## ***File formats specification for GPR software from Malå GeoScience***

### **Rationale**

In order to maintain interoperability between the different software products developed by MGS we need to define a standard set of file formats. The following file formats should be used when saving GPR data. GPR data is here taken to mean only data directly related to the measurement, i.e., the radar data file (\*.RD3), the header file (\*.RAD), the GPS data file (at the moment \*.COR and \*.WSK) and the marker file (\*.MRK). Information such as filter settings, palette settings etc. is considered the responsibility of the applications and will not be transferred together with the data.

### **The data file (\*.RD3)**

Data are stored as signed 16 bit signed integers (short int) in a binary file. The traces are stored sequentially.

### **The header file (\*.RAD)**

The header file is a text file. At the time of writing of this document the specification is as shown below. If information needs to be added to the header file then this should be appended to the end of the file so that older software will not be confused. When changes are made this document should also be updated accordingly.

SAMPLES:402

FREQUENCY:6854.667343

FREQUENCY STEPS:65

SIGNAL POSITION:-0.004731

RAW SIGNAL POSITION:34875

DISTANCE FLAG:0

TIME FLAG:1

PROGRAM FLAG:0

EXTERNAL FLAG:0

TIME INTERVAL:2.000000

DISTANCE INTERVAL:0.000000

OPERATOR:

CUSTOMER:

SITE:

ANTENNAS:500 MHz shielded

ANTENNA ORIENTATION:NOT VALID FIELD

ANTENNA SEPARATION:0.180000

COMMENT:

TIMEWINDOW:58.646172

STACKS:2

STACK EXPONENT:1

STACKING TIME:0.008040

LAST TRACE:9

STOP POSITION:18.000000

SYSTEM CALIBRATION:0.0000022444

START POSITION:0.0000000000

SHORT FLAG:1

INTERMEDIATE FLAG:0

LONG FLAG:0

PREPROCESSING:0

HIGH:0

LOW:0

FIXED INCREMENT:0.0000000000

FIXED MOVES UP:0

FIXED MOVES DOWN:1

FIXED POSITION:0.0000000000

WHEEL CALIBRATION:2490.9000000000

POSITIVE DIRECTION:1

*Comments*

The trigger options section:

DISTANCE FLAG:0

TIME FLAG:1

PROGRAM FLAG:0

EXTERNAL FLAG:0

is an XOR set; only one of the flags can be set to TRUE at any time.

The same applies to the ramp options section:

SHORT FLAG:1

INTERMEDIATE FLAG:0

LONG FLAG:0

only one can be set to TRUE.

### **The GPS files (\*.COR, \*.WSK, \*.UTM)**

There are three valid GPS formats defined:

* GroundVision standard with extension “cor”
* WSKTrans free format with extension “fri”
* UTM coordinates with extension “utm”.

Changes to the file formats should preferably be made by adding new formats to the list so that third-party software that use our data (read REFLEX) does not encounter compatibility problems.

The GroundVision standard file format is simply a parsed version of the NMEA string written with tab separator as follows:

Trace number 🡪 date 🡪 time 🡪 latitude 🡪 ”N” 🡪 longitude 🡪 “E” 🡪 height above MSL 🡪 “M” 🡪 HDOP

Note that this format has now reverted to its old definition.

The WSKTrans free format is a special format compatible with WSKTrans, a coordinate transformation software from the Norwegian land survey. There is probably no reason to support this format in future GPR software. The format is (space separated):

EU89-Geodetisk,P,G,HE

“Trace #” latitude longitude “height above ellipsoid”

The first line is a header that identifies the file type.

The UTM coordinates format contains coordinates in the UTM grid calculated from the latitude and longitude data from the GPS receiver. The transformation is made using Redfearn’s formulas. GV uses a public domain implementation of these contained in the class KCoordinateTransformer. It must always be made clear that we do not accept responsibility for the accuracy of these transformations. The format is (tab separated):

“Trace #” 🡪 northing 🡪 easting 🡪 “height above MSL” 🡪 “UTM zone”

### **The marker file (\*.MRK)**

The marker file should contain information about where in the data the marker is placed together with an index that shows what type of marker it is. There is also a marker label (string) that can be used to pass further information to the application. It is the responsibility of each application to determine what to do with the type information. Like with palette and filter settings, marker settings will not be persistent between different applications. The marker file should be a tab (🡪) separated text file with the following layout:

HEADER VERSION:100

trace number 🡪 sample number 🡪 marker type 🡪 marker label

This version is arbitrarily labelled 100.

The order of the markers in the file isn't important.

Marker: More detailed description:

1. trace number (long);

2. sample number (short);

3. marker type (short);

4. marker label (char[255]).

## ***Appendix 1 – Detailed description of RD3 and RAD formats***

Two files are always created during data acquisition:

**\*.rd3** containing raw data (2 bytes/sample) in straight binary format (nothing else than radar data)

**\*.rad** header file containing parameters and other information connected to the data in the \*.rd3 file

Other files might be created as well but these are of no importance for someone who is interested in reading the radar data for further use. Information on these files is given on request.

### **The \*.rd3 file**

The individual traces can be read by means of a simple algorithm. Below you'll find an example written in C-code, assuming the file is open. The parameter "tracenumber" points out the actual trace number to read and the parameter "buffer" should point to an array of short, large enough for the trace.

void **readtrace**( FILE \*fp, long tracenumber, short \*buffer, int samples) {

long pos;

int nbytes;

pos = samples;

pos \*= sizeof(short);

pos \*= tracenumber; // position of targeted trace calculated

if (fseek(fp,pos,SEEK\_SET)!= 0) {

error(strings8); // check if it works abort otherwise!

}

nbytes = samples \* sizeof(int);

if ( fread((char \*)buffer,1,nbytes,fp) !=nbytes) { // read the trace into the buffer

error(strings9); // abort if something went wrong

}

}

This example show how to read one trace at a time even though the file format actually lends itself to much faster reads, in large chunks.

The file size documented in the \*.rad file should not be taken for granted. The computer could have been turned off before the files where flushed leading to incorrect value of the last trace number (the files are actually flushed regularly during measurements, for safety reasons).

Below you'll find an example of how to calculate the file size, in trace numbers:

long **nbr\_traces\_in\_file**(char \*name, int number\_of\_samples) {

long filesize;

int ret;

FILE \*fp;

char infile[100];

sprintf(infile,"%s%s",name,data\_file\_ext); // data\_file\_ext = ".rd3"

fp = fopen(infile,"rb"); // read & binary

if (fp == NULL) {

return 0; // it doesn’t work!

}

ret = fseek(fp,(long)0,SEEK\_END);

if(ret) {

error(strings13); // abort, it doesn't work

}

filesize = ftell(fp) / (number\_of\_samples \* sizeof(short));

fclose(fp);

return((long) filesize);

}

### **The \*.rad file**

This file is a formatted asci-file. During reads every parameter is checked, so an example of a read function would be rather lengthy. Instead you'll find a write example below, from that any programmer can derive a read function.

void **write\_data\_header\_file**(meas\_par \*mpk) {

char filename[20];

strcpy(filename,mpk->outfile);

strcat(filename,data\_header\_ext); // data\_header\_ext = ".rad"

FILE \*fp;

fp = fopen(filename,"wt");

if(fp == NULL) {

error(strings28);

}

fprintf(fp,"SAMPLES:%d\n",mpk->samples);

fprintf(fp,"FREQUENCY:%f\n",mpk->freq.frequency);

fprintf(fp,"FREQUENCY STEPS:%u\n",mpk->freq.steps);

fprintf(fp,"SIGNAL POSITION:%f\n",mpk->sigpos.pos);

fprintf(fp,"RAW SIGNAL POSITION:%u\n",mpk->sigpos.raw\_pos);

fprintf(fp,"DISTANCE FLAG:%d\n",mpk->trigger.distance\_flag);

fprintf(fp,"TIME FLAG:%d\n",mpk->trigger.time\_flag);

fprintf(fp,"PROGRAM FLAG:%d\n",mpk->trigger.program\_flag);

fprintf(fp,"EXTERNAL FLAG:%d\n",mpk->trigger.external\_flag);

fprintf(fp,"TIME INTERVAL:%f\n",mpk->trigger.time\_interval);

fprintf(fp,"DISTANCE INTERVAL:%f\n",mpk->trigger.distance\_interval);

fprintf(fp,"OPERATOR:%s\n",mpk->info.logger);

fprintf(fp,"CUSTOMER:%s\n",mpk->info.customer);

fprintf(fp,"SITE:%s\n",mpk->info.site);

fprintf(fp,"ANTENNAS:%s\n",mpk->info.antennas);

fprintf(fp,"ANTENNA ORIENTATION:NOT VALID FIELD\n");

fprintf(fp,"ANTENNA SEPARATION:%f\n",mpk->info.antenna\_separation);

fprintf(fp,"COMMENT:%s\n",mpk->info.comment);

fprintf(fp,"TIMEWINDOW:%f\n",mpk->timewindow.timew);

fprintf(fp,"STACKS:%u\n",mpk->stacks.stacks );

fprintf(fp,"STACK EXPONENT:%u\n",mpk->stacks.exponent);

fprintf(fp,"STACKING TIME:%f\n",mpk->stacks.stacking\_time);

fprintf(fp,"LAST TRACE:%ld\n",mpk->final\_trnum);

fprintf(fp,"STOP POSITION:%.2f\n",mpk->final\_pos\_time);

fprintf(fp,"SYSTEM CALIBRATION:%.10f\n",mpk->freq.time\_p\_step);

fprintf(fp,"START POSITION:%.2f\n",mpk->start\_position);

fprintf(fp,"SHORT FLAG:%d\n",mpk->time\_base.short\_flag);

fprintf(fp,"INTERMEDIATE FLAG:%d\n",mpk->time\_base\

.intermediate\_flag);

fprintf(fp,"LONG FLAG:%d\n",mpk->time\_base.long\_flag);

fprintf(fp,"PREPROCESSING:%d\n",mpk->preprocessing);

fprintf(fp,"HIGH:%d\n",mpk->high);

fprintf(fp,"LOW:%d\n",mpk->low);

fprintf(fp,"FIXED INCREMENT:%.3f\n",mpk->tomo.fixed\_inc);

fprintf(fp,"FIXED MOVES UP:%d\n",mpk->tomo.fixed\_up);

fprintf(fp,"FIXED MOVES DOWN:%d\n",mpk->tomo.fixed\_down);

fprintf(fp,"FIXED POSITION:%.3f\n",mpk->tomo.fixed\_pos);

fclose(fp);

}