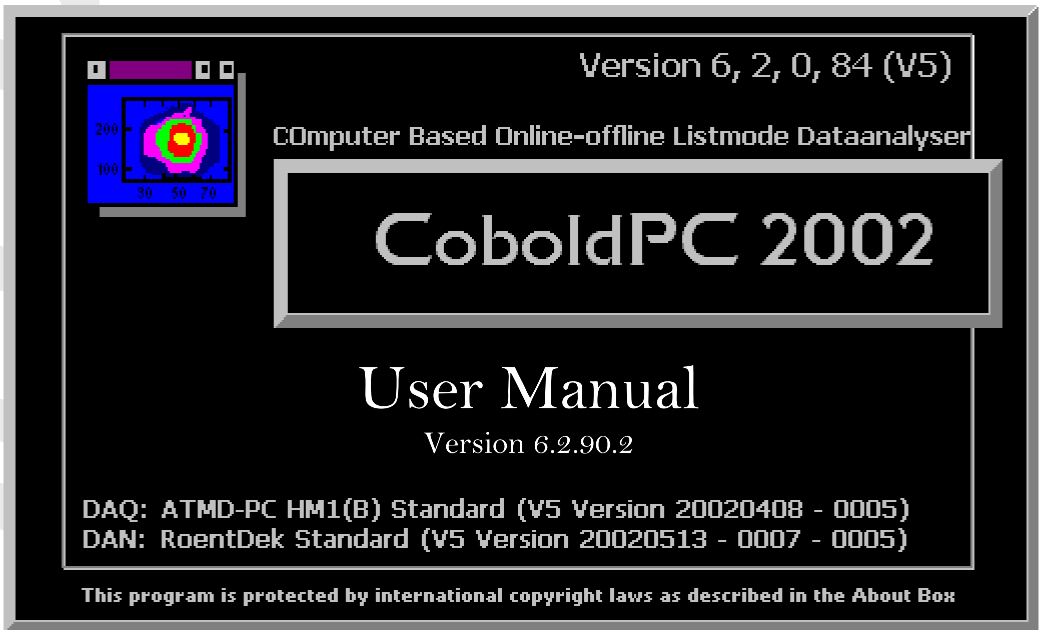


Version 6, 2, 0, 84 (V5)

COmputer Based Online-offline Listmode Dataanalyser

CoboldPC 2002

User Manual

Version 6.2.90.2

DAQ: ATMD-PC HM1(B) Standard (V5 Version 20020408 - 0005)
DAN: RoentDek Standard (V5 Version 20020513 - 0007 - 0005)

Mail Addresses:

Headquarter

RoentDek Handels GmbH
Im Vogelshaag 8
D-65779 Kelkheim-Ruppertshain
Germany

Frankfurt subsidiary

RoentDek Handels GmbH
c/o Institut für Kernphysik
Max-von-Laue Str. 1
D-60438 Frankfurt am Main
Germany

ph: +49-69-798-47108
fax: +49-69-798-47107

Web-Site:

www.roentdek.com

Table of Content

1. General Introduction

1.1. Overview

CoboldPC is designed as a convenient data acquisition (DAQ) and online/offline data analysis (DAN) program for Windows XP (recommended), Windows NT or Windows 95/98.

Due to its modular structure it can potentially address almost any hardware and allows very elaborate data treatments as the analysis plug-in can be modified by the user. The interface between the main program and the hardware is done by a dynamic link library named DAQ.dll while the analysis is performed by another link library named DAN.dll

With this program package you have received a DAQ.dll and at least one standard example analysis file DAN.dll suitable for the purchased hardware. You also received a start-up batch file (CoboldCommandFile, filename.ccf) which will enable you to easily start your first CoboldPC session (e.g. DAQ-hardware read-out with simultaneous on-line control via the example DAN.dll) and already learn about the most frequently used CoboldPC commands.

If additional DAQ plug-ins are necessary due to changes of the hardware setup please contact **RoentDek**. You may change the DAN analysis plug-ins using the MS-Visual C++ 6 compiler, MS-Fortran Power Station 4 or DEC-Visual Fortran 6.

The acquired data (events) received from the DAQ hardware can be stored on disc in a list-mode format (ListModeFile, filename.lmf). The data are stored event by event (consisting of a row of so called coordinates simultaneously acquired) to re-run the analysis offline after the experiment. The size of the ListModeFile is defined by the number of events acquired and the number of coordinates per event.

The online sorting procedure in the DAN.dll and the CoboldCommandFile-script results in a number of one- and two-dimensional spectra showing the acquired data. These spectra can also be stored to disc in the so called (DumpCoboldFile, filename.dcf), printed, exported to other programs (text-editors, display programs or other data analysis programs) or exported as plain ASCII via "Cut and Paste". You can also import ASCII-files in proper format to CoboldPC for data analysis in order to take advantage of the mathematical routines defined in CoboldPC.

All CoboldPC commands are listed and explained in the CoboldPC Help File. Just type "help" in the program command bar to start the help or press F1.

1.2. List-Mode

The program ***CoboldPC*** is a C++ program to analyse List-Mode-Data. ***CoboldPC*** stands for "Computer Based Online offline Listmode Dataanalyser"

List-Mode is a special technique used in collision-physics. In this mode all acquired information (named coordinates in ***CoboldPC***) is stored event by event in a data-list (see Figure 1).

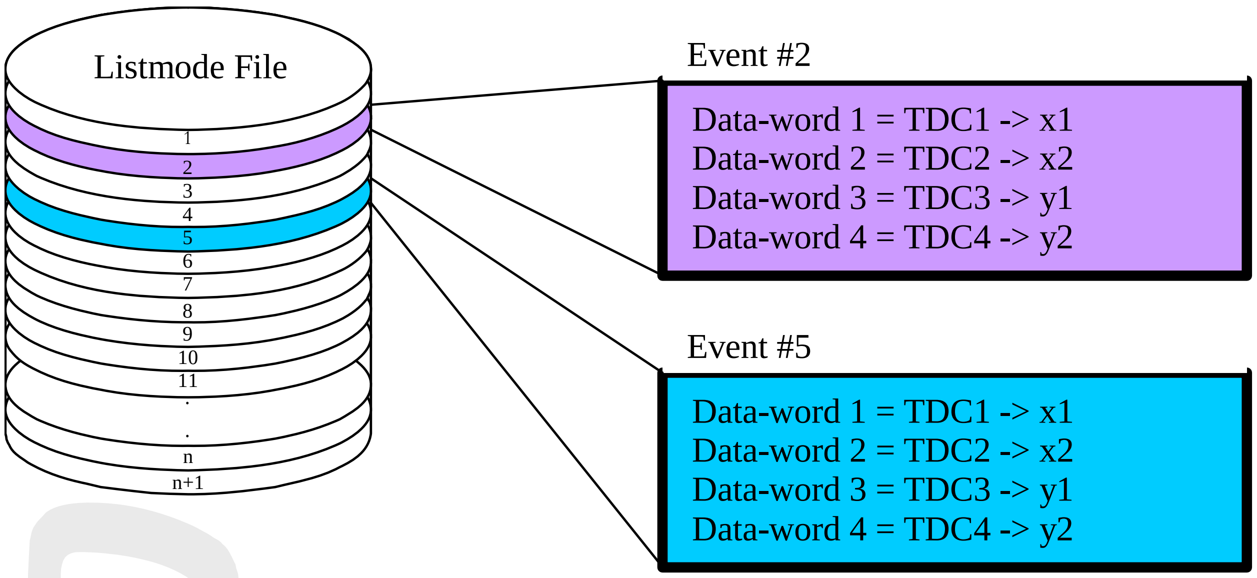
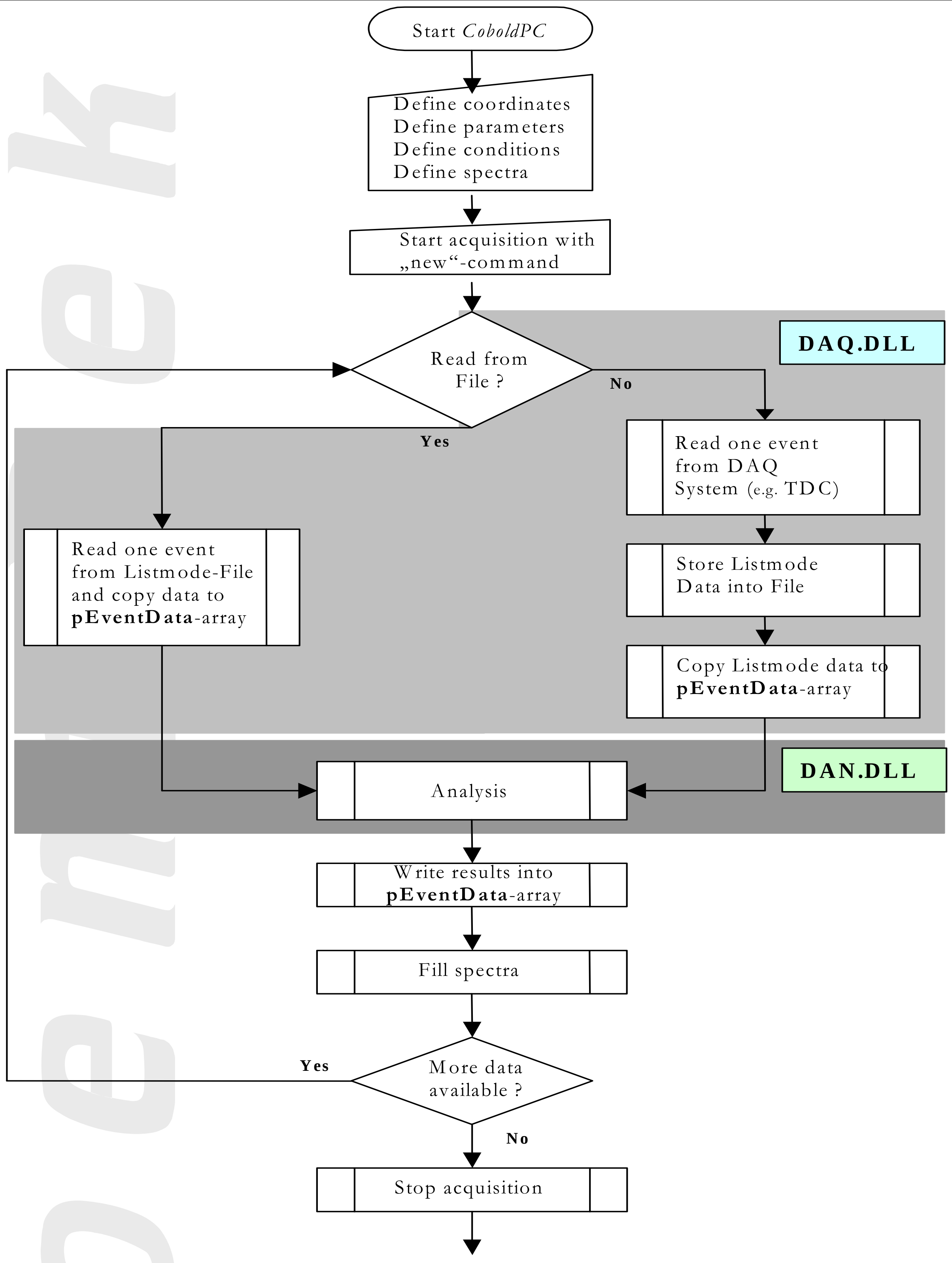


Figure 1: Schematic of a Listmode-File

Either during data acquisition (on-line mode) or after the experiment (off-line mode) the list is processed in ***CoboldPC***. The program allows to sort and to display the information. To combine information of one event complex calculation and mathematics can be performed on the original event-data in the DAN.dll. During this process new coordinates are created and also sorted and displayed by the ***CoboldPC*** program.

Figure 2 shows the flow chart of the ***CoboldPC*** data taking and analysis.


Start *CoboldPC*
Define coordinates
Define parameters
Define conditions
Define spectra
Start acquisition with „new“-command
Read from File ?
No
Yes
DAQ.DLL
Read one event from DAQ System (e.g. TDC)
Store Listmode Data into File
Copy Listmode data to **pEventData**-array
Read one event from Listmode-File and copy data to **pEventData**-array
DAN.DLL
Analysis
Write results into **pEventData**-array
Fill spectra
More data available ?
Yes
No
Stop acquisition

The pEventData-array is one to one related to the list of Coordinates in the ccf-file.

Figure 2: Flowchart of a CoboldPC session

1.3. Organisation

CoboldPC is the main program that controls two sub-programs (DAN.dll and DAQ.dll). For analysis and data acquisition the main program needs these two additional program parts. The DAN.DLL is responsible for the calculations on the original List-Mode-Data. The DAQ.DLL is handling all communications with the data acquisition hardware (e.g. a TDC). It will also handle the writing of the List-Mode-Data to a file if this was selected by the User. Please see chapter 7 for more details.

1.4. Reduced functionality with Windows95/98

Due to memory management differences in Windows95/98 and Windows NT there are a minor restrictions in the functionality of **CoboldPC** running on Windows95/98.

- All redrawing of the **CoboldPC** window has to perform the complete display routine. In Windows NT there is a screen buffering so only the first call of a spectrum will perform a complete drawing.
- Cut of spectrum graphics is disabled in Windows 95/98.
- Cut of spectrum ASCII is only enabled when displaying a one-dimensional spectrum in Windows95/98.

We hope we can support full functionality also in Windows95/98 soon.

1.5. Quick Startup

Start the **CoboldPC** program out of the **CoboldPC** program group as shown in Figure 3.

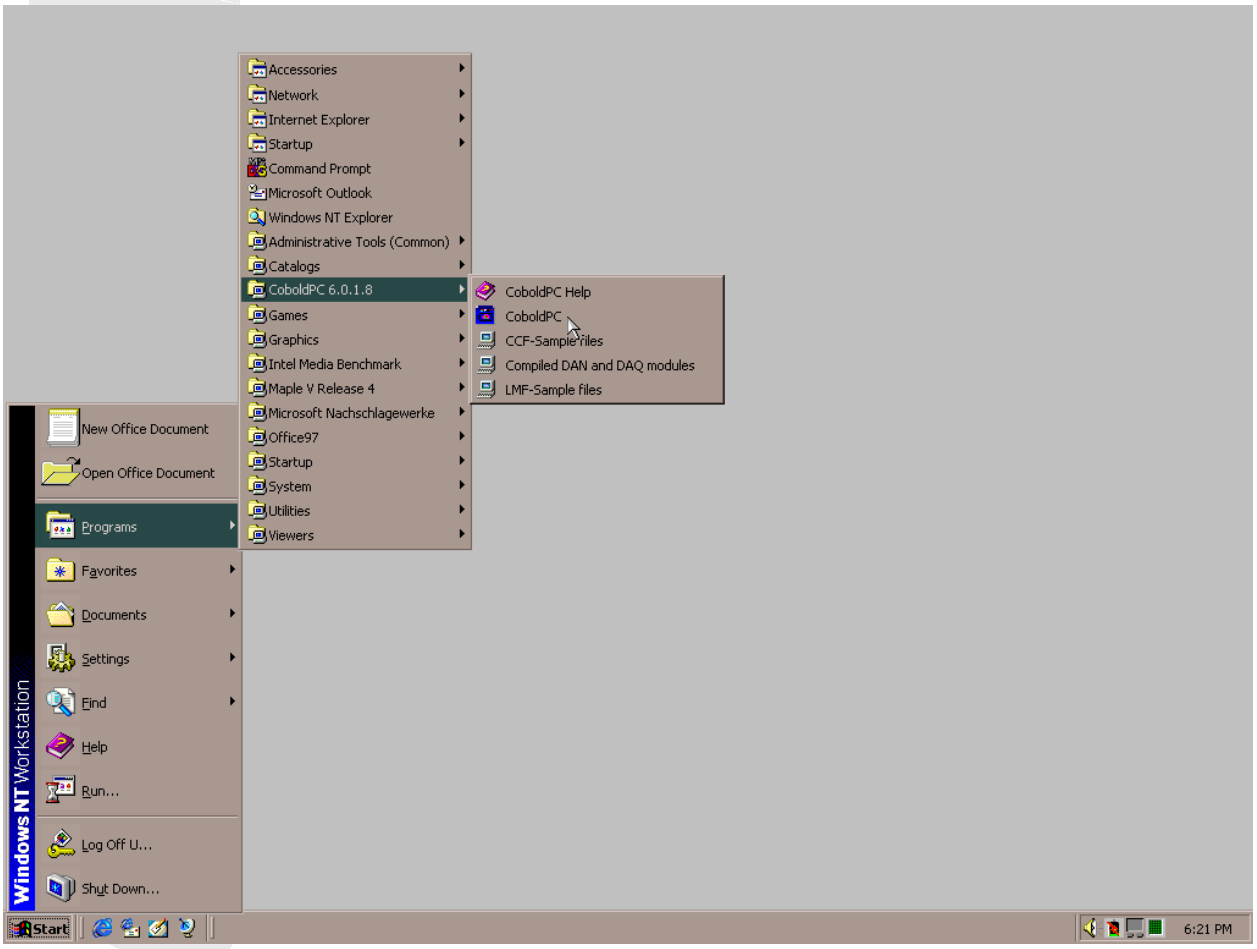
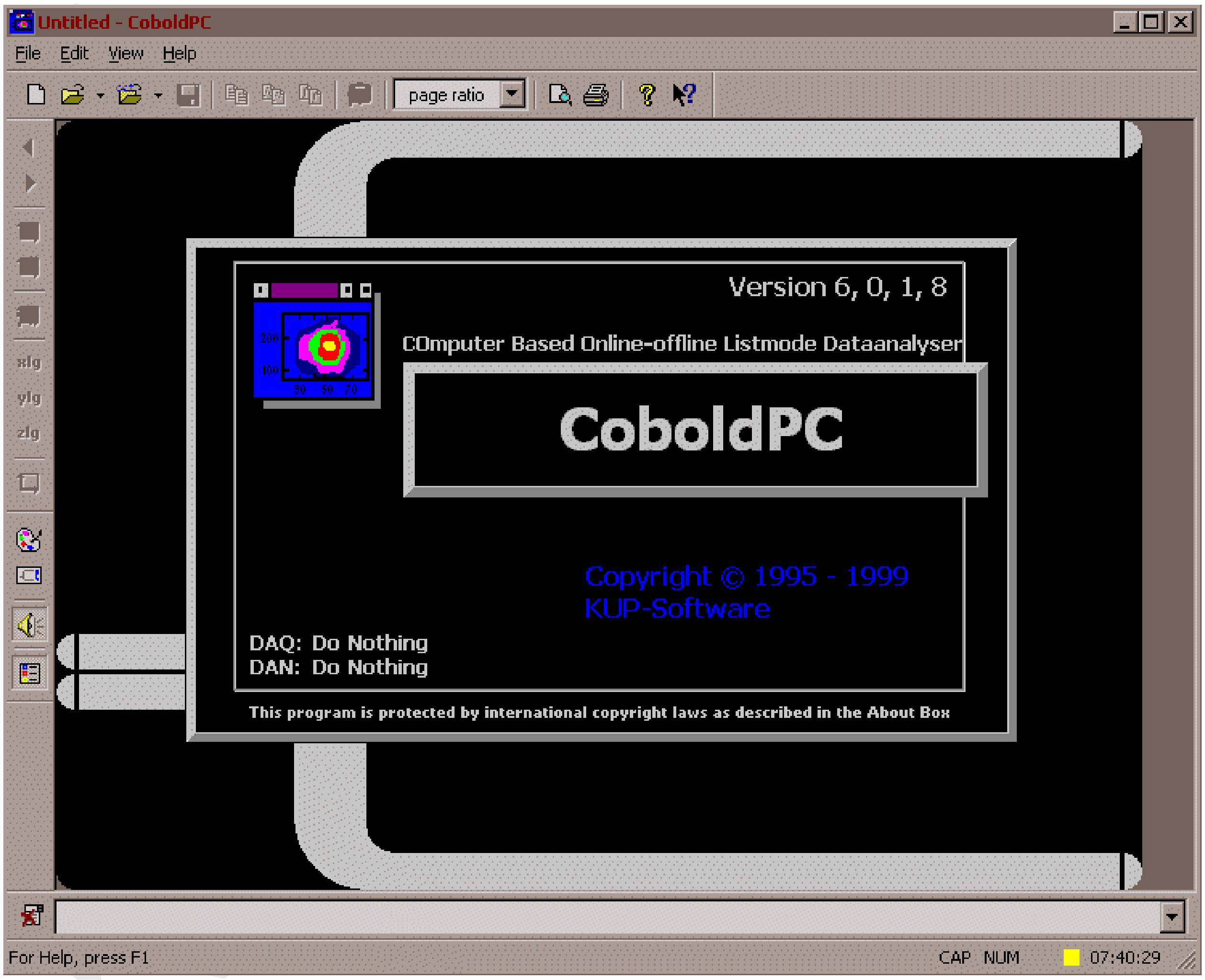


Figure 3: Start CoboldPC program from Start-button under Program-CoboldPC group

The opening window of **CoboldPC** is shown in Figure 4.



Version 6, 0, 1, 8

COmputer Based Online-offline Listmode Dataanalyser

CoboldPC

Copyright © 1995 - 1999
KUP-Software

DAQ: Do Nothing
DAN: Do Nothing

This program is protected by international copyright laws as described in the About Box

For Help, press F1 CAP NUM 07:40:29

Figure 4: Startup window of CoboldPC

2. Installation

2.1. System requirements

The minimum system requirements to run **CoboldPC** are:

- Standard Intel (Pentium) based Computer
- 64MByte Memory
- Windows 95/98 or WindowsNT 4.0/Windows 2000 / Windows XP
- Internet Explorer 3.0 or higher
- latest service releases for Windows95/98 or service packs for Windows NT/XP

We recommend the following system

- Standard Intel Pentium II or Pentium III based Computer with >400MHz
- >=256Mbyte memory
- Windows XP with all available service packs

It is possible that **CoboldPC** will not run on Intel compatible PCs. (For example special japanese PCs)

2.2. Windows NT/Windows 2000 / Windows XP

Be sure you logged on to an account related to the administration group (i.e. ADMINISTRATOR).

- switch to the main directory on the **CoboldPC** CD.
- run the setup.exe program and follow the instructions on screen.
- during the setup of **CoboldPC** (not IE5) there will be a possible message that tells you to reboot. Do not reboot at this point. Continue the installation. After the installation if finished you should reboot the computer manually.

After the program is installed you can run CoboldPC from any standard user account.

If routine fails please contact software.development@roentdek.com .

2.3. After installation requirements

Copy your specific DAN.dll-and DAQ.DLL version you want to use to the main **CoboldPC**-folder. You will find available precompiled versions of these DLLs in the **DAN and DAQ** directory that is located in your installation directory. If you have ordered special versions of these DLLs you will find these in the **DAN and DAQ\ SPECIAL\…** directory.

Note that you have to copy these specific DLLs. After installation there are only dummy DLLs installed in the installation directory to allow immediate start of the **CoboldPC** program.

3. The structure of CoboldPC

The program package of **CoboldPC** consists of a main routine where the general data I/O infrastructure in Windows XP/NT/98/95 are imbedded. A large number of data-treatment, storing and display routines are accessible. It contains exchangeable subroutines that can address different data I/0 hardware (DAQ.dll) and allows the advanced user to compile custom DAN.dll-routines. When you start CoboldPC.exe it will automatically load the DAQ.dll and DAN.dll subroutines. Now your analysis session may begin. The main constituents of a CoboldPC analysis session are:

- Coordinates
- Spectra
- Conditions
- Parameters

These objects have to be defined now after starting the main program by typing the proper commands in the command bar, which is located at the bottom of the **CoboldPC** window. This list of commands is usually written in a script file. The script files of **CoboldPC** have the extension .ccf.

3.1. Coordinates

A "coordinate" is simply a variable. CoboldPC uses a list of these variables for the data transfer between the main program and the DAQ.dll and the DAN.dll. The first part of this list is filled by the DAQ.dll. The DAQ.dll reads out the hardware (e.g. a TDC) and writes this data into the first part of the list of coordinates. Thus you have to define at least as many coordinates as hardware channels are read out by the DAQ.dll. The lower part of the list will be filled by the DAN.dll. Here all the results of the calculations inside the DAN.dll are stored. All coordinates can be displayed in spectra.

The syntax for defining the coordinates named e.g. x,y,z is:

```
coordinate x
coordinate y
coordinate z
```

or

```
coordinate x,y,z
```

This list of coordinates is one to one related to the "`pEventData`"-array that you will notice if you look at the source code of a DAN.dll. Actually, this array is used inside the DAQ.dll and DAN.dll whenever the list of coordinates must be accessed. E.g. this is done twice inside the DAN.dll. First, when the raw data is read inside the DAN.dll and second when the results of the calculations in the DAN are written back into the Coordinate list.

3.2. Spectra

A spectrum is a 1- or 2-dimensional histogram. The required memory is allocated in the RAM of the PC. You need to define each spectrum with a set of command parameters (attributes for this spectrum) that define for example the size and the coordinate you want to visualise. In each hardware read-out cycle the spectra will be filled with new data. This happens in the background. The spectra must be manually updated in order to see the new entries.

The syntax for a 1-dimensional spectrum is:

```
define1 0,1000,5,x,,x-Axis,always,X-Spectrum
```

(the double-comma between x and x-Axis is correct)

which defines a spectrum ranging from channel 0 to channel 1000 with a bin size of 5 sorting coordinate x without any condition (always) (see next subsection), with the Spectrum name „X-Spectrum". The title of the x-axis is "x-Axis". It is possible to define a weight-parameter which can be controlled by the DAN.dll. However, if the standard weigh-parameter of 1.0 is used then this option can be left out („)

The syntax for a 2-dimensional spectrum is:

```
define2 0,1000,5,x,x-Axis,0,1000,5,y,y-Axis,,always,X-Y-Spectrum
```

sorting the x-coordinate on the x-axis and the y-coordinate on the y-axis, both ranging from 0 to 1000 with a bin size of 5 without condition.

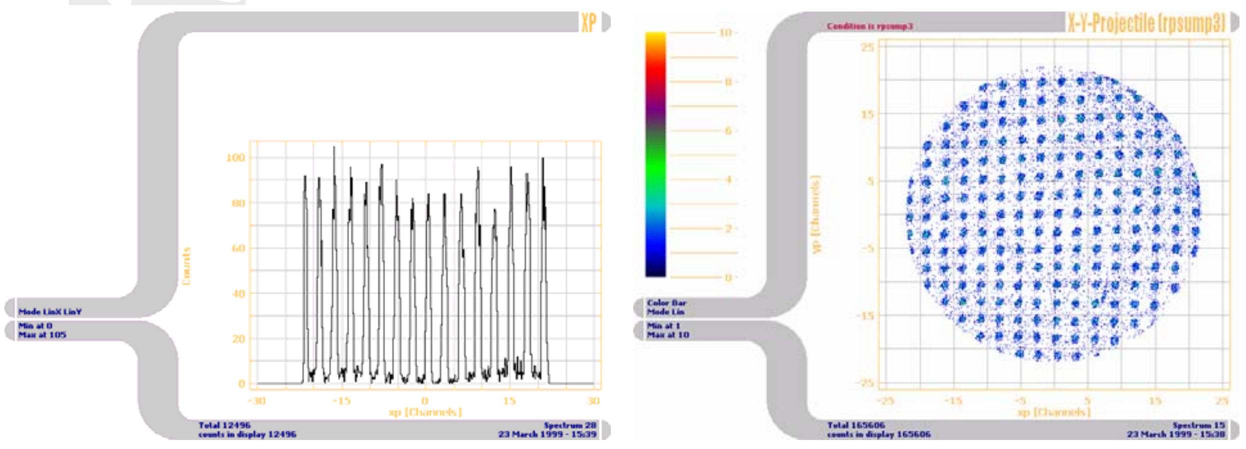


Figure 5: Example for a 1- and a 2-dimensional spectrum

3.3. Conditions

A condition is a one- or multidimensional restriction which is preventing data to be filled into a certain spectrum (defined with that condition as command parameter) if the condition is not fulfilled. Conditions are either single restrictions on a specific coordinate or boolean combinations of two such conditions. The syntax is:

condition x,100,200,x-cond

for a restriction testing the coordinate "x" ranging from channel 100 to 200 with the condition name "x-cond". If for an event the value of coordinate "x" is within that window, the event will be filled into the spectra with the respective condition, otherwise it will be omitted.

Thus defining a spectrum:

define2 0,1000,5,y,y-axis,0,1000,5,z,z-axis,,x-cond,Y-Z-Spectrum

will result in a 2 dimensional spectrum showing the y- and the z- coordinate under the condition that the x-coordinate was between 100 and 200.

The syntax for boolean combinations is:

condition x-cond,and,y-cond,xy-cond

for an "and" combination. Also "or", "not" and "xor" are allowed combinations.

3.4. Parameters

A Parameter is a fixed number which can be set during running the analysis session and will be used within the DAN.dll. This feature enables for example changing a calibration factor or something similar in the analysis without compiling the DAN.dll again. The syntax is

parameter 7,389.56

which gives the parameter 7 the value 389.56. Please start your custom parameters not before index 100, since the first 99 parameter are usually reserved for the communication with the hardware (in the DAQ.dll, such as I/O addresses) and should not be used in the DAN.dll.

The definition of the parameters and coordinates are interlinked with the "pParameters"-array and the "pEventData"-array in the DAQ- and DAN.dll-subroutines. See the example source code for more explanation.

For each **CoboldPC** session the appropriate set of coordinates, control parameters, (condition, and spectra) must be entered from the command line of **CoboldPC** or by loading a **CoboldPC** command file (.ccf-file) with the "exe" command e.g., type 'exe batchfile.ccf'.

To ease the startup and the operation of **CoboldPC** at least one CCF is predefined according to your specified requirements and you received an appropriate startup set of files.

In the LMF-Sample folder you will find a CCF, LMF and two DLLs. You may start the COBOLDPC.exe in this directory and start your first CoboldPC session.

4. The command level

All commands in a **CoboldPC** session have to be entered via the command line or the menu. To ease up the input via command line one can use predefined batchfiles (ext. ccf) with the exe command. If a started batchfile is finished a certain tone sequence will sound. All defined **CoboldPC** commands can be found in the help-file. To access the help menu press F1 or type “help”.

It is not necessary to type the full command name. One needs only to type as many characters to make the beginning unique with respect to other **CoboldPC** commands. If too few characters are given and the command is ambiguous **CoboldPC** will execute the command which is first in alphabetical order.

Example: **`shift`** and **`show`**

Batchfiles can also be addressed without the exe command just by typing the filename (without extension) in the command file. Note that there can be a name conflict with predefined commands.

If a certain character combination is not found in the list of commands or as batchfile in the main folder a certain two-tone combination will sound.

Most commands require one or more command parameters, e.g. ViewSpectrum requires a spectrum number.

A row of command parameters is separated from the command by a space and from each other by comma. The order of the command parameters is defined for each command (see helpfile). Some parameters are always required with the command (as the spectrum number after ViewSpectrum), some are optional as an input of xlow and xhigh after ViewSpectrum n,... (n is the spectrum number here).

If a required parameter is not supplied a Windows menu box will open and require the command parameter input. This is not the case when optional parameters are omitted, then the program uses default values or values given before.

Sometimes a command requires the setting of markers on the displayed spectrum with the mouse. Before the marks are not set the command line is disabled.

Note: in rare cases after mouse operation the command line will be disabled although there is no obvious reason. In this case click on the command line and the blinking command marker indicates that the command line is enabled again.

With arrow up/arrow down you may scroll to the previous command list, you also find the last typed commands in a drop down list (right button next to the command line).

When you use a commands in a batch file that requires for example mouse operation or starts a subroutine you have to follow this command by the wait command. Then the execution of the batch-file is paused until the previous command operation is finished properly. Examples:

Cursor

Marker

IntegrateSpectrum

5. Overview of commands

The following list of **CoboldPC** most frequently used commands gives an overview of the functionality of **CoboldPC**. Please refer to the build-in help for more information.

5.1. General and most frequently used commands

`exe`	executes a command file script
`new hardware`	prepares for starting an acquisition
`start`	starts the acquisition
`pause`	pauses the acquisition (for starting again use the start command)
`stop`	stops the acquisition (for starting again use the new hardware and the start command)
`clear all`	clears the contents of all spectra (instead of all, a certain spectrum number is also possible to clear only one spectrum contents)
`restart`	deletes all coordinates, parameters, conditions and spectra for a total reset
`show status`	shows the status report window
`help`	shows the help file
`wait`	pauses the batch file until the previous command operation is finished

5.2. Data Acquisition Handling Commands

`NewAcquisition`	Prepare CoboldPC for data acquisition.
`PauseAcquisition`	Pauses the data acquisition.
`RewindListModeFile`	Set the read file pointer to the first data in ListModeFile.
`StartAcquisition`	Starts (restarts) the data acquisition.
`StopAcquisition`	Stops data acquisition and closes ListModeFile.
`Wait`	Wait command to block inputs by mouse or keyboard.

5.3. File Handling Commands

`Restart`	Restarts with an empty CoboldPC document.
`Open`	Opens an existing document.
`Execute`	Execute a command file.
`Save`	Saves an opened document using the same file name.
`Save As`	Saves an opened document to a specified file name.
`Print`	Prints a document.
`Print Preview`	Displays the document on the screen as it would appear printed.
`Print Setup`	Selects a printer and printer connection.
`Preferences`	Define global CoboldPC settings
`Exit`	Exits Cobold.

5.4. Definition and Delete Commands

`Condition`	Defines a condition.
`Coordinate`	Defines coordinate(s).
`Define1DimensionalSpectrum`	Defines a 1 dimensional spectrum.
`Define2DimensionalSpectrum`	Defines a 2 dimensional spectrum.
`DefineExperiment`	Define experiment information.
`Delete`	Delete function.
`Grid`	Toggle grid on and off.
`Marker`	Defines a marker in a 1- or 2-dimensional spectrum.
`Mode`	Defines lin- or log-modes for the axes.
`Mode2D`	Defines the display type of a 2-dimensional spectrum.
`Parameter`	Defines the value of a specified parameter.
`SetAxisText`	Define the axis text.
`SetPath`	Define default paths for file io.

5.5. Spectrum handling Commands

`CalibrateSpectrum`	Calibrate a spectrum.
`ClearSpectrum`	Clear one or more spectra.
`CopySpectrum`	Copy a spectrum.

`CutOffNegativeValues`	Set all negative values in spectrum to zero.
`ExpandSpectrum`	Expand a spectrum.
`Remove`	Remove additional data from a spectrum
`SetChannelToValue`	Set a channel in a spectrum to a specific value.
`ShiftSpectrum`	Shift the spectrum.
`ZeroSpectrum`	Set a region in a spectrum to zero.

5.6. Display Commands

`CopyGraphics`	Copy displayed spectrum graphic to clipboard.
`CopyASCII`	Copy displayed spectrum channel values to clipboard.
`CopyIntegrationData`	Copy displayed spectrum integration information to clipboard.
`Cursor`	Display a cursor marker without storing data.
`ExportASCII`	Export displayed spectrum channel values to a file.
`ImportASCII`	Import channel values from a file to the displayed spectrum.
`NextSpectrum`	Display the next available spectrum.
`NoDisplay`	Removes any spectrum from the display area.
`OverlaySpectrum`	Overlay a spectrum to the displayed one.
`PasteASCII`	Copy channel values in ASCII format from the clopboard to the displayed spectrum.
`PreviousSpectrum`	Display the previous available spectrum.
`Show`	Show the specified lists.
`UpdateSpectrum`	Update the display of the spectrum being viewed.
`ViewSpectrum`	View the specified spectrum.

5.7. Mathematical Commands

`AddConstant`	Add a constant value to a spectrum.
`AddSpectrum`	Add two spectra.
`BinomialSmooth`	Perform a binomial smooth.
`ContourSmooth`	Smooth a contour plot.
`DivideConstant`	Divide spectrum by a constant value.
`DivideFunctionQ`	Multiply all channels by X^(1/Q).
`DivideSpectrum`	Divide two spectra.
`FitSpectrum`	Fitting Spectrum Data
`GaussSmooth`	Perform a smooth using a gaussian distribution.
`GP1CORRECT`	Correct the differential nonlinearity of the GP1-TDC
`IntegrateSpectrum`	Integrates a defined area in the displayed spectrum.
`MeanSmooth`	Perform a mean smooth.
`MultiplyConstant`	Multiply a spectrum by a constant value.
`MultiplyFunctionQ`	Multiply all channels by x^Q.
`MultiplySpectrum`	Multiply two spectra.
`ProjectSpectrum`	Makes a projection.
`RotateSpectrum`	Rotate a two dimensional spectru.
`SubtractConstant`	Subtract a constant value from a spectrum.
`SubtractFit`	Subtract the last fit from the given spectrum.
`SubtractSpectrum`	Subtract two spectra.

6. Getting Started

You may now start a **CoboldPC** session either in the example folder or in the main **CoboldPC** folder for example with your own hardware (make sure the hardware is working properly and data is coming in).

You may change, remove or add spectra or condition definitions and observe the effect of your changes of the ccf-file. If you change parameters make sure you don't supply parameter combinations that are not in accordance with the requirements of the DAN.dll or DAQ-routines.

The spectrum numbers will be attributed to the spectra in the order they have been defined.

Note that each spectrum definition will allocate a certain portion of the PC-RAM according to the number of channels and bin-size. Especially 2D spectra might require a large amount of RAM. As the RAM is limited you should choose spectra definitions with minimal size and tolerable bin-size (i.e. spectral resolution) for your respective information needs from this spectrum. The save command will store all spectra information (definition and actual content).

You may store graphics or ASCII of single spectra by clicking with the right mouse button on the displayed spectra (follow pop-up menu instructions). Similar you may import ASCII in proper format to predefined spectra. You may also print a displayed spectrum, colour or gray mode are optional (see help file: **`mode2D`**)

Note again the difference between storing the acquired data in CoboldListMode format (.lmf) and CoboldDumpFile format (.dcf):

The .lmf will store the hardware data in a list if you have defined a filename as parameter in the **`new`** command. This file contains the basic information content of your hardware acquired data. To extract the information you run a **CoboldPC** session with defined spectra and condition, on-line (while acquiring and storing the data in .lmf) or off-line (when analyzing the data in the list-mode file afterwards). Then you eventually store the **CoboldPC** session with the spectra content, etc. to disc. This is a "snapshot" of the data treatment as performed so far. From this save you will be able to do fit routines or spectra calculus later, but there is no way to do changes of conditions for spectra or parameters for the analysis from this CoboldDumpFile.

To re-run the data analysis offline with all options you need the .lmf-file which was written during data acquisition.

7. Building your custom DAN.dll

For most applications the standard DAN.dll subroutine provided with **CoboldPC** is sufficient for most data acquisition and analysis needs.

However for special and more flexible DAN.dll it is possible for the user to adjust the DAN.dll according to his needs by compiling a new DAN.dll.

The main aim of changing the code usually is to define additional coordinates that are computed from other coordinates (for example to make an R/Phi representation of an acquired detector image). Make sure that you adjust your .ccf according to newly defined control parameters or coordinates. The newly defined coordinates will be treated as the other predefined coordinates, i.e. they can be visualized in spectra in on-line and off-line analysis.

Note that in the .lmf only those coordinates defined by the hardware are stored (those containing the complete hardware information). So you may address a given .lmf-file with different DAN.dll, as long as your DAN.dll complies with the DAQ (the control parameters and coordinate definition required by the hardware must be consistent). To use your self-compiled you must copy your new DAN.dll to the main **CoboldPC** folder.

7.1. Compiler Requirements

To be able to build your own DAN.dll you need to fulfill the following software requirements

- MS Visual C++ 6.0 or higher
- (MS Fortran Power Station 4.0 or DEC Visual Fortran 6.0 if you want to program in Fortran)

To build your own DAQ.dll you need to fulfill the following software requirements

- MS Visual C++ 6.0 or higher

For these programming languages you will find ready to use project files in the source folder located in your **CoboldPC** installation directory. (only available if this option was selected during the installation of **CoboldPC**)

Note that the compiler is not part of the program package and the support from **RoentDek** in building your own DAN.dll can only be directive. There are well documented program examples included in the program package, so a medium skilled programmer will be able to compile his own .dll-file if the hardware and software requirements are met. Usually the DAN.dll provided by us is written in C++.

7.2. How to change the source code of the DAN

If you use the Microsoft Visual Studio 6.0 you should open the file "CDAN.dsw". In the case of Microsoft Visual Studio 7.0 or higher please open the file "CDAN.sln". In the compiler environment open the file "SoftwareDependend.cpp".
In this file go to the line "`#include "StandardDAN.cpp"`" and right click on the file "StandardDAN.cpp". A drop down menu will open up. Now chose "Open Document StandardDAN.cpp". Now you have the source code of the DAN on the screen.
When compiling the DAN please make sure to compile in "Release mode" and not in "Debug mode". The debug version of a program is always much slower than the release version.

In the following it is assumed that the MS-Visual-C++-Compiler is used. If you want to program in Fortran please refer to the Fortran source compiler project which is also located in the source folder or contact **RoentDek**.

7.3. General structure of a DAN

In general a DAN contains two important sub-functions:

```
AnalysisInitialize(CDoubleArray *pEventData,CDoubleArray *pParameters, CDoubleArray *pWeighParameter)
```

and

```
AnalysisProcessEvent(CDoubleArray *pEventData,CDoubleArray *pParameters, CDoubleArray *pWeighParameter)
```

"AnalysisInitialize" is called once at the beginning of the data acquisition when the "new" command is executed in **CoboldPC**. In this sub-function some global variables are initialized.

"AnalysisProcessEvent" is called for each event. Here all the calculations on the data are performed.

All data transfers between the DAN, DAQ and **CoboldPC** is done via three arrays. These arrays are one dimensional arrays of type "double". However the elements can not be accessed via the usual syntax "`array[index]`". Instead the two commands are "`->SetAt(index)`" and "->`GetAt(index)`".

1) `pEventData-array`

This array is one to one related to the list of coordinates which is defined in the CCF-file. **CoboldPC** reads the data either online from the TDC or offline from the disc and writes this raw data into the first part of the Coordinate-list (which is the same as the pEventData-array). **CoboldPC** then enters the sub-function "AnalysisProcessEvent". Here all the calculations are done. The results of these calculations are written back into the same array pEventData but below the raw data.

2) `pParameters-array`

This array is filled by **CoboldPC**. When a ccf-file is executed **CoboldPC** takes all Parameter definitions in the ccf-file and fills the values into "`pParameters`". Please note that the first parameter in the CCf-file has the index 1 whereas the first element in "`pParameters`" has index 0 (when programming in C; in Fortran the first index is 1). This is due to the indexing convention for arrays in C (or Fortran).

3) `pWeightParameter-array`

This array can be used to transfer weight-parameters between the DAN and **CoboldPC**. This is needed if some spectra have to be filled with other values than 1.0.

7.4. Sample for a User defined DAN.dll written in C++

This code is based on the Standard DAN which is provided together with **CoboldPC**. However, the code has been simplified in some parts: It works only with the TDC8 and with a standard rectangular delay line detector (DLD).

Naming convention:
In the following code variables beginning with "i" are of type integers, "d" are of type double and "p" are pointers.

```
#define DAQ_VERSION4      20020408
#define DAQ_VERSION       20020408
#define LMF_VERSION       5

#define DAN_VERSION       20020513
#define DAN_SUBVERSION    0x0007

////////////////////////////////////////////////////////////////////////////
// global variables definitions:
LARGE_INTEGER    lipTimeStamp;     // TimeStamp time information
int      ipTimeStamp;              // TimeStamp information type
double   dpTimeScaling;            // Time Scaling (ticks per second)
CTime    ctpLMFStartTime;          // Start Time of LMF
int      iDAQ_ID;                  // parameter 8
double   dpTDCResolution;          // parameter 20
int      ipTDCDataType;            // parameter 21
int      ipNumberOfChannels;       // parameter 32
int      ipNumberOfHits;           // parameter 33
int      ipStartDAQData;           // parameter 105
int      ipStartDANData;           // parameter 106
int      UseHit;                   // parameter 107
double   dpTPCalX;                 // parameter 110
double   dpTPCalY;                 // parameter 111

double   dStartDanCoordinates;
int      iDataOffset;
int      iIndexDataOffset;
int      ipStartDAQTDCData;

double   EventCounter;             // EventCounter for the data in LM-file


//////////////////////////////
// AnalysisGetInformationString
//////////////////////////////
// is called during startup procedure of CoboldPC
// return value is a string that will be displayed in
// the splash box and the about box
CString myInfoString;
CDAN_API LPCTSTR AnalysisGetInformationString()
{
        myInfoString.Format("RoentDek Simple V4 (Version %08d - %04d -
%04d)",DAN_VERSION,DAN_SUBVERSION,LMF_VERSION);
        return LPCTSTR(myInfoString);
}
```

```
/////////////////////
// AnalysisInitialize
/////////////////////
// is called when new command is executed
// return value is false -> error during initialize
//                   true  -> initialize ok

CDAN_API BOOL AnalysisInitialize(CDoubleArray *pEventData,CDoubleArray *pParameters, CDoubleArray
*pWeighParameter)
{
        _dOldEventTime = 0.0;
        _dStartEventTime = 0.0;

        // transfer parameters
        //     in the GetAt lines you'll find the following correction values
        //     0.1 for float to int converion
        //     -1 if parameter is for indexing
        ipTimeStamp       = (int)(pParameters->GetAt(1)+0.1);                // parameter 2
        dpTimeScaling     = pParameters->GetAt(4);                           // parameter 5
        ctpLMFStartTime   = CTime((time_t)((int)(pParameters->GetAt(6)+0.1)));        // parameter 7
        iDAQ_ID           = (int)(pParameters->GetAt(7)+0.1);                // parameter 8
        dpTDCResolution   = pParameters->GetAt(19);                          // parameter 20
        ipTDCDataType     = (int)(pParameters->GetAt(20)+0.1);               // parameter 21
        ipNumberOfChannels = (int)(pParameters->GetAt(31)+0.1);              // parameter 32
        ipNumberOfHits    = (int)(pParameters->GetAt(32)+0.1);               // parameter 33
        ipStartDAQData    = (int)(pParameters->GetAt(104)+0.1) - 1;          // parameter 105
        ipStartDANData    = (int)(pParameters->GetAt(105)+0.1) - 1;          // parameter 106
        dpTPCalX          = pParameters->GetAt(109);                         // parameter 110
        dpTPCalY          = pParameters->GetAt(110);                         // parameter 111

        EventCounter = 0;

        iDataOffset = 0;
        iIndexDataOffset = 2;

        int iDataFormat = (int)(pParameters->GetAt(39)+0.1);
        switch(iDataFormat)
        {
        case LM_BYTE:
        case LM_SBYTE:
                break;
        case LM_SHORT:
        case LM_SSHORT:
        case LM_CAMAC:
                if(ipTimeStamp == 0) { iDataOffset = 0;    iIndexDataOffset = 4; }
                if(ipTimeStamp == 1) { iDataOffset = 2;    iIndexDataOffset = 2; }
                if(ipTimeStamp == 2) { iDataOffset = 4;    iIndexDataOffset = 0; }
                break;
        case LM_LONG:
        case LM_SLONG:
        case LM_FLOAT:
                if(ipTimeStamp == 0) { iDataOffset = 0;    iIndexDataOffset = 2; }
                if(ipTimeStamp == 1) { iDataOffset = 1;    iIndexDataOffset = 1; }
                if(ipTimeStamp == 2) { iDataOffset = 2;    iIndexDataOffset = 0; }
                break;
        case LM_DOUBLELONG:
        case LM_SDOUBLELONG:
        case LM_DOUBLE:
                if(ipTimeStamp == 0) { iDataOffset = 0;    iIndexDataOffset = 1; }
                if(ipTimeStamp == 1) { iDataOffset = 1;    iIndexDataOffset = 0; }
                if(ipTimeStamp == 2) { iDataOffset = 1;    iIndexDataOffset = 0; }
                break;
        default:
                break;
        }

        // correct start address of DAQ coordinates:
        // start address of DAQData in coordinate block in CCF-file:
        if(ipStartDAQData < 0)
                ipStartDAQTDCData = iDataOffset;
        else    ipStartDAQTDCData = ipStartDAQData + iDataOffset;

        pParameters->SetAt(104,double(ipStartDAQTDCData-iDataOffset)+1);

        // start address of DANData in coordinate block in CCF-file:
        if(ipStartDANData < 0) {
                ipStartDANData = ipStartDAQTDCData + (ipNumberOfChannels * (ipNumberOfHits+1));
        }

        pParameters->SetAt(105,double(ipStartDANData)+1);   // write information back to parameter 106

        return true;
}




//////////////////////
// AnalysisProcessEvent
//////////////////////
// is called when “new” command is executed
//CDAQ_API LARGE_INTEGER DAQTimeStamp;
//__declspec(dllimport) int DAQTimeStamp;
__declspec(dllimport) int DAQTimeStamp;
```

```
CDAN_API void AnalysisProcessEvent(CDoubleArray *pEventData,CDoubleArray *pParameters, CDoubleArray
*pWeighParameter)
{
        double   AbsoluteEventTime;        // ns since start
        double   DeltaEventTime;           // This Time - PreviousTime
        double   x1,x2,y1,y2;              // TDC data
        double   x,y;
        double   sumx,sumy;
        int      n1,n2,n3,n4;              // Number of hits for channels 1 to 4
        int      coordinate_address;


        // correct position of Event Data addresses due to time stamp information
        // at the beginning of the coordinate list in ccf-file.
        CorrectEventDataPosistion(pEventData,iIndexDataOffset,ipStartDANData);

        // get time-stamp information
        if(iDataOffset)
        {
                AbsoluteEventTime = GetEventTime(pEventData,pParameters);            // AbsoluteEventTime
                DeltaEventTime = GetDeltaEventTime(pEventData,pParameters);          // AbsoluteDeltaEventTime
        }

        EventCounter = EventCounter + 1;

        // Get Status Information

        // Extract TDC-data from the pEventData-array:
        UseHit = 1;
        coordinate_address = ipStartDAQTDCData+0*(ipNumberOfHits+1);
        n1 = (int)( pEventData->GetAt(coordinate_address) + 0.1 );

        coordinate_address = ipStartDAQTDCData+1*(ipNumberOfHits+1);
        n2 = (int)( pEventData->GetAt(coordinate_address) + 0.1 );

        coordinate_address = ipStartDAQTDCData+2*(ipNumberOfHits+1);
        n3 = (int)( pEventData->GetAt(coordinate_address) + 0.1 );

        coordinate_address = ipStartDAQTDCData+3*(ipNumberOfHits+1);
        n4 = (int)( pEventData->GetAt(coordinate_address) + 0.1 );

        coordinate_address = ipStartDAQTDCData+0*(ipNumberOfHits+1) + UseHit;
        x1 = pEventData->GetAt(coordinate_address);

        coordinate_address = ipStartDAQTDCData+1*(ipNumberOfHits+1) + UseHit;
        x2 = pEventData->GetAt(coordinate_address);

        coordinate_address = ipStartDAQTDCData+2*(ipNumberOfHits+1) + UseHit;
        y1 = pEventData->GetAt(coordinate_address);

        coordinate_address = ipStartDAQTDCData+3*(ipNumberOfHits+1) + UseHit;
        y2 = pEventData->GetAt(coordinate_address);

        // convert to nanoseconds
        x1 = x1 * dpTDCResolution;
        x2 = x2 * dpTDCResolution;
        y1 = y1 * dpTDCResolution;
        y2 = y2 * dpTDCResolution;

        x = dpTPCalX*(x1 - x2)/2.0;
        y = dpTPCalY*(y1 - y2)/2.0;

        // sums and differences
        sumx = x1 + x2;
        sumy = y1 + y2;

        ////////////////////////////////////////////
        // write all data back to coordinates in ccf
        ////////////////////////////////////////////
        pEventData->SetAt(ipStartDANData+0,AbsoluteEventTime);
        pEventData->SetAt(ipStartDANData+1,DeltaEventTime);
        pEventData->SetAt(ipStartDANData+2,EventCounter);
        pEventData->SetAt(ipStartDANData+3,n1);
        pEventData->SetAt(ipStartDANData+4,n2);
        pEventData->SetAt(ipStartDANData+5,n3);
        pEventData->SetAt(ipStartDANData+6,n4);
        pEventData->SetAt(ipStartDANData+7,x1);
        pEventData->SetAt(ipStartDANData+8,x2);
        pEventData->SetAt(ipStartDANData+9,y1);
        pEventData->SetAt(ipStartDANData+10,y2);
        pEventData->SetAt(ipStartDANData+11,x);
        pEventData->SetAt(ipStartDANData+12,y);
        pEventData->SetAt(ipStartDANData+13,sumx);
        pEventData->SetAt(ipStartDANData+14,sumy);
}



//////////////////////
// AnalysisFinalize
//////////////////////
// is called when analysis is stopped (not paused!)
CDAN_API void AnalysisFinalize(CDoubleArray *pEventData,CDoubleArray *pParameters, CDoubleArray
*pWeighParameter)
```

```
{
}
```

7.5. *Sample ccf-file for the above sample-DAN*

The following ccf-file works with the above sample DAN. Please note how the list of Parameters and the list of Coordinates is one to one related to the arrays “pParameters” and “pEventData” in the above source code.

```
restart

Parameter 1,0          ; io-hardwareaddress of the ATMD-Boards
Parameter 2,2          ; save TimeStamp
Parameter 3,5          ; system Timeout Time in s

Parameter 14,0         ; trigger mode for common input

Parameter 30,30        ; Event Open Time in us for detecting an event.
Parameter 32,4         ; number of Channels (reread during offline)
Parameter 33,1         ; number of hits (reread during offline)

parameter 45,0x80      ; gate delay (in common start always use 0x80)
parameter 46,1080      ; gate open
parameter 47,0         ; write empty events
parameter 48,1         ; trigger at falling edge
parameter 49,0         ; trigger at rising edge


Parameter 105,0        ; Parameter 105 = Start of DAQ Data for DAN (Start Coordinate)
                       ; 0 = automatic
Parameter 106,0        ; Parameter 106 = Start of DAN Data (Start Coordinate)
                       ; 0 = automatic
Parameter 110,0        ; Parameter 110 = pTPCalX = Time to Point calibration factor for x (mm/ns)
Parameter 111,0        ; Parameter 111 = pTPCalY = Time to Point calibration factor for y (mm/ns)

; timestamp:
; ----------
Coordinate TRaw1,TRaw2,TRaw3,TRaw4

; TDC raw Data:
; -------------
Coordinate TDC1Hits,TDC1DataHit1
Coordinate TDC2Hits,TDC2DataHit1
Coordinate TDC3Hits,TDC3DataHit1
Coordinate TDC4Hits,TDC4DataHit1


; DAN results:
; ------------
Coordinate AbsoluteEventTime
Coordinate DeltaEventTime
Coordinate EventCounter
Coordinate n1,n2,n3,n4
Coordinate x1,x2
Coordinate y1,y2
Coordinate x,y
Coordinate sumx,sumy


; Spectra definitions:
; --------------------
define1 0,32,1,n1,,none,always,n1
define1 0,32,1,n2,,none,always,n2
define1 0,32,1,n3,,none,always,n3
define1 0,32,1,n4,,none,always,n4

define1 0,2500,1,TDC1DataHit1,,none,always,TDC1 Hit1
define1 0,2500,1,TDC2DataHit1,,none,always,TDC2 Hit1
define1 0,2500,1,TDC3DataHit1,,none,always,TDC3 Hit1
define1 0,2500,1,TDC4DataHit1,,none,always,TDC4 Hit1

define2 -300,300,1,x,,-300,300,1,y,,none,always,xy_plot

new
start
show status
```

A. Appendix – List Mode Data Format

The List-Mode data file consists of 2 sections. The first section is the data-header. The second is the data-area. The main header is constructed as follows.

```
unsigned int   iArchiveFlag = true;
unsigned int   LMDataFormat;
unsigned int   LMNumberOfCoordinates;
unsigned int   LMHeaderSize;
unsigned int   LMUserHeaderSize;
unsigned int   LMNumberOfEvents;
CTime          LMStartTime;
CTime          LMStopTime;
CString        LMVersionString;
CString        LMFilePathName;
CString        LMComment;
```

Unsigned int is an unsigned 32bit integer number.
CTime and CString are classes of the Microsoft Foundation Class 4.x of the Visual C++ Compiler of Microsoft.

LMDataFormat can have the following values:

```
// definitions for main program event data format
#define LM_BYTE                        1     //  8bit integer
#define LM_SHORT                       2     // 16bit integer
#define LM_LONG                        3     // 32bit integer
#define LM_FLOAT                       4     // 32bit IEEE float
#define LM_DOUBLE                      5     // 64bit IEEE float
#define LM_CAMAC                       6     // 24bit integer
#define LM_DOUBLELONG                  7     // 64bit integer
#define LM_SBYTE                       8     // signed 8bit integer
#define LM_SSHORT                      9     // signed 16bit integer
#define LM_SLONG                       10    // signed 32bit integer
#define LM_SDOUBLELONG                 11    // signed 64bit integer
#define LM_USERDEF                     -1    // user will handle the
                                             // reading
                                             // of file and filling of
                                             // EventData Array
                                             // except reading the main
                                             // header file
```

If the LMUserDefinedHeaderSize is not equal to zero then a user defined header will follow. The size of this data block is stored in the LMUserDefineHeaderSize variable of the main header block.

After this header block the event data follows as described in DAQ.DLL source code.