1 Introduction

In the “SCANNER” option, the APCI-8001 and APCI-8008 control products make it possible to record process data in real time. The maximum recording rate is the scan time of the control unit (default value: 1.28 ms).
Process data in the control unit includes, for example, target positions, actual positions, input or output states, axis states and much more. All readable resources which are listed in Section 2.2 of the “Resource Interface” manual can be recorded. Recording takes place in a memory area on the control unit. Since this memory is, of course, limited, in the case of more extensive scans the data area must be read continually in order to prevent a data overrun. However, this means that it is possible to achieve a buffer of several seconds, even with extensive scan records. As a result, data recording is largely decoupled from system utilisation under Windows.
The SCANNER option is not available in all variants of RWMOS.ELF. Availability can be verified in the fwsetup.exe monitor screen.

2 Configuration and access to the scanner module

The scanner model must be configured before it can be used. Access is provided using the “Universal Object Interface”. This access method is described in the “Universal Object Interface” manual.
The properties of the scanner module and the procedure for configuration are described in the “Scanner Interface” manual.

3 Using the scanner module

Using the C programming language, the important components in an application program for using the scanner function are described below. The standard methods for initialising and booting the control module are assumed to be known and are not specifically mentioned here.

3.1 Variable declarations

First, the object descriptor elements must be declared with which the scanner or the resources to be scanned are accessed. This concerns a predefined structure data type.

Example:

```
// Object descriptors for resource interface
struct OptionDescriptorObject
        // Declare resource objects
        odResClean, odRpA1, odDpA1, odScntr, odDigi, odRpA2, odRpA3, odDpA2, odDpA3;
// Object descriptors for scanner interface
struct OptionDescriptorObject
        // Declare scanner objects
        odScannerClean, odScannerHW_Scan_Strobe,
        odScanRpA1, odScanDpA1, odScanRpA2, odScanDpA2, odScanRpA3, odScanDpA3,
        odScanScntr, odScanInit, odScannerRTS, odScannerStart,
        odScannerScanned, odScannerSizeOfRec, odScannerStatus, odScanDigi;
```

3.2 Initialisation of object descriptor elements

The declared elements must be initialised once at the program start using the tables in the relevant manuals. The following structure elements must be assigned for this:

AccessType
DataType
BusNumber
DeviceNumber
Index
SubIndex

In the example below, this assignment takes place in the function SetupOD().

```
/*  Function to initialize the object descriptors
        Parameters: Pointer to od
                        AccessType, DataType, Bus, Device, Index, SubIndex
*/
void SetupOD (struct OptionDescriptorObject *od, ATAccessType at, ATDataType dt,
                int Bus, int Device, int Index, int SubIndex)
{
        od->Handle              = 0;
        od->AccessType          = at;
        od->DataType            = dt;
        od->BusNumber           = Bus;
        od->DeviceNumber        = Device;
        od->Index               = Index;
        od->SubIndex            = SubIndex;
}

// Initialise resources
SetupOD (&odResClean,       ATAccessOutput, ATDataDoubleWord,  1000,   0, 1, 0);
SetupOD (&odRpA1,           ATAccessInput,      ATDataReal,            1000,   2, A1, 0);
SetupOD (&odDpA1,           ATAccessInput,      ATDataReal,            1000,   1, A1, 0);
SetupOD (&odRpA2,           ATAccessInput,      ATDataReal,            1000,   2, A2, 0);
SetupOD (&odDpA2,           ATAccessInput,      ATDataReal,            1000,   1, A2, 0);
SetupOD (&odRpA3,           ATAccessInput,      ATDataReal,            1000,   2, A3, 0);
SetupOD (&odDpA3,           ATAccessInput,      ATDataReal,            1000,   1, A3, 0);
SetupOD (&odScntr,          ATAccessInput,      ATDataDoubleWord,      1000,   5, 0, 0);
SetupOD (&odDigi,           ATAccessInput,      ATDataDoubleWord,      1000,   4, 0, 0);

SetupOD (&odScannerClean, ATAccessOutput, ATDataDoubleWord, 1100, 0, 1, 0);
SetupOD (&odScannerRTS, ATAccessOutput, ATDataDoubleWord, 1100, 0, 7, 0);
SetupOD (&odScanDpA3, ATAccessInputOutput, ATDataNone, 1100, 4, 1, odDpA3.Handle);
SetupOD (&odScanRpA3, ATAccessInputOutput, ATDataNone, 1100, 5, 1, odRpA3.Handle);
.....
etc.
```

The “Handle” element is initially assigned the value 0. On first use, this value is used by the system to make an assignment to the relevant object in the RWMOS operating system software when there are repeated calls.
If the control unit is booted while the program is running, all handle elements must therefore be reset.

3.3 Initialising the scanner module

Using the initialised object descriptor elements, the variables of the scanner module can now be written on and read. The functions wrOptionInt(), rdOptionInt() for integer variables and rdOptionDbl and wrOptionDbl for double floating point numbers are provided in mcug3.dll for reading and writing.

Note: Here, particular care should be taken to ensure that the function corresponding to the data type to be read/written is used. Furthermore, the return value should always be evaluated in order to ensure the success of the desired operations and to detect any errors at the correct point.
Otherwise, troubleshooting on a program which is not working is extremely difficult.

Example:

```
// Cleaning the ressources
int ival = 1;
if (wrOptionInt (&odResClean, &ival) != 4) return -1;
```

3.4 Initialising the resources to be scanned

Before use, at least one successful read access must be performed on the resource elements to be scanned so that the handle data element of the relevant resource is initialised by the system.

Example:

```
// Perform read operations to get a valid handle
if (rdOptionDbl (&odRpA1,  &dval) != 4) return -1;
if (rdOptionDbl (&odDpA1,  &dval) != 4) return -1;
if (rdOptionDbl (&odRpA2,  &dval) != 4) return -1;
if (rdOptionDbl (&odDpA2,  &dval) != 4) return -1;
if (rdOptionDbl (&odRpA3,  &dval) != 4) return -1;
```

3.5 Definition of the scan list

With the resources prepared in this way, the object descriptors for the scan list can now be initialised and the scan list, i.e. the list of objects to be scanned, be created.

Example:

```
ival = 0;
SetupOD (&odScanDpA1, ATAccessInputOutput, ATDataNone, 1100, 4, 1, odDpA1.Handle);
ErrorStatus |= wrOptionInt (&odScanDpA1, &ival);

ival = 0;
SetupOD (&odScanRpA1, ATAccessInputOutput, ATDataNone, 1100, 5, 1, odRpA1.Handle);
ErrorStatus = wrOptionInt (&odScanRpA1, &ival);
```

Special notes: The data type for elements in the scan list is “ATAccessInputOutput”.
The object descriptors for the scan list cannot be initialised until valid handles exist for the objects to be scanned.
The relevant data objects appear in the scan data in reverse order to how they are programmed, i.e. the element which is entered last in the scan list appears first in the scan data.

3.6 Starting the scan

After the scan list is programmed, status information can be read for the system check.
At least, the scanner status (see function 4 “STATUS” or command rdScannerStatus) and the record size of the scan record should be monitored (see function 9 “SIZEOFRECORD”). Before reading this information, function 2 “INIT” must be written on by which the corresponding variables in RWMOS.ELF are prepared.
After error-free initialisation, the scan can be started by writing the value 1 to the variable 3 “STARTSTOP”.

3.7 Reading the scanned data

If an infinite scan is carried out, or if it is not guaranteed that the scan memory on the APCI-8001 or APCI-8008 can record all the data to be scanned, the scan memory must then be read and processed cyclically or stored in the RAM or on the hard drive. The scan memory is read with the DLL function rdScannerBuffer. Here too, the return values of the function must be evaluated in order to detect any errors immediately.
By default, the APCI-8001 or APCI-8008 has a scan memory area of 100,000 bytes. This value can be set using the “SZSCANBUFFER” environment variable. The maximum possible value here is approx. 12 Mbytes. However, the maximum value depends on other options and on the RWMOS.ELF variant used.

The scan data is written to the APCI-8001 or APCI-8008 in a circular buffer. By reading the data, the relevant data area in this circular buffer is made available over and over again.
Any data overrun can be detected with the overrun bit in scanner status.
It is best to read the data to a memory area which is declared as an array of scan records.
As it has shown in practice that reading out the scanner data sometimes takes much time, the new function SendReqScannerBuffer() is provided from mcug3.dll V2.5.3.107 and rwmos.elf V2.5.3.126. It is a sending request to rwmos.elf. With this kind of data transfer, the transfer time can be significantly reduced. However, this method requires the successful request of physical RAM. From experience, this memory is not provided reliably by the Windows operating system, which means that after several program calls or with high usage of the Windows operating system, the function cannot be used.
To evade this constraint, the Windows service rwPhysMemService can be installed by means of the program rwMemMgnt.exe. This service is supplied on the Toolset CD and manages the required memory during the whole runtime of the Windows operating system.

3.8 Stopping the scan

A scan with a fixed number of scan records (variable 7 “RecordsToScan”) stops automatically as soon as the programmed number of records has been recorded. Otherwise, a scan can be stopped by writing 0 to the variable 3 “STARTSTOP”.
After stopping the scan, it should be ensured that all the remaining scan records from the control unit are read.