var as:AS);
C:	void sstps(struct AS far *as);
VISUAL BASIC:	Sub sstps(DASEL As ASEL)
NOTE:	The current command is completely processed. The commands that are in the spooler are preserved and can be continued with SSMS. But please pay attention if the spooler contains traverse commands with target velocities ≠ 0. In error situations, when the spooler contents are to be rejected, it is better to use direct commands to stop the axes concerned (ms or js), as these commands discontinue the current contour and at the same time reject the spooler contents.

4.4.122 sstvl, Spooler Set Target Velocity

DESCRIPTION:	The target velocity of a trajectory of all axes specified in *AS* that is available in the spooler can be set to a defined value with the aid of the sstvl command. The direction of the target velocity corresponds to the direction of the last traverse command. The indicated target velocity must be achievable and must not be limited by system settings such as MdVel, MaxVel or MaxAcc. Otherwise, the target velocity value is reduced accordingly.
BORLAND DELPHI:	procedure sstvl(var as:AS, tvl: double);
C:	void sstvl (struct AS far *as, double tvl);
VISUAL BASIC:	Sub sstvl(DASEL As ASEL, ByVal tvl As Double)
PARAMETER:	In *as*, the axes to be handled are defined. In *tvl*, the desired trajectory velocity is transferred.
NOTE:	This command can be used to continue a trajectory programmed before with SPD commands (spd, spda or spdr) without velocity decrease. Without this command, the profile programmed before would be decelerated to the velocity of 0 at the position transition during active Look-ahead.

4.4.123 ssf, Spool-Special-Function

DESCRIPTION:	This commands allows to enter other commands as traverse commands in the spooler. The command you want to execute is entered with the parameter *command*.
BORLAND DELPHI:	procedure ssf(an: integer; command: integer; value:double); far; stdcall;
C:	void ssf(int axis, int command, double value);
VISUAL BASIC:	Sub ssf(ByVal an As Long, ByVal command As Long, ByVal value As Double)
CALLING PARAMETER:	The value *value* is entered to the axis defined in *axis*. The following commands are available at the moment:

Command	Description
0 .. 999	Describe CI-Variable with *Value*.
1000	Stop the spooler processing, this command is only carried out when the target velocity of the profile last entered is 0.
1001	Set digital outputs, the outputs to be set are specified bitwise in *Value*.
1002	Reset digital outputs, the outputs to be reset are specified bitwise in *Value*.
1003	Stop the spooler processing during the time defined in *Value*. The time unit is 64 µs. The real waiting time is multiplied several times by the scan time. This command is only carried out if the target velocity of the profile last entered is 0. The waiting process can be prematurely ended e.g. with the command SSMS.
1004	Stop the spooler processing until the inputs specified in *Value* are active. The inputs are specified in bit-coded form. This command is only carried out if the target velocity of the profile last entered is 0. The waiting process can be prematurely ended e.g. with the command SSMS. Only inputs of the respective axis group can be specified at a time.
1005	Stop the spooler processing until the value 0 is entered in the common variable CI99. The value entered in *Value* is first entered in CI99. When the command is to be executed the CI99 is set by default and must not be used for any other purpose (see also Chapter 4.4.123.1). **Note:** The PCAP command ClearCI99 **must** be used to delete CI99, because otherwise, the axes may be started asynchronously.
1006	Stop the spooler processing until the command was activated or executed with the same parameter for all bit-coded axes entered in *Value*. With this command the spooler processing can be synchronised by different axes (see also Chapter 4.4.123.1).
1015	The parameter value is added to CI99. Then the spooler processing is stopped until CI99 contains the value 0. The wait command is only executed if the target velocity of the previous profile is 0 (see also Chapter 4.4.123.1). **Note:** The PCAP command ClearCI99 **must** be used to delete CI99, because otherwise, the axes may be started asynchronously.

Command	Description
1025	In the variable CI99, the bit assigned to the axis is set. Then the spooler processing is stopped until this bit is reset in CI99. The wait command is only executed if the target velocity of the previous profile is 0 (see also Chapter 4.4.123.1). **Note:** The PCAP command ClearCI99 **must** be used to delete CI99, because otherwise, the axes may be started asynchronously.
1101	Activate PC interrupt request
1200 ... 1207	Write the Motor-Command-Port mcp of an axis in the system with the index 0..7 1200 = 1. axis, 1201 = 2. axis, etc.
2001	Reset the target velocity in the last spooled traverse profile.
10000 ... 10999	Set bits in CI-variable. The bits to be set are indicated in *Value*.
11000 ... 11999	Reset bits in CI-Variable. The bits to be reset are indicated in *Value*.
20000 ... 20999	Write on CD-variable with *value*.

4.4.123.1 Notes on SSF wait commands

Some of the above described wait commands use the common integer variable CI99.
Here it should be noted that these variables from the PCAP programming are written on via direct PCI memory access and thus asynchronously to the RWMOS operating system software. To achieve error-free synchronicity of axes after a profile continuation via an SSF wait command, the DLL command ClearCI99 must therefore be used.
In some of the commands specified above, bit-coded data, e.g. of inputs, outputs or axes, is expected. Here, the corresponding bit numbers are assigned to the relevant number of the value to be programmed.
Inputs 1 and 3, for example, are to be specified in bit-coded form. In this case, the hexadecimal value 5 must be programmed. This means it is possible to specify multiple axes, inputs or outputs in one data word.

EXAMPLES:	*ssf(A1, 125, 999);*	*// Write CI125 with the value 999*
	ssf(A1, 1001, 1);	*// Set output O1 at axis 1*
	ssf(A1, 1002, 4);	*// Reset output O3 at axis 1*

4.4.124 startcnct, start numeric controller task

DESCRIPTION:	This command can be used to start a previously loaded SAP program. The CNC task selected in *TaskNr* (values 0..3) processes the SAP program right from its beginning. The PCAP command *txbf2()* can be used for loading.
BORLAND DELPHI:	procedure startcnct(TaskNr:integer);
C:	void startcnct(int TaskNr);
VISUAL BASIC:	Sub startcnct(ByVal TaskNr As Long)
NOTE:	A currently running SAP program will be stopped automatically before this command is executed. PCAP command *txbf2()*

4.4.125 stepcnct, step numeric controller task

DESCRIPTION:	This command is used for executing an SAP program line by line.
BORLAND DELPHI:	procedure stepcnct (TaskNr:integer);
C:	void stepcnct(int TaskNr);
VISUAL BASIC:	Sub stepcnct(ByVal TaskNr As Long)
NOTE:	The PCAP command *stepcnct()* has not yet been implemented at present!

4.4.126 stopcnct, stop numeric controller task

DESCRIPTION:	This command causes the SAP program currently being run to stop in the CNC task selected with *TaskNr* (values 0..3) and de-activates this CNC task. The SAP program can be continued with the SAP command *CONTCNCT()* or the PCAP command *contcnct()* .
BORLAND DELPHI:	procedure stopcnct(TaskNr:integer);
C:	void stopcnct(int TaskNr);
VISUAL BASIC:	Sub stopcnct(ByVal TaskNr As Long)
NOTE:	Any EVENT handlers enabled in the SAP program will no longer be processed after the *stopcnct()* command has been executed. Before this command is executed, the drive should be put into a safe operating state.

4.4.127 szpa, set zero position absolute

DESCRIPTION:	This command sets an axis-specific virtual zero position. The parameter *Position* is specified in the axis-specific position unit. The parameter an specifies the axis number. It can be executed in both operating modes: control loop open and control loop closed. In order to prevent jerky motor movements, however it should not be used while the selected axis channel is being moved.
BORLAND DELPHI:	procedure szpa(an: integer; Position: double);
C:	void szpa(int an, double Position);
VISUAL BASIC:	Sub szpa(ByVal an As Long, ByVal Position As Double)
NOTE:	• By calling up szpa with the position value 0, a zero offset which has been possibly set can be deleted. The currently set position value of the zero offset can be read with the command rdZeroOffset Chapter 4.4.110). See also Bit ClearZeroPosition in the register ModeReg. • szpa (szpr) must not be called when traverse commands are entered in the spooler; especially not in case of commands that are tool-radius-corrected or with spline commands.

4.4.128 szpr, set zero position relative

DESCRIPTION:	This commands sets an axis-specific virtual zero position to a relative position. The parameter *Position* is specified in the axis-specific position unit. The parameter an specifies the axis number. It can be executed in both operating modes: control loop open and control loop closed. In order to prevent jerky motor movements, however it should not be used while the selected axis channel is being moved.
BORLAND DELPHI:	procedure szpr(an: integer; Position: double);
C:	void szpr(int an, double Position);
VISUAL BASIC:	Sub szpr(ByVal an As Long, ByVal Position As Double)
NOTE:	• By calling up szpa with the position value 0, a zero offset which has been possibly set can be deleted. The currently set position value of the zero offset can be read with the command rdZeroOffset (Chapter 4.4.110). See also Bit ClearZeroPosition in the register ModeReg. • szpr (szpa) must not be called when traverse commands are entered in the spooler; especially not in case of commands that are tool-radius-corrected or with spline commands.

4.4.129 txbf2, transmit binary file

DESCRIPTION:

This function is used to transfer the file specified in the string or character parameter to the APCI-800x board. The specified file is first searched in the current working directory. Then the directories which are specified in the environment variable PATH are searched. There is an additional function (txbf()) in the fucntion library for compatibility reasons: yet this function txbf() does not support any file name including drive or path information. When the function txbf2 is called up two special file types are essentially permitted.
Firstly, the *system.dat* system file (or files with a compatible structure) and secondly the autocode files (CNC files) with the file extension name ".CNC" generated from the IDE or using the *ncc.exe* command line compiler.
Transferring the *system.dat* system file has the following results:
All axis channels will be initialized with the axis-specific system data. The filter coefficients of the PIDF filter will be recomputed, as with the PCAP command *uf()*. These system data can also be edited in the TOOLSET program *mcfg.exe*. Any system variables previously altered (e.g. axis-specific velocities, accelerations, etc.) are overwritten again by this command.
Important! Transferring CNC files has the following result: the current program main memory of a CNC task is overwritten with the contents of the specified autocode file. This is why the task concerned is automatically halted before the load operation. The CNC file also contains the information on which task it has to be loaded into (Task 0..3). After the CNC file has been successfully transferred, it can be started with the PCAP command *startcnct()* or the PCAP command *STARTCNCT()*.

BORLAND DELPHI:

```
function txbf2(var filename:string):integer;
```

C:

```
int txbf2(char far *filename);
```

VISUAL BASIC:

```
Function txbf2(ByVal filename As String) As Long
```

RETURN VALUE:

The function can return the following values:

Return value	Error description
0	No error
20	File cannot be opened. Possible causes are as follows: - Invalid file name - File does not exist - Path and search drive is invalid
21	The file too large for the CNC task main memory.
22	Invalid file type (Not a SAP file nor a system file)
23	Internal error when reserving memory.
24	Invalid task number is indicated. At a system file this error shows that an invalid or damages system file was used.
25	Data transfer error with remote systems (WebServices)

NOTE:

Normally, the *system.dat* system file need be loaded only once per system start. Please see the particulars given for the PCAP command *mcuinit()* in this context. If you want, you can specify drive and path names in the *filename* parameter.

4.4.130 txbfErrorReport, initialisation error report

DESCRIPTION:	This function gives in plaintext the error return value of the function *txbf2()* described above. A message box is displayed on screen and is to be closed after reading.
BORLAND DELPHI:	procedure txbfErrorReport(filename:PChar; error:integer);
C:	void txbfErrorReport (char *filename, int error);
VISUAL BASIC:	Sub txbfErrorReport (ByVal filename As String, ByVal error As Long)
NOTE:	PCAP commands InitMcuSystem(), InitMcuSystem2() and InitMcuSystem3()
EXAMPLE:	*txbferror = InitMcuSystem3(...); // Execute file transfer* *txbfErrorReport(..., initerror); // In case of error, display error return value*

4.4.131 uf, update filter

DESCRIPTION:	You can use this command to set the APCI-800x PIDF filter for specific axes. Before the command is executed, you must make sure that all the structure components listed above have been initialized. This command can be executed at any time, even during profile generation. This characteristic enables the system to be matched to different load conditions in real time.
BORLAND DELPHI:	procedure uf(var tsrp:TSRP);
C:	void uf(struct TSRP far *tsrp);
VISUAL BASIC:	Sub uf(DTSRP As TSRP)
TSRP COMPONENTS:	TSRP[n].kp, TSRP[n].ki, TSRP[n].kd, TSRP[n].kpl, TSRP[n].kfca, TSRP[n].kfcv n = 0 .. Number of axis present -1
NOTE:	You will find more details on the PIDF filter in chapter 2.1.22, OM / Chapter 4.1.1, CM / Chapter 6.2 and PCAP command *rdf()*.

4.4.132 utrovr, update trajectory override

DESCRIPTION:	The velocity override currently set is taken into account for all axis channels selected in *AS.*
BORLAND DELPHI:	procedure utrovr(var as:AS);
C:	void utrovr(struct AS far *as);
VISUAL BASIC:	Sub utrovr(DASEL As ASEL)
NOTE:	With this command, the last written trajectory override value is adopted for the selected axes. If the bit OvrMode is not set in the Modereg register, this value is adopted for the axis-specific variable jovr. You will find further information under the PCAP command *wrtrovr()*. Depending on the value set through the function wrtrovrst(), the override value is not adopted at once, but is adapted to the given ramp time.

4.4.133 wraux, write auxiliary register

DESCRIPTION:	This function sets the axis-specific auxiliary register to the value set in *aux*.
BORLAND DELPHI:	procedure wraux (var tsrp:TSRP);
C:	void wraux (struct TSRP far *tsrp);
VISUAL BASIC:	Sub wraux (DTSRP As TSRP)
TSRP COMPONENTS:	TSRP[n].aux
NOTE:	See also chapter 4.4.44 and 6.3.3

4.4.134 wrcbcnct, write common buffer CNC-Task

DESCRIPTION:	Each CNC task has a local memory area (referred to as the "Common Buffer"), which can be read and written both by the CNC task concerned and by a PCAP program. This function can be used to write the complete CNC-task-specific buffer (or only a part of it). The *cbcnct* function parameter is used to select the CNC task buffer, the number of bytes to be written and the start address of the block which is to be transferred to the APCI-800x board.
BORLAND DELPHI:	function wrcbcnct(var cbcnct:CBCNCT):integer;
C:	int wrcbcnct(struct CBCNCT far *cbcnct);
VISUAL BASIC:	Sub wrcbcnct(DCBCNCT As CBCNCT)
RETURN VALUE:	The wrcbcnct() function has the following bit-coded return value:

Bit number	
0	0 = No error
0	1 = Task number invalid
1	0 = No error
1	1 = Maximum permitted buffer size exceeded This means that the function in normal circumstances returns the value 0.
2	0 = No error
2	Address error / Memory error

NOTE:	The CNC-task-specific buffer size is 1,000 bytes. The record structure for CBCNCT is to be found in Chapter 4.3.2.9. PCAP command *rdcbcnct()*, SAP commands *RDCBx()* and *WRCBx()*

4.4.135 wrcd, write common double

DESCRIPTION:	This function can be used for write access operations to the common variables, which are predefined variables of the CNC task. The variables concerned are the *rw_SymPas* system variables CD0 .. CD99. The first parameter here specifies the number *ndx* of the double variable to be written. The value range of *ndx* here is 0 to 99. The second parameter is a pointer to the CDBUF structure with 100 double variables. Before the command is executed, the variable to be written must be initialized with the appropriate value you want.
BORLAND DELPHI:	procedure wrcd(ndx: integer; var cdbuf:CDBUF);
C:	void wrcd(int ndx, struct CDBUF far *cdbuf);
VISUAL BASIC:	Sub wrcd(ByVal ndx As Long, CDBUF As CDBUF)
NOTE:	The content of all common variables remains stored in memory even after a system reset operation, which is executed by the *rs()* command, for example. If you do not want this, you should set the variables concerned to the values you want when you start the program.

4.4.136 wrci, write common integer

DESCRIPTION:	This command is identical to the PCAP command *wrcd(),* except that here the variables concerned are the *rw_SymPas* system variables CI0 .. CI999 of the LONGINT type.
BORLAND DELPHI:	procedure wrci(ndx: integer; var cibuf:CIBUF);
C:	void wrci(int ndx, struct CIBUF far *cibuf);
VISUAL BASIC:	Sub wrci(ByVal ndx As Long, CIBUF As CIBUF)
NOTE:	PCAP command *wrcd()*

4.4.137 wrControllerFlags– Write Controller Flags

DESCRIPTION:	This command is used to write on the axis-specific bit-coded ControllerFlags register of the RWMOS operating system software.
BORLAND DELPHI:	procedure wrControllerFlags (an: integer; var value: integer);
C:	void wrControllerFlags (long an, long *value);
VISUAL BASIC:	Sub wrControllerFlags (ByVal an As Long, value As Long)
PARAMETER:	With *an*, the axis channel that has to be accessed is indicated (0, 1, ...). In *value*, the bit-coded value of the ControllerFlags register that has to be written is transferred.
NOTE:	With the aid of flags (bits) in the axis-specific ControllerFlags register, different options in the RWMOS.ELF control algorithm can be activated or controlled (see also Chapters 4.4.51 and 6.3.1.4).

4.4.138 wrdigo, write digital outputs

DESCRIPTION:	This register can be used to set the digital outputs of the APCI-8001. It has to be considered that the digital outputs of the APCI-800x are not grouped in an axis-specific way. If an output is to be set, this can be done by setting the respective bit. The bit-coded structure of the *digo* status word can be found in the table below:

Bit-coded structure of the *digo* word

Bit number	Function	Connector X1 / PIN
0	Output 1	26
1	Output 2	27
2	Output 3	28
3	Output 4	29
4	Output 5	30
5	Output 6	31
6	Output 7	32
7	Output 8	33
8..31	Not assigned	--

BORLAND DELPHI:	procedure wrdigo(var tsrp:TSRP);
C:	void wrdigo(struct TSRP far *tsrp);
VISUAL BASIC:	Sub wrdigo(DTSRP As TSRP)
TSRP COMPONENTS:	TSRP[n].digo

4.4.139 wrdigob, write digital output bit

DESCRIPTION:	This function can be used to set or reset <u>one</u> APCI-8001 digital output. The axis number must be specified in the *an* parameter (0, 1, ... *MAXAXIS-1*). The output is reset with the value 0 or FALSE. Assignment of *bitnr* to the respective APCI-800x digital outputs 'bitnr' Function Connector X1 / PIN 1 Output 1 26 2 Output 2 27 3 Output 3 28 4 Output 4 29 5 Output 5 30 6 Output 6 31 7 Output 7 32 8 Output 8 33 9..32 Not assigned --
BORLAND DELPHI:	procedure wrdigob(an:integer; bitnr:integer; value: integer);
C:	wrdigob(int an, int bitnr, int value);
VISUAL BASIC:	Sub wrdigob(ByVal an As Long, ByVal bitnr As Long, ByVal value As Long)
NOTE:	PCAP command *wrdigo()*

4.4.140 wrdp, write desired position

DESCRIPTION:	You can use this command to write the axis-specific setpoint position (*dp*). This command is normally never needed and should be used only in quite exceptional cases, like testing or commissioning jobs. Alteration of the setpoint position is operative only in the position control operating mode. If there are significant differences between this setpoint position (*dp*) and the current position (*rp*), you must anticipate that the motor will be corrected to this position at maximum system acceleration.
BORLAND DELPHI:	procedure wrdp(var tsrp:TSRP);
C:	void wrdp(struct TSRP far *tsrp) ;
VISUAL BASIC:	Sub wrdp(DTSRP As TSRP)
TSRP COMPONENTS:	TSRP[n].dp
NOTE:	Writing the setpoint position (*dp*) during execution of motion commands may lead to uncontrolled process behaviour and should therefore be avoided. PCAP command *rddp()*

4.4.141 wrdpoffset, write desired position offset

DESCRIPTION:	With this command the axis specific set position offset (*dpoffset*) can be described.
BORLAND DELPHI:	function wrdpoffset (an: integer; var value: double): integer;
C:	int wrdpoffset(int an, double *value);
VISUAL BASIC:	Function wrdpoffset (ByVal an As Long, value As Double) As Long
PARAMETER:	With *an* the axis channel, which has to be called, is indicated (0, 1, ...). In *value* the position offset, which has to be written, is transferred in the axis specific position unit.
RETURN VALUE:	0 for success -1 Command is not available in RWMOS version. -4 Time-out, reason unknown, communication with the motion control board is interrupted Other value <> 0 Unknown error at command execution
NOTE:	In general here only small changes may be programmed, because the set point changes cause a jump of the axs. See also axis-qualifier dpoffset in Table 37. With a dvoffset value (see Chapter 4.4.142), the velocity offset of dpoffset can be parameterised, though. This register can be used e.g. for a regulation, overlaying the position controller, or for a spindle linearisation / spindle correction. **Notice:** This mechanism is used internally by the so-called RTCP (Rotation Tool Center Point) correction. If this correction is used, the setpoint position offset dpOffset must not be written on with the corresponding axes.

4.4.142 wrdvoffset, write desired velocity offset

DESCRIPTION:	With this command, the velocity offset (dvoffset) of the axis-specific setpoint position offset (*dpoffset*) can be written on.
BORLAND DELPHI:	function wrdvoffset (an: integer; var value: double): integer;
C:	int wrdvoffset(int an, double *value);
VISUAL BASIC:	Function wrdvoffset (ByVal an As Long, value As Double) As Long
PARAMETER:	With *an*, the axis channel which has to be accessed is indicated (0, 1, ...). In *value*, the velocity offset which has to be written is returned in the axis-specific position unit.
RETURN VALUE:	0 for success -1 Command is not available in RWMOS version. -4 Time-out, reason unknown, communication with the motion control board is interrupted Other value <> 0 Unknown error at command execution
NOTE:	With the value 0, changes of dpoffset are immediately adopted. The default value is 0.

4.4.143 wrEffRadius – Write Effective Radius

DESCRIPTION:	With the command the effective radius can be written for a rotatory axis.
BORLAND DELPHI:	wrEffRadius (an: integer; var value: double);
C:	void wrEffRadius (long an, double *value);
VISUAL BASIC:	Sub wrEffRadius (an As Long, ByVal value As Double)
PARAMETER:	The axis number is indicated in *an*, the effective radius is transmitted in *value* in the unit which is used per PU.
NOTE:	See chapter 6.3.3

4.4.144 wrGCR, write gear configuration register

DESCRIPTION:	With this function, the axis-specific Gear Configuration Register can be written on. [Chapter 6.3.3]
BORLAND DELPHI:	procedure wrGCR (an: integer; var value: integer);
C:	void wrGCR (long an, long *value);
VISUAL BASIC:	Sub wrGCR (ByVal an As Long, value As Long)
PARAMETER:	With *an*, the axis channel which has to be read out is indicated (0, 1, ...). In *value*, the contents of the GCR register is returned.
RETURN VALUE:	None
NOTE:	See also document on the resource interface - GEAR

4.4.145 wrgf, write gear factor

DESCRIPTION:	You can use this command for resetting the axis-specific gear factor in the appropriate unit. This is necessary, for example, with indexing mechanisms or runtime-entailed alterations to system variables, like workpiece or tool dimensions or other correction factors.
BORLAND DELPHI:	procedure wrgf(var tsrp:TSRP);
C:	void wrgf(struct TSRP far *tsrp);
VISUAL BASIC:	Sub wrgf(DTSRP As TSRP)
TSRP COMPONENTS:	TSRP[n].gf
NOTE:	Remember that (particularly if there are large alterations in the gear factor) the current axis-specific acceleration and velocity parameters have to be matched to this new factor, since this is utilized for converting these system parameters. The value currently set for *gf* can be read with the PCAP command *rdgf()*.

4.4.146 wrgfaux, write gear factor auxiliary channel

DESCRIPTION:	With this function, the axis-specific ratio of stepper motor resolution to encoder channel in stepper systems with encoder verification can be written. The default value is 1.0; the value can only be changed at runtime.
BORLAND DELPHI:	function wrgfaux (an: integer; var value: double) : integer;
C:	int wrgfaux(int an, double *value)
VISUAL BASIC:	Function wrgfaux (ByVal an As Long, value As Double) As Long
RETURN VALUE:	After successful execution, the function returns 0. In this case, the value in *value* could be successfully written to the axis *an*. With a return value ≠ 0, the value could not be written, because e.g. RWMOS.ELF does not support the command. 0 for success -1 Command is not available in RWMOS version. -4 Time-out, reason unknown, communication with the motion control board is interrupted Other value <> 0 Unknown error at command execution
NOTE:	The factor can be read at any time with the PCAP command *rdgfaux()*. See also Chapter 6.3.3.

4.4.147 wrhac, write home acceleration

DESCRIPTION:	You use this command to set the axis-specific maximum acceleration *hac* for all reference travel commands (*home* commands). If this command is not executed, the system will work with the system parameter specified in the TOOLSET program *mcfg.exe*. The system parameter can be overwritten any time you want.
BORLAND DELPHI:	procedure wrhac(var tsrp:TSRP);
C:	void wrhac(struct TSRP far *tsrp);
VISUAL BASIC:	Sub wrhac(DTSRP As TSRP)
TSRP COMPONENTS:	TSRP[n].hac
NOTE:	The value currently set for *hac* can be read with the PCAP command *rdhac()*.

4.4.148 wrhvl, write home velocity

DESCRIPTION:	You use this command to set the axis-specific maximum velocity with the aid of the *hvl* variable for all reference travel commands (*home* commands). If this command is not executed, the system will work with the system parameter specified in the TOOLSET program *mcfg.exe*. The system parameter can be overwritten any time you want.
BORLAND DELPHI:	procedure wrhvl(var tsrp:TSRP);
C:	void wrhvl(struct TSRP far *tsrp);
VISUAL BASIC:	Sub wrhvl(DTSRP As TSRP)
TSRP COMPONENTS:	TSRP[n].hvl
NOTE:	The value currently set for *hac* can be read with the PCAP command *rdhvl()*.

4.4.149 wripw, write in position window

DESCRIPTION:	This command can be used to alter (during the run time) the In-Position Window {ipw} specified using the TSW program *mcfg.exe*. The window is re-specified to the value set in *ipw*. The value is stated in the axis-specific position unit.
BORLAND DELPHI:	procedure wripw(var tsrp:TSRP);
C:	void wripw(struct TSRP far *tsrp);
VISUAL BASIC:	Sub wripw(DTSRP As TSRP)
TSRP COMPONENTS:	TSRP[n].ipw
NOTE:	The "In-Position-Window“ is monitored only when a value greater than 0.0 has been specified. (MCFG / Chapter 1.7.2.1.11) PCAP command *rdipw()*

4.4.150 wrjac, write jog acceleration

DESCRIPTION:	This command is identical to the PCAP command *wrhac(),* except that here the maximum system acceleration is specified for all *jog* commands using the *jac* variable.
BORLAND DELPHI:	procedure wrjac(var tsrp:TSRP);
C:	void wrjac(struct TSRP far *tsrp);
VISUAL BASIC:	Sub wrjac(DTSRP As TSRP)
TSRP COMPONENTS:	TSRP[n].jac
NOTE:	The value currently set for *jac* can be read with the PCAP command *rdjac()*.

4.4.151 wrJerkRel, write jerkrel

DESCRIPTION:	With this command the axis-specific parameter *jerkrel* can be written in.
BORLAND DELPHI:	procedure wrJerkRel (an: integer; var value: double);
C:	void wrJerkRel (long an, double *value);
VISUAL BASIC:	Sub wrJerkRel (an As Long, ByVal value As Double)
PARAMETER:	an = Number of axes (0..n) value = value to be zu written
RETURN VALUE:	None
NOTE:	Only one value between 0 and 1 can be allocated to *jerkrel*. Lower or higher values are limited. See also chapter 4.4.76 and 6.3.3

4.4.152 wrjovr, write jog override

DESCRIPTION:	This command sets the axis-specific velocity correction value. This correction value is taken into account in all *jog* commands. The *jovr* parameter must have a value greater than 0.0. All values smaller than 1.0 will result in a reduction in axis velocity. If *value* has a value greater than 1.0, this will be manifested in an increased velocity.
BORLAND DELPHI:	procedure wrjovr(var trsp:TSRP);
C:	void wrjovr(struct TSRP far *tsrp);
VISUAL BASIC:	Sub wrjovr(DTSRP As TSRP)
TSRP COMPONENTS:	TSRP[n].jovr
NOTE:	Remember that the specified correction value acts equally on the current axis acceleration. If the correction value is increased or reduced too rapidly, this may be manifested in an acceleration jump (jerk) of the axis. The correction factor should therefore be incremented or decremented in linear mode over time-delay loops, until the final value you want has been reached. For execution of the PCAP commands ra(), *rs()* or SAP commands *RA(), RS*, the override factor is initialized to the default value of 1.0. Whenn calling utrovr and selected axis, the value of jovr is set also, if this was not switched off explicitely in the register variable ModeReg. PCAP command *rdjovr()*

4.4.153 wrjtvl, write jog target velocity

DESCRIPTION:	This command is used to set the axis-specific target velocity (jog) with the aid of the *jtvl* variable for the *jog* commands *ja()* and *jr()*. If this command is not executed, the system will work with the system parameter specified in the TOOLSET program *mcfg.exe*. The system parameter can be overwritten any time you want.
BORLAND DELPHI:	procedure wrjtvl(var tsrp:TSRP);
C:	void wrjtvl(struct TSRP far *tsrp);
VISUAL BASIC:	Sub wrjtvl(DTSRP As TSRP)
TSRP COMPONENTS:	TSRP[n].jtvl
NOTE:	The value currently set for *jtvl* can be read with the PCAP command *rdjtvl()*.

4.4.154 wrjvl, write jog velocity

DESCRIPTION:	This command is identical to the PCAP command *wrhvl(),* except that here the maximum traversing velocity is specified using the *jvl* variable for all *jog* commands.
BORLAND DELPHI:	procedure wrjvl(var tsrp:TSRP);
C:	void wrjvl(struct TSRP far *tsrp);
VISUAL BASIC:	Sub wrjvl(DTSRP As TSRP)
TSRP COMPONENTS:	TSRP[n].jvl
NOTE:	The value currently set for *jvl* can be read with the PCAP command *rdjvl()*.

4.4.155 wrledgn, write led green

DESCRIPTION: **APCI-8001:** **APCI-8008:**	 This command can be used to switch the green SMD LED D29 on and off. It is switched on with the value 1 and switched off with the value 0. This command can be used to switch the green SMD LED D53 on and off. It is switched on with the value 1 and switched off with the value 0.
BORLAND DELPHI:	procedure wrledgn(value:integer);
C:	void wrledgn(int value);
VISUAL BASIC:	Sub wrledgn(ByVal value As Long)
NOTE:	This command is primarily used as a testing and diagnostic tool. The SMD-LED is located on the high back end of the solder side of the board.

4.4.156 wrledrd, write led red

DESCRIPTION: **APCI-8001:** **APCI-8008:**	 as for PCAP command *wrledgn()*, but for the red LED D31 as for PCAP command *wrledgn()*, but for the red LED D56
BORLAND DELPHI:	procedure wrledrd(value:integer);
C:	void wrledrd(int value);
VISUAL BASIC:	Sub wrledrd(ByVal value As Long)

4.4.157 wrledyl, write led yellow

DESCRIPTION: **APCI-8001:** **APCI-8008:**	 as for PCAP command *wrledgn()*, but for the yellow LED D30 as for PCAP command *wrledgn()*, but for the yellow LED D55
BORLAND DELPHI:	procedure wrledyl(value:integer);
C:	void wrledyl(int value);
VISUAL BASIC:	Sub wrledyl(ByVal value As Long)

4.4.158 wrlp, write latched position

DESCRIPTION:	This command is used to set the axis-specific latch position to the value set in *lp*. The value is specified in the axis-specific position unit.
BORLAND DELPHI:	procedure wrlp(var tsrp:TSRP);
C:	void wrlp(struct TSRP far *tsrp);
VISUAL BASIC:	Sub wrlp(DTSRP As TSRP)
TSRP COMPONENTS:	TSRP[n].lp
NOTE:	PCAP command *rdlp()*

4.4.159 wrlpndx, write latched position index

DESCRIPTION:	This command is used to set the axis-specific latch position of the zero track (index) to the value set in *lp*. The value is specified in the axis-specific position unit.
BORLAND DELPHI:	procedure wrlpndx (var tsrp:TSRP);
C:	void wrlpndx (struct TSRP far *tsrp);
VISUAL BASIC:	Sub wrlpndx (DTSRP As TSRP)
TSRP COMPONENTS:	TSRP[n].lp
NOTE:	PCAP command *rdlpndx()*

4.4.160 wrMaxAcc – Write Maximum Acceleration Check

DESCRIPTION:	With this command you can write the maximum axis-specific acceleration (*MAXACC*). This value is used by the RWMOS operating system software to limit the trajectory acceleration so that no axis involved in a linear interpolation excedd the maximum acceleration accepted.
BORLAND DELPHI:	wrMaxAcc (an: integer; var value: double);
C:	void wrMaxAcc (long an, double *value);
VISUAL BASIC:	Sub wrMaxAcc (an As Long, ByVal value As Double)
PARAMETER:	The axis number is indicated in *an*, the maximum acceleration accepted is transmitted in *value* in the interpolation-specific acceleration unit (PU and TU).
NOTE:	To check this function, bit 7 in MODEREG register must be set (see chapter 6.3.1.5). When set to 0 the check function is disabled for the axis involved. The function is only available with spooled commands.

4.4.161 wrMaxVel – Write Maximum Velocity Check

DESCRIPTION:	With this command you can write the maximum axis-specific velocity (*MAXVEL*). This value is used by the RWMOS operating system software to limit the trajectory velocity so that no axies involved in a linear interpolation exceeds the maximum velocity accepted.
BORLAND DELPHI:	wrMaxVel (an: integer; var value: double);
C:	void wrMaxVel (long an, double *value);
VISUAL BASIC:	Sub wrMaxVel (an As Long, ByVal value As Double)
PARAMETER:	The axis number is indicated in *an*, the maximum velocity accepted is transmitted in *value* in the axis-specific velocity unit. This value is always interpreted in the interpolation unit.
NOTE:	To check this function, bit 7 in MODEREG register must be set (see chapter 6.3.1.5). When set to 0 the check function is disabled for the axis involved. The function is only available with spooled commands.

4.4.162 wrmcp, write motor command port

DESCRIPTION:	This command is used to write the Motor-Command-Port to the value set in the *mcp* field. You will find this particularly helpful during commissioning work, if you want to check the drive system's setpoint value channel, for example. In idle mode (no position control), the motor axis can be moved with this command in uncontrolled form. This means, for example, that you can check the drive's sense of rotation, or check the pulse acquisition feature and limit switches for correct functioning and so on, before commissioning work is continued in the position control mode.
APCI-8001: **APCI-8008:**	In the case of servo axes, *mcp* can be set to a value between -32,767 and +32,767. This value range corresponds to the analog output voltage range of -10 V to +10 V. It may be necessary to allow for a planned inversion of the analog output signal. In the case of stepping motor axes, *mcp* can be used to specify a time-delay, with the aid of which a stepping signal for stepping motor power output stages is generated. The frequency of this stepping signal can be computed as follows: f_{Pulse} = CLOCK/2/(mcp+1) *Example: with mcp = 999 and CLOCK = 70MHz* f_{Pulse}*=35,000[Hz*] The CLOCK value is 70 MHz for the APCI-8001 and 66.66666 MHz for the APCI-8008. The value range of *mcp* lies between -1,048,574 and +1,048,574. The sign selects the desired traversing direction and influences the axis-specific directional signal. For the stepping signal f_{Pulse}, only the absolute value of *mcp* is determinant. Remember that the value 0 in *mcp* causes a stepping signal of 0 Hz, i.e. the motor halts.
BORLAND DELPHI:	procedure wrmcp(var tsrp:TSRP);
C:	void wrmcp(struct TSRP far *tsrp);
VISUAL BASIC:	Sub wrmcp(DTSRP As TSRP)
TSRP COMPONENTS:	TSRP[n].mcp
NOTE:	If the axis system is in position control, this command will be effective at most for the duration of a scan interval, since the Motor-Command-Ports are set to new values after the PIDF filter has been processed. PCAP command *rdmcp()*

4.4.163 wrMDVel – Write Maximum Velocity Skip

DESCRIPTION:	With this command you can write the maximum axis-specific velocity jump (*MDVEL*). This value is used by the look-ahead functionality RWMOS operating system software verwendet to limit the trajectory velocity so that no axis involved in an interpolation exceeds the maximum velocity accepted.
BORLAND DELPHI:	wrMDVel (an: integer; var value: double);
C:	void wrMDVel (long an, double *value);
VISUAL BASIC:	Sub wrMDVel (an As Long, ByVal value As Double)
PARAMETER:	The axis number is indicated in *an*, the maximum velocity accepted is transmitte in *value* in the activated position and time units (PU and TU) übergeben.
NOTE:	The look-ahead mode is activated by setting the bit 0 in MODEREG register (see chapter 6.3.1.5). The check function of the axis is disabled with the value 0. In this case please consider also the bit 6 of MODEREG. **Important:** In look-ahead mode the different profiles must be programmed with a target velocity > 0 (generally = maximum velocity) so that the look ahead is really effective.

4.4.164 wrModeReg – Write MODEREG

DESCRIPTION:	With this command the register MODEREG of the RWMOS operating system software can be described.
BORLAND DELPHI:	wrModeReg (var value: integer);
C:	void wrModeReg(long *value);
VISUAL BASIC:	wrModeReg (ByVal value As Long)
PARAMETER:	bitcodierter value für ModeReg
NOTE:	With flags (bits) in the ModeReg register different options can be activated and monitored in RWMOS.ELF such as e.g. look-ahead, S profile etc. (see chapter 6.3.1.5).

4.4.165 wrmpe, write maximum position error

DESCRIPTION:	This command can be used to alter, during the run time, the position error limit {mpe} specified with the aid of the TSW program *mcfg.exe*. The axis-specific maximum permitted position error is reset to the value set in *mpe*. The value is specified in the axis-specific position unit.
BORLAND DELPHI:	procedure wrmpe(var tsrp:TSRP);
C:	void wrmpe(struct TSRP far *tsrp);
VISUAL BASIC:	Sub wrmpe(DTSRP As TSRP)
TSRP COMPONENTS:	TSRP[n].mpe
NOTE:	Position error monitoring is performed only if a value greater than 0.0 has been specified and the control loop is closed. (MCFG / Chapter 1.7.2.1.9) PCAP command *rdmpe()*

4.4.166 wrnfrax, write No-Feed-Rate-Axis

DESCRIPTION:	With this command on the NFRAX register of the RWMOS operating system software is written.
BORLAND DELPHI:	wrnfax (var value: integer);
C:	void wrnfax (long *value);
VISUAL BASIC:	Sub wrnfax (VyVal value As Long)
PARAMETER:	Bit coded value for NFRAX
NOTE:	In the register NFRAX so-called No-Feed-Rate axex can be defined bit coded. These axes are not used for the calculation of the velocity at interpolation commands; in spite of taking part in the interpolation. In this way the influence of other axes for the velocity in interpolation profiles can be prevented. See also rdnfrax function in chapter 4.4.92.

4.4.167 wrrp, write real position

DESCRIPTION:	This command sets the axis-specific current position register to the value set in *rp and* is operative only in open-loop mode (no position control). The value is specified in the axis-specific position unit.
BORLAND DELPHI:	procedure wrmpe(var tsrp:TSRP);
C:	void wrmpe(struct TSRP far *tsrp);
VISUAL BASIC:	Sub wrmpe(DTSRP As TSRP)
TSRP COMPONENTS:	TSRP[n].mpe
NOTE:	This command will cause the machine zero to be shifted automatically!

4.4.168 wrsdec, write stop deceleration

DESCRIPTION:	This command is used to set the axis-specific stop deceleration *sdec* for the following: the PCAP command *js()* [chapter 4.4.24], SAP command *JS()* [chapter 6.6.26], the software end positions (MCFG / Chapters 1.7.2.1.10 and 1.7.2.2.3) planned with SMD and the digital inputs planned with LSL_SMD or LSR_SMD projected digital inputs (MCFG / Chapter 1.7.2.5). If *wrsdec()* is not executed, the system will work with the system parameter specified in the TOOLSET program *mcfg.exe*. The system parameter can be overwritten any time you want.
BORLAND DELPHI:	procedure wrsdec(var tsrp:TSRP);
C:	void wrsdec(struct TSRP far *tsrp);
VISUAL BASIC:	Sub wrsdec(DTSRP As TSRP)
TSRP COMPONENTS:	TSRP[n].sdec
NOTE:	The value currently set for *sdec* can be read with the PCAP command *rdsdec()* (see chapter 4.4.98).

4.4.169 wrsll, write software limit left

DESCRIPTION:	This command can be used to alter, during the run time, the axis-specific left software limit position {sll} defined with the aid of the TSW program *mcfg.exe.* The left software limit is reset to the value set in *sll*. The value is specified in the axis-specific position unit.
BORLAND DELPHI:	procedure wrsll(var tsrp:TSRP);
C:	void wrsll(struct TSRP far *tsrp);
VISUAL BASIC:	Sub wrsll(DTSRP As TSRP)
TSRP COMPONENTS:	TSRP[n].sll
NOTE:	The software limit set is taken into account only if the home position of the axis channel involved has already been defined or is set after execution of this command. (MCFG / Chapter 1.7.2.1.10) PCAP commands *rdsll()*, *shp()*, SAP command *SHP()*

4.4.170 wrslr, write software limit right

DESCRIPTION:	This command is identical to the PCAP command *wrsll()*, but the right software limit is redefined with the value set in the *slr* parameter.
BORLAND DELPHI:	procedure wrslr(var tsrp:TSRP);
C:	void wrslr(struct TSRP far *tsrp);
VISUAL BASIC:	Sub wrslr(DTSRP As TSRP)
TSRP COMPONENTS:	TSRP[n].slr

4.4.171 wrslsp, write Slits / Stepperpulses

DESCRIPTION:	This command sets the axis-specific resolution per motor turn {slsp}. The default value is determined with the TOOLSET program *mcfg.exe*.
BORLAND DELPHI:	procedure wrslsp (an: integer; var value: double);
C:	void wrslsp (long an, double *value);
VISUAL BASIC:	Sub wrslsp (ByVal an As Long, value As Double)
TSRP COMPONENTS:	None
NOTE:	slsp can be read with the PCAP commaand *rdslsp().* See also axis qualifier slsp. For slsp only numeric values > 0.0 are allowed.

4.4.172 wrtp – write target position

DESCRIPTION:	With this command, you can write the axis-specific target position (*tp*). This command is usually only necessary and used in special cases.
BORLAND DELPHI:	procedure wrtp(var tsrp:TSRP);
C:	void wrtp(struct TSRP far *tsrp) ;
VISUAL BASIC:	Sub wrtp(DTSRP As TSRP)
TSRP COMPONENTS:	TSRP[n].tp
NOTE:	By writing the target position (*tp*) when motion commands are executed an uncontrolled process course can occur in certain cases. This is why it must be avoided. See also PCAP command *rdtp().*

4.4.173 wrtrac, write trajectory acceleration

DESCRIPTION:	Write the RWMOS system variable TRAC
BORLAND DELPHI:	function wrtrac (var value:double) : integer;
C:	int wrtrac (double *value);
VISUAL BASIC:	Function wrtrac (value As Double) as long
RETURN VALUE:	0 for success -1 Command is not available in RWMOS version. -4 Time-out, reason unknown, communication with the motion control board is interrupted Other value <> 0 Unknown error at command execution
NOTE:	TRAC is the interpolation trajectory acceleration which is used with interpolation commands that are called from the rw_SymPas programming environment (see also *rw_SymPas* system parameter in Table 32). With PCAP programming, this parameter is transferred in the data structure LMP, CMP or HMP when it is called. This acceleration is used only with the PCAP call of Motion-Stop (ms) if bit 15 (MS_DECEL) is set in the ModeReg register (Table 36).

4.4.174 wrtrovr, write trajectory override

DESCRIPTION:	This command sets the trajectory velocity correction value for all interpolation commands (*move* commands). The *value* parameter must have a value greater than 0.0. All values smaller than 1.0 result in a reduction in the trajectory velocity. If *value* has a value greater than 1.0, this will be manifested in an increase in trajectory velocity. The correction value specified in *value* is placed in intermediate storage on the APCI-800x board in a system variable and does not become operative until after execution of the PCAP command *utrovr()*, or the SAP command *UTROVR()*. The axis channels selected there will be decelerated or accelerated even during trajectory travel, depending on the *value* correction factor.
BORLAND DELPHI:	procedure wrtrovr(var value:double);
C:	void wrtrovr(double *value);
VISUAL BASIC:	Sub wrtrovr(value As Double)
NOTE:	Remember that the specified correction value acts equally on the current trajectory acceleration. If the correction value is increased or decreased too quickly, this may be manifested in an abrupt acceleration (jerk) of the axes. The correction factor should therefore be incremented or decremented over time-delay loops in linear mode until the final value you want has been reached. When the PCAP command *rs()* or the SAP command *RS* is executed, the override factor is initialized to the default value of 1.0. PCAP commands *wrtrovr()*, *wrjovr()*, *rdtrovr()* and *rdjovr()*

4.4.175 wrtrovrst, write trajectory override settling time

DESCRIPTION:	With this command a „soft“ adaptation of the override value TROVR can be done after the calling of utrovr() realisieren. In the parameter „value“ before the calling of utrover() a time in seconds is indicated that is the adaptation time between the values 0 and 1. In this way velocity jumps, which are caused by programming, can be avoided. Indicated times that are smaller than the sampling interval are not taken into consideration. The value 0 disables the function.
BORLAND DELPHI:	function wrtrovr(var value:double) : integer;
C:	int wrtrovr(double *value);
VISUAL BASIC:	Function wrtrovr(value As Double) as long
RETURN VALUE:	0 for success -1 Command is not available in RWMOS version. -4 Time-out, reason unknown, communication with the motion control board is interrupted Other value <> 0 Unknown error at command execution
NOTE:	Please also see the commands rdtrovrst, wrtrovr, rdtrovr, utrovr and rw_SymPas system parameter TROVRST Setting the Jog-Override using wrjovr is not affected by this function. See also Bit 25 in the MODEREG register (Chapter 6.3.1.4). Reading the TROVR parameter always returns the programmed target value, even during the adaptation phase. If the current effective override value is to be read during the adaptation phase, the current Jog-Override of one of the axes involved can be used.

4.4.176 wrtrvl, write trajectory velocity

DESCRIPTION	Write the RWMOS system variable TRVL
BORLAND DELPHI:	function wrtrvl (var value:double) : integer;
C:	int wrtrvl (double *value);
VISUAL BASIC:	Function wrtrvl (value As Double) as long
RETURN VALUE:	0 for success -1 Command is not available in RWMOS version. -4 Time-out, reason unknown, communication with the motion control board is interrupted Other value <> 0 Unknown error at command execution
NOTE:	TRVL is the interpolation trajectory acceleration which is used with interpolation commands that are called from the rw_SymPas programming environment (see also *rw_SymPas* system parameter in Table 32). With PCAP programming, this parameter is transferred in the data structure LMP, CMP or HMP when it is called.

4.4.177 wrtrtvl, write trajectory target velocity

DESCRIPTION:	Write the RWMOS system variable TRTVL
BORLAND DELPHI:	function wrtrtvl (var value:double) : integer;
C:	int wrtrtvl (double *value);
VISUAL BASIC:	Function wrtrtvl (value As Double) as long
RETURN VALUE:	0 for success -1 Command is not available in RWMOS version. -4 Time-out, reason unknown, communication with the motion control board is interrupted Other value <> 0 Unknown error at command execution
NOTE:	TRTVL is the interpolation trajectory acceleration which is used with interpolation commands that are called from the rw_SymPas programming environment (see also *rw_SymPas* system parameter in Table 32). With PCAP programming, this parameter is transferred in the data structure LMP, CMP or HMP when it is called.