

REFLEXW data-format

One REFLEX raw-data profile consists of two files: the first is the data file with the extension DAT, the second the header file with the extension PAR. The name without the extension must be the same (e.g. YP000000).

The data files and header files always have to be seen in relation with each other. All the information of the header file is necessary for the correct reading of the data file. The two files are saved separately in order to facilitate a quick modification of the header files.

1. data file with the extension DAT or ??T:

Each data file consists of a number of traces which consist of a trace header and the trace samples. The number of traces (ScansMeasured) and the number of trace samples (MeasureSamples) are saved in the header file. The maximum number of traces is 256000. The maximum number of samples is 32768. The trace header and the trace samples are comprised in one record having the following structure (PASCAL example):

```
CONST
    MaxSampleNumber = 32768;

TYPE
    TraceArray = Array[0..MaxSampleNumber] of SmallInt;
    TraceSingleArray = Array[0..MaxSampleNumber] of Single;

TraceRecord =
    RECORD
        TraceNo          : Word;      { ** actual trace number ** }
        NoOfSamples      : Word;      { ** number of samples ** }
        IKomp            : Word;      { ** component for 3-comp. registration ** }
        EnsembleNo       : Word;      { ** Ensemble-number ** }
        CDPNo            : Word;      { ** CDP-number ** }
        ShotNo           : Word;      { ** Shot-number ** }
        GeophoneNo       : Word;      { ** Geophone-number ** }
        TraceMarker      : SmallInt;  { ** <> 0 : Marker ** }
        TraceTime        : Integer;   { *** time *** }
        TimeDel          : Single;    { ** timedelay ** }
        ShotElevation    : Single;    { ** shot elevation ** }
        Distance         : Single;    { ** distance ** }
        ShotOrt          : Array[1..2] of Single; { ** shot-coordinates ** }
        GeophoneOrt      : Array[1..2] of Single; { ** Geophone-coordinates ** }
        CDPOrt          : Array[1..2] of Single; { ** CDP-coordinates ** }
        RecElevation     : Single;    { ** actual elevation ** }

        TraceData       : TraceArray; { ** data ** }
        OR SingleTraceData: TraceSingleArray; { ** data ** }
    END;
```

Each data trace therefore consists of a trace header of constant size and the trace samples (SingleTrace), an array whose size is determined dynamically by the number of samples in the file header (MeasureSamples+1). The maximum number of samples per trace is 32768. The trace samples are either saved as 16 Bit SmallInt-Precision or 32 bit floating point format. The dataformat is given within the fileheader (see parameter format code in section 2). In Pascal the reading of a complete data trace is declared as follows:

```
VAR
    Trace : TraceRecord;
    OutputFile : File;
    MeasureSamples, Number, NumRead : Integer;
    InputSize : LongInt;

BEGIN
    MeasureSamples := 512; { for example }
    InputSize := SizeOf(Trace);
    InputSize := InputSize-2*(MaxSampleNumber-MeasureSamples);
    Assign(OutputFile, 'C:\REFLEX\DEMO\ROHDATA\TEST_000.DAT');
    Rewrite(OutputFile, InputSize);
    for Number := 1 to ScansMeasured do
    BEGIN
        With Trace do
        BEGIN
            { *** Define the traceheader *** }
            TraceNo := Number;
            NoOfSamples := MeasureSamples;
            ..... and so on
            { *** Define the trace data *** }
            for i := 1 to MeasureSamples do
                SingleTrace[i] := ????
            END;
            BlockWrite(OutputFile, Trace, 1);
        END;
    END;
    Close(OutputFile);

END;
```

2. Header files with the extension PAR or ??R

The names of these files only differ from the data files in their extension PAR instead of DAT for raw data files or ??R instead of ??T for processed files, respectively. Here information is stored concerning the whole file (e.g. profile coordinates, time increment etc.).

Since the internal structure of the headerfile is only of interest for users who want to read in data by own programs or create own data files (e.g. digital input of a signal trace for crosscorrelation), the file format (Type HeaderRecord) and the reading of the header file is explained with a small Turbo-Pascal program (please notice: SmallInt - 2 byte, Word - 2 Byte, Integer - 4 Byte, Single - 4 Byte):

```
Const
    MaxNumComments = 12;
    MaxNumberOfProcessingSteps = 10;

Type
    ColorArray = ARRAY[1..16,1..2] OF SmallInt;
    Processing = RECORD
        Sign : String[20];
        Parameters : Array[1..10] of SmallInt;
        SingleParameters : Array[1..5] of Single;
    END;

    ProcessingArray = Array[1..MaxNumberOfProcessingSteps] of Processing;
    HeaderRecord = RECORD
        ParmStrings : Array[1..20] of String[20];
        ParmIntegers : Array[1..16] of SmallInt;
        ParmLongInt : Array[1..2] of Integer;
        ParmSingles : Array[1..20] of Single;
        ParmColors : ColorArray; { 64 byte }
        ParmComments : Array[1..MaxNumComments] of String[20]; { 252 byte }
        ParmGainValues : Array[1..2,1..MaxGainValues] of Single; { 64 byte }
        ParmSingleDummy : Array[1..4] of Single; { 16 byte }
        ParmIntegerDummy : Array[1..20] of SmallInt; { 40 byte }
        ParmProcessingFlow : ProcessingArray; { 610 byte }
    END;

PROCEDURE GetParmRecord(DataHeader : HeaderRecord);

{ *** Procedure for reading the informations from the headerrecord *** }

VAR
    ihelp : Integer;

BEGIN
    With DataHeader do
        BEGIN
            ProjectName := ParmStrings[1]; { project name }
            ProfileDirection := ParmStrings[2]; { profile direction (X,Y oder Z) }
            ProfileConstant := ParmStrings[3]; { profile constant (X,Y oder Z) }
            MeasureArt := ParmStrings[4]; { measure art }
            ProfileCoord := ParmStrings[5]; { free }
            HelpString := ParmStrings[6]; { free }
            HelpString := ParmStrings[7]; { free }
            HelpString := ParmStrings[8]; { free }
            PlotArt := ParmStrings[9]; { plot mode e.g. POINT-MODE }
            WiggleArt := ParmStrings[10]; { WiggleArt(LINE,BLACK and so on) }
            AutoName := ParmStrings[11]; { automatic file name }
            DistanceDimension := ParmStrings[13]; { distance dimension }
            TimeDimension := ParmStrings[14]; { time dimension }
            ReceiverSequence := ParmStrings[15]; { receiver sequence (plot) }
            DistanceAxisName := ParmStrings[16]; { name - distance axis }
            TimeAxisName := ParmStrings[17]; { name - time axis }
            AmplitudeDimension := ParmStrings[18]; { amplitude dimension }
            MeasureSamples := ParmIntegers[1]; { number of samples }
            ScansMeasured := ParmLongInt[1]; { number of traces }
            WiggleBlack := ParmIntegers[3]; { blackening parameter in percent }
            WiggleInc := ParmIntegers[4]; { increment for wiggle-mode }
            PointXArea := ParmIntegers[5]; { range in x-direction for plotting in point-mode};
            HelpInteger := ParmIntegers[6]; { free }
            CoordNumber := ParmIntegers[7]; { profile number }
            WiggleClip := ParmIntegers[8]; { clipping parameter in percent }
            FormatCode := ParmIntegers[9]; { data format Code: 0: 16 bit small-integer; 1: 32 floating }
            AGCWindowPlot := ParmIntegers[10]; { AGC-window in samples }
            HeaderMarker := ParmIntegers[11]; { 0:datamarker, 1:headermarker }
            NumberOfGainValues := ParmIntegers[12]; { number of gain values }
            TraceIncrement := ParmSingles[1]; { trace increment }
            TimeIncrement := ParmSingles[2]; { time increment }
            TimeMeasured := ParmSingles[3]; { measured time }
            TimeBegin := ParmSingles[4]; { time begin }
```

```

MarkerIncrement := ParmSingles[5]; { free }
WiggleScale     := ParmSingles[6]; { scaling for wiggle mode }
ValueScaling    := ParmSingles[7]; { scaling for ASCII-conversion }
XCoordBegin     := ParmSingles[8]; { x-coordinate start of the profile }
YCoordBegin     := ParmSingles[9]; { y-coordinate start of the profile }
ZCoordBegin     := ParmSingles[10]; { z-coordinate start of the profile }
XCoordEnd       := ParmSingles[11]; { x-coordinate end of the profile }
YCoordEnd       := ParmSingles[12]; { y-coordinate end of the profile }
ZCoordEnd       := ParmSingles[13]; { z-coordinate end of the profile }
PointValueScale := ParmSingles[14]; { plot scaling }
ShotPosition    := ParmSingles[15]; { shot position }
ShotReceiverDistance := ParmSingles[16]; { shot receiver distance }
NominalFrequency := ParmSingles[17]; { nominal frequency }
ProfileIncrement := ParmSingles[18]; { increment between 2 parallel lines of a 3D-data block }
CMPBin          := ParmSingles[19]; { CMP-bin for the CMP-sorting }
OffsetBin       := ParmSingles[19]; { offset-bin for the CMP-sorting }

for ihelp := 1 to 16 do
BEGIN
  Colors[ihelp,1] := ParmColors[ihelp,1]; {start amplitude-color index}
  Colors[ihelp,2] := ParmColors[ihelp,2]; { end amplitude-color index}
END;
for ihelp := 1 to MaxNumbComments do
  Comments[ihelp] := ParmComments[ihelp]; { 10 comments, 20 char. each}
ProcessingFlow := ParmProcessingFlow; {processing flow }
end;

```

The reading (writing) of a header file is performed by using a blockread(blockwrite) command. For writing a header file only the most important "HeaderRecord-Parameters" have to be set. For writing a file the setting of the following parameters is necessary:

- a) ProfileDirection ParmStrings[2] := ProfileDirection;
- b) ProfileConstant ParmStrings[3] := ProfileConstant;
- c) MeasureSamples ParmIntegers[1] := MeasureSamples;
- d) TimeMeasured
- e) ScansMeasured
- f) TimeBegin
- g) TimeIncrement
- h) TraceIncrement
- i) XBegin-ZEnd

```

VAR
  Header : HeaderRecord;
  OutputFile : File;

BEGIN
  MeasureSamples := 512; { for example }
  Assign(OutputFile, 'C:\REFLEX\DEMO\ROHDATA\TEST_000.PAR');
  Rewrite(OutputFile, SizeOf(Header));
  { *** Define Header *** }
  With Header do
  BEGIN
    ParmStrings[2] := 'X';
    ParmStrings[3] := 'Y';
    .....
    ParmIntegers[1] := 512; { *** for example **** }
    ... and so on
  END;
  BlockWrite(OutputFile, Header, 1);
  Close(OutputFile);
END;

```