Model MPA-3

32 bit DLL Software Interface

User Manual

Version 2.1, 12. August 2011

Model MPA-3
32 bit DLL Software Interface Manual
Table of Contents

1. Introduction

The 32 bit MPANT software for the multiparameter system MPA-3 consists of a hardware-dependent server program and a general graphics program that controls the hardware via a DLL. Any other Windows application can also control the hardware via the DLL. To support the programming of such customer-specific user interfaces, as an option we deliver this documentation including sourcecode and example programs for LabVIEW, Visual Basic, C and Delphi. The complete sourcecode of the DLL that controls the hardware via the server program is included in the appendix. A special new FMPA3.DLL including source code with examples like F*(x-x‘)/(x+x‘) where x‘ is marked by a routing bit allows to display calculated spectra including calculated error bars in the MPANT program.

The server program MPA3.EXE is a rather compact Windows application. It controls the hardware and data and allows to perform measurements with the system. MPANT is just a user interface to control the server program. It has access to the data in a shared memory region and can display spectra. It is not necessary that MPANT is running during an acquisition. It can be exited and restarted without stopping a running acquisition. If you don't want MPANT automatically started when starting the server, just rename the file MPANT.EXE in the working directory (default C:\MPA3) for example into MYMPANT.EXE.

The DLL DMPA3.DLL is an interface providing functions to communicate with the server program. Most of these functions send messages to the server as you do it when operating the server program by sending Windows messages via mouse clicks. Please do not expect any functions in this DLL for controlling the hardware directly. All software described in this manual requires that the server program is running. The DLL was mainly developed as an interface between the server program and MPANT, not as a nice developing tool for customers. But by looking at the programming examples and the following hints it should be easy to develop own programs that are able to control the server like MPANT does.

1.1 Some hints

The server program has a built-in command interpreter. The syntax of these commands is described in the MPA-3 manual, chapter 5.2, and in the MPANT on-line help (look for: "How to use the command language.."). It is recommended to send commands like "range=16384" to the server via the RunCmd DLL function, i.e. RunCmd(0, "range=16384"); if you want to set parameters like a spectra length. The alternative method is to store all settings parameters into the DLL by calling the DLL function StoreSettingData(setting, 0); and then calling NewSetting(0); to send a message to the server to read new settings from the DLL. This method will not work for changing a spectra length to avoid the problem of any undefined memory pointers. The range parameter should be changed by the server program only (or by sending a "range=.." command). The recommended usage of the DLL is reading parameters like Status, Settings, Strings, Cnt numbers, ROI boundaries using the corresponding DLL functions, but for any actions or setting any parameters the command interpreter should be used.

If your application that controls the MPA-3 server via the DLL is a true Windows application with a main window and corresponding message loop handling messages sent to this window, you can fetch a special Windows message to react immediately on actions like an acquisition status change without permanently polling the status:

Declare a global int MM_STATUS and somewhere when initializing your program register a Windows message using a code line like:

```
  MM_STATUS=RegisterWindowMessage("MPANTStatus");
```

furthermore, declare somewhere in your headers constants

```
  #define ID_NEWSTATUS   162
  #define ID_NEWSETTING  139
  #define ID_NEWDATA       160
```

I assume your main Window Procedure is declared like

```
  DWORD WINAPI MyMainWndProc(HWND hwnd, UINT msg, WPARAM wParam,
  LPARAM lParam)
```

you can then insert here code like

```
  if (msg == MM_STATUS) {
    if (wParam == ID_NEWSTATUS) {
      // status change, read acquisition status and react accordingly...
    }
    else if (wParam == ID_NEWDATA) {
      //  release all pointers, the server will reallocate some spectra..
    }
    else if (wParam == ID_NEWSETTING) {
      // the server has reallocated some spectra, get new pointers..
    }
  }
```

On any status change the server sends a NEW_STATUS message. The lParam value is usually zero, but after a stop of an acquisition the lParam is equal to 1, so your program is able to react accordingly.

It is important that the DLL is loaded first by the Server program and that it is loaded from the same path by all programs using it. Otherwise it does not work to access the shared memory. The dmpa3.dll is installed into the Windows\System32 directory. Please make sure that there is nowhere else any file dmpa3.dll. Start MPA3.exe by hand before starting your program, or by a call from your program for example like

```
{
  STARTUPINFO startupinfo = {0};
  PROCESS_INFORMATION procinfo = {0};
  startupinfo.cb = sizeof(STARTUPINFO);
  return CreateProcess ( "MPA3.EXE", NULL, NULL, NULL, FALSE, HIGH_PRIORITY_CLASS, NULL,
NULL, &startupinfo,&procinfo);
}
```

, but before your program loads the DLL. It is recommended not to link the DLL to your program using a dmpa3.LIB file, but explicitely load it at runtime as demonstrated in our example tstmpa3.c.

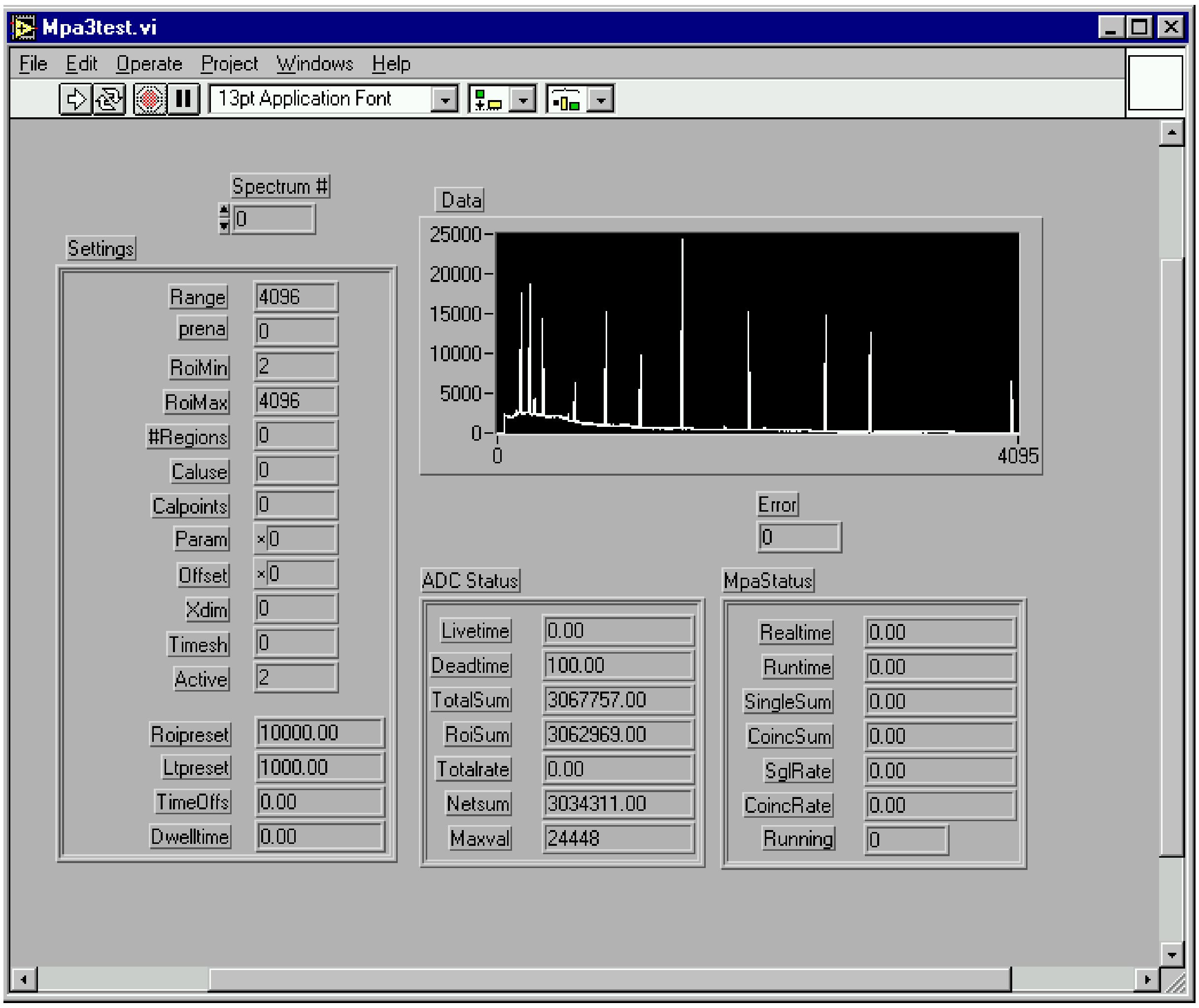
2. Using the DMPA3 DLL from LabVIEW

To access the MPA3 data directly from LabView via the DLL, some LabView VI's („Virtual Instruments") contained in MPA3.LLB and the MPA3TEST.VI are provided.

2.1 Installation

Files: MPA3.LLB, MPA3TEST.VI

The distribution floppy contains in a directory \LV the following files: MPA3.LLB and MPA3TEST.VI. Copy these files into your working directory of the MPANT software. Please start now MPA3.EXE and then LabVIEW and open the MPA3TEST.VI (or just double click on MPA3TEST.VI). You may load some data with MPANT or the server and then run the VI.



Spectrum # 0

Error 0

Settings	
Range	4096
prena	0
RoiMin	2
RoiMax	4096
#Regions	0
Caluse	0
Calpoints	0
Param	x0
Offset	x0
Xdim	0
Timesh	0
Active	2
Roipreset	10000.00
Ltpreset	1000.00
TimeOffs	0.00
Dwelltime	0.00

ADC Status	
Livetime	0.00
Deadtime	100.00
TotalSum	3067757.00
RoiSum	3062969.00
Totalrate	0.00
Netsum	3034311.00
Maxval	24448

MpaStatus	
Realtime	0.00
Runtime	0.00
SingleSum	0.00
CoincSum	0.00
SglRate	0.00
CoincRate	0.00
Running	0

Figure 2.1 The MPA3TEST VI.

In the Windows menu, click on Show Diagram to display the diagram, and on Show Help Window to display the Help window.

The Demo VI contains the VI's to get the settings, status and spectrum data.

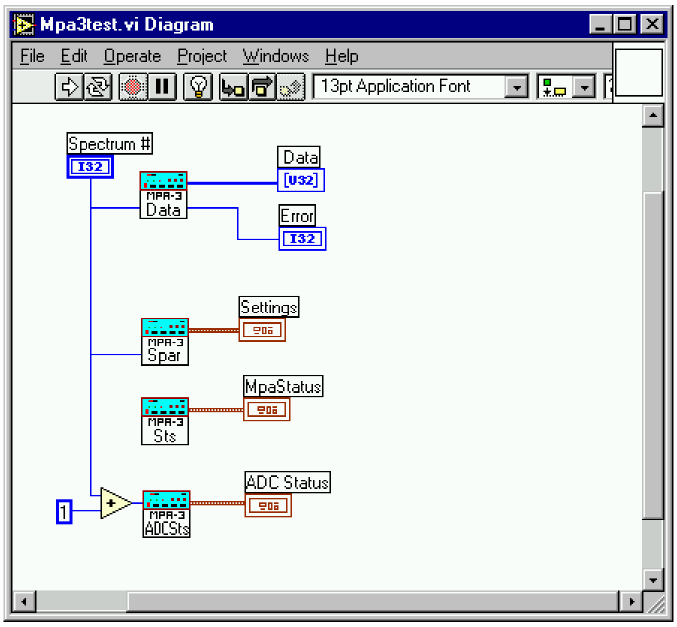
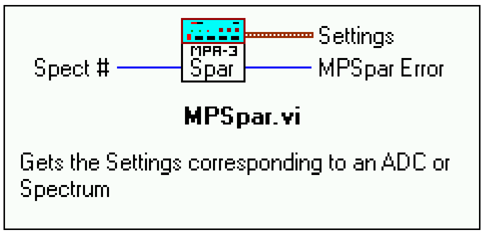


Fig. 2.2: Diagram of MPA3test.VI

2.2 Getting Parameters



The Settings are obtained with the MPSpar.vi. It results a 32 bit integer as an error code and the settings contained in a cluster.

You can use the help window to get information: on the front panel as the active window, just move the mouse over the item you are interested and observe the help window. The cluster has the components known from the DLL structure definitions:

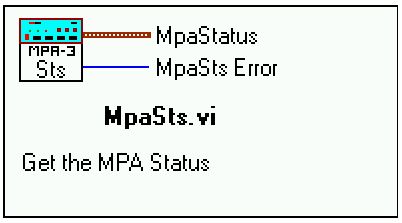
1. Range (I32)	2. Prena (I32)
3. RoiMin (I32)	4. RoiMax (I32)
5. #Regions (I32)	6. Caluse (I32)
7. Calpoints (I32)	8. Param (hex) (I32)
9. Offset (hex) (I32)	10. Xdim (I32)
11. Timesh (I32)	12. Active(hex) (I32)
13. Roipreset (DBL)	14. Ltpreset (DBL)
15. TimeOffs (DBL)	16. Dwelltime (DBL)

For the detailed meaning of each parameter please refer to the LabView on-line help window or the DLL description in this manual.

The error code has the following meanings:

1: DMPA3.DLL not found
2: GetSettingData function in DLL not found
4: No Parameters available

2.3 Getting the Status



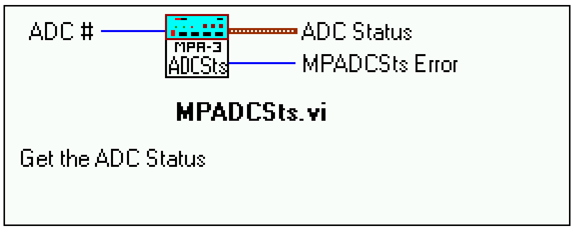
The MPA Status is obtained with the MpaSts.VI. It results a 32 bit integer as an error code and the status parameters contained in a cluster.

The cluster has the following components:

1: Realtime (DBL)
2. Runtime (DBL)
3: SingleSum (DBL)
4: CoincSum (DBL)
5: SlgRate (DBL)
6: CoincRate (DBL)
7: Running (I32)

The error code has the following meanings:

1: DMPA3.DLL not found
2: GetStatusData function in DLL not found
4: No Parameters available



The ADC Status is obtained with the MPADCSts.VI. The Input is the ADC number with 1 for ADC 1A and so on. It results a 32 bit integer as an error code and the status parameters contained in a cluster.

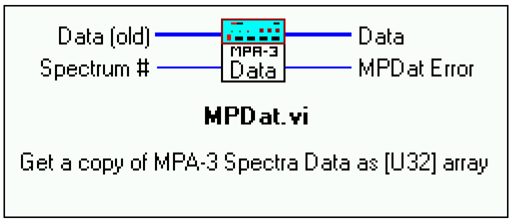
The cluster has the following components:

1: Livetime (DBL)
2: Deadtime (DBL)
3: TotalSum (DBL)
4: RoiSum (DBL)
5: Totalrate (DBL)
7: Netsum (DBL)
8: Maxval (U32)

The error code has the following meanings:

1: DMPA3.DLL not found
2: GetStatusData function in DLL not found
4: No Parameters available

2.4 Getting the Spectrum Data

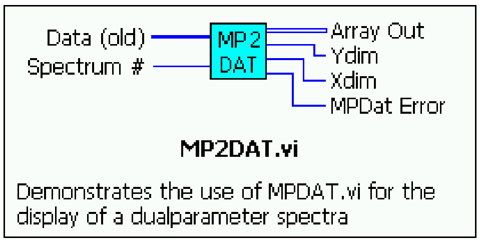


The Spectrum data is obtained with the MPDat.vi. It results a 32 bit integer as an error code and the spectrum as a [U32] array.

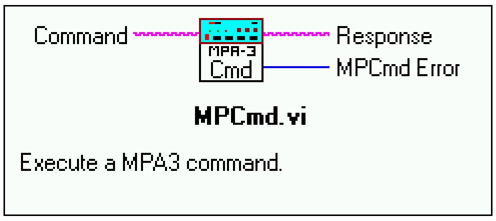
The error code has the following meanings:

1: DMPA3.DLL not found
2: GetSettingData or LVGetDat function in DLL not found
4: No Data available

How to use MPDat.vi to get a display of a dualparameter spectra in LabVIEW is demonstrated in MP2Dat.vi


Data (old)
Spectrum #
MP2
DAT
Array Out
Ydim
Xdim
MPDat Error
MP2DAT.vi
Demonstrates the use of MPDAT.vi for the display of a dualparameter spectra

2.5 Executing a command

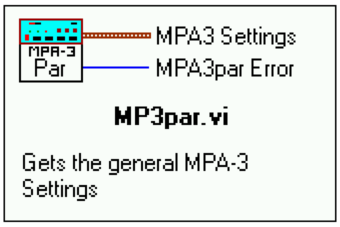


The MPA3.LLB contains some more VIs that are not used by MPA3TEST.VI. Any command for the MPA3 server can be executed by MPCmd.VI. It results an response string and a 32 bit integer as an error code.

The error code has the following meanings:

1: DMPA3.DLL not found
2: RunCmd function in DLL not found
4: Memory Allocation failure

2.6 Getting the MPA-3 general parameters



The general MPA-3 settings can be obtained by MP3par.vi. It results the MPA3 Settings in a cluster and a 32 bit integer as an error code.

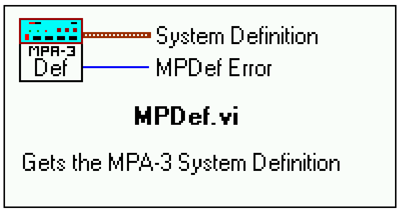
The MPA3 Settings have following components:

1. sen (I32)	2. coi (I32)
3. ctm (I32)	4. dtm (I32)
5. tct (I32)	6. tp0 (I32)
7. tp1 (I32)	8. tp2 (I32)
9. aui (I32)	10. auo (I32)
11. dor (I32	12. bk0 (I32)
13. bk1 (I32)	14. rtcuse (I32)
15. dac (I32)	16. diguse (I32)
17. digval (I32)	18. rtprena (I32)
19. rtpreset (DBL)	

For the detailed meaning of each parameter please refer to the LabView on-line help window or the DLL description in this manual. The error code has the following meanings:

1: DMPA3.DLL not found
2: GetMP3SettingData function in DLL not found
4: No parameters available

2.7 Getting the MPA-3 System definition



MPDef.vi

Gets the MPA-3 System Definition

The MPA-3 System Definition are obtained by MPDef.vi. It results the System Definition parameters in a cluster and a 32 bit integer as an error code. The System Definition has following components:

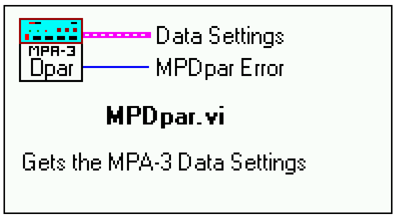
1. nDevices (I32)	nDevices is the number of ADCs
2. nDisplays (I32)	nDisplays is the number of spectra
3. nSystems (I32)	nSystems the number of independent systems which is 1 in the present version
4. bRemote (I32)	bRemote indicates whether the MPA3 server is controlled by MPANT
5. auxsys (I32)	auxsys describes the use of the particular AUX connector which can be used in the coincidence event handling:

```
auxsys (I32):   System definition word for AUXx
auxsys & 0xFF ==0 not used
              ==2 coinc with any
             bit 4..7 in group 1..4
```

The error code has the following meanings:

1: DMPA3.DLL not found
2: GetDefData function in DLL not found
4: No parameters available

2.8 Getting the MPA-3 Data Settings


MPA-3 Dpar

Data Settings

MPDpar Error

MPDpar.vi

Gets the MPA-3 Data Settings

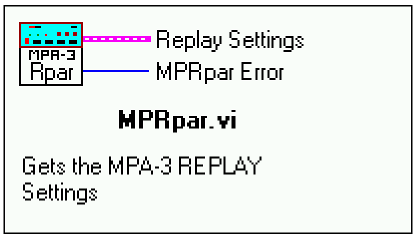
The MPA-3 Data Settings can be obtained by MPDpar.vi. It results the Data Settings in a cluster and a 32 bit integer as an error code. The Data Settings are:

1. savedata (I32):	1 means auto save after stop
2. autoinc (I32):	autoincrement MPA data filename
3. fmt (I32):	MPA format type (0=ASCII, 1=binary, 2=GANAAS)
4. sepfmt (I32):	format type for seperate spectra
5. sephead (I32)	1 means seperate header
6. smpts (I32)	number of points for smoothing operation
7. caluse (I32)	1 means using calibration for shifted spectra summing
8. filename (STR)	MPA data file name
9. specfile (STR)	spectrum file name
10. command (STR)	

The error code has the following meanings:

1: DMPA3.DLL not found
2: GetDatSetting function in DLL not found
4: No parameters available

2.9 Getting the Replay Settings


MPA-3 Rpar

Replay Settings

MPRpar Error

MPRpar.vi

Gets the MPA-3 REPLAY Settings

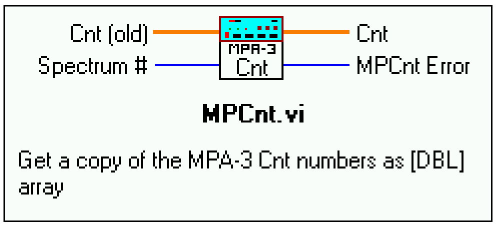
The Replay Settings can be obtained by MPCnt.VI. It needs as input the spectrum number and results a 32 bit integer as an error code and the Replay Settings:

1. use (I32):	1 if Replay Mode ON
2. modified (I32):	1 if different settings are used from measurement time
3. limit (I32):	0=all, 1 = limited time range
4. speed (I32):	replay speed in units of 100 kByte per sec
5. timefrom (DBL):	first time (sec)
6. timeto (DBL):	last time (sec)
7. timepreset (DBL):	last time - first time
8. filename (STR):	listfile for replay

The error code has the following meanings:

1: DMPA3.DLL not found
2: GetReplaySetting function in DLL not found
4: No parameters available

2.10 Getting the Cnt numbers



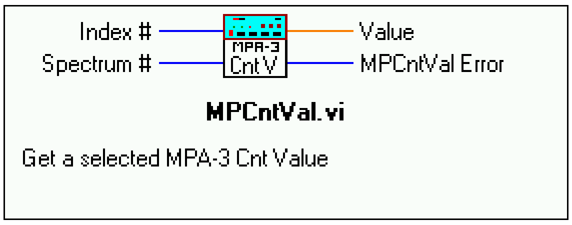
MPCnt.vi

Get a copy of the MPA-3 Cnt numbers as [DBL] array

The Cnt numbers can be obtained by MPCnt.VI. It needs as input the spectrum number and results a 32 bit integer as an error code and the Cnt numbers as an array of 448 DBL containing the Cnt numbers.

Cnt[0] = Realtime, Cnt[1] = Totalsum,
Cnt[2] = ROIsum, Cnt[3] = Totalrate,
Cnt[4] = Net sum, Cnt[5] = Livetime,
Cnt[6] = Deadtime,
Cnt[11] = c0 cal. coeff., Cnt[12] = c1,
Cnt[13] = c2, Cnt[14] = c3,
Cnt[19] = calch0, Cnt[35] = calval0, calib. points
Cnt[20] = calch1, Cnt[36] = calval1,...
Cnt[64]..Cnt[191] : Peak values in
corresponding Roi 0..127 for Calibration
Cnt[192], Cnt[193] ,... Cnt[447]: Roi Sum and Roi Net Sum
in corresponding Roi 0 (, 1, ..127)
(actualized by MPANT when selected)

The error code has the following meanings:
1: DMPA3.DLL not found
2: LVGetCnt function in DLL not found
4: No Data available

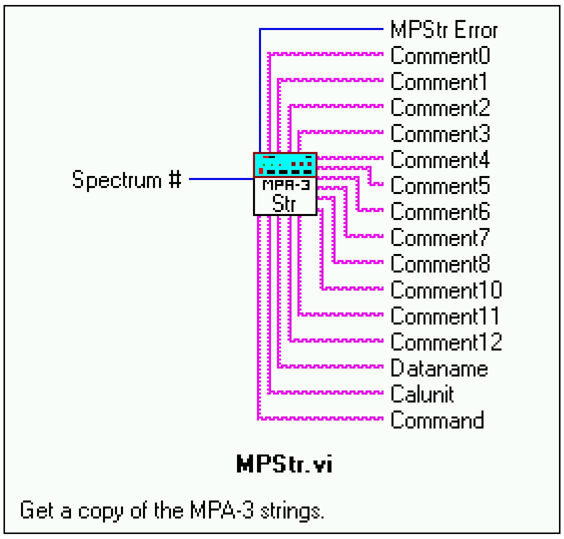


MPCntVal.vi

Get a selected MPA-3 Cnt Value

A selected Cnt Value can be obtained by MPCntVal.VI. Inputs are the spectrum number and the index into the Cnt array. It results a 32 bit integer as an error code and the selected Cnt value as DBL.

2.11 Getting the Strings

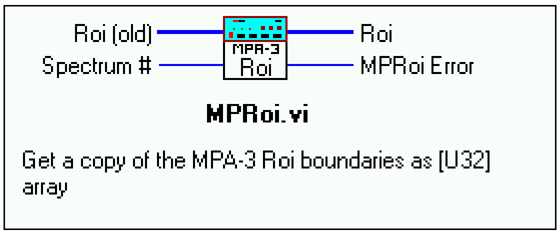


The Strings can be obtained by MPStr.VI. It results a 32 bit integer as an error code and the Strings.

The error code has the following meanings:

1: DMPA3.DLL not found
2: LVGetStr function in DLL not found
4: No Data available

2.12 Getting the ROI boundaries



The ROI boundaries can be obtained by MPRoi.VI. It results a 32 bit integer as an error code and the ROI boundaries for single spectra or rectangle ROIs for dualparameter spectra, respectively, contained in a [U32] array.

The error code has the following meanings:

1: DMPA3.DLL not found
2: LVGetRoi function in DLL not found
4: No Data available

2.13 Getting ROI data from slice or rectangular ROIs

RROI1DAT.VI and RROI2DAT.VI contained in MPA3.LLB demonstrate how to access data in slice and rectangular ROIs. Using the MPANT software create a rectangular ROI in a single- or dualparameter spectra. Then start the VI, enter the spectra # (0 for the first spectra, usually ADC_1A) and the ROI # shown in the MPANT spectra display and run the VI:

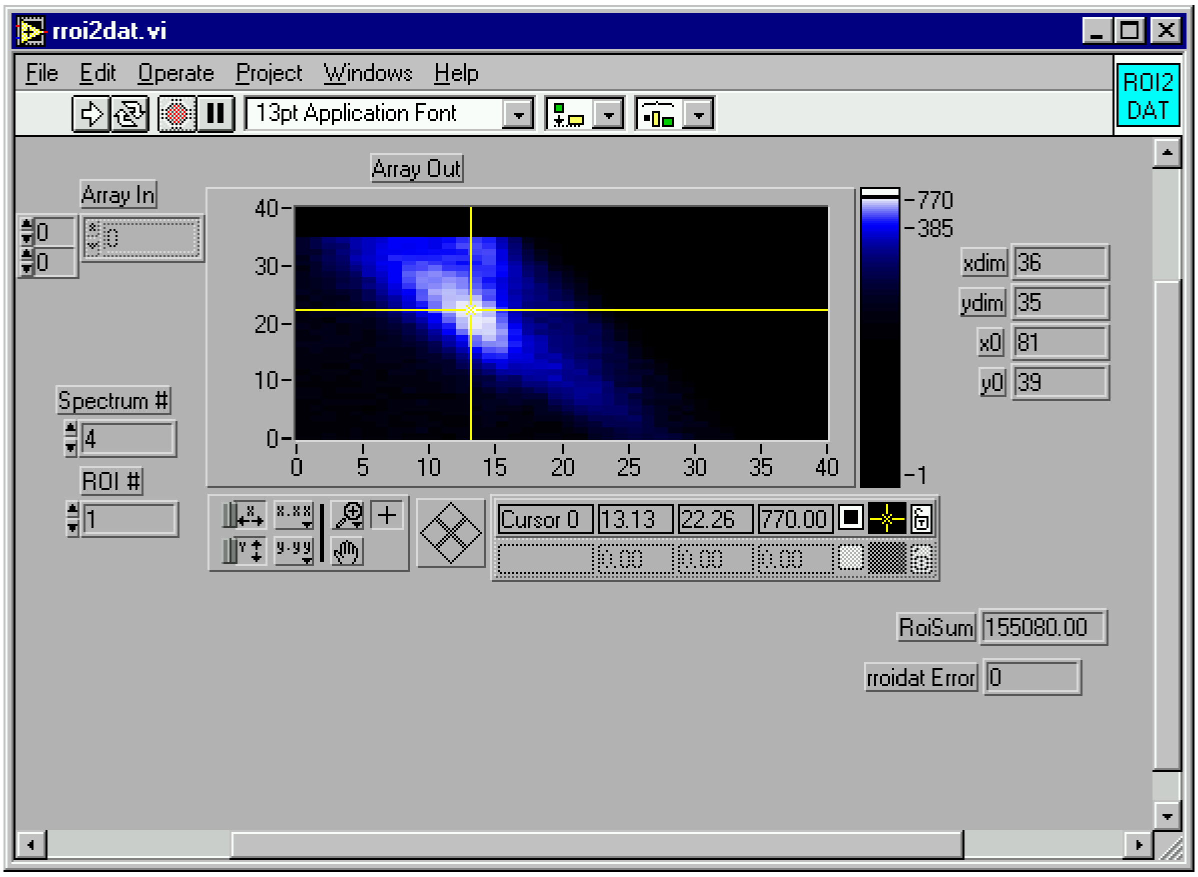
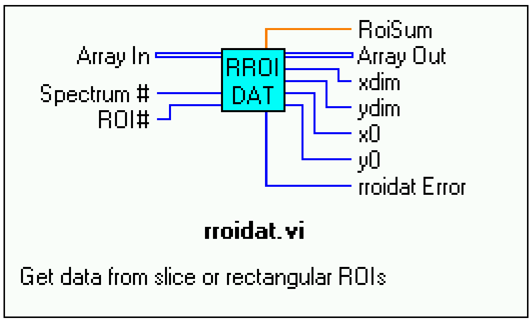


Figure 2.3 Rectangular ROI data.

The RROI2DAT.VI demonstrates the use of rroidat.VI contained in MPA3.LLB.



RROIDAT.VI has the following inputs:

Array In:	Two dimensional array [U32] to be resized. May be left open.
Spectrum # :	(I32) It is 0 for the first spectra (usually ADC_1A).
ROI # :	(I32) The ROI number is displayed in the MPANT display window for the presently activated ROI. The number starts from 1.

The outputs of RROIDAT.VI are:

RoiSum:	Sum of counts (DBL) inside the Roi.
Array Out:	Two dmensional array [U32] containing the data.
xdim:	x-dimension (I32) of the Array.
ydim:	y-dimension (I32) of the Array.
x0:	x-coordinate of lower left corner of the ROI rectangle (I32).
y0:	y-coordinate of lower left corner of the ROI rectangle (I32).

rroidat Error: (I32)

-2: DMPA3.DLL not found.
-4: ROI not found.
-5: Memory allocation failure.
-6: Display not found

2.14 Getting ROI data from curved or polygonal two dimensional ROIs

ROI2DAT.VI contained in MPA3.LLB demonstrates how to access polygonal ROI data. Using the MPANT software create a curved or polygonal ROI in a dualparameter spectra. Then start this VI, enter the ROI Id shown in the MPANT spectra display and run the VI:

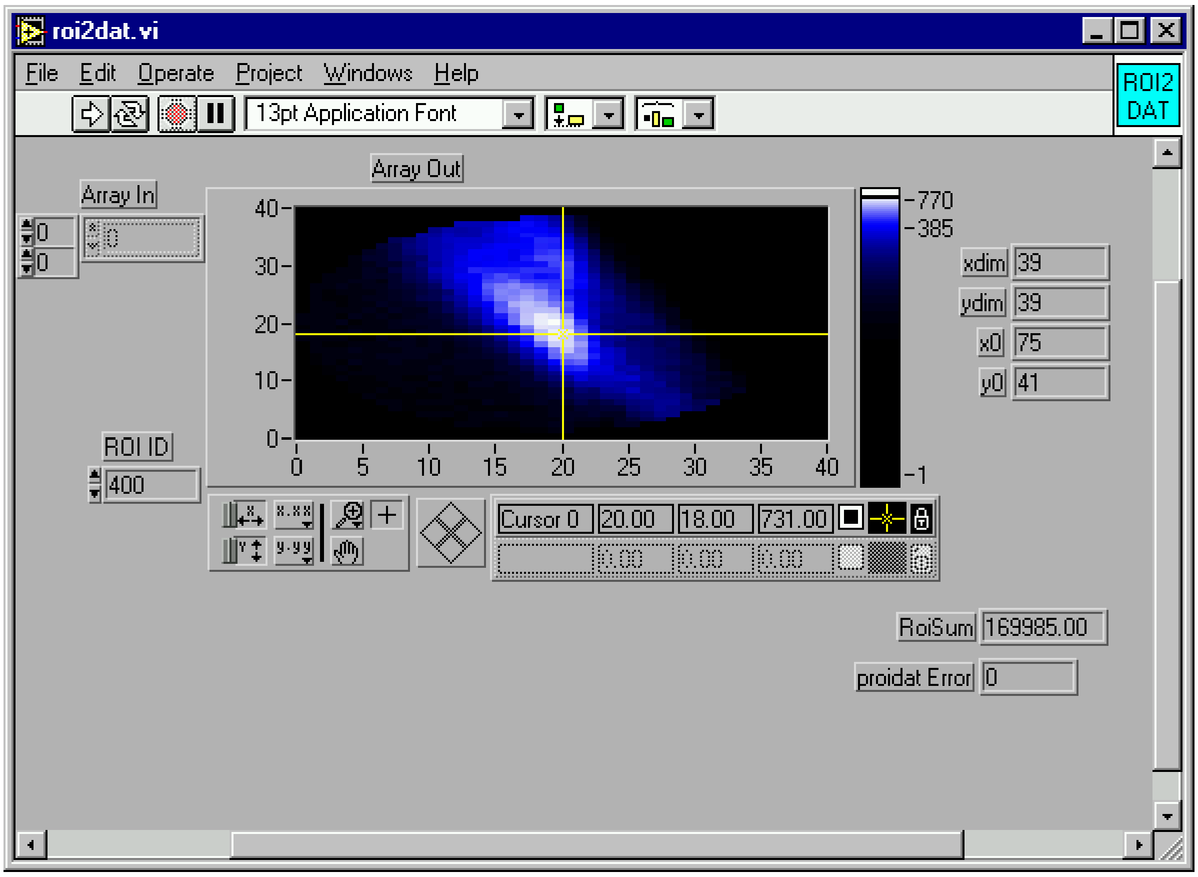
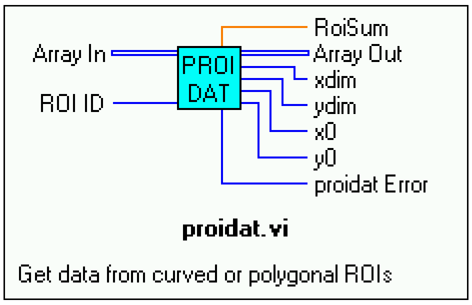


Figure 2.4 Circular ROI data.

The ROI2DAT.VI demonstrates the use of proidat.VI contained in MPA3.LLB.



PROIDAT.VI has the following inputs:

Array In:	Two dimensional array [U32] to be resized. May be left open.
ROI ID:	ROI ID (I32). The ROI Id is displayed in the MPANT display window for the presently activated ROI. The number starts from 100 * spectra Id and is incremented for each new ROI by 1.

The outputs of PROIDAT.VI are:

RoiSum:	Sum of counts (DBL) inside the Roi.
Array Out:	Two dmensional array [U32] containing the data.
xdim:	x-dimension (I32) of the Array.
ydim:	y-dimension (I32) of the Array.
x0:	x-coordinate of lower left corner of the ROI minimum surrounding rectangle (I32).
y0:	y-coordinate of lower left corner of the ROI minimum surrounding rectangle (I32).

proidat Error: (I32)

-2: DMPA3.DLL not found.
-3: MPA3ROI.DLL not found.
-4: ROI not found.
-5: Memory allocation failure.
-6: Display not found

2.14a The CIN source of proidat.vi

The source code of the code interface node in proidat.vi is contained on the distribution disk in a file cinsrc.ex$ (rename it to .exe and run it to extract the archive). It demonstrates how to call the DLL's dmpa3.dll and mpa3roi.dll directly from a "C" program.

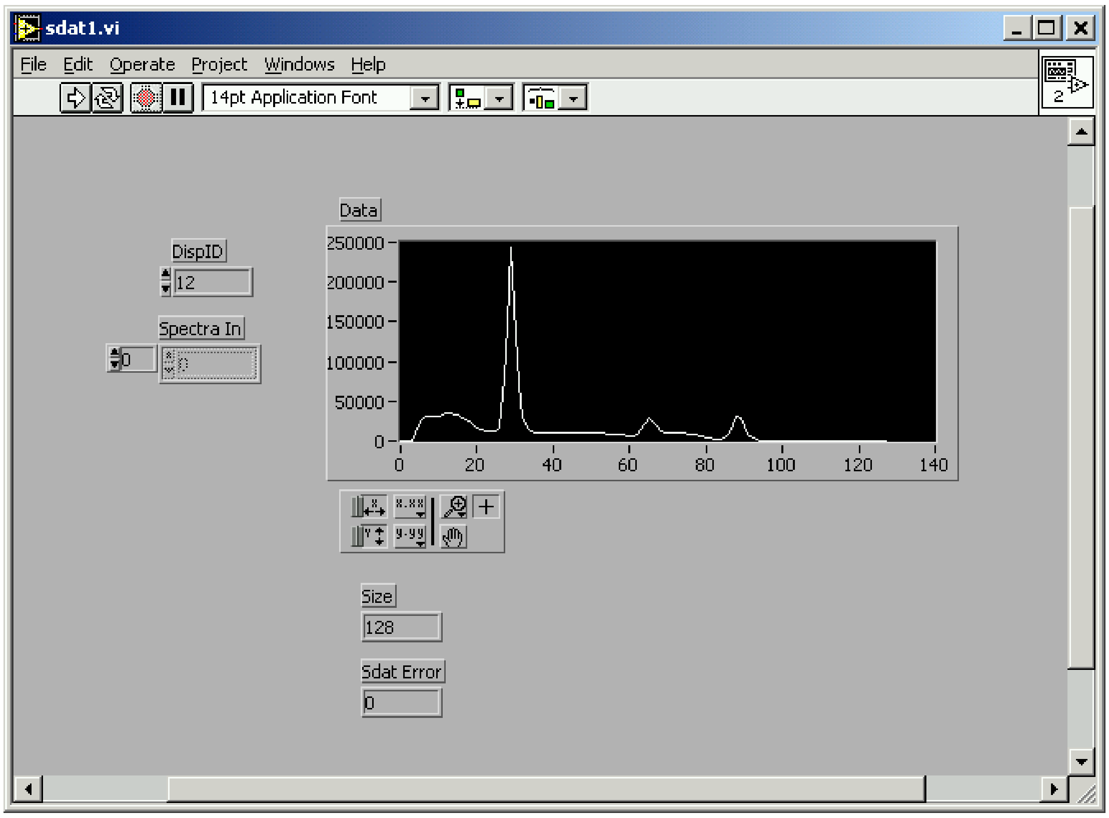
```
/*
 * proidat.c CIN source file
 */

#include "extcode.h"
#include "hosttype.h"
#include <windows.h>

/* stubs for advanced CIN functions */

UseDefaultCINInit
UseDefaultCINDispose
UseDefaultCINAbort
//UseDefaultCINLoad
//UseDefaultCINUnload
UseDefaultCINSave

/*
 * typedefs
 */

typedef struct {
              int32 dimSizes[2];
              uInt32 arg1[1];
              } TD1;
typedef TD1 **TD1Hdl;

typedef struct{
//REGION descriptions
    int   wLockCount;         // (reserved)
    int   wID;                // identification
    int   wClass;             // (reserved)
    int   wType;              // 3=POLY,4=CIRCLE, 5=RING, 6=PIE
    int   wSegments;          // number of polygons
    int   wPoints;            // number of points in all segments
    RECT  rcMaxSurr;          // map rect
    RECT  rcMinSurr;          // minimal surrounding rect
    long  lParam;             // (reserved)
    DWORD  bitsize;           // size of Bits object
    DWORD  chansize;          // size of channels object
    DWORD  counsize;          // size of counts object
    LPSTR  Bits;              // pointer to Bits
    POINT  *Channels;         // pointer to Channels
    int    *Counts;           // pointer to Counts
    HANDLE hBits;       // Handle to mask bits for all segments
    HANDLE hChannels;   // Handle to channels for all segments
    HANDLE hCounts;     // Handle to number of points in each segment
}ROI;
#define LPROI ROI FAR*


HINSTANCE hRoiDLL = 0;
HINSTANCE hDLL = 0;

CIN MgErr CINLoad(RsrcFile rf)
{
  ENTERLVSB
  hDLL = LoadLibrary("dmpa3.DLL");
  hRoiDLL = LoadLibrary("mpa3roi.DLL");
  LEAVELVSB
  return noErr;
}

CIN MgErr CINUnload(void)
{
  ENTERLVSB
  if (hRoiDLL)
    FreeLibrary(hRoiDLL);
  hRoiDLL = 0;
  if (hDLL)
    FreeLibrary(hDLL);
  hDLL = 0;
  LEAVELVSB
  return noErr;
}

CIN MgErr CINRun(TD1Hdl Array_In,
                 int32 *xdim,
                 int32 *ydim,
                 int32 *ROI_ID,
                 int32 *x0,
                 int32 *y0,
                 int32 *proidat_Error,
                 float64 *RoiSum);

CIN MgErr CINRun(TD1Hdl var1, int32 *xdim, int32 *ydim,
                 int32 *ROI_ID, int32 *x0, int32 *y0, int32 *err,
                 float64 *RoiSum)
{

  FARPROC lpFindEntry = NULL;
  FARPROC lpOpenRoi = NULL;
  FARPROC lpGetRoi = NULL;
  FARPROC lpReleaseRoi = NULL;
  FARPROC lpPtInRoi = NULL;
  FARPROC lpGetDatPtr = NULL;
  FARPROC lpReleaseDatPtr = NULL;
  MgErr   cinErr = noErr;
  int   cb=0;
  unsigned long *pt = NULL;
  int y, y1, x, x1, bIn, Entry = -1, ndisp;
  unsigned long val;
```

```
long xmax, ymax, pos;
uInt32 *datp;
LPROI roi;

ENTERLVSB
*err=0;

if (!hDLL) {
  *err = -2;  // DMPA3.DLL not found
  goto out;
}
if (!hRoiDLL) {
  *err = -3;  // MPA3ROI.DLL not found
  goto out;
}
lpFindEntry = GetProcAddress(hRoiDLL, "FindEntry");
lpOpenRoi = GetProcAddress(hRoiDLL, "OpenNewProi");
lpGetRoi = GetProcAddress(hRoiDLL, "GetRoi");
lpReleaseRoi = GetProcAddress(hRoiDLL, "CloseProi");
lpPtInRoi = GetProcAddress(hRoiDLL, "PointInProi");
if (!lpPtInRoi) {*err = -3; goto out; }
if (lpFindEntry) Entry = (*lpFindEntry)(*ROI_ID);
if (Entry < 0) {*err = -4; goto out; } // ROI not found
if (lpOpenRoi) cb = (*lpOpenRoi)(Entry);
if (cb < 0) {*err = -4; goto out; }     // ROI not found
if (lpGetRoi) cb = (*lpGetRoi)(Entry, &roi);
if (!cb) {*err = -4; goto out; }        // ROI not found
*x0 = roi->rcMinSurr.left;
*y0 = roi->rcMinSurr.bottom;
x1 = roi->rcMinSurr.right;
y1 = roi->rcMinSurr.top;
*xdim = x1  - roi->rcMinSurr.left;
*ydim = y1  - roi->rcMinSurr.bottom;

if (cinErr = NumericArrayResize(uL, 2, (UHandle *)(&var1), *xdim *
            (*ydim))) {
  *err = -5;  // Memory allocation failure
  goto out;
}
(*var1)->dimSizes[0] = *xdim;
(*var1)->dimSizes[1] = *ydim;

ndisp = *ROI_ID / 100;

lpGetDatPtr = GetProcAddress(hDLL, "GetDatPtr");
lpReleaseDatPtr = GetProcAddress(hDLL, "ReleaseDatPtr");
if (!lpGetDatPtr) {*err = -2; goto out;}
if (!(*lpGetDatPtr)(ndisp, &xmax, &ymax, &pt)) {
  *err = -6; goto out;}                 // Display not found
datp = (*var1)->arg1;
*RoiSum = 0.;
for (x = *x0; x < x1; x++) {
  for (y = *y0; y < y1; y++) {
    val = 0;
    bIn = (*lpPtInRoi)(roi, x, y);
    if (bIn) {
      pos = y * xmax + x;
      val = *(pt + pos);
      *RoiSum += val;
    }
    *datp = val;
    datp++;
  }
```

```
        }
        (*lpReleaseDatPtr)();

    out:
        (*lpReleaseRoi)(Entry);
        LEAVELVSB

        return cinErr;
    }
```

2.15 Getting data from special views in MPANT like projections or slices

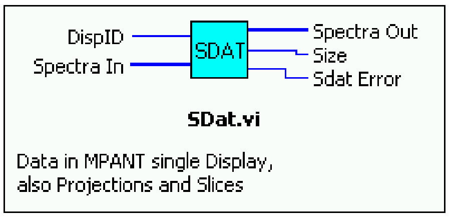
SDAT1.VI contained in MPA3.LLB demonstrates how to access special single spectra like slices or projections of ROIs calculated within the MPANT program. Using the MPANT software create a projection of a ROI or a slice from a dualparameter spectra. Then start this VI, enter the Display Id shown within curved brackets in the title bar of the display window in MPANT and run the VI:



DispID 12
Spectra In 0
Size 128
Sdat Error 0

Figure 2.5 Data from special MPANT views like slices or projections.

The SDAT1.VI demonstrates the use of sdat.VI contained in MPA3.LLB.



SDAT.VI has the following inputs:

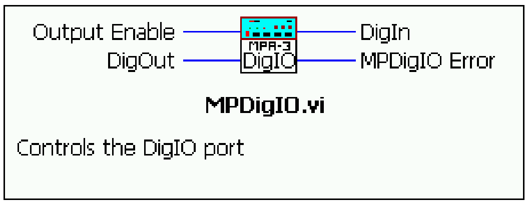
DispID: (I32) ID number of the display view as shown in round brackets in the title bar of the Window in MPANT

Spectra In: One dimensional array [I32] to be resized. May be left open.

The outputs of SDAT.VI are:

Spectra Out: One dimensional array [I32] containing the data.
Size: dimension (I32) of the Array.
Sdat Error: (I32)
-2: DMPA3.DLL not found.
-3: GetSVal function in DMPA3.DLL not found.
-4: MPANT not running
-5: Memory allocation failure.

2.16 Controlling the Dig I/O

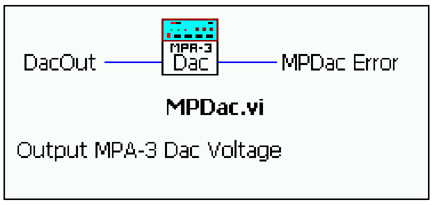


MPDigIO.VI controls input and output of the DIG I/O port including the output enable register by a combined vi. It results the Dig input and a 32 bit integer as an error code.

The error code has the following meanings:

1: DMPA3.DLL not found
2: DigInOut function in DLL not found

2.17 Controlling the DAC Output

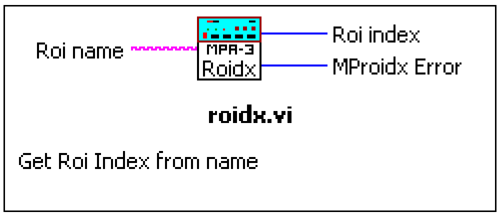


The DAC output voltage can be controlled MPDac.VI. It can set the 8 bit DAC output value.

The error code has the following meanings:

1: DMPA3.DLL not found
2: DacOut function in DLL not found

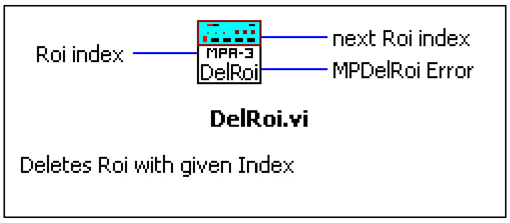
2.18 Get Roi Index from Roi name



Roidx.vi gets a unique index to address ROIs from named ROI's.
Rectangular or 1D ROIs:
LOWORD is the spectra number,
HIWORD is the ROI number (1,2,..)
Polygonal ROIs:
LOWORD is an entry number
HIWORD is the roiid = 100 * spectra_number + ROI_number
returns 0 if not found.
The error code has the following meanings:

1: DMPA3.DLL not found
2: GetRoiIndex function in DLL not found

2.19 Delete Roi with given Index

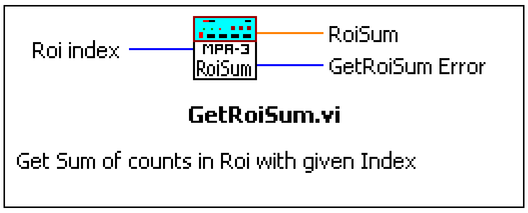


DelRoi.vi deletes a ROI with given index.
The error code has the following meanings:

1: DMPA3.DLL not found
2: DeleteRoi function in DLL not found

A combined DelNamedRoi.vi in MPA3.LLB can be used to delete a named ROI.

2.20 Get Roi Sum

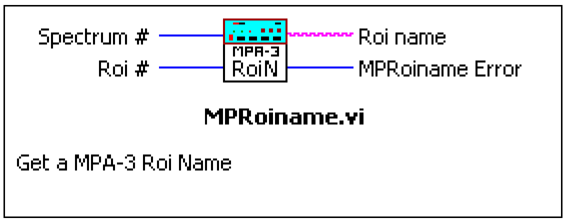


GetRoiSum.vi gets the sum of counts in a ROI with given index.
The error code has the following meanings:

1: DMPA3.DLL not found
2: GetRoiSum function in DLL not found

A combined NamedRoiSum.vi in MPA3.LLB can be used to get the ROISum of a named ROI

2.21 Get Roi Name



MPRoiname.vi gets the name of an ROI in a spectra. ROI # 0..127 are 2-point (rectangular or slice) ROIs, ROI #128 .. 191 are polygonal ROIs.
The error code has the following meanings:

1: DMPA3.DLL not found
2: LVGetRoinam function in DLL not found

3. Using the DMPA3 DLL from Visual Basic

In the following an example is shown how to control the mpa-3 from a simple program in Visual Basic version 5.0.

3.1 The Include File

The include file DECLMPA3.BAS contains the structure and function definitions of the DLL.

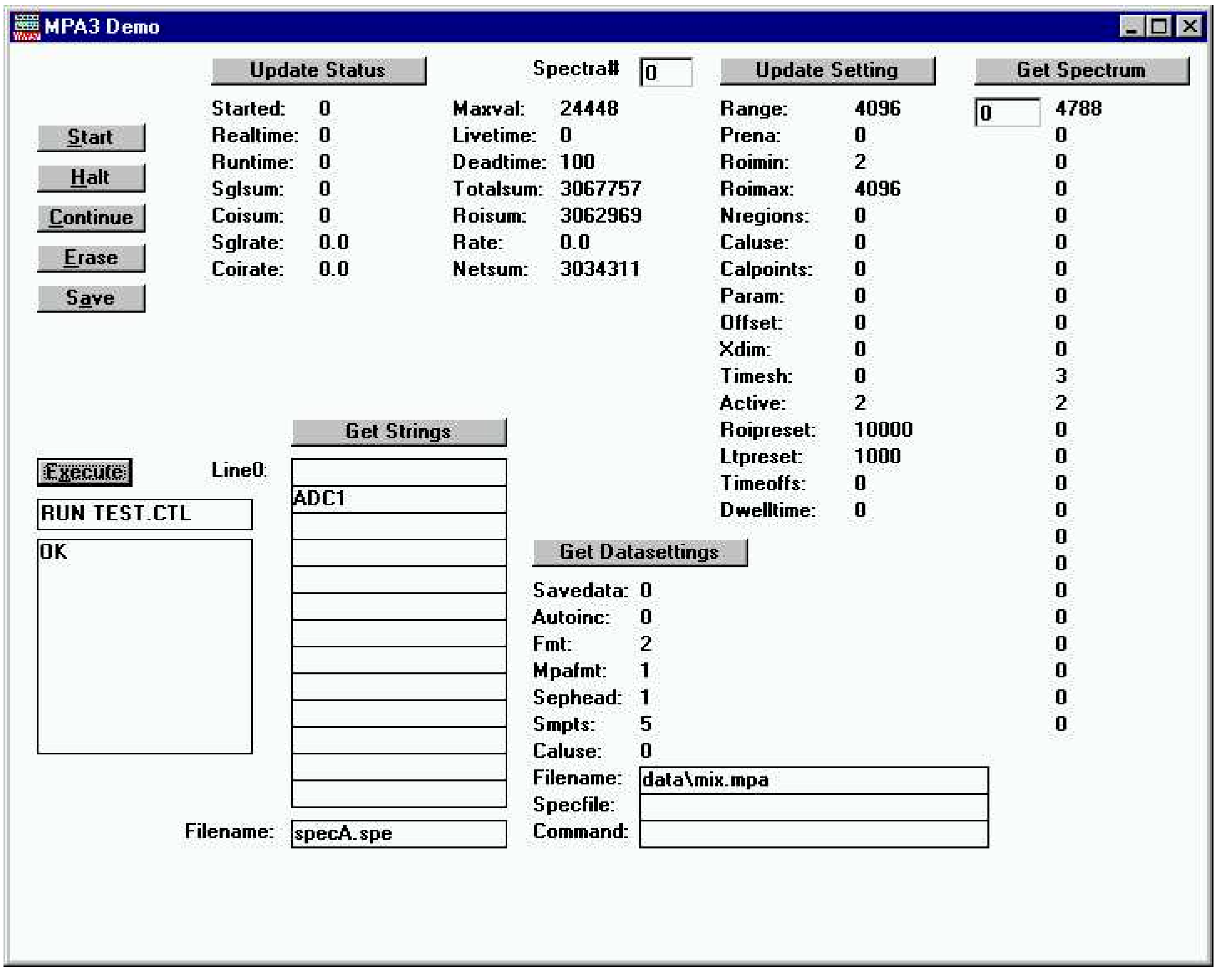
```
Attribute VB_Name = "DECLMPA3"
Type Acqstatus
   Val As Long
   Val1 As Long
   Cnt(0 To 5) As Double
End Type

Type Acqsetting
   Range As Long
   Prena As Long
   Roimin As Long
   Roimax As Long
   Nregions As Long
   Caluse As Long
   Calpoints As Long
   Param As Long
   Offset As Long
   Xdim As Long
   Timesh As Long
   Active As Long
   Roipreset As Double
   Ltpreset As Double
   Timeoffs As Double
   Dwelltime As Double
End Type

Type Replaysetting
   Use As Long
   Modified As Long
   Limit As Long
   Speed As Long
   Timefrom As Double
   Timeto As Double
   Timepreset As Double
   Filename As String * 256
End Type

Type Datsetting
   SaveData As Long
   Autoinc As Long
   Fmt As Long
   Mpafmt As Long
   Sephead As Long
   Smpts As Long
   Caluse As Long
   Filename As String * 256
   Specfile As String * 256
   Command As String * 256
End Type

Type Mp3setting
```

```
   Sen As Long
   Coi As Long
   Ctm As Long
   Dtm As Long
   Tct As Long
   Tp0 As Long
   Tp1 As Long
   Tp2 As Long
   Aui As Long
   Auo As Long
   Dor As Long
   Bk0 As Long
   Bk1 As Long
   Rtcuse As Long
   Dac As Long
   Diguse As Long
   Digval As Long
   Rtprena As Long
   Rtpreset As Double
End Type

Type Acqdef
  Ndevices As Long
  Ndisplays As Long
  Nsystems As Long
  Bremote As Long
  Auxsys As Long
  Sys0(16) As Long
  Sys1(16) As Long
End Type

Declare Sub StoreSettingData Lib "DMPA3.DLL" Alias "#2" (Setting As Acqsetting, ByVal Ndisplay As Long)
Declare Function GetSettingData Lib "DMPA3.DLL" Alias "#3" (Setting As Acqsetting, ByVal Ndisplay As
Long) As Long
Declare Function GetStatusData Lib "DMPA3.DLL" Alias "#5" (Status As Acqstatus, ByVal Ndevice As Long)
As Long
Declare Sub Start Lib "DMPA3.DLL" Alias "#6" (ByVal Nsystem As Long)
Declare Sub Halt Lib "DMPA3.DLL" Alias "#7" (ByVal Nsystem As Long)
Declare Sub Continue Lib "DMPA3.DLL" Alias "#8" (ByVal Nsystem As Long)
Declare Sub NewSetting Lib "DMPA3.DLL" Alias "#9" (ByVal Ndisplay As Long)
Declare Function ServExec Lib "DMPA3.DLL" Alias "#10" (ByVal Clwnd As Long) As Long
Declare Function GetSpec Lib "DMPA3.DLL" Alias "#13" (ByVal I As Long, ByVal Ndisplay As Long) As
Long
Declare Sub SaveSetting Lib "DMPA3.DLL" Alias "#14" ()
Declare Function GetStatus Lib "DMPA3.DLL" Alias "#15" (ByVal Ndevice As Long) As Long
Declare Sub EraseData Lib "DMPA3.DLL" Alias "#16" (ByVal Nsystem As Long)
Declare Sub SaveData Lib "DMPA3.DLL" Alias "#17" (ByVal Ndevice As Long, ByVal All As Long)
Declare Sub GetBlock Lib "DMPA3.DLL" Alias "#18" (Hist As Long, ByVal Start As Long, ByVal Size As
Long, ByVal Stp As Long, ByVal Ndisplay As Long)
Declare Function GetDefData Lib "DMPA3.DLL" Alias "#20" (Def As Acqdef) As Long
Declare Sub LoadData Lib "DMPA3.DLL" Alias "#21" (ByVal Ndevice As Long, ByVal All As Long)
Declare Sub NewData Lib "DMPA3.DLL" Alias "#22" ()
Declare Sub HardwareDlg Lib "DMPA3.DLL" Alias "#23" (ByVal Item As Long)
Declare Sub UnregisterClient Lib "DMPA3.DLL" Alias "#24" ()
Declare Sub DestroyClient Lib "DMPA3.DLL" Alias "#25" ()
Declare Sub RunCmd Lib "DMPA3.DLL" Alias "#28" (ByVal Ndevice As Long, ByVal Cmd As String)
Declare Sub AddData Lib "DMPA3.DLL" Alias "#29" (ByVal Ndisplay As Long, ByVal All As Long)
Declare Function LVGetRoi Lib "DMPA3.DLL" Alias "#30" (Roi As Long, ByVal Ndisplay As Long) As Long
Declare Function LVGetCnt Lib "DMPA3.DLL" Alias "#31" (Cnt As Double, ByVal Ndisplay As Long) As
Long
Declare Function LVGetOneCnt Lib "DMPA3.DLL" Alias "#32" (Cnt As Double, ByVal Ndisplay As Long,
ByVal Cntnum As Long) As Long
```

```
Declare Function LVGetStr Lib "DMPA3.DLL" Alias "#33" (ByVal Comment As String, ByVal Ndisplay As
Long) As Long
Declare Sub SubData Lib "DMPA3.DLL" Alias "#34" (ByVal Ndisplay As Long, ByVal All As Long)
Declare Sub Smooth Lib "DMPA3.DLL" Alias "#35" (ByVal Ndisplay As Long)
Declare Function GetMP3Setting Lib "DMPA3.DLL" Alias "#39" (Msetting As Mp3setting) As Long
Declare Function GetDatSetting Lib "DMPA3.DLL" Alias "#41" (Dsetting As Datsetting) As Long
Declare Function GetReplaySetting Lib "DMPA3.DLL" Alias "#43" (Rsetting As Replaysetting) As Long
Declare Function DigInOut Lib "DMPA3.DLL" Alias "#48" (ByVal Value As Long, ByVal Enable As Long) As
Long
Declare Function DacOut Lib "DMPA3.DLL" Alias "#49" (ByVal Value As Long) As Long
```

3.2 The Visual Basic demo program

The simple Visual Basic program is shown here: It allows to get Status, Settings, spectrum data and strings for any MPA-3 spectrum, perform actions like start, halt, continue, erase, or any control command. To use the example, it is recommended to copy the DMPA3.DLL into your WINNT\SYSTEM32 directory and delete or rename it in your working directory for example into DMPA3.DL_ . It is essential that the DLL is loaded from Visual Basic and the MPA3 server program from the same path.



```
Started:   0          Maxval:    24448      Range:      4096     0   4788
Realtime:  0          Livetime:  0          Prena:      0            0
Runtime:   0          Deadtime:  100        Roimin:     2            0
Sglsum:    0          Totalsum:  3067757    Roimax:     4096         0
Coisum:    0          Roisum:    3062969    Nregions:   0            0
Sglrate:   0.0        Rate:      0.0        Caluse:     0            0
Coirate:   0.0        Netsum:    3034311    Calpoints:  0            0
                                            Param:      0            0
                                            Offset:     0            0
                                            Xdim:       0            0
                                            Timesh:     0            3
                                            Active:     2            2
                                            Roipreset:  10000        0
                                            Ltpreset:   1000         0
                                            Timeoffs:   0            0
                                            Dwelltime:  0            0
                                                                     0
                                                                     0
RUN TEST.CTL      Line0:                    Savedata:  0             0
OK                ADC1                      Autoinc:   0             0
                                            Fmt:       2             0
                                            Mpafmt:    1             0
                                            Sephead:   1             0
                                            Smpts:     5             0
                                            Caluse:    0
                                            Filename:  data\mix.mpa
                                            Specfile:
                  Filename: specA.spe       Command:
```

Fig. 3.1: The Visual Basic MPA3 Demo program

This is the complete program code:

```
Attribute VB_Name = "Form1"
Attribute VB_GlobalNameSpace = False
Attribute VB_Creatable = False
Attribute VB_PredeclaredId = True
Attribute VB_Exposed = False
Dim Status As Acqstatus
Dim Setting As Acqsetting
Dim Dsetting As Datsetting
Dim OldStarted As Integer
Dim Mcano As Long
Dim Sysno As Long
Dim Chan As Long
Dim Hist(24) As Long
Dim Toggle As Integer
```

```
Private Sub CommandContinue_Click()
Call Continue(0)
End Sub

Private Sub CommandDatasettings_Click()
  Call GetDatSetting(Dsetting)
  LabelSavedata.Caption = Dsetting.SaveData
  LabelAutoinc.Caption = Dsetting.Autoinc
  LabelFmt.Caption = Dsetting.Fmt
  LabelMpafmt.Caption = Dsetting.Mpafmt
  LabelSephead.Caption = Dsetting.Sephead
  LabelSmpts.Caption = Dsetting.Smpts
  LabelAddcal.Caption = Dsetting.Caluse
  LabelMpafilename.Caption = Dsetting.Filename
  Labelspecfile.Caption = Dsetting.Specfile
  LabelCommand.Caption = Dsetting.Command
End Sub

Private Sub CommandErase_Click()
  Call EraseData(0)
End Sub

Private Sub CommandExecute_Click()
  Dim a As String * 1024
  Mid$(a, 1) = TextCommand.Text
  Call RunCmd(0, a)
  LabelRespons.Caption = a
  Mcano = Val(TextMC.Text)
  Ret = GetStatus(Mcano + 1)
  Ret = GetStatusData(Status, 0)
  Call UpdateMpStatus
  Ret = GetStatusData(Status, Mcano + 1)
  Call UpdateStatus
End Sub

Private Sub CommandGetspec_Click()
  Mcano = Val(TextMC.Text)
  Chan = Val(TextChan.Text)
  Call GetBlock(Hist(0), Chan, Chan + 24, 1, Mcano)
  For I = 0 To 23 Step 1
    LabelData(I).Caption = Hist(I)
  Next I
End Sub

Private Sub CommandGetstring_Click()
  Dim b As String * 1024
  Mcano = Val(TextMC.Text)
  Ret = LVGetStr(b, Mcano)
  LabelLine(0).Caption = Mid$(b, 1, 60)
  LabelLine(1).Caption = Mid$(b, 61, 60)
  LabelLine(2).Caption = Mid$(b, 121, 60)
  LabelLine(3).Caption = Mid$(b, 181, 60)
  LabelLine(4).Caption = Mid$(b, 241, 60)
  LabelLine(5).Caption = Mid$(b, 301, 60)
  LabelLine(6).Caption = Mid$(b, 361, 60)
  LabelLine(7).Caption = Mid$(b, 421, 60)
  LabelLine(8).Caption = Mid$(b, 481, 60)
  LabelLine(9).Caption = Mid$(b, 541, 60)
  LabelLine(10).Caption = Mid$(b, 601, 60)
  LabelLine(11).Caption = Mid$(b, 881, 60)
  LabelLine(12).Caption = Mid$(b, 961, 60)
  LabelLine(13).Caption = Mid$(b, 661, 100)
End Sub

Private Sub CommandHalt_Click()
  Call Halt(0)
End Sub


Private Sub CommandSave_Click()
  Call SaveData(0, 1)
End Sub

Private Sub CommandSetting_Click()
  Mcano = Val(TextMC.Text)
  If GetSettingData(Setting, Mcano) = 1 Then
     Call UpdateSetting
  End If
End Sub

Private Sub CommandStart_Click()
  Call Start(0)
End Sub

Private Sub CommandUpdate_Click()
  Mcano = Val(TextMC.Text)
  Ret = GetStatus(Mcano + 1)
  If GetStatusData(Status, Mcano + 1) = 1 Then
     Call UpdateStatus
  End If
End Sub

Private Sub Form_Load()
  OldStarted = 0
  Mcano = 0
  Sysno = 0
  Chan = 0
  Ret = ServExec(0)
  Ret = GetStatus(0)
  Ret = GetStatusData(Status, 0)
  Call UpdateMpStatus
  Ret = GetStatusData(Status, Mcano + 1)
  Call UpdateStatus
  Ret = GetSettingData(Setting, 0)
  Call UpdateSetting
End Sub

Private Sub Timer1_Timer()
  Mcano = Val(TextMC.Text)
  If Mcano > 16 Then
    Mcano = 16
  End If
  If Mcano < 0 Then
    Mcano = 0
  End If
  Ret = GetStatus(0)
  If GetStatusData(Status, 0) = 1 Then
    If Status.Val = 1 Or OldStarted = 1 Then
      Toggle = Not Toggle
      OldStarted = Status.Val
      Call UpdateMpStatus
      If GetStatusData(Status, Mcano + 1) = 1 Then
```

```
        Call UpdateStatus
      End If
    End If
  End If
End Sub

Private Sub UpdateMpStatus()
      LabelStarted.Caption = Status.Val
      LabelRealtime.Caption = Status.Cnt(0)
      LabelRuntime.Caption = Status.Cnt(1)
      LabelSglsum.Caption = Status.Cnt(2)
      LabelCoisum.Caption = Status.Cnt(3)
      LabelSglrate.Caption = Format$(Status.Cnt(4), "######0.0#")
      LabelCoirate.Caption = Format$(Status.Cnt(5), "######0.0#")
End Sub

Private Sub UpdateStatus()
      LabelMaxval.Caption = Status.Val
      LabelLivetime.Caption = Status.Cnt(0)
      LabelDeadtime.Caption = Status.Cnt(1)
      LabelTotalsum.Caption = Status.Cnt(2)
      LabelRoisum.Caption = Status.Cnt(3)
      LabelTotalrate.Caption = Format$(Status.Cnt(4), "######0.0#")
      LabelNetsum.Caption = Status.Cnt(5)
End Sub

Private Sub UpdateSetting()
      LabelRange.Caption = Setting.Range
      Labelprena.Caption = Setting.Prena
      LabelRoimin.Caption = Setting.Roimin
      LabelRoimax.Caption = Setting.Roimax
      LabelNregions.Caption = Setting.Nregions
      LabelCaluse.Caption = Setting.Caluse
      LabelCalpoints.Caption = Setting.Calpoints
      LabelParam.Caption = Setting.Param
      LabelOffset.Caption = Setting.Offset
      LabelXdim.Caption = Setting.Xdim
      LabelTimesh.Caption = Setting.Timesh
      LabelActive.Caption = Setting.Active
      LabelRoipreset.Caption = Setting.Roipreset
      LabelLtpreset.Caption = Setting.Ltpreset
      LabelTimeoffs.Caption = Setting.Timeoffs
      LabelDwelltime.Caption = Setting.Dwelltime
End Sub
```

4. Using the DMPA3 DLL from C

In the following an example is shown how to control the MPA-3 from a simple console application written in Microsoft C.

4.1 The Include File

The include file dmpa3.h contains the function definitions of the DLL. It includes also the structure definitions from struct.h listed in the appendix A.1.

```
#ifdef __cplusplus
extern "C"
{
#endif

#include "struct.h"

#define MAXCNT              448
#define MAXDEV              17

#ifdef WINDOWS31
#define MAXDSP        32
#else
#define MAXDSP        50
#endif

#define ID_SAVE             103
#define ID_CONTINUE         106
#define ID_START            109
#define ID_BREAK            137
#define ID_NEWSETTING       139
#define ID_GETSTATUS        141
#define ID_SAVEFILE         151
#define ID_ERASE            154
#define ID_LOADFILE         155
#define ID_NEWDATA          160
#define ID_HARDWDLG         161
#define ID_SAVEFILE2        194
#define ID_LOADFILE2        203
#define ID_SAVEFILE3        217
#define ID_LOADFILE3        219
#define ID_SAVEFILE4        223
#define ID_LOADFILE4        225
#define ID_LOADFILE5        226
#define ID_LOADFILE6        227
#define ID_LOADFILE7        228
#define ID_LOADFILE8        229
#define ID_SAVEFILE5        230
#define ID_SAVEFILE6        231
#define ID_SAVEFILE7        232
#define ID_SAVEFILE8        233
#define ID_SUMFILE          234
#define ID_SUMFILE2         235
#define ID_SUMFILE3         236
#define ID_SUMFILE4         237
#define ID_SUMFILE5         238
#define ID_SUMFILE6         239
#define ID_SUMFILE7         240
#define ID_SUMFILE8         241
#define ID_SUBTRACT         289
#define ID_SMOOTH           290
```

```
#define ID_SUBTRACT2          296
#define ID_SMOOTH2            297
#define ID_SUBTRACT3          298
#define ID_SMOOTH3            299
#define ID_SUBTRACT4          300
#define ID_SMOOTH4            301
#define ID_SUBTRACT5          302
#define ID_SMOOTH5            303
#define ID_SUBTRACT6          304
#define ID_SMOOTH6            305
#define ID_SUBTRACT7          306
#define ID_SMOOTH7            307
#define ID_SUBTRACT8          308
#define ID_SMOOTH8            309
#define ID_COMBDLG            401
#define ID_DATADLG            402
#define ID_MAPLSTDLG          403
#define ID_REPLDLG            404
#define ID_ERASE2             1108
#define ID_ERASE3             1109
#define ID_ERASE4             1110
#define ID_ERASEFILE2         1111
#define ID_ERASEFILE3         1112
#define ID_ERASEFILE4         1113
#define ID_START2             1114
#define ID_BREAK2             1115
#define ID_CONTINUE2          1116
#define ID_START3             1117
#define ID_BREAK3             1118
#define ID_CONTINUE3          1119
#define ID_START4             1120
#define ID_BREAK4             1121
#define ID_CONTINUE4          1122
#define ID_RUNCMD             1123
#define ID_RUNCMD2            1124
#define ID_RUNCMD3            1125
#define ID_RUNCMD4            1126
#define ID_RUNCMD5            1127
#define ID_RUNCMD6            1128
#define ID_RUNCMD7            1129
#define ID_RUNCMD8            1130
#define ID_ERASEFILE5         1131
#define ID_ERASEFILE6         1132
#define ID_ERASEFILE7         1133
#define ID_ERASEFILE8         1134
#define ID_DIGINOUT           1137
#define ID_DACOUT             1138

/*** FUNCTION PROTOTYPES (do not change) ***/
#ifdef DLL
BOOL APIENTRY DllMain(HANDLE hInst, DWORD ul_reason_being_called, LPVOID lpReserved);

VOID APIENTRY StoreSettingData(ACQSETTING *Setting, int nDisplay);
                        // Stores Spectra Settings into the DLL
int APIENTRY GetSettingData(ACQSETTING *Setting, int nDisplay);
                        // Get Spectra Settings stored in the DLL
VOID APIENTRY StoreStatusData(ACQSTATUS *Status, int nDev);
                        // Store the Status into the DLL
int APIENTRY GetStatusData(ACQSTATUS *Status, int nDev);
                        // Get the Status
VOID APIENTRY Start(int nSystem);       // Start
VOID APIENTRY Halt(int nSystem);        // Halt
VOID APIENTRY Continue(int nSystem);     // Continue
VOID APIENTRY NewSetting(int nDisplay);   // Indicate new Settings to Server
UINT APIENTRY ServExec(HWND ClientWnd);  // Register client at Server MPA3.EXE
VOID APIENTRY StoreData(ACQDATA *Data, int nDisplay);
                              // Stores Data pointers into the DLL
long APIENTRY GetSpec(long i, int nDisplay);
                              // Get a spectrum value
VOID APIENTRY SaveSetting(void);        // Save Settings
int APIENTRY GetStatus(int nDev);           // Request actual Status from Server
VOID APIENTRY Erase(int nSystem);       // Erase spectrum
VOID APIENTRY SaveData(int nDisplay, int all);     // Saves data
VOID APIENTRY GetBlock(long *hist, int start, int end, int step,
  int nDisplay);                          // Get a block of spectrum data
VOID APIENTRY StoreDefData(ACQDEF *Def);
                                          // Store System Definition into DLL
int APIENTRY GetDefData(ACQDEF *Def);             // Get System Definition
VOID APIENTRY LoadData(int nDisplay, int all);     // Loads data
VOID APIENTRY NewData(void);                        // Indicate new ROI or string Data
VOID APIENTRY HardwareDlg(int item);              // item=0: Calls the Settings dialog
                              // 1: data dialog, 2: system dialog
VOID APIENTRY UnregisterClient(void);              // Clears remote mode from MPANT
VOID APIENTRY DestroyClient(void);                 // Close MPANT
UINT APIENTRY ClientExec(HWND ServerWnd);
                              // Execute the Client MPANT.EXE
int APIENTRY LVGetDat(unsigned long HUGE *datp, int nDisplay);
                              // Copies the spectrum to an array
VOID APIENTRY RunCmd(int nDisplay, LPSTR Cmd);
                              // Executes command
VOID APIENTRY AddData(int nDisplay, int all);      // Adds data
VOID APIENTRY SubData(int nDisplay, int all);      // Subtracts data
VOID APIENTRY Smooth(int nDisplay);       // Smooth data
int APIENTRY LVGetRoi(unsigned long FAR *roip, int nDisplay);
                          // Copies the ROI boundaries to an array
int APIENTRY LVGetOneRoi(int nDisplay, int roinum, long *x1, long *x2);
                                                    // Get one ROI boundary
int APIENTRY LVGetCnt(double *cntp, int nDisplay);
                                          // Copies Cnt numbers to an array
int APIENTRY LVGetOneCnt(double *cntp, int nDisplay, int cntnum);
                                          // Get one Cnt number
int APIENTRY LVGetStr(char *strp, int nDisplay);
                              // Copies strings to an array
VOID APIENTRY StoreMP3Setting(ACQMP3 *Defmp3);
                              // Store MP3 Settings into DLL
int APIENTRY GetMP3Setting(ACQMP3 *Defmp3);
                              // Get MP3 Settings from DLL
VOID APIENTRY StoreDatSetting(DATSETTING *Defdat);
                              // Store Data Format Definition into DLL
int APIENTRY GetDatSetting(DATSETTING *Defdat);
                              // Get Data Format Definition from DLL
VOID APIENTRY StoreReplaySetting(REPLAYSETTING *Repldat);
                              // Store Replay Settings into DLL
int APIENTRY GetReplaySetting(REPLAYSETTING *Repldat);
                              // Get Replay Settings from DLL
int APIENTRY GetDatPtr(int nDisplay, long *xmax, long *ymax, LPSTR *pt);
                              // Get a temporary pointer to spectra data
int APIENTRY ReleaseDatPtr(void);
                              // Release temporary data pointer
long APIENTRY GetSVal(int DspID, long xval);
                              // Get special display data like projections from MPANT
int APIENTRY DigInOut(int value, int enable);        // controls Dig I/0 ,
                                                    // returns digin
int APIENTRY DacOut(int value);   // output Dac value as analogue voltage
```

```
#else
typedef int (WINAPI *IMPAGETSETTING) (ACQSETTING FAR *Setting, int nDisplay);
                              // Get Spectra Settings stored in the DLL
typedef int (WINAPI *IMPAGETSTATUS) (ACQSTATUS FAR *Status, int nDisplay);
                              // Get the Status
typedef VOID (WINAPI *IMPARUNCMD) (int nDisplay, LPSTR Cmd);
                              // Executes command
typedef int (WINAPI *IMPAGETCNT) (double FAR *cntp, int nDisplay);
                              // Copies Cnt numbers to an array
typedef int (WINAPI *IMPAGETROI) (unsigned long FAR *roip, int nDisplay);
                              // Copies the ROI boundaries to an array
typedef int (WINAPI *IMPAGETDEF) (ACQDEF FAR *Def);
                              // Get System Definition
typedef int (WINAPI *IMPAGETDAT) (unsigned long HUGE *datp, int nDisplay);
                              // Copies the spectrum to an array
typedef int (WINAPI *IMPAGETSTR) (char FAR *strp, int nDisplay);
                              // Copies strings to an array
typedef UINT (WINAPI *IMPASERVEXEC) (HWND ClientWnd);  // Register client at server MPA3.EXE

typedef int (WINAPI *IMPANEWSTATUS) (int nDev);     // Request actual Status from Server

typedef int (WINAPI *IMPAGETMP3SET) (ACQMP3 *Defmp3);
                           // Get MPA3 Settings from DLL
typedef int (WINAPI *IMPAGETDATSET) (DATSETTING *Defdat);
                           // Get Data Format Definition from DLL
typedef int (WINAPI *IMPADIGINOUT) (int value, int enable);  // controls Dig I/0 ,
                                                              // returns digin
typedef int (WINAPI *IMPADACOUT) (int value);   // output Dac value as analogue voltage
typedef VOID (WINAPI *IMPASTART) (int nSystem);        // Start
typedef VOID (WINAPI *IMPAHALT) (int nSystem);         // Halt
typedef VOID (WINAPI *IMPACONTINUE) (int nSystem);     // Continue
typedef VOID (WINAPI *IMPAERASE) (int nSystem);        // Erase spectrum
#endif

#ifdef __cplusplus
}
#endif
```

4.2 The C demo program

The source of the simple C program is shown here: It shows how to access the DLL and to get Status, Settings and spectrum data. To perform actions like start, halt, continue, erase, just send the corresponding commands using the command language.

```
// -----------------------------------------------------------------------
// TSTMPA3.C : DMPA3.DLL Software driver C example
// -----------------------------------------------------------------------

#include <stdio.h>
#include <string.h>
#include <windows.h>
#include <time.h>

#undef DLL
#include "dmpa3.h"


HANDLE          hDLL = 0;
```

```
IMPAGETSETTING  lpSet=NULL;
IMPANEWSTATUS   lpNewStat=NULL;
IMPAGETSTATUS   lpStat=NULL;
IMPARUNCMD      lpRun=NULL;
IMPAGETCNT      lpCnt=NULL;
IMPAGETROI      lpRoi=NULL;
IMPAGETDAT      lpDat=NULL;
IMPAGETSTR      lpStr=NULL;
IMPASERVEXEC    lpServ=NULL;
IMPAGETDATSET   lpGetDatSet=NULL;
IMPAGETMP3SET   lpGetMp3Set=NULL;
IMPADIGINOUT    lpDigInOut=NULL;
IMPADACOUT      lpDacOut=NULL;
IMPASTART               lpStart=NULL;
IMPAHALT                lpHalt=NULL;
IMPACONTINUE    lpContinue=NULL;
IMPAERASE               lpErase=NULL;


ACQSETTING     Setting={0};
ACQDATA        Data={0};
ACQDEF         Def={0};
ACQSTATUS      Status={0};
DATSETTING     DatSetting={0};
ACQMP3                  MP3Setting={0};

short nDev=0;

void help()
{
        printf("Commands:\n");
        printf("Q                       Quit\n");
        printf("?           Help\n");
        printf("S       Show Status\n");
        printf("H                       Halt\n");
        printf("T       Show Setting\n");
        printf("ADC=x   Switch to ADC #x (0=MPA)\n");
   printf("(... more see command language in MPANT help)\n");
   printf("\n");
}

void PrintMpaStatus(ACQSTATUS *Stat)
{
  if(Stat->val) printf("ON\n"); else printf("OFF\n");
  printf("realtime= %.2lf\n", Stat->cnt[ST_REALTIME]);
  printf("runtime= %.2lf\n", Stat->cnt[ST_RUNTIME]);
  printf("single_ev= %lf\n", Stat->cnt[ST_SINGLESUM]);
  printf("coinc_ev= %lf\n", Stat->cnt[ST_COINCSUM]);
  printf("sglrate= %.2lf\n", Stat->cnt[ST_SGLRATE]);
  printf("coirate= %.2lf\n", Stat->cnt[ST_COIRATE]);
}

void PrintStatus(ACQSTATUS *Stat)
{
  printf("livetime= %.2lf\n", Stat->cnt[ST_LIVETIME]);
  printf("deadtime%%= %.2lf\n", Stat->cnt[ST_DEADTIME]);
  printf("totalsum= %lf\n", Stat->cnt[ST_TOTALSUM]);
  printf("roisum= %lf\n", Stat->cnt[ST_ROISUM]);
  printf("netsum= %lf\n", Stat->cnt[ST_NETSUM]);
  printf("totalrate= %.2lf\n", Stat->cnt[ST_TOTALRATE]);
}
```

```
void PrintDatSetting(DATSETTING *Set)
{
 printf("savedata= %d\n", Set->savedata);
 printf("autoinc= %d\n", Set->autoinc);
 printf("fmt= %d\n", Set->fmt);
 printf("sepfmt= %d\n", Set->sepfmt);
 printf("sephead= %d\n", Set->sephead);
 printf("filename= %s\n", Set->filename);
}

void PrintMP3Setting(ACQMP3 *Set)
{
 printf("rtcuse= %x\n", Set->rtcuse);
 printf("dac= %d\n", Set->dac);
 printf("diguse= %x\n", Set->diguse);
 printf("digval= %d\n", Set->digval);
 printf("rtprena= %x\n", Set->rtprena);
 printf("rtpreset= %lg\n", Set->rtpreset);
}

void PrintSetting(ACQSETTING *Set)
{
 printf("range= %ld\n", Set->range);
 printf("prena= %x\n", Set->prena);
 printf("roimin= %ld\n", Set->roimin);
 printf("roimax= %ld\n", Set->roimax);
 printf("nregions= %d\n", Set->nregions);
 printf("caluse= %d\n", Set->caluse);
 printf("calpoints= %d\n", Set->calpoints);
 printf("param= %lx\n", Set->param);
 printf("offset= %lx\n", Set->offset);
 printf("xdim= %d\n", Set->xdim);
 printf("timesh= %d\n", Set->timesh);
 printf("active= %x\n", Set->active);
 printf("roipreset= %lg\n", Set->roipreset);
 printf("ltpreset= %lg\n", Set->ltpreset);
 printf("timeoffs= %lg\n", Set->timeoffs);
 printf("dwelltime= %lg\n", Set->dwelltime);
}

int run(char *command)
{
   int err;
   if (!stricmp(command, "?"))           help();
   else if (!stricmp(command,"Q"))       return 1;
   else if (!stricmp(command,"S")) {
     err = (*lpStat)(&Status, nDev);
     if (nDev) PrintStatus(&Status);
     else PrintMpaStatus(&Status);
   }
   else if (!stricmp(command,"T")) {
     if (nDev) {     // spectra settings
       err = (*lpSet)(&Setting, nDev-1);
       printf("ADC %d:\n", nDev);
       PrintSetting(&Setting);
     }
    else {        // MPA3 settings
       err = (*lpGetMp3Set)(&MP3Setting);
       PrintMP3Setting(&MP3Setting);
                                  // DATSettings
       err = (*lpGetDatSet)(&DatSetting);
       PrintDatSetting(&DatSetting);
     }
   }
   else if (!stricmp(command,"H")) {
     (*lpHalt)(0);
   }
   else if(!strnicmp(command, "ADC=", 4)) {
     sscanf(command+4, "%d", &nDev);
     (*lpRun)(0, command);
   }
   else if (!stricmp(command,"MPA")) {
     nDev=0;
     (*lpRun)(0, command);
   }
   else {
     (*lpRun)(0, command);
     printf("%s\n", command);
   }
   return 0;
}

//int PASCAL WinMain(HINSTANCE hInst, HINSTANCE hPrevInst, LPSTR lpCmd, int nShow)
void main(int argc, char *argv[])
{
 long Errset=0, Erracq=0, Errread=0;
 char command[80];

 hDLL = LoadLibrary("DMPA3.DLL");
 if(hDLL){
     lpSet=(IMPAGETSETTING)GetProcAddress(hDLL,"GetSettingData");
     lpNewStat=(IMPANEWSTATUS)GetProcAddress(hDLL,"GetStatus");
     lpStat=(IMPAGETSTATUS)GetProcAddress(hDLL,"GetStatusData");
     lpRun=(IMPARUNCMD)GetProcAddress(hDLL,"RunCmd");
     lpCnt=(IMPAGETCNT)GetProcAddress(hDLL,"LVGetCnt");
     lpRoi=(IMPAGETROI)GetProcAddress(hDLL,"LVGetRoi");
     lpDat=(IMPAGETDAT)GetProcAddress(hDLL,"LVGetDat");
     lpStr=(IMPAGETSTR)GetProcAddress(hDLL,"LVGetStr");
     lpServ=(IMPASERVEXEC)GetProcAddress(hDLL,"ServExec");
     lpGetDatSet=(IMPAGETDATSET)GetProcAddress(hDLL,"GetDatSetting");
     lpGetMp3Set=(IMPAGETMP3SET)GetProcAddress(hDLL,"GetMP3Setting");
     lpDigInOut=(IMPADIGINOUT)GetProcAddress(hDLL,"DigInOut");
     lpDacOut=(IMPADACOUT)GetProcAddress(hDLL,"DacOut");
     lpStart=(IMPASTART)GetProcAddress(hDLL,"Start");
     lpHalt=(IMPAHALT)GetProcAddress(hDLL,"Halt");
     lpContinue=(IMPACONTINUE)GetProcAddress(hDLL,"Continue");
     lpErase=(IMPAERASE)GetProcAddress(hDLL,"Erase");
 }
 else return;

 // Initialize parameters
 Errset = (*lpNewStat)(0);
 Errset = (*lpStat)(&Status, 0);
 PrintMpaStatus(&Status);

 /*
 (*lpSet)(&Setting, 0);
 PrintSetting(&Setting);
 */

 printf("\nCommands:\n");
 help();

 while(TRUE)
```

```
{
        scanf("%s", command);
        if (run(command)) break;
}

  FreeLibrary(hDLL);

  return;
}
```

5. Using the DMPA3 DLL from Delphi

In the following an example is shown how to control the MPA-3 from a simple console application written in Delphi.

```
program Testmpa3;
{$APPTYPE CONSOLE}
{$X+}
uses
  Windows, sysutils;

const   ST_LIVETIME  =  0;
        ST_DEADTIME  =  1;
        ST_TOTALSUM  =  2;
        ST_ROISUM    =  3;
        ST_TOTALRATE =  4;
        ST_NETSUM    =  5;
        ST_REALTIME  = 0;
        ST_RUNTIME   = 1;
        ST_SINGLESUM = 2;
        ST_COINCSUM  = 3;
        ST_SGLRATE   = 4;
        ST_COIRATE   = 5;

type    SmallIntPointer = ^SmallInt;


{These Type definitions was ported from file struct.h to Delphi}
  AcqStatusTyp = RECORD          // Status information
           val      : Cardinal;           // ADC : max value or MPA: aquisition status
           val1     : Cardinal;           // reserved
                   cnt      : array [0..5] of Double;       // ADC status: Livetime in msec,
                                          // Deadtime in percent, total sum,
                                          // roi sum, total rate, net sum
                                          // MPA: realtime, runtime, singlesum,
                                          // coincsum, sglrate, coirate
           End;
  AcqStatusTypPointer = ^AcqStatusTyp;

  DatSettingTyp = RECORD         // Data settings
           savedata    : Cardinal;        // 1 if auto save after stop
           autoinc     : Cardinal;        // 1 if auto increment filename
           fmt         : Cardinal;        // format type: 0 == ASCII, 1 == binary,
                                          // 2 == GANAAS
           sepfmt      : Cardinal;        // format for seperate spectra
           sephead     : Cardinal;        // seperate Header
           smpts       : Cardinal;        // number of points for smoothing operation
           caluse      : Cardinal;        // 1 for using calibration for shifted spectra summing
           filename    : array [0..255] of Char;            // mpa data filename
           specfile    : array [0..255] of Char;            // seperate spectra filename
           command     : array [0..255] of Char;            // command
           End;

  ReplaySettingTyp = RECORD      // Replay settings
           use         : Cardinal;        // 1 if Replay Mode ON
           modified    : Cardinal;        // 1 if different settings are used
           limit       : Cardinal;        // 0: all, 1: limited time range
           speed       : Cardinal;        // replay speed in units of 100 kB / sec
           timefrom    : Double;          // first time (sec)
```

```
          timeto      : Double;          // last time (sec)
          timepreset  : Double;          // last time - first time
          filename    : array [0..255] of Char;
          End;

  AcqSettingTyp = RECORD          // ADC or spectra Settings
                  range       : Cardinal;         // spectrum length, ADC range
                  prena       : Cardinal;         // bit 0: livetime preset enabled
                                                  // bit 1: ROI preset enabled
                  roimin      : Cardinal;         // lower ROI limit
                  roimax      : Cardinal;         // upper limit: roimin <= channel < roimax
                  nregions    : Cardinal;         // number of regions
                  caluse      : Cardinal;         // bit 0 == 1 if calibration used, higher bits: formula
                  calpoints   : Cardinal;         // number of calibration points
                  param       : Cardinal;         // for MAP and POS: LOWORD=x,
                                                  // HIGHWORD=y (rtime=256, RTC=257)
                  offset      : Cardinal;         // zoomed MAPS: LOWORD: xoffset, HIGHWORD: yoffset
                  xdim        : Cardinal;         // x resolution of maps
                  timesh      : Cardinal;         // bitshift for timespectra
                  active      : Cardinal;         // Spectrum definition words for ADC1..16:
                                                  // active & 0xF  ==0 not used
                                                  //               ==1 single
                                                  //               ==2 coinc with any
                                                  // bit 4..7 in group 1..4 for ADCs
                                                  // Spectrum definition words for calc. spectra:
                                   // active & 0xF  ==3 MAP, ((x-xoffs)>>xsh) x ((y-yoffs)>>ysh)
                                   //               ==0xB TIM, MAP with RTC or rtime as x or y
                                   //                 ((x-xoffs)>>xsh) x ((y-timeoffs)>>timesh)
                                   //               or ((x-timeoffs)>>timesh x ((y-yoffs)>>ysh)
                                   //         bit4=1: x zoomed MAP
                                   //         bit5=1: y zoomed MAP
                                   //               ==4 POS, (y<<xsh) /(x + y)
                                   //               ==5 SUM, (x + y)
                                   //               ==6 DIV, (x<<xsh) / y
                                   // bit 8..11 xsh, bit 12..15 ysh or bit 8..15 xsh
                                   // bit 8..11 xsh, bit 12..15 ysh
                  roipreset   : Double;           // ROI preset value
                  ltpreset    : Double;           // Livetime preset value (MCA)
                  timeoffs    : Double;           // offset for time spectra
                  dwelltime   : Double;           // binsize for time spectra
          End;
  AcqSettingTypPointer = ^AcqSettingTyp;

  ExtAcqSettingTyp = RECORD                       // Settings
                  range       : Cardinal;         // spectrum length, ADC range
                  prena       : Cardinal;         // bit 0: livetime preset enabled
                                                  // bit 1: ROI preset enabled
                  roimin      : Cardinal;         // lower ROI limit
                  roimax      : Cardinal;         // upper limit: roimin <= channel < roimax
                  nregions    : Cardinal;         // number of regions
                  caluse      : Cardinal;         // bit 0 == 1 if calibration used, higher bits: formula
                  calpoints   : Cardinal;         // number of calibration points
                  param       : Cardinal;         // for MAP and POS: LOWORD=x,
                                                  // HIGHWORD=y (rtime=256, RTC=257)
                  offset      : Cardinal;         // zoomed MAPS: LOWORD: xoffset, HIGHWORD: yoffset
                  xdim        : Cardinal;         // x resolution of maps
                  timesh      : Cardinal;         // bitshift for timespectra
                  active      : Cardinal;         // Spectrum definition words for ADC1..16:
                                                  // active & 0xF  ==0 not used
                                                  //               ==1 single
                                                  //               ==2 coinc with any
                                                  // bit 4..7 in group 1..4 for ADCs
```

```
                                          // Spectrum definition words for calc. spectra:
                                          // active & 0xF  ==3 MAP, ((x-xoffs)>>xsh) x ((y-yoffs)>>ysh)
                                          //             ==0xB TIM, MAP with RTC or rtime as x or y
                                          //               ((x-xoffs)>>xsh) x ((y-timeoffs)>>timesh)
                                          //            or ((x-timeoffs)>>timesh x ((y-yoffs)>>ysh)
                                          //        bit4=1: x zoomed MAP
                                          //        bit5=1: y zoomed MAP
                                          //             ==4 POS, (y<<xsh) /(x + y)
                                          //             ==5 SUM, (x + y)
                                          //             ==6 DIV, (x<<xsh) / y
                                          // bit 8..11 xsh, bit 12..15 ysh or bit 8..15 xsh
                                          // bit 8..11 xsh, bit 12..15 ysh
                roipreset   : Double;     // ROI preset value
                ltpreset    : Double;     // Livetime preset value (MCA)
                timeoffs    : Double;     // offset for time spectra
                dwelltime   : Double;     // binsize for time spectra
                vtype       : Cardinal;   // 0=single, 1=MAP, 2=ISO...
                ydim        : Cardinal;   // y resolution of maps
                reserved    : array [0..12] of LongInt;
        End;
ExtAcqSettingTypPointer = ^ExtAcqSettingTyp;

AcqDataTyp = RECORD
                s0          : Array of LongInt;      // pointer to spectrum
                region      : Array of Cardinal;     // pointer to regions
                comment0    : Array of Char;         // pointer to strings
                cnt         : Array of Double;       // pointer to counters
                hs0         : Integer;
                hrg         : Integer;
                hcm         : Integer;
                hct         : Integer;
                End;

AcqMp3Typ = RECORD
        sen     : Cardinal;     // Start Enable Register
                                // 1 in Bit 0(..15) means ADC 1A(..4D)
                                // starts coincidence window
        coi     : Cardinal;     // Coincidence Control Register
                                // 1 in Bit 0(..15) means ADC 1A(..4D)
                                // is in coincidence mode,
                                // otherwise in single mode
        ctm     : Cardinal;     // Coinc. Time in units of 50 ns
        dtm     : Cardinal;     // Data Ready Timeout in units of 50 ns
        tct     : Cardinal;     // Time Stamp Control Register
                                // Bit 0..1: Count Source:
                                //          0=20MHz,    1=AUX2edge,
                                //          2=AUX1edge, 3=REJedge
                                // Bit 2..3: Counter Enable:
                                //          0=ON,       1=AUX2&ON,
                                //    2=AUX1&ON,  3=REJ&ON
                                // Bit 4..7: Time Capture:
                                //                  0=OR-ed DEAD edge(DOR),
                                //                  1=DOR+AUX2,
                                //                  2=DOR+AUX1,
                                //                  3=DOR+REJ,
                                //                  4=coinc_start
                                //                  5=coinc_end
                                //                  6=coinc_ok
                                //                  7=Software
                                // Bit8..10: Timer Load:
                                //                  0=Software, 1=AUX2
                                //                  2=AUX1, 3=REJ,
                                //              4=Wrap around
                                // Bit 11: timer 'preset reached' stops acquisition
                                // Bit 12: timer 'preset reached' in PCI status
                                // Bit 13: enable preset
        tp0     : Cardinal;     // Timer Preset 0 Register  0..65535
        tp1     : Cardinal;     // Timer Preset 1 Register
        tp2     : Cardinal;     // Timer Preset 2 Register
                                // The timer preset is
                                // (tp2 * 65536 + tp1) * 65536 + tp0
        aui     : Cardinal;     // Aux In Control
                                // Bit 0: AUX2 polarity, 1=active low
                                // Bit 1: AUX1 polarity
                                // Bit 2: AUX2 start coinc.window enable
                                // Bit 3: AUX1 start enable
                                // Bit 4: AUX2 coinc. mode
                                // Bit 5: AUX1 coinc. mode
                                // Bit 12: REJ coinc. mode
                                // Bit 13: REJ polarity
                                // Bit 15: Reject mode:
                                //         0=instant, 1=at end of coinc.time
        auo      : Cardinal;    // Aux Out Control
                                // Bit 0..3: AUX2 output:
                                //              0=coinc_start,  1=coinc_run,
                                //              2=coinc_active, 3=coinc_ok,
                                //              4=dead_store,   5=enca (ON),
                                //              6=1, 7=0
                                // Bit 4..7: AUX1 output:
                                //              0=coinc_start,  1=coinc_run,
                                //              2=coinc_active, 3=coinc_ok,
                                //              4=dead_store,   5=enca (ON),
                                //              6=1, 7=0,
                                //              8=preset reached, 9=timer load
                                // Bit 8:  AUX2 output enable
                                // Bit 9:  AUX1 output enable
                                // Bit 10: AUX2 mirror GO line (Bit0..3=7)
                                // Bit 11: AUX1 mirror GO line (Bit4..7=7)
        dor     : Cardinal;     // Timer Dead OR
                                // 1 in Bit 0 (..15) for ADC 1A(..4D)
                                // to capture RTC time, see tct description
        bk0      : Cardinal;    // Block Routing Control 0
                                // Allows to route more than 16 FMP bus subadresses
                                // default=0 for 1 to 1 routing of 16 ADC interfaces
        bk1      : Cardinal;    // Block Routing Control 1, default=0
        rtcuse    : Cardinal;   // Bit 0: Timestamp in datastream
                                // Bit 1: Halt when preset reached
        dac     : Cardinal;     // Bit 0..7: DAC output value (8 bit)
        diguse    : Cardinal;   // Usage of DIG I/O
        // Bit 0:  DIG I/O bit 7 output status
        // Bit 1:  Invert Polarity
        // Bit 2:  Input bit 6 Trigger System
        // Bit 6:  Output digval and increment digval after stop
        // Bit 7:  Input Mode: =0 Resistive, =1 Tristate
        // Bit 8:  Input Mode: =0 Pullup, =1 Pulldown
        digval    : Cardinal;   // DIG I/O Output value
        rtprena   : Cardinal;   // Realtime Preset enable
        rtpreset  : Double;     // Realtime Preset
        End;


AcqDefTyp = RECORD
                nDevices   : Cardinal;   // Number of connected ADC Interfaces = max. 16
                nDisplays  : Cardinal;   // Number of histograms = nDevices + Positions + Maps
```

```
                    nSystems    : Cardinal;       // Number of independent systems = 1
                    bRemote    : Cardinal;       // 1 if server controlled by MPANT
                    auxsys      : Cardinal;       // System definition words for AUXx
                                                  // auxsys & 0xFF ==0 not used
                                                  //              ==2 coinc with any
                                                  // bit 4..7 in group 1..4
           sys0        : array [0..15] of Cardinal;
                                                  // System definition words for ADC1..16:
                                                  // see active definition in ACQSETTING
           sys1        : array [0..15] of Cardinal;
                                                  // ADC in System (always 1)
           End;
  AcqDefTypPointer = ^AcqDefTyp;


  LpGet = function (var Setting : AcqSettingTyp;       // Get Settings stored in the DLL
               nDisplay : Cardinal) : LongInt; stdcall;

  LpStat = function (var Status  : AcqSTatusTyp;       // Get Status stored in the DLL
               nDisplay : Cardinal) : LongInt; stdcall;

  LpRun  = procedure (nDisplay : LongInt;              // Executes command
                          Cmd    : PChar) ; stdcall;

  LpCnt  = function (var cntp    : Double;              // Copies Cnt numbers to an array
               nDisplay : Cardinal) : LongInt; stdcall;

  LpRoi  = function (var roip    : Cardinal;            // Copies the ROI boundaries to an array
               nDisplay : Cardinal) : LongInt; stdcall;

  LpDat  = function (var datp    : LongInt;             // Copies the spectrum to an array
               nDisplay : Cardinal) : LongInt; stdcall;

  LpStr  = function (var strp    : Char;                // Copies strings to an array
              nDisplay : Cardinal) : LongInt; stdcall;

  LpServ = function (ClientWnd : Cardinal) : Cardinal; stdcall;            // Register Client MCDWIN.EXE

  LpNewStat = function (nDevice : Cardinal) : LongInt; stdcall;           // Request actual Status from Server

  LpGetMp3 = function (var Defmp3 : AcqMp3Typ) : LongInt; stdcall; // Get MPA3 Settings from DLL

  LpGetDatSet = function (var Defdat : DatSettingTyp) : LongInt; stdcall;
                                                       // Get Data Format Definition from DLL

  LpDigInOut = function (value  :  Cardinal;            // controls Dig I/0, returns digin
                    enable :  Cardinal) : LongInt; stdcall;

  LpDacOut = function (value : Cardinal)  : LongInt; stdcall;
                                                       // output Dac value as analogue voltage


var  Handle          : Integer;
    TGet : LpGet;
    TStat             : LpStat;
    TRun: LpRun;
    TCnt : LpCnt;
    TRoi       : LpRoi;
    TDat       : LpDat;
    TStr       : LpStr;
    TServ      : LpServ;
    TNewStat   : LpNewStat;
    TMp3       : LpGetMp3;
    TDatset    : LpGetDatSet;
    TDiginout  : LpDigInOut;
    Setting    : AcqSettingTyp;
    Mp3set     : AcqMp3Typ;
    {Data              : AcqDataTyp;
    Def       : AcqDefTyp;}
    Status     : AcqStatusTyp;
    Adc        : Cardinal;
    cmd        : String;
    Err       : LongInt;
    Spec       : Array[0..8191] of LongInt;

procedure PrintMpaStatus(var stat: AcqStatusTyp);
begin
  with stat do
  begin
    if val <> 0 then
      writeln('ON')
    else
      writeln('OFF');
    writeln('realtime= ', cnt[ST_REALTIME]);
    writeln('runtime=  ', cnt[ST_RUNTIME]);
    writeln('single_ev= ', cnt[ST_SINGLESUM]);
    writeln('coinc_ev= ', cnt[ST_COINCSUM]);
    writeln('sglrate=  ', cnt[ST_SGLRATE]);
    writeln('coirate=  ', cnt[ST_COIRATE]);
  end;
end;

procedure PrintStatus(var stat: AcqStatusTyp);
begin
  with stat do
  begin
    writeln('livetime=  ', cnt[ST_LIVETIME]);
    writeln('deadtime%= ', cnt[ST_DEADTIME]);
    writeln('totalsum=  ', cnt[ST_TOTALSUM]);
    writeln('roisum=    ', cnt[ST_ROISUM]);
    writeln('netsum=    ', cnt[ST_NETSUM]);
    writeln('totalrate= ', cnt[ST_TOTALRATE]);
  end;
end;

procedure PrintDatSetting(var datsett: DatSettingTyp);
begin
  with datsett do
  begin
    writeln('savedata= ', savedata);
    writeln('autoinc=  ', autoinc);
    writeln('fmt=      ', fmt);
    writeln('sepfmt=   ', sepfmt);
    writeln('sephead=  ', sephead);
    writeln('filename= ', String(filename));
  end;
end;

procedure PrintMp3Setting(var mpsett: AcqMp3Typ);
begin
  with mpsett do
  begin
    writeln('rtcuse=   ', rtcuse);
    writeln('dac=      ', dac);
    writeln('diguse=   ', diguse);
    writeln('digval=   ', digval);
    writeln('rtprena=  ', rtprena);
    writeln('rtpreset= ', rtpreset);
  end;
end;

procedure PrintSetting(var sett: AcqSettingTyp);
begin
  with sett do
  begin
    writeln('range=    ', range);
    writeln('prena=    ', prena);
    writeln('roimin=   ', roimin);
    writeln('roimax=   ', roimax);
    writeln('nregions= ', nregions);
    writeln('caluse=   ', caluse);
    writeln('calpoints=', calpoints);
    writeln('param=    ', param);
    writeln('offset=   ', offset);
    writeln('xdim=     ', xdim);
    writeln('timesh=   ', timesh);
    writeln('active=   ', active);
    writeln('roipreset=', roipreset);
    writeln('ltpreset= ', ltpreset);
    writeln('timeoffs= ', timeoffs);
    writeln('dwelltime=', dwelltime);
  end;
end;

procedure PrintDat(len: Cardinal);
  var i: Integer;
begin
  writeln('first 30 of ', len, ' datapoints:');
  for i:= 0 to 29 do
    writeln(Spec[i]);
end;

procedure help;
begin
  writeln('Commands:');
  writeln('Q     Quit');
  writeln('H     Help');
  writeln('S     Status');
  writeln('G     Settings');
  writeln('ADC=x Switch to ADC #x (0=MPA)');
  writeln('D     Get Data');
  writeln('... more see command language in MPANT Help');
end;

function run(command : String) : LongInt;
begin
  run := 0;
  if command = 'H' then
  help
  else if command = 'Q' then
  begin
    run := 1;
  end
  else if command = 'S' then
  begin
    if @TStat <> nil then
    begin
      Err := TStat(Status, Adc);
      PrintStatus(Status);
    end;
  end
  else if command = 'G' then
  begin
    if @TGet <> nil then
    begin
      if Adc > 0 then
      begin
        Err := TGet(Setting, Adc-1);
        PrintSetting(Setting);
      end
      else
      begin
        Err := TMp3(Mp3set);
        PrintMp3Setting(Mp3set);
      end;
    end;
  end
  else if AnsiPos('ADC=',command) = 1 then
  begin
     Val(String(PChar(command)+4), Adc, Err);
     TRun(0, PChar(command));
  end
  else if command = 'D' then
  begin
    if @TGet <> nil then
    begin
      if Adc > 0 then
      begin
        Err := TGet(Setting, Adc-1);
        if @TDat <> nil then
        begin
          Err := TDat(Spec[0], Adc-1);
          PrintDat(Setting.range);
        end;
      end;
    end;
  end
  else
  begin
    if @TRun <> nil then
    begin
      TRun(0, PChar(command));
      writeln(command);
    end;
  end;
end;

begin
  SetLength(cmd, 100);
  Adc := 0;
  Handle := LoadLibrary('dmpa3.dll');
  if Handle <> 0 then
  begin
    @TGet := GetProcAddress(Handle, 'GetSettingData');
    @TStat := GetProcAddress(Handle, 'GetStatusData');
    @TRun := GetProcAddress(Handle, 'RunCmd');
    @TCnt := GetProcAddress(Handle, 'LVGetCnt');
    @TRoi := GetProcAddress(Handle, 'LVGetRoi');
    @TDat := GetProcAddress(Handle, 'LVGetDat');
    @TStr := GetProcAddress(Handle, 'LVGetStr');
    @TServ := GetProcAddress(Handle, 'ServExec');
    @TNewStat := GetProcAddress(Handle, 'GetStatus');
    @TDatset := GetProcAddress(Handle, 'GetDatSetting');
    @TMp3 := GetProcAddress(Handle, 'GetMP3Setting');
    @TDiginout := GetProcAddress(Handle, 'DigInOut');

    if @TNewStat <> nil then
      Err := TNewStat(0);

 {   if @TStat <> nil then
    begin
      Err := TStat(Status, 0);
      PrintStatus(Status);
    end;

    if @TGet <> nil then
    begin
      Err := TGet(Setting, 0);
      PrintSetting(Setting);
    end; }
    help;

    repeat
      readln(cmd);
    until run(cmd) <> 0;

    FreeLibrary(Handle);
  end;
end.
```

APPENDIX: The DMPA3 DLL

The Dynamic Link Library DMPA3.DLL provides an interface to the server program MPA3.EXE that is used by the MPANT software, but can also be used by any Windows program. Custom DLL functions allow user-defined calculated parameter spectra. In the following this DLL is described in detail including the complete sourcecode.

A.1 The Structures

In struct.h some important structures are defined. A structure of type ACQSTATUS contains parameters describing the status of an acquisition. There is an array of these structures stored in the DLL, status[0] contains general mpa status data, and status[1]..status[16] ADC status data.

```
#define  WINDOWSNT
#undef  WINDOWS95
#undef  WINDOWS31

#ifdef WINDOWS31

#define GET_WM_COMMAND_ID(w)  w
#define GET_WM_COMMAND_CMD(w,l) HIWORD(l)
#define GET_WM_COMMAND_HWND(l) LOWORD(l)
#define GET_WM_SCRHWND(l) HIWORD(l)
#define GET_WM_SCROLLPOS(w,l) LOWORD(l)
#define FIND_WINDOW(a,b) FindWindow(b,a)
#define HUGE huge
#define USHORT unsigned short
#define SetForegroundWindow(w)
#define APIENTRY FAR PASCAL

#else

#define GET_WM_COMMAND_ID(w)  LOWORD(w)
#define GET_WM_COMMAND_CMD(w,l) HIWORD(w)
#define GET_WM_COMMAND_HWND(l) l
#define GET_WM_SCRHWND(l) l
#define GET_WM_SCROLLPOS(w,l) (short)HIWORD(w)
#define FIND_WINDOW(a,b) FindWindow(a,b)
#define HUGE
#define _fmemcpy memcpy
#define _fstrcpy strcpy

#endif


#define ST_LIVETIME   0
#define ST_DEADTIME   1
#define ST_TOTALSUM   2
#define ST_ROISUM     3
#define ST_TOTALRATE  4
#define ST_NETSUM     5

typedef struct{
  unsigned long val;        // Maximum value in spectrum
  unsigned long val1;       // reserved
  double cnt[6];            // ADC status: Livetime in msec, Deadtime in percent,
                            // total sum, roi sum, total rate, net sum
} ACQSTATUS;

#define ST_REALTIME   0
```

```
#define ST_RUNTIME    1
#define ST_SINGLESUM  2
#define ST_COINCSUM   3
#define ST_SGLRATE    4
#define ST_COIRATE    5

                              // MPA status: real time in msec, elapsed computer-runtime, total single events
                              // total coinc events, rate of single events, rate of coinc events
                              // val=aquisition status: 0= HALT, 1= ON
```

DATSETTING is a structure type containing data format settings.

```
typedef struct {
long savedata;                // bit0=1 if auto save after stop
                              // bit1 write listfile
                              // bit2 no histogramming
                              // bit3 reduce timer data by 10^n
                              // bit4..5 n-1
                              // bit6 drop 'zero events'
  long autoinc;               // 1 if auto increment filename
  long fmt;                   // format type: 0 == ASCII, 1 == binary,
                              // 2 == GANAAS, 3 == CSV=MPAWIN ASC(2D)
  long mpafmt;                // format used in mpa datafiles
  long sephead;               // seperate Header
  long smpts;                 // number of points for smoothing operation
  long caluse;                // 1 for using calibration for shifted spectra summing
  char filename[256];         // mpa data filename
  char specfile[256];         // seperate spectra filename
  char command[256];          // command
} DATSETTING;
```

REPLAYSETTING is a structure type containing Replay settings.

```
typedef struct {
  long use;                   // 1 if Replay Mode ON
  long modified;              // 1 if different settings are used
  long limit;                 // 0: all,
                              // 1: limited time range
  long speed;                 // replay speed in units of 100 kB / sec
  double timefrom;            // first time (sec)
  double timeto;              // last time (sec)
  double timepreset;          // last time - first time
  char filename[256];
} REPLAYSETTING;
```

ACQSETTING is a structure type containing all the spectra settings .

```
typedef struct{
  long range;                 // spectrum length
  long prena;                 // bit 0: livetime preset enabled
                              // bit 1: ROI preset enabled
  long roimin;                // lower ROI limit
  long roimax;                // upper limit: roimin <= channel < roimax
  long nregions;              // number of regions
  long caluse;                // bit0: 1 if calibration used, higher bits: formula
  long calpoints;             // number of calibration points
  long param;                 // for MAP and POS: LOWORD=x, HIGHWORD=y (rtime=256, RTC=257)
  long offset;                // zoomed MAPS: LOWORD: xoffset, HIGHWORD: yoffset
  long xdim;                  // x resolution of maps
  long timesh;                // bitshift for timespectra
  long active;       // Spectrum definition words for ADC1..16:
```

```
                    // active & 0xF  ==0 not used
                    //             ==1 single
                    //             ==2 coinc with any
                    // bit 4..7 in group 1..4 for ADCs
                    // Spectrum definition words for calc. spectra:
                    // active & 0xF  ==3 MAP, ((x-xoffs)>>xsh) x ((y-yoffs)>>ysh)
                    //             ==0xB TIM, MAP with RTC or rtime as x or y
                    //               ((x-xoffs)>>xsh) x ((y-timeoffs)>>timesh)
                    //             or ((x-timeoffs)>>timesh x ((y-yoffs)>>ysh)
                    //        bit4=1: x zoomed MAP
                    //        bit5=1: y zoomed MAP
                    //             ==4 POS, (y<<xsh) /(x + y)
                    //             ==5 SUM, (x + y)>>xsh
                    //             ==6 DIV, (x<<xsh) / y
                    //             ==9 DLL, fDLL(x,y,z)
                    //             ==10 HISTORY, x
                    // bit 8..11 xsh, bit 12..15 ysh or bit 8..15 xsh
                    // HIWORD(active) = 1+condition no. (0=no condition)
  double roipreset;          // ROI preset value
  double ltpreset;           // livetime preset value
  double timeoffs;           // offset for time spectra
  double dwelltime;          // binsize for time spectra
} ACQSETTING;
```

ACQDATA is a structure type containing pointers to the data belonging to a measurement. The data is stored in a named memory-mapped file (see DLL source).

```
typedef struct{
  unsigned long HUGE *s0;           // pointer to spectrum
  unsigned long far *region;        // pointer to regions
  unsigned char far *comment0;      // pointer to strings
  double far *cnt;                  // pointer to counters
  HANDLE hs0;
  HANDLE hrg;
  HANDLE hcm;
  HANDLE hct;
} ACQDATA;
```

Data[nDisplay].s0 points to a block memory of unsigned long numbers containing the spectra data.

Data[nDisplay].region points to a block of 256 unsigned long numbers containing the Roi (Region of interest) boundaries as defined in the MPANT program. The first Roi is:
Data[nDisplay].region[0] <= x < Data[nDisplay].region[1], the second Data[nDisplay].region[2] <= x < Data[nDisplay].region[3] and so on, 128 Rois are possible. These Rois have nothing to do with the special Roi defined in the ACQSETTING structure for the Roi Preset.

Data[nDisplay].comment0 points to a block of 1024 bytes containing the strings.
Data[nDisplay].comment0[0] is the first byte of the 0. commentline,
Data[nDisplay].comment0[60] is the first byte of the 1. commentline,
Data[nDisplay].comment0[120] is the first byte of the 2. commentline,
Data[nDisplay].comment0[180] is the first byte of the 3. commentline,
Data[nDisplay].comment0[240] is the first byte of the 4. commentline,
Data[nDisplay].comment0[300] is the first byte of the 5. commentline,
Data[nDisplay].comment0[360] is the first byte of the 6. commentline,
Data[nDisplay].comment0[420] is the first byte of the 7. commentline,

Data[nDisplay].comment0[480] is the first byte of the 8. commentline,
Data[nDisplay].comment0[540] is the first byte of the 9. commentline,
Data[nDisplay].comment0[600] is the first byte of the 10. commentline,
Data[nDisplay].comment0[660] is the first byte of the data filename,
Data[nDisplay].comment0[760] is the first byte of the calibration unit name,
Data[nDisplay].comment0[800] is the first byte of the commandstring.
Data[nDisplay].comment0[880] is the first byte of the 11. commentline,
Data[nDisplay].comment0[960] is the first byte of the 12. commentline

Data[nDisplay].cnt points to a block of 448 double numbers containing:
Data[nDisplay].cnt[0] = Realtime
Data[nDisplay].cnt[1] = Totalsum
Data[nDisplay].cnt[2] = ROIsum
Data[nDisplay].cnt[3] = Totalrate
Data[nDisplay].cnt[4] = Net ROIsum
Data[nDisplay].cnt[5] = Livetime
Data[nDisplay].cnt[6] = Deadtime (%)
Data[nDisplay].cnt[11] = c0 Calibration parameter
Data[nDisplay].cnt[12] = c1 Calibration parameter
Data[nDisplay].cnt[13] = c2 Calibration parameter
Data[nDisplay].cnt[14] = c3 Calibration parameter
Data[nDisplay].cnt[19] = Channel number of first calibration Point
Data[nDisplay].cnt[35] = Energy value at first calibration Point
Data[nDisplay].cnt[20] = Channel number of 2. calibration Point
Data[nDisplay].cnt[36] = Energy value at 2. calibration Point...

Data[nDisplay].cnt[64..191] = Energy value for calibration peak in ROI 0..127
Data[nDisplay].cnt[192] = ROI Sum in ROI 0 (actualized by MPANT when selected in any spectra display)
Data[nDisplay].cnt[193] = ROI Net Sum in ROI 0 ...
Data[nDisplay].cnt[447] = ROI Net Sum in ROI 127

ACQMP3 is a structure type describing special MPA-3 hardware settings.

```
typedef struct {
  int sen;            // Start Enable Register
                      // 1 in Bit 0(..15) means ADC 1A(..4D)
                      // starts coincidence window
  int coi;            // Coincidence Control Register
                      // 1 in Bit 0(..15) means ADC 1A(..4D)
                      // is in coincidence mode,
                      // otherwise in single mode
  int ctm;            // Coinc. Time in units of 50 ns
  int dtm;            // Data Ready Timeout
  int tct;            // Time Stamp Control Register
                      // Bit 0..1: Count Source:
                      //          0=20MHz,    1=AUX2edge,
                      //          2=AUX1edge, 3=REJedge
                      // Bit 2..3: Counter Enable:
                      //          0=ON,       1=AUX2&ON,
                      //    2=AUX1&ON,  3=REJ&ON
                      // Bit 4..7: Time Capture:
                      //                0=OR-ed DEAD edge(DOR),
                      //                1=DOR+AUX2,
                      //                2=DOR+AUX1,
                      //                3=DOR+REJ,
                      //                4=coinc_start
                      //                5=coinc_end
                      //                6=coinc_ok
                      //                7=Software
```

```
                    // Bit8..10: Timer Load:
                    //              0=Software, 1=AUX2
                    //              2=AUX1, 3=REJ,
                    //              4=Wrap around
                    // Bit 11: timer 'preset reached' stops acquisition
                    // Bit 12: timer 'preset reached' in PCI status
                    // Bit 13: enable preset
  int tp0;          // Timer Preset 0 Register  0..65535
  int tp1;          // Timer Preset 1 Register
  int tp2;          // Timer Preset 2 Register
                    // The timer preset is
                    // (tp2 * 65536 + tp1) * 65536 + tp0
  int aui;          // Aux In Control
                    // Bit 0: AUX2 polarity, 1=active low
                    // Bit 1: AUX1 polarity
                    // Bit 2: AUX2 start coinc.window enable
                    // Bit 3: AUX1 start enable
                    // Bit 4: AUX2 coinc. mode
                    // Bit 5: AUX1 coinc. mode
                    // Bit 12: REJ coinc. mode
                    // Bit 13: REJ polarity
                    // Bit 15: Reject mode:
                    //       0=instant, 1=at end of coinc.time
  int auo;          // Aux Out Control
                    // Bit 0..3: AUX2 output:
                    //              0=coinc_start,  1=coinc_run,
                    //              2=coinc_active, 3=coinc_ok,
                    //              4=dead_store,   5=enca (ON),
                    //              6=1, 7=0
                    // Bit 4..7: AUX1 output:
                    //              0=coinc_start,  1=coinc_run,
                    //              2=coinc_active, 3=coinc_ok,
                    //              4=dead_store,   5=enca (ON),
                    //              6=1, 7=0,
                    //              8=preset reached, 9=timer load
                    // Bit 8:  AUX2 output enable
                    // Bit 9:  AUX1 output enable
                    // Bit 10: AUX2 mirror GO line (Bit0..3=7)
                    // Bit 11: AUX1 mirror GO line (Bit4..7=7)
  int dor;// Timer Dead OR
                    // 1 in Bit 0 (..15) for ADC 1A(..4D)
                    // to capture RTC time, see tct description
  int bk0;          // Block Routing Control 0
                    // Allows to route more than 16 FMP bus subadresses
                    // default=0 for 1 to 1 routing of 16 ADC interfaces
  int bk1;          // Block Routing Control 1, default=0
  int rtcuse;       // Bit 0: Timestamp in datastream
                    // Bit 1: Halt when preset reached
  int dac;          // Bit 0..7: DAC output value (8 bit)
  int diguse;       // Usage of DIG I/O
                    // Bit 0:  DIG I/O bit 7 output status
                    // Bit 1:  Invert Polarity
                    // Bit 2:  Input bit 6 Trigger System
                    // Bit 6:  Output digval and increment digval after stop
                    // Bit 7:  Input Mode: =0 Resistive, =1 Tristate
                    // Bit 8:  Input Mode: =0 Pullup, =1 Pulldown
  int digval ;      // DIG I/O Output value
  int rtprena;      // Realtime Preset enable
  double rtpreset;  // Realtime Preset
} ACQMP3;
```

ACQDEF is a structure type describing the system definition.

```
typedef struct {
  int nDevices;            // Number of connected ADC Interfaces = max. 16
  int nDisplays;           // Number of histograms = nDevices + Positions + Maps
  int nSystems;            // Number of independent systems = 1
  int bRemote;             // 1 if server controlled by MPANT
  int auxsys;              // System definition words for AUXx
                   // auxsys & 0xFF ==0 not used
                   //              ==2 coinc with any
                           // bit 4..7 in group 1..4
  int sys0[16];            // System definition words for ADC1..16:
                   // see active definition in ACQSETTING
  int sys1[16];            // ADC in System (always 1)
} ACQDEF;
```

A.2 The Library Functions

In the header file dmpa3.h all functions are declared. The arguments named nDevice, nDisplay pertain to the ADC number and is zero for the first ADC.

```
#ifdef __cplusplus
extern "C"
{
#endif

#include "struct.h"

#define MAXCNT              448
#define MAXDEV              17
#define MAXDSP              50

#define ID_SAVE             103
#define ID_CONTINUE         106
#define ID_START            109
#define ID_BREAK            137
#define ID_NEWSETTING       139
#define ID_GETSTATUS        141
#define ID_SAVEFILE         151
#define ID_ERASE            154
#define ID_LOADFILE         155
#define ID_NEWDATA          160
#define ID_HARDWDLG         161
#define ID_SAVEFILE2        194
#define ID_LOADFILE2        203
#define ID_SAVEFILE3        217
#define ID_LOADFILE3        219
#define ID_SAVEFILE4        223
#define ID_LOADFILE4        225
#define ID_LOADFILE5        226
#define ID_LOADFILE6        227
#define ID_LOADFILE7        228
#define ID_LOADFILE8        229
#define ID_SAVEFILE5        230
#define ID_SAVEFILE6        231
#define ID_SAVEFILE7        232
#define ID_SAVEFILE8        233
#define ID_SUMFILE          234
#define ID_SUMFILE2         235
#define ID_SUMFILE3         236
#define ID_SUMFILE4         237
#define ID_SUMFILE5         238
#define ID_SUMFILE6         239
#define ID_SUMFILE7         240
#define ID_SUMFILE8         241
#define ID_SUBTRACT         289
#define ID_SMOOTH           290
#define ID_SUBTRACT2        296
#define ID_SMOOTH2          297
#define ID_SUBTRACT3        298
#define ID_SMOOTH3          299
#define ID_SUBTRACT4        300
#define ID_SMOOTH4          301
#define ID_SUBTRACT5        302
#define ID_SMOOTH5          303
#define ID_SUBTRACT6        304
#define ID_SMOOTH6          305
```

```
#define ID_SUBTRACT7            306
#define ID_SMOOTH7              307
#define ID_SUBTRACT8            308
#define ID_SMOOTH8              309
#define ID_COMBDLG              401
#define ID_DATADLG              402
#define ID_MAPLSTDLG            403
#define ID_REPLDLG              404
#define ID_ERASE2               1108
#define ID_ERASE3               1109
#define ID_ERASE4               1110
#define ID_ERASEFILE2           1111
#define ID_ERASEFILE3           1112
#define ID_ERASEFILE4           1113
#define ID_START2               1114
#define ID_BREAK2               1115
#define ID_CONTINUE2            1116
#define ID_START3               1117
#define ID_BREAK3               1118
#define ID_CONTINUE3            1119
#define ID_START4               1120
#define ID_BREAK4               1121
#define ID_CONTINUE4            1122
#define ID_RUNCMD               1123
#define ID_RUNCMD2              1124
#define ID_RUNCMD3              1125
#define ID_RUNCMD4              1126
#define ID_RUNCMD5              1127
#define ID_RUNCMD6              1128
#define ID_RUNCMD7              1129
#define ID_RUNCMD8              1130
#define ID_ERASEFILE5           1131
#define ID_ERASEFILE6           1132
#define ID_ERASEFILE7           1133
#define ID_ERASEFILE8           1134
#define ID_DIGINOUT             1137
#define ID_DACOUT               1138
#define ID_GETROIINDEX          1139
#define ID_GETROISUM            1141
#define ID_DELETEROI            1301
#define ID_SELECTROI            1302

/*** FUNCTION PROTOTYPES (do not change) ***/
BOOL APIENTRY DllMain(HANDLE hInst, DWORD ul_reason_being_called, LPVOID lpReserved);

VOID APIENTRY StoreSettingData(ACQSETTING *Setting, int nDisplay);
                              // Stores Settings into the DLL
int APIENTRY GetSettingData(ACQSETTING *Setting, int nDisplay);
                              // Get Settings stored in the DLL
VOID APIENTRY StoreStatusData(ACQSTATUS *Status, int nDev);
                              // Store the Status into the DLL
int APIENTRY GetStatusData(ACQSTATUS *Status, int nDev);
                              // Get the Status
VOID APIENTRY Start(int nSystem);       // Start
VOID APIENTRY Halt(int nSystem);        // Halt
VOID APIENTRY Continue(int nSystem);            // Continue
VOID APIENTRY NewSetting(int nDisplay);         // Indicate new Settings to Server
UINT APIENTRY ServExec(HWND ClientWnd);   // Execute the Server MPA3.EXE
VOID APIENTRY StoreData(ACQDATA *Data, int nDisplay);
                              // Stores Data pointers into the DLL
int APIENTRY GetData(ACQDATA *Data, int nDisplay);
                              // Get Data pointers
```

```
long APIENTRY GetSpec(long i, int nDisplay);
                                   // Get a spectrum value
VOID APIENTRY SaveSetting(void);        // Save Settings
int APIENTRY GetStatus(int nDev);                 // Request actual Status from Server
VOID APIENTRY Erase(int nSystem);                 // Erase spectrum
VOID APIENTRY SaveData(int nDisplay, int all);   // Saves data
VOID APIENTRY GetBlock(long *hist, int start, int end, int step,
  int nDisplay);                   // Get a block of spectrum data
VOID APIENTRY StoreDefData(ACQDEF *Def);
                                   // Store System Definition into DLL
int APIENTRY GetDefData(ACQDEF *Def);             // Get System Definition
VOID APIENTRY LoadData(int nDisplay, int all);   // Loads data
VOID APIENTRY NewData(void);            // Indicate new ROI or string Data
VOID APIENTRY HardwareDlg(int item);              // item=0: Calls the Settings dialog
                                   // 1: data dialog, 2: system dialog
VOID APIENTRY UnregisterClient(void);             // Clears remote mode from MPANT
VOID APIENTRY DestroyClient(void);                // Close MPANT
UINT APIENTRY ClientExec(HWND ServerWnd);
                                   // Execute the Client MPANT.EXE
int APIENTRY LVGetDat(unsigned long HUGE *datp, int nDisplay);
                                   // Copies the spectrum to an array
VOID APIENTRY RunCmd(int nDisplay, LPSTR Cmd);
                                   // Executes command
VOID APIENTRY AddData(int nDisplay, int all);    // Adds data
VOID APIENTRY SubData(int nDisplay, int all);    // Subtracts data
VOID APIENTRY Smooth(int nDisplay);               // Smooth data
int APIENTRY LVGetRoi(unsigned long FAR *roip, int nDisplay);
                                   // Copies the ROI boundaries to an array
int APIENTRY LVGetOneRoi(int nDisplay, int roinum, long *x1, long *x2);
                                   // Get one ROI boundary
int APIENTRY LVGetCnt(double *cntp, int nDisplay);
                                   // Copies Cnt numbers to an array
int APIENTRY LVGetOneCnt(double *cntp, int nDisplay, int cntnum);
                                   // Get one Cnt number
int APIENTRY LVGetStr(char *strp, int nDisplay);
                                   // Copies strings to an array
VOID APIENTRY StoreMP3Setting(ACQMP3 *Defmp3);
                                   // Store MP3 System Definition into DLL
int APIENTRY GetMP3Setting(ACQMP3 *Defmp3);
                                   // Get MP3 System Definition from DLL
VOID APIENTRY StoreDatSetting(DATSETTING *Defdat);
                                   // Store Data Format Definition into DLL
int APIENTRY GetDatSetting(DATSETTING *Defdat);
                                   // Get Data Format Definition from DLL
int APIENTRY GetDatPtr(int nDisplay, long *xmax, long *ymax, LPSTR *pt);
                                   // Get a temporary pointer to spectra data
                                   // xmax and ymax return the spectra dimensions
int APIENTRY ReleaseDatPtr(void);
                                   // Release temporary pointer to spectra data
long APIENTRY GetSVal(int DspID, long xval);
                             // Get special display data like projections from MPANT
int APIENTRY DigInOut(int value, int enable);     // controls Dig I/0 ,returns digin
int APIENTRY DacOut(int value);         // output Dac value as analogue voltage
long APIENTRY GetRoiIndex(LPSTR roiname);
    // get a unique index to address ROIs from named ROI's.
    // rectangular or 1D ROIs:
    //    LOWORD is the spectra number,
    //    HIWORD is the ROI number (1,2,..)
    // polygonal ROIs:
    //    LOWORD is an entry number
    //    HIWORD is the roiid = 100 * spectra_number + ROI_number
    // returns 0 if not found.
```

```
int APIENTRY DeleteRoi(DWORD roiindex);
     // deletes ROI with given index
int APIENTRY SelectRoi(DWORD roiindex);
     // selects ROI with given index
int APIENTRY GetRoiSum(DWORD roiindex, double *sum);
             // get sum of counts in ROI,
             // returns roiindex, or 0 if not found

#ifdef __cplusplus
}
#endif
```

A.3 The Ordinal numbers of the functions

In the Definition file dmpa3.def the ordinal numbers of the library fuctions are defined:

```
;*Dmpa3.def
;*Version:          NT/95 1.0
;*Date:             Aug-15-1998
;*Hardware:         MPA3
;*Op System:        Windows NT 3.51/4.0, Windows 95
;*Compiler:         MSVC++ 4.2

;*

LIBRARY DMPA3

SECTIONS
     dmpa3sh        READ WRITE SHARED

EXPORTS
;    Functions in dmpa3.c
     StoreSettingData    @2
     GetSettingData      @3
     StoreStatusData     @4
     GetStatusData       @5
     Start               @6
     Halt                @7
     Continue            @8
     NewSetting          @9
     ServExec            @10
     StoreData           @11
     GetData             @12
     GetSpec             @13
     SaveSetting         @14
     GetStatus           @15
     Erase               @16
     SaveData            @17
     GetBlock            @18
     StoreDefData        @19
     GetDefData          @20
     LoadData            @21
     NewData             @22
     HardwareDlg         @23
     UnregisterClient    @24
     DestroyClient       @25
     ClientExec          @26
     LVGetDat            @27
     RunCmd              @28
     AddData             @29
```

```
        LVGetRoi             @30
        LVGetCnt             @31
        LVGetOneCnt          @32
        LVGetStr             @33
        SubData              @34
        Smooth               @35
        StoreExtSettingData  @36
        GetExtSettingData    @37
        StoreMP3Setting      @38
        GetMP3Setting        @39
        StoreDatSetting      @40
        GetDatSetting        @41
        StoreReplaySetting   @42
        GetReplaySetting     @43
        GetDatPtr            @44
        ReleaseDatPtr        @45
        LVGetOneRoi          @46
        GetSVal              @47
        DigInOut             @48
        DacOut               @49
        GetRoiIndex          @50
        DeleteRoi            @51
        GetRoiSum            @52
        SelectRoi            @53
;       Functions in custom.c
        CloseTab
        IniTab
        RTab
        IniSweep
        IncSweep
```

A.4 The sourcecode of the functions

In the source file dmpa3.c the body of the library functions is coded:

```
/****************************************************************************
 MODUL:    DMPA3.C
 PURPOSE:  DLL to communicate with MPA3 Server
****************************************************************************/

#include "windows.h"
#include <string.h>
#include <stdio.h>
#define DLL
#include "dmpa3.h"

#pragma data_seg("dmpa3sh")
BOOL    bRemote=0;
BOOL    bDef=FALSE;
BOOL    bMp3=FALSE;
BOOL    bDat=FALSE;
BOOL    bRepl=FALSE;
HWND    hwndServer=0;
HWND    hwndClient=0;
HWND    hwndMPANT=0;
int     MM_NEARCONTROL=0;
int     MM_GETVAL=0;
BOOL    bStatus[MAXDEV]={0};
BOOL    bSetting[MAXDSP]={0};

ACQSTATUS DLLStatus[MAXDEV] = {0};
EXTACQSETTING DLLSetting[MAXDSP] = {0};
#ifdef WINDOWS31
ACQDATA DLLData[MAXDSP] = {0};
HANDLE  hInst=0;
#endif
ACQDEF DLLDef = {0};
ACQMP3 DLLMp3 = {0};
DATSETTING DLLdat = {0};
REPLAYSETTING DLLRepl = {0};

#pragma data_seg()

#ifdef WINDOWS31
/****************************************************************************
   FUNCTION:  WEP(int)

   PURPOSE:  Performs cleanup tasks when the DLL is unloaded.  WEP() is
         called automatically by Windows when the DLL is unloaded (no
         remaining tasks still have the DLL loaded).  It is strongly
         recommended that a DLL have a WEP() function, even if it does
         nothing but returns success (1), as in this example.

*****************************************************************************/
int FAR PASCAL WEP (int bSystemExit)
{
   return(1);
}

/****************************************************************************
  FUNCTION: LibMain(HANDLE, WORD, WORD, LPSTR)
  PURPOSE:  Is called by LibEntry.  LibEntry is called by Windows when
        the DLL is loaded.  The LibEntry routine is provided in
        the LIBENTRY.OBJ in the SDK Link Libraries disk.  (The
        source LIBENTRY.ASM is also provided.)

        LibEntry initializes the DLL's heap, if a HEAPSIZE value is
        specified in the DLL's DEF file.  Then LibEntry calls
            LibMain.

            LibMain should return a value of 1 if the initialization is
            successful.
*******************************************************************************/
int FAR PASCAL LibMain(hModule, wDataSeg, cbHeapSize, lpszCmdLine)
HANDLE      hModule;
WORD    wDataSeg;
WORD    cbHeapSize;
LPSTR   lpszCmdLine;
{
  DLLDef.bRemote = 0;
  hInst=hModule;
  MM_NEARCONTROL = RegisterWindowMessage((LPSTR)"MPANEARCONTROL");
  if(cbHeapSize)
   UnlockData(0);

   return 1;
}

#else

BOOL APIENTRY DllMain(HANDLE hInst, DWORD ul_reason_being_called, LPVOID lpReserved)
{
   return 1;
      UNREFERENCED_PARAMETER(hInst);
      UNREFERENCED_PARAMETER(ul_reason_being_called);
      UNREFERENCED_PARAMETER(lpReserved);
}

#endif

VOID APIENTRY StoreDefData(ACQDEF *Def)
{
 int i;
/* {
          char txt[100];
          int nDisplay = 0;
          sprintf(txt,"StoreDefData %d %d %ld", nDisplay, bSetting[nDisplay], DLLSetting[nDisplay].range);
          MessageBox(NULL, txt, "DMPA3.DLL", MB_OK);
} */
 if(Def == NULL) {
   bDef = FALSE;
   for (i=0; i<MAXDSP; i++) {
     bSetting[i] = FALSE;
   }
 }
 else{
   _fmemcpy((LPSTR FAR *)&DLLDef,(LPSTR FAR *)Def,sizeof(ACQDEF));
   bDef = TRUE;
 }
/* {
          char txt[100];
          int nDisplay = 0;
          sprintf(txt,"StoreDefData %d %d %ld", nDisplay, bSetting[nDisplay], DLLSetting[nDisplay].range);
		MessageBox(NULL, txt, "DMPA3.DLL", MB_OK);
} */
}

int APIENTRY GetDefData(ACQDEF *Def)
{
  if (bDef) {
    DLLDef.bRemote = bRemote;
    _fmemcpy((LPSTR FAR *)Def,(LPSTR FAR *)&DLLDef,sizeof(ACQDEF));
  }
  return bDef;
}

VOID APIENTRY StoreMP3Setting(ACQMP3 *Defmp3)
{
  if(Defmp3 == NULL) {
    bMp3 = FALSE;
  }
  else{
    _fmemcpy((LPSTR FAR *)&DLLMp3,(LPSTR FAR *)Defmp3,sizeof(ACQMP3));
    bMp3 = TRUE;
  }
}

int APIENTRY GetMP3Setting(ACQMP3 *Defmp3)
{
  if (bMp3) {
    _fmemcpy((LPSTR FAR *)Defmp3,(LPSTR FAR *)&DLLMp3,sizeof(ACQMP3));
  }
  return bMp3;
}

VOID APIENTRY StoreDatSetting(DATSETTING *Defdat)
{
  if(Defdat == NULL) {
    bDat = FALSE;
  }
  else{
    _fmemcpy((LPSTR FAR *)&DLLdat,(LPSTR FAR *)Defdat,sizeof(DATSETTING));
    bDat = TRUE;
  }
}

int APIENTRY GetDatSetting(DATSETTING *Defdat)
{
  if (bDat) {
    _fmemcpy((LPSTR FAR *)Defdat,(LPSTR FAR *)&DLLdat,sizeof(DATSETTING));
  }
  return bDat;
}

VOID APIENTRY StoreReplaySetting(REPLAYSETTING *Repldat)
{
  if(Repldat == NULL) {
    bRepl = FALSE;
  }
  else{
    _fmemcpy((LPSTR FAR *)&DLLRepl,(LPSTR FAR *)Repldat,sizeof(REPLAYSETTING));
    bRepl = TRUE;
  }
}
```

```
int APIENTRY GetReplaySetting(REPLAYSETTING *Repldat)
{
  if (bRepl) {
    _fmemcpy((LPSTR FAR *)Repldat,(LPSTR FAR *)&DLLRepl,sizeof(REPLAYSETTING));
  }
  return bRepl;
}

VOID APIENTRY StoreSettingData(ACQSETTING * Setting, int nDisplay)
{
  if (nDisplay < 0 || nDisplay >= MAXDSP) return;
  if(Setting == NULL) {
    bSetting[nDisplay] = FALSE;
  }
  else{
    _fmemcpy((LPSTR FAR *)&DLLSetting[nDisplay],
          (LPSTR FAR *)Setting,sizeof(ACQSETTING));
    bSetting[nDisplay] = TRUE;
    if(Setting->range == 0L) {
      bSetting[nDisplay] = FALSE;
    }
/* {
                char txt[100];
                sprintf(txt,"StoreSettingData %d %d %ld", nDisplay, bSetting[nDisplay], Setting->range);
                MessageBox(NULL, txt, "DMPA3.DLL", MB_OK);
} */
  }
}

int APIENTRY GetSettingData(ACQSETTING *Setting, int nDisplay)
{
  if (nDisplay < 0 || nDisplay >= MAXDSP) return 0;
  if (bSetting[nDisplay]) {
    _fmemcpy((LPSTR FAR *)Setting,
          (LPSTR FAR *)&DLLSetting[nDisplay],sizeof(ACQSETTING));
  }
  return bSetting[nDisplay];
}

VOID APIENTRY StoreExtSettingData(EXTACQSETTING *Setting, int nDisplay)
{
  if (nDisplay < 0 || nDisplay >= MAXDSP) return;
  if(Setting == NULL) {
    bSetting[nDisplay] = FALSE;
  }
  else{
    _fmemcpy((LPSTR FAR *)&DLLSetting[nDisplay],
          (LPSTR FAR *)Setting,sizeof(EXTACQSETTING));
    bSetting[nDisplay] = TRUE;
    if(Setting->range == 0L) {
      bSetting[nDisplay] = FALSE;
    }
  }
/* {
                char txt[100];
                sprintf(txt,"StoreExtSettingData %d %d %ld", nDisplay, bSetting[nDisplay],
DLLSetting[nDisplay].range);
                MessageBox(NULL, txt, "DMPA3.DLL", MB_OK);
} */
}

int APIENTRY GetExtSettingData(EXTACQSETTING *Setting, int nDisplay)
```

```
{
  if (nDisplay < 0 || nDisplay >= MAXDSP) return 0;
  if (bSetting[nDisplay]) {
    _fmemcpy((LPSTR FAR *)Setting,
          (LPSTR FAR *)&DLLSetting[nDisplay],sizeof(EXTACQSETTING));
  }
/* {
              char txt[100];
              sprintf(txt,"GetExtSettingData %d %d %ld", nDisplay, bSetting[nDisplay],
DLLSetting[nDisplay].range);
              MessageBox(NULL, txt, "DMPA3.DLL", MB_OK);
} */
  return bSetting[nDisplay];
}

VOID APIENTRY StoreData(ACQDATA *Data, int nDisplay)
{
  if (nDisplay < 0 || nDisplay >= MAXDSP) return;
  if(Data == NULL) {
    bSetting[nDisplay] = FALSE;
  }
#ifdef WINDOWS31
  else
    _fmemcpy((LPSTR FAR *)&DLLData[nDisplay],(LPSTR FAR *)Data,sizeof(ACQDATA));
#endif
}

int APIENTRY GetData(ACQDATA *Data, int nDisplay)
{
  if (nDisplay < 0 || nDisplay >= MAXDSP) return 0;
#ifdef WINDOWS31
  if (bSetting[nDisplay]) {
    _fmemcpy((LPSTR FAR *)Data,(LPSTR FAR *)&DLLData[nDisplay],sizeof(ACQDATA));
  }
#endif
  return bSetting[nDisplay];
}

long APIENTRY GetSpec(long i, int nDisplay)
{
#ifdef WINDOWS31
  if (nDisplay < 0 || nDisplay >= MAXDSP) return 0;
  if (bSetting[nDisplay] && i < DLLSetting[nDisplay].range)
    return (DLLData[nDisplay].s0[i]);
  else
    return 0L;
#else
  char sz[40];
  HANDLE hs0;
  unsigned long *s0;
  unsigned long val;
  if (nDisplay < 0 || nDisplay >= MAXDSP) return 0;
  if (!bSetting[nDisplay]) return 0;
  if (i > DLLSetting[nDisplay].range) return 0;
  sprintf(sz,"MPA3_S0_%d",nDisplay);
  if (!(hs0 = OpenFileMapping(FILE_MAP_READ, FALSE, sz)))
            return 0;
  if (!(s0 = (unsigned long *)MapViewOfFile(hs0,
         FILE_MAP_READ, 0, 0, 0))) {
    CloseHandle(hs0);
    return 0;
  }
  val = s0[i];
  UnmapViewOfFile(s0);
  CloseHandle(hs0);
  return val;
#endif
}

VOID APIENTRY GetBlock(long *hist, int start, int end, int step,
  int nDisplay)
{
#ifdef WINDOWS31
  int i,j=0;
  if (nDisplay < 0 || nDisplay >= MAXDSP) return;
  if (end > DLLSetting[nDisplay].range) end = DLLSetting[nDisplay].range;
  for (i=start; i<end; i+=step, j++)
    *(hist + j) = DLLData[nDisplay].s0[i];
#else
  int i,j=0;
  char sz[40];
  HANDLE hs0;
  unsigned long *s0;
  if (nDisplay < 0 || nDisplay >= MAXDSP) return;
  if (!bSetting[nDisplay]) return;
  if (end > DLLSetting[nDisplay].range) end = DLLSetting[nDisplay].range;
  sprintf(sz,"MPA3_S0_%d",nDisplay);
  if (!(hs0 = OpenFileMapping(FILE_MAP_READ, FALSE, sz)))
            return;
  if (!(s0 = (unsigned long *)MapViewOfFile(hs0,
        FILE_MAP_READ, 0, 0, 0))) {
    CloseHandle(hs0);
    return;
  }
  for (i=start; i<end; i+=step, j++)
    *(hist + j) = s0[i];
  UnmapViewOfFile(s0);
  CloseHandle(hs0);
  return;
#endif
}

int APIENTRY LVGetDat(unsigned long HUGE *datp, int nDisplay)
{
#ifdef WINDOWS31
  if (bSetting[nDisplay]) {
    long i;
    for (i=0; i<DLLSetting[nDisplay].range; i++)
      datp[i] = DLLData[nDisplay].s0[i];
    return 0;
  }
  else return 4;
#else
  long i;
  char sz[40];
  HANDLE hs0;
  unsigned long *s0;
  if (nDisplay < 0 || nDisplay >= MAXDSP) return 4;
  if (!bSetting[nDisplay]) return 4;
  sprintf(sz,"MPA3_S0_%d",nDisplay);
  if (!(hs0 = OpenFileMapping(FILE_MAP_READ, FALSE, sz)))
            return 4;
  if (!(s0 = (unsigned long *)MapViewOfFile(hs0,
        FILE_MAP_READ, 0, 0, 0))) {
```

```
    CloseHandle(hs0);
    return 4;
  }
  for (i=0; i<DLLSetting[nDisplay].range; i++)
      datp[i] = s0[i];
  UnmapViewOfFile(s0);
  CloseHandle(hs0);
#endif
}

HANDLE hEXMDisplay=0;
LPSTR EXMDisplay=NULL;

int APIENTRY GetDatPtr(int nDisplay, long *xmax, long *ymax, LPSTR *pt)
{
  char sz[40];
  if (nDisplay < 0 || nDisplay >= MAXDSP) return -1;
  if (!bSetting[nDisplay]) return -1;
  *xmax = DLLSetting[nDisplay].xdim;
  *ymax = DLLSetting[nDisplay].range;
  if (*xmax) *ymax /= *xmax;
  else {*xmax = *ymax; *ymax = 1;}
  sprintf(sz,"MPA3_S0_%d",nDisplay);
  ReleaseDatPtr();
  if (!(hEXMDisplay = OpenFileMapping(FILE_MAP_READ, FALSE, sz)))
            return 0;
  if (!(EXMDisplay = MapViewOfFile(hEXMDisplay,
         FILE_MAP_READ, 0, 0, 0))) {
    CloseHandle(hEXMDisplay);
    return 0;
  }
  *pt = EXMDisplay;
  return (int)hEXMDisplay;
}

int APIENTRY ReleaseDatPtr()
{
  if(EXMDisplay)
             UnmapViewOfFile(EXMDisplay);
  EXMDisplay = NULL;
  if(hEXMDisplay)
             CloseHandle(hEXMDisplay);
  hEXMDisplay = 0;
  return 0;
}

int APIENTRY LVGetRoi(unsigned long FAR *roip, int nDisplay)
{
#ifdef WINDOWS31
  if (bSetting[nDisplay]) {
    int i,n;
    n = 2 * DLLSetting[nDisplay].nregions;
    for (i=0; i<n; i++)
      roip[i] = DLLData[nDisplay].region[i];
    return 0;
  }
  else return 4;
#else
  int i,n;
  char sz[40];
  HANDLE hrg;
  unsigned long *region;
  if (nDisplay < 0 || nDisplay >= MAXDSP) return 4;
  if (!bSetting[nDisplay]) return 4;
  sprintf(sz,"MPA3_RG_%d",nDisplay);
  if (!(hrg = OpenFileMapping(FILE_MAP_READ, FALSE, sz)))
            return 4;
  if (!(region = (unsigned long *)MapViewOfFile(hrg,
        FILE_MAP_READ, 0, 0, 0))) {
    CloseHandle(hrg);
    return 4;
  }
  n = 2 * DLLSetting[nDisplay].nregions;
  for (i=0; i<n; i++)
    roip[i] = region[i];
  UnmapViewOfFile(region);
  CloseHandle(hrg);
  return 0;
#endif
}

int APIENTRY LVGetOneRoi(int nDisplay, int roinum, long *x1, long *x2)
{
#ifdef WINDOWS31
  if (bSetting[nDisplay] && (roinum > 0 && (roinum <= 128)) {
    *x1 = DLLData[nDisplay].region[2*(roinum-1)];
    *x2 = DLLData[nDisplay].region[2*(roinum-1)+1];
    return 0;
  }
  else return 4;
#else
  char sz[40];
  HANDLE hrg;
  unsigned long *region;
  if (nDisplay < 0 || nDisplay >= MAXDSP) return 4;
  if (!bSetting[nDisplay] || (roinum < 1) || (roinum > 128)) return 4;
  sprintf(sz,"MPA3_RG_%d",nDisplay);
  if (!(hrg = OpenFileMapping(FILE_MAP_READ, FALSE, sz)))
            return 4;
  if (!(region = (unsigned long *)MapViewOfFile(hrg,
        FILE_MAP_READ, 0, 0, 0))) {
    CloseHandle(hrg);
    return 4;
  }
  *x1 = region[2*(roinum-1)];
  *x2 = region[2*(roinum-1)+1];
  UnmapViewOfFile(region);
  CloseHandle(hrg);
  return 0;
#endif
}

int APIENTRY LVGetCnt(double *cntp, int nDisplay)
{
#ifdef WINDOWS31
  if (bSetting[nDisplay]) {
    int i;
    for (i=0; i<MAXCNT; i++)
      cntp[i] = DLLData[nDisplay].cnt[i];
    return 0;
  }
  else return 4;
#else
  int i;
  char sz[40];
  HANDLE hct;
  double *cnt;
  if (nDisplay < 0 || nDisplay >= MAXDSP) return 4;
  if (!bSetting[nDisplay]) return 4;
  sprintf(sz,"MPA3_CT_%d",nDisplay);
  if (!(hct = OpenFileMapping(FILE_MAP_READ, FALSE, sz)))
            return 4;
  if (!(cnt = (double *)MapViewOfFile(hct,
        FILE_MAP_READ, 0, 0, 0))) {
    CloseHandle(hct);
    return 4;
  }
  for (i=0; i<MAXCNT; i++)
    cntp[i] = cnt[i];
  UnmapViewOfFile(cnt);
  CloseHandle(hct);
  return 0;
#endif
}

int APIENTRY LVGetOneCnt(double *cntp, int nDisplay, int cntnum)
                            // Get one Cnt number
{
#ifdef WINDOWS31
  if (bSetting[nDisplay]) {
    *cntp = DLLData[nDisplay].cnt[cntnum];
    return 0;
  }
  else return 4;
#else
  char sz[40];
  HANDLE hct;
  double *cnt;
  if (nDisplay < 0 || nDisplay >= MAXDSP) return 4;
  if (!bSetting[nDisplay]) return 4;
  sprintf(sz,"MPA3_CT_%d",nDisplay);
  if (!(hct = OpenFileMapping(FILE_MAP_READ, FALSE, sz)))
            return 4;
  if (!(cnt = (double *)MapViewOfFile(hct,
        FILE_MAP_READ, 0, 0, 0))) {
    CloseHandle(hct);
    return 4;
  }
  *cntp = cnt[cntnum];
  UnmapViewOfFile(cnt);
  CloseHandle(hct);
  return 0;
#endif
}

int APIENTRY LVGetStr(char *strp, int nDisplay)
{
#ifdef WINDOWS31
  if (bSetting[nDisplay]) {
    int i;
    for (i=0; i<1024; i++)
      strp[i] = DLLData[nDisplay].comment0[i];
    return 0;
  }
  else return 4;
#else
```

```
  int i;
  char sz[40];
  HANDLE hcm;
  char *comment0;
  if (nDisplay < 0 || nDisplay >= MAXDSP) return 4;
  if (!bSetting[nDisplay]) return 4;
  sprintf(sz,"MPA3_CM_%d",nDisplay);
  if (!(hcm = OpenFileMapping(FILE_MAP_READ, FALSE, sz)))
            return 4;
  if (!(comment0 = (char *)MapViewOfFile(hcm,
        FILE_MAP_READ, 0, 0, 0))) {
    CloseHandle(hcm);
    return 4;
  }
  for (i=0; i<1024; i++)
    strp[i] = comment0[i];
  UnmapViewOfFile(comment0);
  CloseHandle(hcm);
  return 0;
#endif
}

VOID APIENTRY StoreStatusData(ACQSTATUS *Status, int nDev)
{
  if (nDev < 0 || nDev >= MAXDEV) return;
  if(Status == NULL)
    bStatus[nDev] = FALSE;
  else{
    _fmemcpy((LPSTR FAR *)&DLLStatus[nDev],
         (LPSTR FAR *)Status,sizeof(ACQSTATUS));
    bStatus[nDev] = TRUE;
  }
/*  {
                char txt[100];
                sprintf(txt,"StoreStatusData %d %d", nDisplay, bStatus[nDisplay]);
                MessageBox(NULL, txt, "DMPA3.DLL", MB_OK);
  } */
}

int APIENTRY GetStatusData(ACQSTATUS *Status, int nDev)
{
  //DebugBreak();
  /* {
                char txt[100];
                sprintf(txt,"GetStatusData %d %d", nDev, bStatus[nDev]);
                MessageBox(NULL, txt, "DMPA3.DLL", MB_OK);
  } */
  if (nDev < 0 || nDev > DLLDef.nDevices) return 0;
  if (bStatus[nDev]) {
    _fmemcpy((LPSTR FAR *)Status,
         (LPSTR FAR *)&DLLStatus[nDev],sizeof(ACQSTATUS));
  }
  return bStatus[nDev];
}

VOID APIENTRY Start(int nSystem)
{
  if (nSystem < 0 || nSystem > 3) return;
  if (!hwndServer) hwndServer = FIND_WINDOW("MPA-3 Server",NULL);
  switch (nSystem) {
    case 0:
     PostMessage(hwndServer, WM_COMMAND, ID_START, 0L);
     break;
    case 1:
     PostMessage(hwndServer, WM_COMMAND, ID_START2, 0L);
     break;
    case 2:
     PostMessage(hwndServer, WM_COMMAND, ID_START3, 0L);
     break;
    case 3:
     PostMessage(hwndServer, WM_COMMAND, ID_START4, 0L);
     break;
  }
}

VOID APIENTRY Halt(int nSystem)
{
  if (nSystem < 0 || nSystem > 3) return;
  if (!hwndServer) hwndServer = FIND_WINDOW("MPA-3 Server",NULL);
  switch (nSystem) {
    case 0:
     PostMessage(hwndServer, WM_COMMAND, ID_BREAK, 0L);
     break;
    case 1:
     PostMessage(hwndServer, WM_COMMAND, ID_BREAK2, 0L);
     break;
    case 2:
     PostMessage(hwndServer, WM_COMMAND, ID_BREAK3, 0L);
     break;
    case 3:
     PostMessage(hwndServer, WM_COMMAND, ID_BREAK4, 0L);
     break;
  }
}

VOID APIENTRY Continue(int nSystem)
{
  if (nSystem < 0 || nSystem > 3) return;
  if (!hwndServer) hwndServer = FIND_WINDOW("MPA-3 Server",NULL);
  switch (nSystem) {
    case 0:
     PostMessage(hwndServer, WM_COMMAND, ID_CONTINUE, 0L);
     break;
    case 1:
     PostMessage(hwndServer, WM_COMMAND, ID_CONTINUE2, 0L);
     break;
    case 2:
     PostMessage(hwndServer, WM_COMMAND, ID_CONTINUE3, 0L);
     break;
    case 3:
     PostMessage(hwndServer, WM_COMMAND, ID_CONTINUE4, 0L);
     break;
  }
}

VOID APIENTRY SaveSetting()
{
  if (!hwndServer) hwndServer = FIND_WINDOW("MPA-3 Server",NULL);
  PostMessage(hwndServer, WM_COMMAND, ID_SAVE, 0L);
}

VOID APIENTRY NewSetting(int nDev)
{
  if (!hwndServer) hwndServer = FIND_WINDOW("MPA-3 Server",NULL);
  //if (nDev>=0 && nDev<8) bStatus[nDev] = FALSE;
  SendMessage(hwndServer, WM_COMMAND, ID_NEWSETTING, 0L);
}

VOID APIENTRY NewData()
{
  if (!hwndServer) hwndServer = FIND_WINDOW("MPA-3 Server",NULL);
  PostMessage(hwndServer, WM_COMMAND, ID_NEWDATA, 0L);
}

int APIENTRY GetStatus(int nDev)
{
  if (!hwndServer) hwndServer = FIND_WINDOW("MPA-3 Server",NULL);
  if (bStatus[nDev]) {
      SendMessage(hwndServer, WM_COMMAND, ID_GETSTATUS, 0L);
  }
  /* {
              char txt[100];
              sprintf(txt,"GetStatus %d %d", nDev, bStatus[nDev]);
              MessageBox(NULL, txt, "DMCD2.DLL", MB_OK);
  } */
  return bStatus[nDev];
}

UINT APIENTRY ServExec(HWND ClientWnd)
{
  bRemote = 1;
  hwndClient = ClientWnd;
  if (!hwndServer) hwndServer = FIND_WINDOW("MPA-3 Server",NULL);
  if (hwndServer) {
    ShowWindow(hwndServer, SW_MINIMIZE);
    return 32;
  }
  else
    return WinExec("MPA3.EXE", SW_SHOW);
}

UINT APIENTRY ClientExec(HWND ServerWnd)
{
  if (ServerWnd) hwndServer = ServerWnd;
  return WinExec((LPSTR)"MPANT /device=MPA3", SW_SHOW);
}

VOID APIENTRY UnregisterClient()
{
  hwndClient = 0;
  bRemote = 0;
}

VOID APIENTRY DestroyClient()
{
  bRemote = 0;
  if (hwndClient) SendMessage(hwndClient, WM_CLOSE, 0, 0L);
  hwndClient = 0;
}

VOID APIENTRY Erase(int nSystem)
{
  if (nSystem < 0 || nSystem > 3) return;
  if (!hwndServer) hwndServer = FIND_WINDOW("MPA-3 Server",NULL);
  switch (nSystem) {
    case 0:
     PostMessage(hwndServer, WM_COMMAND, ID_ERASE, 0L);
     break;
    case 1:
     PostMessage(hwndServer, WM_COMMAND, ID_ERASE2, 0L);
     break;
    case 2:
     PostMessage(hwndServer, WM_COMMAND, ID_ERASE3, 0L);
     break;
    case 3:
     PostMessage(hwndServer, WM_COMMAND, ID_ERASE4, 0L);
     break;
  }
}

VOID APIENTRY SaveData(int nDisplay, int all)
{
  if (nDisplay < 0 || nDisplay >= MAXDSP) return;
  if (!hwndServer) hwndServer = FIND_WINDOW("MPA-3 Server",NULL);
  PostMessage(hwndServer, WM_COMMAND, ID_SAVEFILE,
                        MAKELPARAM((WORD)nDisplay, (WORD)all));
}

VOID APIENTRY LoadData(int nDisplay, int all)
{
  if (nDisplay < 0 || nDisplay >= MAXDSP) return;
  if (!hwndServer) hwndServer = FIND_WINDOW("MPA-3 Server",NULL);
//  bStatus[nDisplay] = FALSE;
  PostMessage(hwndServer, WM_COMMAND, ID_LOADFILE,
                        MAKELPARAM((WORD)nDisplay, (WORD)all));
}

VOID APIENTRY AddData(int nDisplay, int all)
{
  if (nDisplay < 0 || nDisplay >= MAXDSP) return;
  if (!hwndServer) hwndServer = FIND_WINDOW("MPA-3 Server",NULL);
//  bStatus[nDisplay] = FALSE;
  PostMessage(hwndServer, WM_COMMAND, ID_SUMFILE,
                        MAKELPARAM((WORD)nDisplay, (WORD)all));
}

VOID APIENTRY SubData(int nDisplay, int all)
{
  if (nDisplay < 0 || nDisplay >= MAXDSP) return;
  if (!hwndServer) hwndServer = FIND_WINDOW("MPA-3 Server",NULL);
//  bStatus[nDisplay] = FALSE;
  PostMessage(hwndServer, WM_COMMAND, ID_SUBTRACT,
                        MAKELPARAM((WORD)nDisplay, (WORD)all));
}

VOID APIENTRY Smooth(int nDisplay)
{
  if (nDisplay < 0 || nDisplay >= MAXDSP) return;
  if (!hwndServer) hwndServer = FIND_WINDOW("MPA-3 Server",NULL);
//  bStatus[nDisplay] = FALSE;
  PostMessage(hwndServer, WM_COMMAND, ID_SMOOTH,
                        MAKELPARAM((WORD)nDisplay, (WORD)0));
}

VOID APIENTRY HardwareDlg(int item)
{
  if (!hwndServer) hwndServer = FIND_WINDOW("MPA-3 Server",NULL);
  switch (item) {
    case 0:
      PostMessage(hwndServer, WM_COMMAND, ID_HARDWDLG, 0L);
      break;
    case 1:
      PostMessage(hwndServer, WM_COMMAND, ID_DATADLG, 0L);
      break;
    case 2:
      PostMessage(hwndServer, WM_COMMAND, ID_COMBDLG, 0L);
      break;
    case 3:
      PostMessage(hwndServer, WM_COMMAND, ID_MAPLSTDLG, 0L);
      break;
    case 4:
      PostMessage(hwndServer, WM_COMMAND, ID_REPLDLG, 0L);
      break;
  }
}

VOID APIENTRY RunCmd(int nDisplay, LPSTR Cmd)
{
// nDisplay must be zero
#ifdef WINDOWS31
  if (!hwndServer) hwndServer = FIND_WINDOW("MPA-3 Server",NULL);
  if (!MM_NEARCONTROL) MM_NEARCONTROL =
RegisterWindowMessage((LPSTR)"MPANEARCONTROL");
  if (Cmd != NULL) {
    _fstrcpy(&DLLData[0].comment0[800], Cmd);
  }
#else
  char sz[40];
  HANDLE hcm;
  char *comment0;
  if (nDisplay < 0 || nDisplay >= MAXDSP) return;
  if (!bSetting[nDisplay]) return;
  if (!hwndServer) hwndServer = FIND_WINDOW("MPA-3 Server",NULL);
  if (!MM_NEARCONTROL) MM_NEARCONTROL =
RegisterWindowMessage((LPSTR)"MPANEARCONTROL");
  sprintf(sz,"MPA3_CM_%d",nDisplay);
  if (!(hcm = OpenFileMapping(FILE_MAP_WRITE, FALSE, sz)))
            return;
  if (!(comment0 = (char *)MapViewOfFile(hcm,
        FILE_MAP_WRITE, 0, 0, 0))) {
    CloseHandle(hcm);
    return;
  }
  strcpy(&comment0[800], Cmd);
#endif
SendMessage(hwndServer, MM_NEARCONTROL, (WPARAM)ID_RUNCMD, (LONG)(LPSTR)Cmd);
#ifndef WINDOWS31
  strcpy(Cmd, &comment0[1024]);
  UnmapViewOfFile(comment0);
  CloseHandle(hcm);
#endif
}

long APIENTRY GetSVal(int DspID, long xval)
{
  long val=0;
  if (xval == -2) {
              hwndMPANT = FIND_WINDOW("mpwframe",NULL);
              return (long)hwndMPANT;   // should be called first to be sure that MPANT is started
  }
  if (!hwndMPANT) hwndMPANT = FIND_WINDOW("mpwframe",NULL);
  if (!MM_GETVAL) MM_GETVAL = RegisterWindowMessage((LPSTR)"MPANTGetval");
  val = SendMessage(hwndMPANT, MM_GETVAL, (WPARAM)DspID, (LPARAM)xval);
    // for xval == -1 returns Display size
  return val;
}

int APIENTRY DigInOut(int value, int enable)  // controls Dig I/0 ,
                                                                 // returns digin
{
  int val=0;
  long lval;
  if (!hwndServer) hwndServer = FIND_WINDOW("MPA-3 Server",NULL);
  if (!MM_NEARCONTROL) MM_NEARCONTROL =
RegisterWindowMessage((LPSTR)"MPANEARCONTROL");
  lval = ((long)value & 0xFF) | ((enable & 0xFF)<<8);
  val = SendMessage(hwndServer, MM_NEARCONTROL, ID_DIGINOUT, (LONG)lval);
  return val;
}

int APIENTRY DacOut(int value)    // output Dac value as analogue voltage
{
  int val=0;
  long lval;
  if (!hwndServer) hwndServer = FIND_WINDOW("MPA-3 Server",NULL);
  if (!MM_NEARCONTROL) MM_NEARCONTROL =
RegisterWindowMessage((LPSTR)"MPANEARCONTROL");
  value &= 0xFF;
  lval = (long)value;
  val = SendMessage(hwndServer, MM_NEARCONTROL, ID_DACOUT, (LONG)lval);
  return val;
}

long APIENTRY GetRoiIndex(LPSTR roiname)
            // for named ROI's returns in LOWORD the spectra number, in HIWORD the roiid,
    // returns 0 if not found.
{
  int val=0;
  char sz[40];
  HANDLE hcm;
  char *comment0;
  if (!hwndServer) hwndServer = FIND_WINDOW("MPA-3 Server",NULL);
  if (!MM_NEARCONTROL) MM_NEARCONTROL =
RegisterWindowMessage((LPSTR)"MPANEARCONTROL");
  strcpy(sz,"MPA3_CM_0");
  if (!(hcm = OpenFileMapping(FILE_MAP_WRITE, FALSE, sz)))
            return 0;
  if (!(comment0 = (char *)MapViewOfFile(hcm,
        FILE_MAP_WRITE, 0, 0, 0))) {
    CloseHandle(hcm);
    return 0;
  }
  strncpy(&comment0[800], roiname, 20);
  comment0[820] = '\0';

  val = SendMessage(hwndServer, MM_NEARCONTROL, ID_GETROIINDEX, 0);
  UnmapViewOfFile(comment0);
  CloseHandle(hcm);

  return val;
}
```

```
int APIENTRY DeleteRoi(DWORD roiindex)
   // deletes ROI
{
  int val=0;
  HWND hwndClient = FIND_WINDOW("mpwframe",NULL);
  if (hwndClient)
    return SendMessage(hwndClient, WM_COMMAND, ID_DELETEROI, roiindex);
  if (!hwndServer) hwndServer = FIND_WINDOW("MPA-3 Server",NULL);
  if (!MM_NEARCONTROL) MM_NEARCONTROL =
RegisterWindowMessage((LPSTR)"MPANEARCONTROL");
  val = SendMessage(hwndServer, MM_NEARCONTROL, ID_DELETEROI, (LONG)roiindex);
  return val;
}

int APIENTRY SelectRoi(DWORD roiindex)
   // selects ROI
{
  int val=0;
  HWND hwndClient = FIND_WINDOW("mpwframe",NULL);
  if (hwndClient)
    return SendMessage(hwndClient, WM_COMMAND, ID_SELECTROI, roiindex);
  return val;
}

int APIENTRY GetRoiSum(DWORD roiindex, double *sum)
        // get sum of counts in ROI,
        // returns roiindex= in LOWORD the spectra number, in HIWORD the roiid, or 0 if not found
{
  int val=0, nDisplay, roiid;
  if (!hwndServer) hwndServer = FIND_WINDOW("MPA-3 Server",NULL);
  if (!MM_NEARCONTROL) MM_NEARCONTROL =
RegisterWindowMessage((LPSTR)"MPANEARCONTROL");
  val = SendMessage(hwndServer, MM_NEARCONTROL, ID_GETROISUM, (LONG)roiindex);
  if (val) {
         roiid = HIWORD(roiindex);
         if (roiid < 100) nDisplay = LOWORD(roiindex);
         else nDisplay = roiid/100;
         if(LVGetOneCnt(sum, nDisplay, 7)) return 0;
// cnt[7] contains the sum, cnt[8] contains the area
  }
  return val;
}
```

A.5 The custom functions

Version 1.32 (Sep-26-2000) of MPA-3 software:
- The Calculated Spectrum Setting dialog was expanded by a new option named "DLL function" and a button named "Edit". By selecting the "DLL function" and pressing this button a dialog "DLL Function" is opened. The calculated spectrum is a function programmed in the DMPA3.DLL RTab(Xpar, Ypar, Zpar).
  Run the TSTRXY.EXE program to create a sample table file and try the dlltest.cnf configuration: For a 2D position dependent energy spectra correction now as a sample a function named

unsigned long RTab(long x, long y, long z)

is implemented in the DLL. It is calculated by looking up in a table as
z * Tab[x,y] and must be initialised by using the DLL function named

int IniTab(LPSTR filename);
The Cleanup function CloseTab() frees the memory allocated by IniTab. A sample table can be calculated by using a program tstrxy.exe. It assumes a Gaussion function for the position dependent energy calibration of a 2D position dependent detector. The names of these DLL functions and the filename containing the table must be provided in the DLL Function dialog, and also the dimensions XRange and YRange of the table. The Xpar and Ypar parameters are shifted to fit their spectra range into the table dimensions. The complete sources of the DMPA3.DLL and TSTRXY.EXE are included in the DLL software interface to allow own written applications for calculated spectra based on this example.

In the source files custom.c and custom.h special functions are coded to allow the calculation of user-defined parameter spectra. As an example for a 2D position dependent detector a position-dependent energy spectra correction is implemented via a lookup table.

Here is the header file custom.h:

```
VOID APIENTRY CloseTab(void);
int APIENTRY IniTab(LPSTR filename);
unsigned long APIENTRY RTab(long x, long y, long z);
```

Here is custom.c:

```
/**************************************************************************
  MODUL:    custom.C
  PURPOSE:  Customer module of DMAP3.DLL to communicate with MPA3 Server
***************************************************************************/

#include "windows.h"
#include <string.h>
#include <stdio.h>
#define DLL
#include "dmpa3.h"
#include "custom.h"

long xdim=256, ydim=256, ndim;
HANDLE hTab=0;
double *lpTab=NULL;

int freadstr(FILE *stream, char *buff, int buflen)
{
  int i=0,ic;
```

```
 while ((ic=getc(stream)) != 10) {
   if (ic == EOF) {
     buff[i]='\0';
     return 1;
   }
   if (ic == 13) ic=0;
   buff[i]=(char)ic;
   i++;
   if (i==buflen-1) break;
 }
 buff[i]='\0';
 return 0;
}

VOID APIENTRY CloseTab()
{
 if (lpTab)
   GlobalUnlock(lpTab);
 if (hTab)
   GlobalFree(hTab);
 hTab=0;
 lpTab=NULL;
}

int APIENTRY IniTab(LPSTR filename)
{
        FILE *f;
        long i, j, x, y;
        int ret;
        double *pf;
        char txt[256];

        if (!filename) return -1;
        if (!(f = fopen(filename, "rb"))) {
          sprintf(txt, "Filename %s not found!", filename);
          MessageBox(NULL, txt, "DMPA3.DLL", MB_OK);
          return -1;
        }
        while(!freadstr(f,txt,256)) {
            if(!strnicmp("xdim=", txt, 5)) {
                sscanf(txt+5,"%ld",&xdim);
            }
            else if(!strnicmp("ydim=", txt, 5)) {
                sscanf(txt+5,"%ld",&ydim);
            }
            if(!strnicmp("[DATA", txt, 5)) {
                break;
            }
        }
        ndim = xdim * ydim;
        hTab = GlobalAlloc(GMEM_DDESHARE|GMEM_MOVEABLE,(DWORD)sizeof(double)*ndim);
        if (!hTab) {
          MessageBox(NULL, "No memory!", "DMPA3.DLL", MB_OK);
          return -1;
        }
        lpTab = (double *) GlobalLock(hTab);
        pf = lpTab;
        for (j=0; j<ydim; j++) {
          for (i=0; i<xdim; i++) {
            ret = (freadstr(f, txt, 256));
            sscanf(txt, "%ld %ld %lf", &x, &y, pf);
```

```
                pf++;
                if (ret) goto out;
              }
            }
out:
            if (j < ydim) {
              sprintf(txt, "Table ends at x=%ld y=%ld", i, j);
              MessageBox(NULL, txt, "DMPA3.DLL", MB_OK);
              CloseTab();
              return -1;
            }
            return 0;
}

unsigned long APIENTRY RTab(long x, long y, long z)
{
  long val;
  if (lpTab==NULL) return 0;
  if ((x >= xdim) || (y >= ydim)) return 0;
  val = (long) (z * lpTab[y*xdim + x]);
  if (val > 0) return (unsigned long)val;
  else return 0;
}
```

Version 1.48 (Nov-18-2002) of MPA-3 software:
The custom.c module was expanded by a function that allows to use an ADC input as a sweep marker in the data stream and form a sweep count parameter from an ADC signal. It is then possible to form two dimensional spectra with the sweep counter as one parameter. Use for example the ADC1D as a sweep counter. In the "Calculated Spectrum Setting " dialog choose then ADC1D as the "left" and "right" Parameter, and click "Edit" to open the "DLL Function" dialog. Choose ADC1D also as the Zpar Parameter. Enter the following strings:

Initialise DLL function:	IniSweep
Table filename:	Dummy
Cleanup DLL function:	CloseTab
DLL function:	IncSweep

Instead of "Dummy" also the name of a file containing some ASCII lines defining some parameters can be entered, especially a ROI boundary for a valid ADC signal. It is obvious from the custom.c source how to use it.

A.6 How to compile the DLL

The 32 bit DLL can be compiled with the Microsoft Visual C/C++ compiler version 4.2 or higher. To recompile the DLL, use the makefile dmpa3.mak.