A PTG data set actually contains multiple files and a folder. The top level PTG file contains ASCII text describing the names of the binary files that are part of the collection; in addition, there are one or more PTG files in a subfolder that contain binary scan data. Each binary file in the subfolder contains data from a different scan. A PTG data set may contain any number of scans.

For example, given a data set with 3 scans and a base file name of *MyFilename* (the base file name can be any valid file name), the layout of a PTG data set would look like:

MyFilename.PTG
MyFilename\ MyFilename-0.PTG
MyFilename\ MyFilename-1.PTG
MyFilename\ MyFilename-2.PTG

Top Level ASCII PTG File

The top level PTG file lists the locations of the binary PTG files. As shown below, it starts with two lines saying it is a PTG index file, and then lists the file names including the subfolder name. In the above example, the top level *MyFilename.PTG* file would contain the text:

```
PTG index file
--------------
MyFilename\MyFilename-0.PTG
MyFilename\MyFilename-1.PTG
MyFilename\MyFilename-2.PTG
```

Binary PTG Scan Files

All binary PTG files are saved in a subdirectory whose name is the base file name.

General conventions about binary PTG file

- Binary encoding format: uses Little Endian (Intel) format.
- Units: All positions and offsets are in meters; all angles are in radians.
- Strings: All strings are written as a 32-bit integer containing the length of the string (including the trailing NULL character), followed by the string characters and the trailing NULL character. For example, the string “ab” would be represented by the following bytes: 3 0 0 0 \a \b \0 (note that the 4 byte integer could be read as a unit instead of as bytes, at least on a Little Endian machine).

Structure of the binary PTG file

The PTG file contains the following sequence of data.

[4 bytes] File type tag: start with ‘P’, ‘T’, ‘G’, ‘\0’.

[4 bytes] Magic number as 32-bit integer = 2458887111 (or 0x928FA3C7).

[variable bytes] The metadata section contains a sequence of mandatory and optional key/value pairs in the order given by the table below. The keys are all strings beginning with “%%” and the type of the values (if any) are described in the following table.

Mand / Opt [a]	Key	Value
M	String: %%header_begin	(none: marks beginning of metadata)
M	String: %%version	Int32: version number (1)
O	String: %%sw_name	String: scanworld name
O	String: %%scan_name	String: scan name
O	String: %%scanner_name	String: scanner name
O	String: %%scanner_model	String: scanner model
O	String: %%scanner_ip_addr	String: ip address
O	String: %%creation_date	String: date as y/m/d
O	String: %%creation_time	String: time as hh:mm:ss
Om	String: %%texte_*	String: to be ignored
Om	String: %%text_*	String: to be ignored
M	String: %%cols	Int32: NumCols, number of columns
M	String: %%rows	Int32: NumRows, number of rows
O	String: %%rows_total	Int32: to be ignored
O	String: %%azim_min	Double: minimum azimuth angle
O	String: %%azim_max	Double: maximum azimuth angle
O	String: %%elev_min	Double: minimum elevation angle
O	String: %%elev_max	Double: maximum elevation angle
O	String: %%transform	Double[4][4]: transformation matrix [b]
M	String: %%properties	Int32: property bits [c]
M	String: %%header_end	(none: marks end of metadata)

Note:

a. M/O/Om: M is mandatory, O is optional, Om is optional and may occur multiple times.

b. If the transform key/value is not present then no transformation should be applied to the points. If present, then the elements are read in row order (see subscript order below). All 16 values are in the file, even though the last column contains only values of zero and one. Points read from the subsequent point records in the PTG file can be transformed to the final coordinate system as follows:

$$\begin{pmatrix} x_w & y_w & z_w & 1 \end{pmatrix} = \begin{pmatrix} x & y & z & 1 \end{pmatrix} \cdot \begin{bmatrix} a_{(0)} & b_{(1)} & c_{(2)} & 0_{(3)} \\ d_{(4)} & e_{(5)} & f_{(6)} & 0_{(7)} \\ g_{(8)} & h_{(9)} & i_{(10)} & 0_{(11)} \\ dx_{(12)} & dy_{(13)} & dz_{(14)} & 1_{(15)} \end{bmatrix}$$

c. The property bits determine what is saved in a single point data record. The order of the attributes, though some are optional, is xyz coordinate, followed by intensity value, and finally RGB values. The xyz coordinate is always present (0x1 or 0x2 is always set) but the property bits determines if floats or doubles are used. If the intensity bit is 1 then the point data record contains a float intensity value with range [0-1]. If the RGB bit is 1 then the point data record contains a unsigned char[3] RGB value with each component in the range [0-255].

0x1: if set then xyz is in float[3] format (mutually exclusive with 0x2)
0x2: if set then xyz is in double[3] format (mutually exclusive with 0x1)
0x4: if set then intensity is available as a float (range [0-1])
0x8: if set then RGB is available as unsigned char[3] (each with range [0-255])

[8 * NumCols bytes] Array of Int64 values specifying the absolute file offsets in bytes to the start of each column of point record data. The scan is stored as a collection of column data sets; as described below, each

column of data has variable length due to missing points. The absolute offsets described here can be used to seek to the beginning of each column data set.

[variable bytes] The n^{th} column data set begins at the offset specified in the n^{th} position of the offset array described above. Each column data set consists of a bitmask that specifies which points exist (a 1 bit in the mask) and which points were missed (a 0 bit in the mask), followed by the point data records for all points that exist (skipping missed points). The column data set consists of:

a. [ceil(NumRows/8) bytes] A column bitmask describing existing (1 bit) and missed (0 bit) points for all positions in the column. The highest bit in each byte corresponds to the lowest row number for that byte. For example, the highest bit of the first byte represents row 0, and the highest bit of the second byte represents row 8.
b. [number of non-missed points * point record size] A sequence of point records for non-missed points. Missed points are not stored. Each point record contains the data that is specified by the property bits in the metadata section. The xyz values are in scanner local coordinates with the scanner at (0,0,0).

High Level Procedure for Reading a Binary PTG File

The following pseudo-code shows how an entire PTG file could be imported.

```
PROC ImportTopLevelAsciiPTG
    Open top level ASCII PTG file for reading as text;
    If the first line is "PTG index file" then
        Read next line (to skip the ------ line);
        For each line read file name
            Import binary PTG file as a scan with transformation;
    Else
        Close PTG file. This may be a single binary PTG file;
        Import binary PTG file as a scan with transformation;
    Endif
END PROC

! Import PTG binary file as scan with transformation
PROC ImportBinaryPTG
    Open binary PTG file for read as binary;
    Read 4 bytes to verify file tag: "PTG";
    Read Int32 to verify magic number;
    Read string tag and verify it to be "%%header_begin";
    do
        Read string tag;
        Read followed value based on its type;
        ! make sure you successfully read key data:
        ! NumRows, NumCols, transform, property bits
    until tag is "%%header_end";
    Read Int64 ColumnPositions[NumCols];
    For each column
        Seek to ColumnPositions[column];
        Read row validity bits ValidBits[] (length is ceil(NumRows/8));
        Convert bits to bool ValidPoints[] (length is now NumRows);
        For each row
            If ValidPoints[row] is true
                Read point record (size determined by property bits);
                ! use point record data as desired
            End if
        End For
    End For
END PROC
```