dgnlib

Contents

Chapter 1

Class Index

1.1 Class List

Here are the classes, structs, unions and interfaces with brief descriptions:

Chapter 2

File Index

2.1 File List

Here is a list of all documented files with brief descriptions:

Chapter 3

Class Documentation

3.1 _DGNTagDef Struct Reference

`#include <dgnlib.h>`

Public Attributes

- char * name
- int id
- char * prompt
- int type
- tagValueUnion defaultValue

3.1.1 Detailed Description

Tag definition.

Structure holding definition of one tag within a DGNTagSet.

3.1.2 Member Data Documentation

3.1.2.1 defaultValue

`tagValueUnion _DGNTagDef::defaultValue`

Default tag value

3.1.2.2 id

```
int _DGNTagDef::id
```

Tag index/identifier.

3.1.2.3 name

```
char* _DGNTagDef::name
```

Name of this tag.

3.1.2.4 prompt

```
char* _DGNTagDef::prompt
```

User prompt when requesting value.

3.1.2.5 type

```
int _DGNTagDef::type
```

Tag type (one of DGNTT_STRING(1), DGNTT_INTEGER(3) or DGNTT_FLOAT(4).

The documentation for this struct was generated from the following file:

- dgnlib.h

3.2 DGNElemArc Struct Reference

```
#include <dgnlib.h>
```

Public Attributes

- DGNElemCore **core**
- DGNPoint origin
- double primary_axis
- double secondary_axis
- double rotation
- int **quat** [4]
- double startang
- double sweepang

3.2.1 Detailed Description

Ellipse element

The core.stype code is DGNST_ARC.

Used for: DGNT_ELLIPSE(15), DGNT_ARC(16)

3.2.2 Member Data Documentation

3.2.2.1 origin

```
DGNPoint DGNElemArc::origin
```

Origin of ellipse

3.2.2.2 primary_axis

```
double DGNElemArc::primary_axis
```

Primary axis length

3.2.2.3 rotation

```
double DGNElemArc::rotation
```

Counterclockwise rotation in degrees

3.2.2.4 secondary_axis

```
double DGNElemArc::secondary_axis
```

Secondary axis length

3.2.2.5 startang

```
double DGNElemArc::startang
```

Start angle (degrees counterclockwise of primary axis)

3.2.2.6 sweepang

```
double DGNElemArc::sweepang
```

Sweep angle (degrees)

The documentation for this struct was generated from the following file:

- dgnlib.h

3.3 DGNElemBSplineCurveHeader Struct Reference

```
#include <dgnlib.h>
```

Public Attributes

- DGNElemCore **core**
- long desc_words
- unsigned char order
- unsigned char properties
- unsigned char curve_type
- short num_poles
- short num_knots

3.3.1 Detailed Description

B-Spline Curve Header element

The core.stype code is DGNST_BSPLINE_CURVE_HEADER.

Used for: DGNT_BSPLINE_CURVE_HEADER(27)

3.3.2 Member Data Documentation

3.3.2.1 curve_type

```
unsigned char DGNElemBSplineCurveHeader::curve_type
```

curve type

3.3.2.2 desc_words

```
long DGNElemBSplineCurveHeader::desc_words
```

Total length of B-Spline curve in words, excluding the first 20 words (header + desc_words field)

3.3.2.3 num_knots

```
short DGNElemBSplineCurveHeader::num_knots
```

number of knots

3.3.2.4 num_poles

```
short DGNElemBSplineCurveHeader::num_poles
```

number of poles, max. 101

3.3.2.5 order

```
unsigned char DGNElemBSplineCurveHeader::order
```

B-spline order: 2-15

3.3.2.6 properties

```
unsigned char DGNElemBSplineCurveHeader::properties
```

Properties: ORing of DGNBSC_ flags

The documentation for this struct was generated from the following file:

- dgnlib.h

3.4 DGNElemBSplineSurfaceBoundary Struct Reference

```
#include <dgnlib.h>
```

Public Attributes

- DGNElemCore **core**
- short number
- short numverts
- DGNPoint vertices [1]

3.4.1 Detailed Description

B-Spline Surface Boundary element

The core.stype code is DGNST_BSPLINE_SURFACE_BOUNDARY

Used for: DGNT_BSPLINE_SURFACE_BOUNDARY(25)

3.4.2 Member Data Documentation

3.4.2.1 number

```
short DGNElemBSplineSurfaceBoundary::number
```

boundary number

3.4.2.2 numverts

```
short DGNElemBSplineSurfaceBoundary::numverts
```

number of boundary vertices

3.4.2.3 vertices

```
DGNPoint DGNElemBSplineSurfaceBoundary::vertices[1]
```

Array of 1 or more 2D boundary vertices (in UV space)

The documentation for this struct was generated from the following file:

- dgnlib.h

3.5 DGNElemBSplineSurfaceHeader Struct Reference

```
#include <dgnlib.h>
```

Public Attributes

- DGNElemCore **core**
- long desc_words
- unsigned char curve_type
- unsigned char u_order
- unsigned short u_properties
- short num_poles_u
- short num_knots_u
- short rule_lines_u
- unsigned char v_order
- unsigned short v_properties
- short num_poles_v
- short num_knots_v
- short rule_lines_v
- short num_bounds

3.5.1 Detailed Description

B-Spline Surface Header element

The core.stype code is DGNST_BSPLINE_SURFACE_HEADER.

Used for: DGNT_BSPLINE_SURFACE_HEADER(24)

3.5.2 Member Data Documentation

3.5.2.1 curve_type

```
unsigned char DGNElemBSplineSurfaceHeader::curve_type
```

curve type

3.5.2.2 desc_words

```
long DGNElemBSplineSurfaceHeader::desc_words
```

Total length of B-Spline surface in words, excluding the first 20 words (header + desc_words field)

3.5.2.3 num_bounds

```
short DGNElemBSplineSurfaceHeader::num_bounds
```

number of boundaries

3.5.2.4 num_knots_u

```
short DGNElemBSplineSurfaceHeader::num_knots_u
```

number of knots

3.5.2.5 num_knots_v

```
short DGNElemBSplineSurfaceHeader::num_knots_v
```

number of knots

3.5.2.6 num_poles_u

```
short DGNElemBSplineSurfaceHeader::num_poles_u
```

number of poles

3.5.2.7 num_poles_v

```
short DGNElemBSplineSurfaceHeader::num_poles_v
```

number of poles

3.5.2.8 rule_lines_u

```
short DGNElemBSplineSurfaceHeader::rule_lines_u
```

number of rule lines

3.5.2.9 rule_lines_v

```
short DGNElemBSplineSurfaceHeader::rule_lines_v
```

number of rule lines

3.5.2.10 u_order

```
unsigned char DGNElemBSplineSurfaceHeader::u_order
```

B-spline U order: 2-15

3.5.2.11 u_properties

```
unsigned short DGNElemBSplineSurfaceHeader::u_properties
```

surface U properties: ORing of DGNBSC_ flags

3.5.2.12 v_order

```
unsigned char DGNElemBSplineSurfaceHeader::v_order
```

B-spline V order: 2-15

3.5.2.13 v_properties

```
unsigned short DGNElemBSplineSurfaceHeader::v_properties
```

surface V properties: Oring of DGNBSS_ flags

The documentation for this struct was generated from the following file:

- dgnlib.h

3.6 DGNElemCellHeader Struct Reference

`#include <dgnlib.h>`

Public Attributes

- DGNElemCore **core**
- int totlength
- char name [7]
- unsigned short cclass
- unsigned short levels [4]
- DGNPoint rnglow
- DGNPoint rnghigh
- double trans [9]
- DGNPoint origin
- double **xscale**
- double **yscale**
- double **rotation**

3.6.1 Detailed Description

Cell Header.

The core.stype code is DGNST_CELL_HEADER.

Returned for DGNT_CELL_HEADER(2).

3.6.2 Member Data Documentation

3.6.2.1 cclass

`unsigned short DGNElemCellHeader::cclass`

Class bitmap

3.6.2.2 levels

`unsigned short DGNElemCellHeader::levels[4]`

Levels used in cell

3.6.2.3 name

`char DGNElemCellHeader::name[7]`

Cell name

3.6.2.4 origin

```
DGNPoint DGNElemCellHeader::origin
```

Cell Origin

3.6.2.5 rnghigh

```
DGNPoint DGNElemCellHeader::rnghigh
```

X/Y/Z maximums for cell

3.6.2.6 rnglow

```
DGNPoint DGNElemCellHeader::rnglow
```

X/Y/Z minimums for cell

3.6.2.7 totlength

```
int DGNElemCellHeader::totlength
```

Total length of cell in words, excluding the first 19 words (header + totlength field)

3.6.2.8 trans

```
double DGNElemCellHeader::trans[9]
```

2D/3D Transformation Matrix

The documentation for this struct was generated from the following file:

- dgnlib.h

3.7 DGNElemCellLibrary Struct Reference

```
#include <dgnlib.h>
```

Public Attributes

- DGNElemCore **core**
- short celltype
- short attindx
- char name [7]
- int numwords
- short dispsymb
- unsigned short cclass
- unsigned short levels [4]
- char description [28]

3.7.1 Detailed Description

Cell Library.

The core.stype code is DGNST_CELL_LIBRARY.

Returned for DGNT_CELL_LIBRARY(1).

3.7.2 Member Data Documentation

3.7.2.1 attindx

```
short DGNElemCellLibrary::attindx
```

Attribute linkage.

3.7.2.2 cclass

```
unsigned short DGNElemCellLibrary::cclass
```

Class bitmap

3.7.2.3 celltype

```
short DGNElemCellLibrary::celltype
```

Cell type.

3.7.2.4 description

```
char DGNElemCellLibrary::description[28]
```

Description

3.7.2.5 dispsymb

```
short DGNElemCellLibrary::dispsymb
```

Display symbol

3.7.2.6 levels

```
unsigned short DGNElemCellLibrary::levels[4]
```

Levels used in cell

3.7.2.7 name

```
char DGNElemCellLibrary::name[7]
```

Cell name

3.7.2.8 numwords

```
int DGNElemCellLibrary::numwords
```

Number of words in cell definition

The documentation for this struct was generated from the following file:

- dgnlib.h

3.8 DGNElemColorTable Struct Reference

```
#include <dgnlib.h>
```

Public Attributes

- DGNElemCore **core**
- int **screen_flag**
- GByte color_info [256][3]

3.8.1 Detailed Description

Color table.

The core.stype code is DGNST_COLORTABLE.

Returned for DGNT_GROUP_DATA(5) elements, with a level number of DGN_GDL_COLOR_TABLE(1).

3.8.2 Member Data Documentation

3.8.2.1 color_info

```
GByte DGNElemColorTable::color_info[256][3]
```

Color table, 256 colors by red (0), green(1) and blue(2) component.

The documentation for this struct was generated from the following file:

- dgnlib.h

3.9 DGNElemComplexHeader Struct Reference

`#include <dgnlib.h>`

Public Attributes

- DGNElemCore **core**
- int totlength
- int numelems
- int surftype
- int boundelms

3.9.1 Detailed Description

Complex header element

The core.stype code is DGNST_COMPLEX_HEADER.

Used for: DGNT_COMPLEX_CHAIN_HEADER(12), DGNT_COMPLEX_SHAPE_HEADER(14), DGNT_3DSURFACE_HEADER(18) and DGNT_3DSOLID_HEADER(19).

Compatible with DGNT_TEXT_NODE (7), see DGNAddRawAttrLink()

3.9.2 Member Data Documentation

3.9.2.1 boundelms

`int DGNElemComplexHeader::boundelms`

of elements in each boundary

(only used for 3D surface/solid).

3.9.2.2 numelems

`int DGNElemComplexHeader::numelems`

of elements in surface

3.9.2.3 surftype

```
int DGNElemComplexHeader::surftype
```

surface/solid type (only used for 3D surface/solid). One of DGNSUT_* or DGNSOT_*.

3.9.2.4 totlength

```
int DGNElemComplexHeader::totlength
```

Total length of surface in words, excluding the first 19 words (header + totlength field)

The documentation for this struct was generated from the following file:

- dgnlib.h

3.10 DGNElemCone Struct Reference

```
#include <dgnlib.h>
```

Public Attributes

- DGNElemCore **core**
- short unknown
- int quat [4]
- DGNPoint center_1
- double radius_1
- DGNPoint center_2
- double radius_2

3.10.1 Detailed Description

Cone element

The core.stype code is DGNST_CONE.

Used for: DGNT_CONE(23)

3.10.2 Member Data Documentation

3.10.2.1 center_1

```
DGNPoint DGNElemCone::center_1
```

center of first circle

3.10.2.2 center_2

```
DGNPoint DGNElemCone::center_2
```

center of second circle

3.10.2.3 quat

```
int DGNElemCone::quat[4]
```

Orientation quaternion

3.10.2.4 radius_1

```
double DGNElemCone::radius_1
```

radius of first circle

3.10.2.5 radius_2

```
double DGNElemCone::radius_2
```

radius of second circle

3.10.2.6 unknown

```
short DGNElemCone::unknown
```

Unknown data

The documentation for this struct was generated from the following file:

- dgnlib.h

3.11 DGNElemCore Struct Reference

```
#include <dgnlib.h>
```

Public Attributes

- int **offset**
- int **size**
- int element_id
- int stype
- int level
- int type
- int complex
- int deleted
- int graphic_group
- int properties
- int color
- int weight
- int style
- int attr_bytes
- unsigned char * attr_data
- int raw_bytes
- unsigned char * raw_data

3.11.1 Detailed Description

Core element structure.

Core information kept about each element that can be read from a DGN file. This structure is the first component of each specific element structure (like DGNElemMultiPoint). Normally the DGNElemCore.stype field would be used to decide what specific structure type to case the DGNElemCore pointer to.

3.11.2 Member Data Documentation

3.11.2.1 attr_bytes

```
int DGNElemCore::attr_bytes
```

Bytes of attribute data, usually zero.

3.11.2.2 attr_data

```
unsigned char* DGNElemCore::attr_data
```

Raw attribute data

3.11.2.3 color

```
int DGNElemCore::color
```

Color index (0-255)

3.11.2.4 complex

```
int DGNElemCore::complex
```

Is element complex?

3.11.2.5 deleted

```
int DGNElemCore::deleted
```

Is element deleted?

3.11.2.6 element_id

```
int DGNElemCore::element_id
```

Element number (zero based)

3.11.2.7 graphic_group

```
int DGNElemCore::graphic_group
```

Graphic group number

3.11.2.8 level

```
int DGNElemCore::level
```

Element Level: 0-63

3.11.2.9 properties

```
int DGNElemCore::properties
```

Properties: ORing of DGNPF_ flags

3.11.2.10 raw_bytes

```
int DGNElemCore::raw_bytes
```

Bytes of raw data, usually zero.

3.11.2.11 raw_data

```
unsigned char* DGNElemCore::raw_data
```

All raw element data including header.

3.11.2.12 style

```
int DGNElemCore::style
```

Line Style: One of DGNS_* values

3.11.2.13 stype

```
int DGNElemCore::stype
```

Structure type: (DGNST_*)

3.11.2.14 type

```
int DGNElemCore::type
```

Element type (DGNT_)

3.11.2.15 weight

```
int DGNElemCore::weight
```

Line Weight (0-31)

The documentation for this struct was generated from the following file:

- dgnlib.h

3.12 DGNElementInfo Struct Reference

```
#include <dgnlib.h>
```

Public Attributes

- unsigned char level
- unsigned char type
- unsigned char stype
- unsigned char flags
- vsi_l_offset offset

3.12.1 Detailed Description

Element summary information.

Minimal information kept about each element if an element summary index is built for a file by DGNGetElementIndex().

3.12.2 Member Data Documentation

3.12.2.1 flags

```
unsigned char DGNElementInfo::flags
```

Other flags

3.12.2.2 level

```
unsigned char DGNElementInfo::level
```

Element Level: 0-63

3.12.2.3 offset

```
vsi_l_offset DGNElementInfo::offset
```

Offset within file (private)

3.12.2.4 stype

```
unsigned char DGNElementInfo::stype
```

Structure type (DGNST_*)

3.12.2.5 type

```
unsigned char DGNElementInfo::type
```

Element type (DGNT_*)

The documentation for this struct was generated from the following file:

- dgnlib.h

3.13 DGNElemKnotWeight Struct Reference

```
#include <dgnlib.h>
```

Public Attributes

- DGNElemCore **core**
- float array [1]

3.13.1 Detailed Description

B-Spline Knot/Weight element

The core.stype code is DGNST_KNOT_WEIGHT

Used for: DGNT_BSPLINE_KNOT(26), DGNT_BSPLINE_WEIGHT_FACTOR(28)

3.13.2 Member Data Documentation

3.13.2.1 array

```
float DGNElemKnotWeight::array[1]
```

array (variable length). Length is given in the corresponding B-Spline header.

The documentation for this struct was generated from the following file:

- dgnlib.h

3.14 DGNElemMultiPoint Struct Reference

```
#include <dgnlib.h>
```

Public Attributes

- DGNElemCore **core**
- int num_vertices
- DGNPoint vertices [1]

3.14.1 Detailed Description

Multipoint element

The core.stype code is DGNST_MULTIPOINT.

Used for: DGNT_LINE(3), DGNT_LINE_STRING(4), DGNT_SHAPE(6), DGNT_CURVE(11), DGNT_BSPLINE_↩ POLE(21)

3.14.2 Member Data Documentation

3.14.2.1 num_vertices

```
int DGNElemMultiPoint::num_vertices
```

Number of vertices in "vertices"

3.14.2.2 vertices

```
DGNPoint DGNElemMultiPoint::vertices[1]
```

Array of two or more vertices

The documentation for this struct was generated from the following file:

- dgnlib.h

3.15 DGNElemSharedCellDefn Struct Reference

```
#include <dgnlib.h>
```

Public Attributes

- DGNElemCore **core**
- int totlength

3.15.1 Detailed Description

Shared Cell Definition.

The core.stype code is DGNST_SHARED_CELL_DEFN.

Returned for DGNT_SHARED_CELL_DEFN(2).

3.15.2 Member Data Documentation

3.15.2.1 totlength

```
int DGNElemSharedCellDefn::totlength
```

Total length of cell in words, excluding the first 19 words (header + totlength field)

The documentation for this struct was generated from the following file:

- dgnlib.h

3.16 DGNElemTagSet Struct Reference

```
#include <dgnlib.h>
```

Public Attributes

- DGNElemCore **core**
- int tagCount
- int tagSet
- int flags
- char * tagSetName
- DGNTagDef * tagList

3.16.1 Detailed Description

Tag Set.

The core.stype code is DGNST_TAG_SET.

Returned for DGNT_APPLICATION_ELEM(66), Level: 24.

3.16.2 Member Data Documentation

3.16.2.1 flags

```
int DGNElemTagSet::flags
```

Tag flags - not too much known.

3.16.2.2 tagCount

```
int DGNElemTagSet::tagCount
```

Number of tags in tagList.

3.16.2.3 tagList

```
DGNTagDef* DGNElemTagSet::tagList
```

List of tag definitions in this set.

3.16.2.4 tagSet

```
int DGNElemTagSet::tagSet
```

Tag set index.

3.16.2.5 tagSetName

```
char* DGNElemTagSet::tagSetName
```

Tag set name.

The documentation for this struct was generated from the following file:

- dgnlib.h

3.17 DGNElemTagValue Struct Reference

```
#include <dgnlib.h>
```

Public Attributes

- DGNElemCore **core**
- int tagType
- int tagSet
- int tagIndex
- int tagLength
- tagValueUnion tagValue

3.17.1 Detailed Description

Tag Value.

The core.stype code is DGNST_TAG_VALUE.

Returned for DGNT_TAG_VALUE(37).

3.17.2 Member Data Documentation

3.17.2.1 tagIndex

```
int DGNElemTagValue::tagIndex
```

Tag index within tag set.

3.17.2.2 tagLength

```
int DGNElemTagValue::tagLength
```

Length of tag information (text)

3.17.2.3 tagSet

```
int DGNElemTagValue::tagSet
```

Which tag set does this relate to?

3.17.2.4 tagType

```
int DGNElemTagValue::tagType
```

Tag type indicator, DGNTT_*

3.17.2.5 tagValue

```
tagValueUnion DGNElemTagValue::tagValue
```

Textual value of tag

The documentation for this struct was generated from the following file:

- dgnlib.h

3.18 DGNElemTCB Struct Reference

```
#include <dgnlib.h>
```

Public Attributes

- DGNElemCore **core**
- int dimension
- double origin_x
- double origin_y
- double origin_z
- long uor_per_subunit
- char sub_units [3]
- long subunits_per_master
- char master_units [3]
- DGNViewInfo **views** [8]

3.18.1 Detailed Description

Terminal Control Block (header).

The core.stype code is DGNST_TCB.

Returned for DGNT_TCB(9).

The first TCB in the file is used to determine the dimension (2D vs. 3D), and transformation from UOR (units of resolution) to subunits, and subunits to master units. This is handled transparently within DGNReadElement(), so it is not normally necessary to handle this element type at the application level, though it can be useful to get the sub_units, and master_units names.

3.18.2 Member Data Documentation

3.18.2.1 dimension

```
int DGNElemTCB::dimension
```

Dimension (2 or 3)

3.18.2.2 master_units

```
char DGNElemTCB::master_units[3]
```

User name for master units (2 chars)

3.18.2.3 origin_x

```
double DGNElemTCB::origin_x
```

X origin of UOR space in master units(?)

3.18.2.4 origin_y

```
double DGNElemTCB::origin_y
```

Y origin of UOR space in master units(?)

3.18.2.5 origin_z

```
double DGNElemTCB::origin_z
```

Z origin of UOR space in master units(?)

3.18.2.6 sub_units

```
char DGNElemTCB::sub_units[3]
```

User name for subunits (2 chars)

3.18.2.7 subunits_per_master

```
long DGNElemTCB::subunits_per_master
```

Subunits per master unit.

3.18.2.8 uor_per_subunit

```
long DGNElemTCB::uor_per_subunit
```

UOR per subunit.

The documentation for this struct was generated from the following file:

- dgnlib.h

3.19 DGNElemText Struct Reference

```
#include <dgnlib.h>
```

Public Attributes

- DGNElemCore **core**
- int font_id
- int justification
- double length_mult
- double height_mult
- double rotation
- DGNPoint origin
- char string [1]

3.19.1 Detailed Description

Text element

The core.stype code is DGNST_TEXT.

NOTE: Currently we are not capturing the "editable fields" information.

Used for: DGNT_TEXT(17).

3.19.2 Member Data Documentation

3.19.2.1 font_id

```
int DGNElemText::font_id
```

Microstation font id, no list available

3.19.2.2 height_mult

```
double DGNElemText::height_mult
```

Char height in master units

3.19.2.3 justification

```
int DGNElemText::justification
```

Justification, see DGNJ_∗

3.19.2.4 length_mult

```
double DGNElemText::length_mult
```

Char width in master (if square)

3.19.2.5 origin

```
DGNPoint DGNElemText::origin
```

Bottom left corner of text.

3.19.2.6 rotation

```
double DGNElemText::rotation
```

Counterclockwise rotation in degrees

3.19.2.7 string

```
char DGNElemText::string[1]
```

Actual text (length varies, \0 terminated

The documentation for this struct was generated from the following file:

- dgnlib.h

3.20 DGNElemTextNode Struct Reference

`#include <dgnlib.h>`

Public Attributes

- DGNElemCore **core**
- int totlength
- int numelems
- int node_number
- short max_length
- short max_used
- short font_id
- short justification
- long line_spacing
- double length_mult
- double height_mult
- double rotation
- DGNPoint origin

3.20.1 Detailed Description

Text Node Header.

The core.stype code is DGNST_TEXT_NODE.

Used for DGNT_TEXT_NODE (7). First fields (up to numelems) are compatible with DGNT_COMPLEX_HEADER (7),

See also

DGNAddRawAttrLink()

3.20.2 Member Data Documentation

3.20.2.1 font_id

`short DGNElemTextNode::font_id`

text font used

3.20.2.2 height_mult

`double DGNElemTextNode::height_mult`

height multiplier

3.20.2.3 justification

```
short DGNElemTextNode::justification
```

justification type, see DGNJ_

3.20.2.4 length_mult

```
double DGNElemTextNode::length_mult
```

length multiplier

3.20.2.5 line_spacing

```
long DGNElemTextNode::line_spacing
```

spacing between text strings

3.20.2.6 max_length

```
short DGNElemTextNode::max_length
```

maximum length allowed, characters

3.20.2.7 max_used

```
short DGNElemTextNode::max_used
```

maximum length used

3.20.2.8 node_number

```
int DGNElemTextNode::node_number
```

text node number

3.20.2.9 numelems

```
int DGNElemTextNode::numelems
```

Number of text strings

3.20.2.10 origin

```
DGNPoint DGNElemTextNode::origin
```

Snap origin (as defined by user)

3.20.2.11 rotation

```
double DGNElemTextNode::rotation
```

rotation angle (2d)

3.20.2.12 totlength

```
int DGNElemTextNode::totlength
```

Total length of the node (bytes = totlength * 2 + 38)

The documentation for this struct was generated from the following file:

- dgnlib.h

3.21 DGNPoint Struct Reference

```
#include <dgnlib.h>
```

Public Attributes

- double x
- double y
- double z

3.21.1 Detailed Description

DGN Point structure.

Note that the DGNReadElement() function transforms points into "master" coordinate system space when they are in the file in UOR (units of resolution) coordinates.

3.21.2 Member Data Documentation

3.21.2.1 x

```
double DGNPoint::x
```

x (normally eastwards) coordinate.

3.21.2.2 y

```
double DGNPoint::y
```

y (normally northwards) coordinate.

3.21.2.3 z

```
double DGNPoint::z
```

z, up coordinate. Zero for 2D objects.

The documentation for this struct was generated from the following file:

- dgnlib.h

3.22 DGNViewInfo Struct Reference

Public Attributes

- int **flags**
- unsigned char **levels** [8]
- DGNPoint **origin**
- DGNPoint **delta**
- double **transmatrx** [9]
- double **conversion**
- unsigned long **activez**

The documentation for this struct was generated from the following file:

- dgnlib.h

3.23 tagValueUnion Union Reference

Public Attributes

- char * **string**
- GInt32 **integer**
- double **real**

The documentation for this union was generated from the following file:

- dgnlib.h

Chapter 4

File Documentation

4.1 dgnlib.h File Reference

```
#include "cpl_conv.h"
```

Classes

- struct DGNPoint
- struct DGNElementInfo
- struct DGNElemCore
- struct DGNElemMultiPoint
- struct DGNElemArc
- struct DGNElemText
- struct DGNElemComplexHeader
- struct DGNElemColorTable
- struct DGNViewInfo
- struct DGNElemTCB
- struct DGNElemCellHeader
- struct DGNElemCellLibrary
- struct DGNElemSharedCellDefn
- union tagValueUnion
- struct DGNElemTagValue
- struct _DGNTagDef
- struct DGNElemTagSet
- struct DGNElemCone
- struct DGNElemTextNode
- struct DGNElemBSplineSurfaceHeader
- struct DGNElemBSplineCurveHeader
- struct DGNElemBSplineSurfaceBoundary
- struct DGNElemKnotWeight

Macros

- #define **CPLE_DGN_ERROR_BASE**
- #define **CPLE_ElementTooBig** CPLE_DGN_ERROR_BASE+1
- #define **DGNTT_STRING** 1
- #define **DGNTT_INTEGER** 3
- #define **DGNTT_FLOAT** 4
- #define DGNST_CORE 1
- #define DGNST_MULTIPOINT 2
- #define DGNST_COLORTABLE 3
- #define DGNST_TCB 4
- #define DGNST_ARC 5
- #define DGNST_TEXT 6
- #define DGNST_COMPLEX_HEADER 7
- #define DGNST_CELL_HEADER 8
- #define DGNST_TAG_VALUE 9
- #define DGNST_TAG_SET 10
- #define DGNST_CELL_LIBRARY 11
- #define DGNST_CONE 12
- #define DGNST_TEXT_NODE 13
- #define DGNST_BSPLINE_SURFACE_HEADER 14
- #define DGNST_BSPLINE_CURVE_HEADER 15
- #define DGNST_BSPLINE_SURFACE_BOUNDARY 16
- #define DGNST_KNOT_WEIGHT 17
- #define DGNST_SHARED_CELL_DEFN 18
- #define **DGNT_CELL_LIBRARY** 1
- #define **DGNT_CELL_HEADER** 2
- #define **DGNT_LINE** 3
- #define **DGNT_LINE_STRING** 4
- #define **DGNT_GROUP_DATA** 5
- #define **DGNT_SHAPE** 6
- #define **DGNT_TEXT_NODE** 7
- #define **DGNT_DIGITIZER_SETUP** 8
- #define **DGNT_TCB** 9
- #define **DGNT_LEVEL_SYMBOLOGY** 10
- #define **DGNT_CURVE** 11
- #define **DGNT_COMPLEX_CHAIN_HEADER** 12
- #define **DGNT_COMPLEX_SHAPE_HEADER** 14
- #define **DGNT_ELLIPSE** 15
- #define **DGNT_ARC** 16
- #define **DGNT_TEXT** 17
- #define **DGNT_3DSURFACE_HEADER** 18
- #define **DGNT_3DSOLID_HEADER** 19
- #define **DGNT_BSPLINE_POLE** 21
- #define **DGNT_POINT_STRING** 22
- #define **DGNT_BSPLINE_SURFACE_HEADER** 24
- #define **DGNT_BSPLINE_SURFACE_BOUNDARY** 25
- #define **DGNT_BSPLINE_KNOT** 26
- #define **DGNT_BSPLINE_CURVE_HEADER** 27
- #define **DGNT_BSPLINE_WEIGHT_FACTOR** 28
- #define **DGNT_CONE** 23
- #define **DGNT_SHARED_CELL_DEFN** 34
- #define **DGNT_SHARED_CELL_ELEM** 35
- #define **DGNT_TAG_VALUE** 37
- #define **DGNT_APPLICATION_ELEM** 66

- #define **DGNS_SOLID** 0
- #define **DGNS_DOTTED** 1
- #define **DGNS_MEDIUM_DASH** 2
- #define **DGNS_LONG_DASH** 3
- #define **DGNS_DOT_DASH** 4
- #define **DGNS_SHORT_DASH** 5
- #define **DGNS_DASH_DOUBLE_DOT** 6
- #define **DGNS_LONG_DASH_SHORT_DASH** 7
- #define **DGNSUT_SURFACE_OF_PROJECTION** 0
- #define **DGNSUT_BOUNDED_PLANE** 1
- #define **DGNSUT_BOUNDED_PLANE2** 2
- #define **DGNSUT_RIGHT_CIRCULAR_CYLINDER** 3
- #define **DGNSUT_RIGHT_CIRCULAR_CONE** 4
- #define **DGNSUT_TABULATED_CYLINDER** 5
- #define **DGNSUT_TABULATED_CONE** 6
- #define **DGNSUT_CONVOLUTE** 7
- #define **DGNSUT_SURFACE_OF_REVOLUTION** 8
- #define **DGNSUT_WARPED_SURFACE** 9
- #define **DGNSOT_VOLUME_OF_PROJECTION** 0
- #define **DGNSOT_VOLUME_OF_REVOLUTION** 1
- #define **DGNSOT_BOUNDED_VOLUME** 2
- #define **DGNC_PRIMARY** 0
- #define **DGNC_PATTERN_COMPONENT** 1
- #define **DGNC_CONSTRUCTION_ELEMENT** 2
- #define **DGNC_DIMENSION_ELEMENT** 3
- #define **DGNC_PRIMARY_RULE_ELEMENT** 4
- #define **DGNC_LINEAR_PATTERNED_ELEMENT** 5
- #define **DGNC_CONSTRUCTION_RULE_ELEMENT** 6
- #define **DGN_GDL_COLOR_TABLE** 1
- #define **DGN_GDL_NAMED_VIEW** 3
- #define **DGN_GDL_REF_FILE** 9
- #define **DGNPF_HOLE** 0x8000
- #define **DGNPF_SNAPPABLE** 0x4000
- #define **DGNPF_PLANAR** 0x2000
- #define **DGNPF_ORIENTATION** 0x1000
- #define **DGNPF_ATTRIBUTES** 0x0800
- #define **DGNPF_MODIFIED** 0x0400
- #define **DGNPF_NEW** 0x0200
- #define **DGNPF_LOCKED** 0x0100
- #define **DGNPF_CLASS** 0x000f
- #define **DGNEIF_DELETED** 0x01
- #define **DGNEIF_COMPLEX** 0x02
- #define **DGNJ_LEFT_TOP** 0
- #define **DGNJ_LEFT_CENTER** 1
- #define **DGNJ_LEFT_BOTTOM** 2
- #define **DGNJ_LEFTMARGIN_TOP** 3 /∗ text node header only ∗/
- #define **DGNJ_LEFTMARGIN_CENTER** 4 /∗ text node header only ∗/
- #define **DGNJ_LEFTMARGIN_BOTTOM** 5 /∗ text node header only ∗/
- #define **DGNJ_CENTER_TOP** 6
- #define **DGNJ_CENTER_CENTER** 7
- #define **DGNJ_CENTER_BOTTOM** 8
- #define **DGNJ_RIGHTMARGIN_TOP** 9 /∗ text node header only ∗/
- #define **DGNJ_RIGHTMARGIN_CENTER** 10 /∗ text node header only ∗/
- #define **DGNJ_RIGHTMARGIN_BOTTOM** 11 /∗ text node header only ∗/
- #define **DGNJ_RIGHT_TOP** 12

- #define **DGNJ_RIGHT_CENTER** 13
- #define **DGNJ_RIGHT_BOTTOM** 14
- #define **DGNO_CAPTURE_RAW_DATA** 0x01
- #define **DGNLT_DMRS** 0x0000
- #define **DGNLT_INFORMIX** 0x3848
- #define **DGNLT_ODBC** 0x5e62
- #define **DGNLT_ORACLE** 0x6091
- #define **DGNLT_RIS** 0x71FB
- #define **DGNLT_SYBASE** 0x4f58
- #define **DGNLT_XBASE** 0x1971
- #define **DGNLT_SHAPE_FILL** 0x0041
- #define **DGNLT_ASSOC_ID** 0x7D2F
- #define **DGNCF_USE_SEED_UNITS** 0x01
- #define **DGNCF_USE_SEED_ORIGIN** 0x02
- #define **DGNCF_COPY_SEED_FILE_COLOR_TABLE** 0x04
- #define **DGNCF_COPY_WHOLE_SEED_FILE** 0x08
- #define **DGNBSC_CURVE_DISPLAY** 0x10
- #define **DGNBSC_POLY_DISPLAY** 0x20
- #define **DGNBSC_RATIONAL** 0x40
- #define **DGNBSC_CLOSED** 0x80
- #define **DGNBSS_ARC_SPACING** 0x40
- #define **DGNBSS_CLOSED** 0x80

Typedefs

- typedef struct _DGNTagDef DGNTagDef
- typedef void * DGNHandle

Functions

- DGNHandle CPL_DLL DGNOpen (const char *, int)
- void CPL_DLL DGNSetOptions (DGNHandle, int)
- int CPL_DLL DGNTestOpen (GByte *, int)
- const DGNElementInfo CPL_DLL * DGNGetElementIndex (DGNHandle, int *)
- int CPL_DLL DGNGetExtents (DGNHandle, double *)
- int CPL_DLL DGNGetDimension (DGNHandle)
- DGNElemCore CPL_DLL * DGNReadElement (DGNHandle)
- void CPL_DLL **DGNFreeElement** (DGNHandle, DGNElemCore *)
- void CPL_DLL DGNRewind (DGNHandle)
- int CPL_DLL DGNGotoElement (DGNHandle, int)
- void CPL_DLL DGNClose (DGNHandle)
- int CPL_DLL DGNLoadTCB (DGNHandle)
- int CPL_DLL DGNLookupColor (DGNHandle, int, int *, int *, int *)
- int CPL_DLL DGNGetShapeFillInfo (DGNHandle, DGNElemCore *, int *)
- int CPL_DLL DGNGetAssocID (DGNHandle, DGNElemCore *)
- int CPL_DLL DGNGetElementExtents (DGNHandle, DGNElemCore *, DGNPoint *, DGNPoint *)
- void CPL_DLL DGNDumpElement (DGNHandle, DGNElemCore *, FILE *)
- const char CPL_DLL * DGNTypeToName (int)
- void CPL_DLL **DGNRotationToQuaternion** (double, int *)
- void CPL_DLL **DGNQuaternionToMatrix** (int *, float *)
- int CPL_DLL **DGNStrokeArc** (DGNHandle, DGNElemArc *, int, DGNPoint *)
- int CPL_DLL **DGNStrokeCurve** (DGNHandle, DGNElemMultiPoint *, int, DGNPoint *)

- void CPL_DLL DGNSetSpatialFilter (DGNHandle hDGN, double dfXMin, double dfYMin, double dfXMax, double dfYMax)
- int CPL_DLL **DGNGetAttrLinkSize** (DGNHandle, DGNElemCore *, int)
- unsigned char CPL_DLL * DGNGetLinkage (DGNHandle hDGN, DGNElemCore *psElement, int iIndex, int *pnLinkageType, int *pnEntityNum, int *pnMSLink, int *pnLinkSize)
- int CPL_DLL DGNWriteElement (DGNHandle, DGNElemCore *)
- int CPL_DLL DGNResizeElement (DGNHandle, DGNElemCore *, int)
- DGNHandle CPL_DLL DGNCreate (const char *pszNewFilename, const char *pszSeedFile, int nCreation↩ Flags, double dfOriginX, double dfOriginY, double dfOriginZ, int nMasterUnitPerSubUnit, int nUORPerSub↩ Unit, const char *pszMasterUnits, const char *pszSubUnits)
- DGNElemCore CPL_DLL * **DGNCloneElement** (DGNHandle hDGNSrc, DGNHandle hDGNDst, DGNElemCore *psSrcElement)
- int CPL_DLL DGNUpdateElemCore (DGNHandle hDGN, DGNElemCore *psElement, int nLevel, int n↩ GraphicGroup, int nColor, int nWeight, int nStyle)
- int CPL_DLL **DGNUpdateElemCoreExtended** (DGNHandle hDGN, DGNElemCore *psElement)
- DGNElemCore CPL_DLL * DGNCreateMultiPointElem (DGNHandle hDGN, int nType, int nPointCount, DGNPoint *pasVertices)
- DGNElemCore CPL_DLL * **DGNCreateArcElem2D** (DGNHandle hDGN, int nType, double dfOriginX, double dfOriginY, double dfPrimaryAxis, double dfSecondaryAxis, double dfRotation, double dfStartAngle, double dfSweepAngle)
- DGNElemCore CPL_DLL * DGNCreateArcElem (DGNHandle hDGN, int nType, double dfOriginX, double dfOriginY, double dfOriginZ, double dfPrimaryAxis, double dfSecondaryAxis, double dfStartAngle, double dfSweepAngle, double dfRotation, int *panQuaternion)
- DGNElemCore CPL_DLL * DGNCreateConeElem (DGNHandle hDGN, double center_1X, double center_1Y, double center_1Z, double radius_1, double center_2X, double center_2Y, double center_2Z, double radius_2, int *panQuaternion)
- DGNElemCore CPL_DLL * DGNCreateTextElem (DGNHandle hDGN, const char *pszText, int nFontId, int nJustification, double dfLengthMult, double dfHeightMult, double dfRotation, int *panQuaternion, double df↩ OriginX, double dfOriginY, double dfOriginZ)
- DGNElemCore CPL_DLL * DGNCreateColorTableElem (DGNHandle hDGN, int nScreenFlag, GByte aby↩ ColorInfo[256][3])
- DGNElemCore CPL_DLL * DGNCreateComplexHeaderElem (DGNHandle hDGN, int nType, int nTotLength, int nNumElems)
- DGNElemCore CPL_DLL * DGNCreateComplexHeaderFromGroup (DGNHandle hDGN, int nType, int n↩ NumElems, DGNElemCore **papsElems)
- DGNElemCore CPL_DLL * DGNCreateSolidHeaderElem (DGNHandle hDGN, int nType, int nSurfType, int nBoundElems, int nTotLength, int nNumElems)
- DGNElemCore CPL_DLL * DGNCreateSolidHeaderFromGroup (DGNHandle hDGN, int nType, int nSurf↩ Type, int nBoundElems, int nNumElems, DGNElemCore **papsElems)
- DGNElemCore CPL_DLL * DGNCreateCellHeaderElem (DGNHandle hDGN, int nTotLength, const char *pszName, short nClass, short *panLevels, DGNPoint *psRangeLow, DGNPoint *psRangeHigh, DGNPoint *psOrigin, double dfXScale, double dfYScale, double dfRotation)
- DGNElemCore CPL_DLL * DGNCreateCellHeaderFromGroup (DGNHandle hDGN, const char *pszName, short nClass, short *panLevels, int nNumElems, DGNElemCore **papsElems, DGNPoint *psOrigin, double dfXScale, double dfYScale, double dfRotation)
- int CPL_DLL DGNAddMSLink (DGNHandle hDGN, DGNElemCore *psElement, int nLinkageType, int n↩ EntityNum, int nMSLink)
- int CPL_DLL DGNAddRawAttrLink (DGNHandle hDGN, DGNElemCore *psElement, int nLinkSize, unsigned char *pabyRawLinkData)
- int CPL_DLL DGNAddShapeFillInfo (DGNHandle hDGN, DGNElemCore *psElement, int nColor)
- int CPL_DLL DGNElemTypeHasDispHdr (int nElemType)

4.1.1 Detailed Description

Definitions of public structures and API of DGN Library.

4.1.2 Macro Definition Documentation

4.1.2.1 DGNST_ARC

```
#define DGNST_ARC 5
```

DGNElemCore style: Element uses DGNElemArc structure

4.1.2.2 DGNST_BSPLINE_CURVE_HEADER

```
#define DGNST_BSPLINE_CURVE_HEADER 15
```

DGNElemCore style: Element uses DGNElemBSplineCurveHeader structure

4.1.2.3 DGNST_BSPLINE_SURFACE_BOUNDARY

```
#define DGNST_BSPLINE_SURFACE_BOUNDARY 16
```

DGNElemCore style: Element uses DGNElemBSplineSurfaceBoundary structure

4.1.2.4 DGNST_BSPLINE_SURFACE_HEADER

```
#define DGNST_BSPLINE_SURFACE_HEADER 14
```

DGNElemCore style: Element uses DGNElemBSplineSurfaceHeader structure

4.1.2.5 DGNST_CELL_HEADER

```
#define DGNST_CELL_HEADER 8
```

DGNElemCore style: Element uses DGNElemCellHeader structure

4.1.2.6 DGNST_CELL_LIBRARY

```
#define DGNST_CELL_LIBRARY 11
```

DGNElemCore style: Element uses DGNElemCellLibrary structure

4.1.2.7 DGNST_COLORTABLE

```
#define DGNST_COLORTABLE 3
```

DGNElemCore style: Element uses DGNElemColorTable structure

4.1.2.8 DGNST_COMPLEX_HEADER

```
#define DGNST_COMPLEX_HEADER 7
```

DGNElemCore style: Element uses DGNElemComplexHeader structure

4.1.2.9 DGNST_CONE

```
#define DGNST_CONE 12
```

DGNElemCore style: Element uses DGNElemCone structure

4.1.2.10 DGNST_CORE

```
#define DGNST_CORE 1
```

DGNElemCore style: Element uses DGNElemCore structure

4.1.2.11 DGNST_KNOT_WEIGHT

```
#define DGNST_KNOT_WEIGHT 17
```

DGNElemCore style: Element uses DGNElemKnotWeight structure

4.1.2.12 DGNST_MULTIPOINT

```
#define DGNST_MULTIPOINT 2
```

DGNElemCore style: Element uses DGNElemMultiPoint structure

4.1.2.13 DGNST_SHARED_CELL_DEFN

```
#define DGNST_SHARED_CELL_DEFN 18
```

DGNElemCore style: Element uses DGNElemSharedCellDefn structure

4.1.2.14 DGNST_TAG_SET

```
#define DGNST_TAG_SET 10
```

DGNElemCore style: Element uses DGNElemTagSet structure

4.1.2.15 DGNST_TAG_VALUE

```
#define DGNST_TAG_VALUE 9
```

DGNElemCore style: Element uses DGNElemTagValue structure

4.1.2.16 DGNST_TCB

```
#define DGNST_TCB 4
```

DGNElemCore style: Element uses DGNElemTCB structure

4.1.2.17 DGNST_TEXT

```
#define DGNST_TEXT 6
```

DGNElemCore style: Element uses DGNElemText structure

4.1.2.18 DGNST_TEXT_NODE

```
#define DGNST_TEXT_NODE 13
```

DGNElemCore style: Element uses DGNElemTextNode structure

4.1.3 Typedef Documentation

4.1.3.1 DGNHandle

```
typedef void* DGNHandle
```

Opaque handle representing DGN file, used with DGN API.

4.1.3.2 DGNTagDef

```
typedef struct _DGNTagDef DGNTagDef
```

Tag definition.

Structure holding definition of one tag within a DGNTagSet.

4.1.4 Function Documentation

4.1.4.1 DGNAddMSLink()

```
int CPL_DLL DGNAddMSLink (
            DGNHandle hDGN,
            DGNElemCore * psElement,
            int nLinkageType,
            int nEntityNum,
            int nMSLink )
```

Add a database link to element.

The target element must already have raw_data loaded, and it will be resized (see DGNResizeElement()) as needed for the new attribute data. Note that the element is not written to disk immediate. Use DGNWriteElement() for that.

Parameters

hDGN	the file to which the element corresponds.
psElement	the element being updated.
nLinkageType	link type (DGNLT_*). Usually one of DGNLT_DMRS, DGNLT_INFORMIX, DGNLT_ODBC, DGNLT_ORACLE, DGNLT_RIS, DGNLT_SYBASE, or DGNLT_XBASE.
nEntityNum	indicator of the table referenced on target database.
nMSLink	indicator of the record referenced on target table.

Returns

-1 on failure, or the link index.

4.1.4.2 DGNAddRawAttrLink()

```
int CPL_DLL DGNAddRawAttrLink (
            DGNHandle hDGN,
            DGNElemCore * psElement,
            int nLinkSize,
            unsigned char * pabyRawLinkData )
```

Add a raw attribute linkage to element.

Given a raw data buffer, append it to this element as an attribute linkage without trying to interpret the linkage data.

The target element must already have raw_data loaded, and it will be resized (see DGNResizeElement()) as needed for the new attribute data. Note that the element is not written to disk immediate. Use DGNWriteElement() for that.

This function will take care of updating the "totlength" field of complex chain or shape headers to account for the extra attribute space consumed in the header element.

Parameters

hDGN	the file to which the element corresponds.
psElement	the element being updated.
nLinkSize	the size of the linkage in bytes.
pabyRawLinkData	the raw linkage data (nLinkSize bytes worth).

Returns

-1 on failure, or the link index.

4.1.4.3 DGNAddShapeFillInfo()

```
int CPL_DLL DGNAddShapeFillInfo (
            DGNHandle hDGN,
```

```
            DGNElemCore * psElement,
            int nColor )
```

Add a shape fill attribute linkage.

The target element must already have raw_data loaded, and it will be resized (see DGNResizeElement()) as needed for the new attribute data. Note that the element is not written to disk immediate. Use DGNWriteElement() for that.

Parameters

hDGN	the file to which the element corresponds.
psElement	the element being updated.
nColor	fill color (color index from palette).

Returns

-1 on failure, or the link index.

4.1.4.4 DGNClose()

```
void CPL_DLL DGNClose (
            DGNHandle hDGN )
```

Close DGN file.

Parameters

hDGN	Handle from DGNOpen() for file to close.

4.1.4.5 DGNCreate()

```
DGNHandle CPL_DLL DGNCreate (
            const char * pszNewFilename,
            const char * pszSeedFile,
            int nCreationFlags,
            double dfOriginX,
            double dfOriginY,
            double dfOriginZ,
            int nSubUnitsPerMasterUnit,
            int nUORPerSubUnit,
            const char * pszMasterUnits,
            const char * pszSubUnits )
```

Create new DGN file.

This function will create a new DGN file based on the provided seed file, and return a handle on which elements may be read and written.

The following creation flags may be passed:

- DGNCF_USE_SEED_UNITS: The master and subunit resolutions and names from the seed file will be used in the new file. The nMasterUnitPerSubUnit, nUORPerSubUnit, pszMasterUnits, and pszSubUnits arguments will be ignored.
- DGNCF_USE_SEED_ORIGIN: The origin from the seed file will be used and the X, Y and Z origin passed into the call will be ignored.
- DGNCF_COPY_SEED_FILE_COLOR_TABLE: Should the first color table occurring in the seed file also be copied?
- DGNCF_COPY_WHOLE_SEED_FILE: By default only the first three elements (TCB, Digitizer Setup and Level Symbology) are copied from the seed file. If this flag is provided the entire seed file is copied verbatim (with the TCB origin and units possibly updated).

Parameters

pszNewFilename	the filename to create. If it already exists it will be overwritten.
pszSeedFile	the seed file to copy header from.
nCreationFlags	An ORing of DGNCF_* flags that are to take effect.
dfOriginX	the X origin for the file.
dfOriginY	the Y origin for the file.
dfOriginZ	the Z origin for the file.
nSubUnitsPerMasterUnit	the number of subunits in one master unit.
nUORPerSubUnit	the number of UOR (units of resolution) per subunit.
pszMasterUnits	the name of the master units (2 characters).
pszSubUnits	the name of the subunits (2 characters).

4.1.4.6 DGNCreateArcElem()

```
DGNElemCore CPL_DLL* DGNCreateArcElem (
            DGNHandle hDGN,
            int nType,
            double dfOriginX,
            double dfOriginY,
            double dfOriginZ,
            double dfPrimaryAxis,
            double dfSecondaryAxis,
            double dfStartAngle,
            double dfSweepAngle,
            double dfRotation,
            int * panQuaternion )
```

Create Arc or Ellipse element.

Create a new 2D or 3D arc or ellipse element. The start angle, and sweep angle are ignored for DGNT_ELLIPSE but used for DGNT_ARC.

The newly created element will still need to be written to file using DGNWriteElement(). Also the level and other core values will be defaulted. Use DGNUpdateElemCore() on the element before writing to set these values.

Parameters

hDGN	the DGN file on which the element will eventually be written.
nType	either DGNT_ELLIPSE or DGNT_ARC to select element type.
dfOriginX	the origin (center of rotation) of the arc (X).
dfOriginY	the origin (center of rotation) of the arc (Y).
dfOriginZ	the origin (center of rotation) of the arc (Y).
dfPrimaryAxis	the length of the primary axis.
dfSecondaryAxis	the length of the secondary axis.
dfStartAngle	start angle, degrees counterclockwise of primary axis.
dfSweepAngle	sweep angle, degrees
dfRotation	Counterclockwise rotation in degrees.
panQuaternion	3D orientation quaternion (NULL to use rotation).

Returns

the new element (DGNElemArc) or NULL on failure.

4.1.4.7 DGNCreateCellHeaderElem()

```
DGNElemCore CPL_DLL* DGNCreateCellHeaderElem (
            DGNHandle hDGN,
            int nTotLength,
            const char * pszName,
            short nClass,
            short * panLevels,
            DGNPoint * psRangeLow,
            DGNPoint * psRangeHigh,
            DGNPoint * psOrigin,
            double dfXScale,
            double dfYScale,
            double dfRotation )
```

Create cell header.

The newly created element will still need to be written to file using DGNWriteElement(). Also the level and other core values will be defaulted. Use DGNUpdateElemCore() on the element before writing to set these values.

Generally speaking the function DGNCreateCellHeaderFromGroup() should be used instead of this function.

Parameters

hDGN	the file handle on which the element is to be written.
nTotLength	total length of cell in words not including the 38 bytes of the cell header that occur before the totlength indicator.
nClass	the class value for the cell.
panLevels	an array of shorts holding the bit mask of levels in effect for this cell. This array should contain 4 shorts (64 bits).
psRangeLow	the cell diagonal origin in original cell file coordinates.
psRangeHigh	the cell diagonal top left corner in original cell file coordinates.

Parameters

psOrigin	the origin of the cell in output file coordinates.
dfXScale	the amount of scaling applied in the X dimension in mapping from cell file coordinates to output file coordinates.
dfYScale	the amount of scaling applied in the Y dimension in mapping from cell file coordinates to output file coordinates.
dfRotation	the amount of rotation (degrees counterclockwise) in mapping from cell coordinates to output file coordinates.

Returns

the new element (DGNElemCellHeader) or NULL on failure.

4.1.4.8 DGNCreateCellHeaderFromGroup()

```
DGNElemCore CPL_DLL* DGNCreateCellHeaderFromGroup (
            DGNHandle hDGN,
            const char * pszName,
            short nClass,
            short * panLevels,
            int nNumElems,
            DGNElemCore ** papsElems,
            DGNPoint * psOrigin,
            double dfXScale,
            double dfYScale,
            double dfRotation )
```

Create cell header from a group of elements.

The newly created element will still need to be written to file using DGNWriteElement(). Also the level and other core values will be defaulted. Use DGNUpdateElemCore() on the element before writing to set these values.

This function will compute the total length, bounding box, and diagonal range values from the set of provided elements. Note that the proper diagonal range values will only be written if 1.0 is used for the x and y scale values, and 0.0 for the rotation. Use of other values will result in incorrect scaling handles being presented to the user in Microstation when they select the element.

Parameters

hDGN	the file handle on which the element is to be written.
nClass	the class value for the cell.
panLevels	an array of shorts holding the bit mask of levels in effect for this cell. This array should contain 4 shorts (64 bits). This array would normally be passed in as NULL, and the function will build a mask from the passed list of elements.
psOrigin	the origin of the cell in output file coordinates.
dfXScale	the amount of scaling applied in the X dimension in mapping from cell file coordinates to output file coordinates.
dfYScale	the amount of scaling applied in the Y dimension in mapping from cell file coordinates to output file coordinates.
dfRotation	the amount of rotation (degrees counterclockwise) in mapping from cell coordinates to output file coordinates.

Returns

the new element (DGNElemCellHeader) or NULL on failure.

4.1.4.9 DGNCreateColorTableElem()

```
DGNElemCore CPL_DLL* DGNCreateColorTableElem (
            DGNHandle hDGN,
            int nScreenFlag,
            GByte abyColorInfo[256][3] )
```

Create color table element.

Creates a color table element with the indicated color table.

Note that color table elements are actually of type DGNT_GROUP_DATA(5) and always on level 1. Do not alter the level with DGNUpdateElemCore() or the element will essentially be corrupt.

The newly created element will still need to be written to file using DGNWriteElement(). Also the level and other core values will be defaulted. Use DGNUpdateElemCore() on the element before writing to set these values.

Parameters

hDGN	the file to which the element will eventually be written.
nScreenFlag	the screen to which the color table applies (0 = left, 1 = right).
abyColorInfo	array of 256 color entries. The first is the background color.

Returns

the new element (DGNElemColorTable) or NULL on failure.

4.1.4.10 DGNCreateComplexHeaderElem()

```
DGNElemCore CPL_DLL* DGNCreateComplexHeaderElem (
            DGNHandle hDGN,
            int nType,
            int nTotLength,
            int nNumElems )
```

Create complex chain/shape header.

The newly created element will still need to be written to file using DGNWriteElement(). Also the level and other core values will be defaulted. Use DGNUpdateElemCore() on the element before writing to set these values.

The nTotLength is the sum of the size of all elements in the complex group plus 5. The DGNCreateComplexHeaderFromGroup() can be used to build a complex element from the members more conveniently.

Parameters

hDGN	the file on which the element will be written.
nType	DGNT_COMPLEX_CHAIN_HEADER or DGNT_COMPLEX_SHAPE_HEADER. depending on whether the list is open or closed (last point equal to last) or if the object represents a surface or a solid.
nTotLength	the value of the totlength field in the element.
nNumElems	the number of elements in the complex group not including the header element.

Returns

the new element (DGNElemComplexHeader) or NULL on failure.

4.1.4.11 DGNCreateComplexHeaderFromGroup()

```
DGNElemCore CPL_DLL* DGNCreateComplexHeaderFromGroup (
            DGNHandle hDGN,
            int nType,
            int nNumElems,
            DGNElemCore ** papsElems )
```

Create complex chain/shape header.

This function is similar to DGNCreateComplexHeaderElem(), but it takes care of computing the total size of the set of elements being written, and collecting the bounding extents. It also takes care of some other convenience issues, like marking all the member elements as complex, and setting the level based on the level of the member elements.

Parameters

hDGN	the file on which the element will be written.
nType	DGNT_COMPLEX_CHAIN_HEADER or DGNT_COMPLEX_SHAPE_HEADER. depending on whether the list is open or closed (last point equal to last) or if the object represents a surface or a solid.
nNumElems	the number of elements in the complex group not including the header element.
papsElems	array of pointers to nNumElems elements in the complex group. Some updates may be made to these elements.

Returns

the new element (DGNElemComplexHeader) or NULL on failure.

4.1.4.12 DGNCreateConeElem()

```
DGNElemCore CPL_DLL* DGNCreateConeElem (
            DGNHandle hDGN,
```

```
            double dfCenter_1X,
            double dfCenter_1Y,
            double dfCenter_1Z,
            double dfRadius_1,
            double dfCenter_2X,
            double dfCenter_2Y,
            double dfCenter_2Z,
            double dfRadius_2,
            int * panQuaternion )
```

Create Cone element.

Create a new 3D cone element.

The newly created element will still need to be written to file using DGNWriteElement(). Also the level and other core values will be defaulted. Use DGNUpdateElemCore() on the element before writing to set these values.

Parameters

hDGN	the DGN file on which the element will eventually be written.
dfCenter_1X	the center of the first bounding circle (X).
dfCenter_1Y	the center of the first bounding circle (Y).
dfCenter_1Z	the center of the first bounding circle (Z).
dfRadius_1	the radius of the first bounding circle.
dfCenter_2X	the center of the second bounding circle (X).
dfCenter_2Y	the center of the second bounding circle (Y).
dfCenter_2Z	the center of the second bounding circle (Z).
dfRadius_2	the radius of the second bounding circle.
panQuaternion	3D orientation quaternion (NULL for default orientation - circles parallel to the X-Y plane).

Returns

the new element (DGNElemCone) or NULL on failure.

4.1.4.13 DGNCreateMultiPointElem()

```
DGNElemCore CPL_DLL* DGNCreateMultiPointElem (
            DGNHandle hDGN,
            int nType,
            int nPointCount,
            DGNPoint * pasVertices )
```

Create new multi-point element.

The newly created element will still need to be written to file using DGNWriteElement(). Also the level and other core values will be defaulted. Use DGNUpdateElemCore() on the element before writing to set these values.

NOTE: There are restrictions on the nPointCount for some elements. For instance, DGNT_LINE can only have 2 points. Maximum element size precludes very large numbers of points.

Parameters

hDGN	the file on which the element will eventually be written.
nType	the type of the element to be created. It must be one of DGNT_LINE, DGNT_LINE_STRING, DGNT_SHAPE, DGNT_CURVE or DGNT_BSPLINE_POLE.
nPointCount	the number of points in the pasVertices list.
pasVertices	the list of points to be written.

Returns

the new element (a DGNElemMultiPoint structure) or NULL on failure.

4.1.4.14 DGNCreateSolidHeaderElem()

```
DGNElemCore CPL_DLL* DGNCreateSolidHeaderElem (
            DGNHandle hDGN,
            int nType,
            int nSurfType,
            int nBoundElems,
            int nTotLength,
            int nNumElems )
```

Create 3D solid/surface.

The newly created element will still need to be written to file using DGNWriteElement(). Also the level and other core values will be defaulted. Use DGNUpdateElemCore() on the element before writing to set these values.

The nTotLength is the sum of the size of all elements in the solid group plus 6. The DGNCreateSolidHeaderFromGroup() can be used to build a solid element from the members more conveniently.

Parameters

hDGN	the file on which the element will be written.
nType	DGNT_3DSURFACE_HEADER or DGNT_3DSOLID_HEADER.
nSurfType	the surface/solid type, one of DGNSUT_* or DGNSOT_*.
nBoundElems	the number of elements in each boundary.
nTotLength	the value of the totlength field in the element.
nNumElems	the number of elements in the solid not including the header element.

Returns

the new element (DGNElemComplexHeader) or NULL on failure.

4.1.4.15 DGNCreateSolidHeaderFromGroup()

```
DGNElemCore CPL_DLL* DGNCreateSolidHeaderFromGroup (
            DGNHandle hDGN,
```

```
            int nType,
            int nSurfType,
            int nBoundElems,
            int nNumElems,
            DGNElemCore ** papsElems )
```

Create 3D solid/surface header.

This function is similar to DGNCreateSolidHeaderElem(), but it takes care of computing the total size of the set of elements being written, and collecting the bounding extents. It also takes care of some other convenience issues, like marking all the member elements as complex, and setting the level based on the level of the member elements.

Parameters

hDGN	the file on which the element will be written.
nType	DGNT_3DSURFACE_HEADER or DGNT_3DSOLID_HEADER.
nSurfType	the surface/solid type, one of DGNSUT_* or DGNSOT_*.
nBoundElems	the number of boundary elements.
nNumElems	the number of elements in the solid not including the header element.
papsElems	array of pointers to nNumElems elements in the solid. Some updates may be made to these elements.

Returns

the new element (DGNElemComplexHeader) or NULL on failure.

4.1.4.16 DGNCreateTextElem()

```
DGNElemCore CPL_DLL* DGNCreateTextElem (
            DGNHandle hDGN,
            const char * pszText,
            int nFontId,
            int nJustification,
            double dfLengthMult,
            double dfHeightMult,
            double dfRotation,
            int * panQuaternion,
            double dfOriginX,
            double dfOriginY,
            double dfOriginZ )
```

Create text element.

The newly created element will still need to be written to file using DGNWriteElement(). Also the level and other core values will be defaulted. Use DGNUpdateElemCore() on the element before writing to set these values.

Parameters

hDGN	the file on which the element will eventually be written.
pszText	the string of text.
nFontId	microstation font id for the text. 1 may be used as default.

Parameters

nJustification	text justification. One of DGNJ_LEFT_TOP, DGNJ_LEFT_CENTER, DGNJ_LEFT_BOTTOM, DGNJ_CENTER_TOP, DGNJ_CENTER_CENTER, DGNJ_CENTER_BOTTOM, DGNJ_RIGHT_TOP, DGNJ_RIGHT_CENTER, DGNJ_RIGHT_BOTTOM.
dfLengthMult	character width in master units.
dfHeightMult	character height in master units.
dfRotation	Counterclockwise text rotation in degrees.
panQuaternion	3D orientation quaternion (NULL to use rotation).
dfOriginX	Text origin (X).
dfOriginY	Text origin (Y).
dfOriginZ	Text origin (Z).

Returns

the new element (DGNElemText) or NULL on failure.

4.1.4.17 DGNDumpElement()

```
void CPL_DLL DGNDumpElement (
            DGNHandle hDGN,
            DGNElemCore * psElement,
            FILE * fp )
```

Emit textual report of an element.

This function exists primarily for debugging, and will produce a textual report about any element type to the designated file.

Parameters

hDGN	the file from which the element originated.
psElement	the element to report on.
fp	the file (such as stdout) to report the element information to.

4.1.4.18 DGNElemTypeHasDispHdr()

```
int CPL_DLL DGNElemTypeHasDispHdr (
            int nElemType )
```

Does element type have display header.

Parameters

nElemType	element type (0-63) to test.

Returns

TRUE if elements of passed in type have a display header after the core element header, or FALSE otherwise.

4.1.4.19 DGNGetAssocID()

```
int CPL_DLL DGNGetAssocID (
            DGNHandle hDGN,
            DGNElemCore * psElem )
```

Fetch association id for an element.

This method will check if an element has an association id, and if so returns it, otherwise returning -1. Association ids are kept as a user attribute linkage where present.

Parameters

hDGN	the file.
psElem	the element.

Returns

The id or -1 on failure.

4.1.4.20 DGNGetDimension()

```
int CPL_DLL DGNGetDimension (
            DGNHandle hDGN )
```

Return 2D/3D dimension of file.

Return 2 or 3 depending on the dimension value of the provided file.

4.1.4.21 DGNGetElementExtents()

```
int CPL_DLL DGNGetElementExtents (
            DGNHandle hDGN,
            DGNElemCore * psElement,
            DGNPoint * psMin,
            DGNPoint * psMax )
```

Fetch extents of an element.

This function will return the extents of the passed element if possible. The extents are extracted from the element header if it contains them, and transformed into master georeferenced format. Some element types do not have extents at all and will fail.

This call will also fail if the extents raw data for the element is not available. This will occur if it was not the most recently read element, and if the raw_data field is not loaded.

Parameters

hDGN	the handle of the file to read from.
psElement	the element to extract extents from.
psMin	structure loaded with X, Y and Z minimum values for the extent.
psMax	structure loaded with X, Y and Z maximum values for the extent.

Returns

TRUE on success of FALSE if extracting extents fails.

4.1.4.22 DGNGetElementIndex()

```
const DGNElementInfo CPL_DLL* DGNGetElementIndex (
            DGNHandle hDGN,
            int * pnElementCount )
```

Fetch element index.

This function will return an array with brief information about every element in a DGN file. It requires one pass through the entire file to generate (this is not repeated on subsequent calls).

The returned array of DGNElementInfo structures contain the level, type, stype, and other flags for each element in the file. This can facilitate application level code representing the number of elements of various types efficiently.

Note that while building the index requires one pass through the whole file, it does not generally request much processing for each element.

Parameters

hDGN	the file to get an index for.
pnElementCount	the integer to put the total element count into.

Returns

a pointer to an internal array of DGNElementInfo structures (there will be ∗pnElementCount entries in the array), or NULL on failure. The returned array should not be modified or freed, and will last only as long as the DGN file remains open.

4.1.4.23 DGNGetExtents()

```
int CPL_DLL DGNGetExtents (
            DGNHandle hDGN,
            double * padfExtents )
```

Fetch overall file extents.

The extents are collected for each element while building an index, so if an index has not already been built, it will be built when DGNGetExtents() is called.

The Z min/max values are generally meaningless (0 and 0xffffffff in uor space).

Parameters

hDGN	the file to get extents for.
padfExtents	pointer to an array of six doubles into which are loaded the values xmin, ymin, zmin, xmax, ymax, and zmax.

Returns

TRUE on success or FALSE on failure.

4.1.4.24 DGNGetLinkage()

```
unsigned char CPL_DLL* DGNGetLinkage (
            DGNHandle hDGN,
            DGNElemCore * psElement,
            int iIndex,
            int * pnLinkageType,
            int * pnEntityNum,
            int * pnMSLink,
            int * pnLength )
```

Returns requested linkage raw data.

A pointer to the raw data for the requested attribute linkage is returned as well as (potentially) various information about the linkage including the linkage type, database entity number and MSLink value, and the length of the raw linkage data in bytes.

If the requested linkage (iIndex) does not exist a value of zero is returned.

The entity number is (loosely speaking) the index of the table within the current database to which the MSLINK value will refer. The entity number should be used to lookup the table name in the MSCATALOG table. The MSLINK value is the key value for the record in the target table.

Parameters

hDGN	the file from which the element originated.
psElement	the element to report on.
iIndex	the zero based index of the linkage to fetch.
pnLinkageType	variable to return linkage type. This may be one of the predefined DGNLT_ values or a different value. This pointer may be NULL.
pnEntityNum	variable to return the entity number in or NULL if not required.
pnMSLink	variable to return the MSLINK value in, or NULL if not required.
pnLength	variable to returned the linkage size in bytes or NULL.

Returns

pointer to raw internal linkage data. This data should not be altered or freed. NULL returned on failure.

4.1.4.25 DGNGetShapeFillInfo()

```
int CPL_DLL DGNGetShapeFillInfo (
            DGNHandle hDGN,
            DGNElemCore * psElem,
            int * pnColor )
```

Fetch fill color for a shape.

This method will check for a 0x0041 user attribute linkaged with fill color information for the element. If found the function returns TRUE, and places the fill color in ∗pnColor, otherwise FALSE is returned and ∗pnColor is not updated.

Parameters

hDGN	the file.
psElem	the element.
pnColor	the location to return the fill color.

Returns

TRUE on success or FALSE on failure.

4.1.4.26 DGNGotoElement()

```
int CPL_DLL DGNGotoElement (
            DGNHandle hDGN,
            int element_id )
```

Seek to indicated element.

Changes what element will be read on the next call to DGNReadElement(). Note that this function requires and index, and one will be built if not already available.

Parameters

hDGN	the file to affect.
element↩ _id	the element to seek to. These values are sequentially ordered starting at zero for the first element.

Returns

returns TRUE on success or FALSE on failure.

4.1.4.27 DGNLoadTCB()

```
int CPL_DLL DGNLoadTCB (
            DGNHandle hDGN )
```

Load TCB if not already loaded.

This function will load the TCB element if it is not already loaded. It is used primarily to ensure the TCB is loaded before doing any operations that require TCB values (like creating new elements).

Returns

FALSE on failure or TRUE on success.

4.1.4.28 DGNLookupColor()

```
int CPL_DLL DGNLookupColor (
            DGNHandle hDGN,
            int color_index,
            int * red,
            int * green,
            int * blue )
```

Translate color index into RGB values.

If no color table has yet been encountered in the file a hard-coded "default" color table will be used. This seems to be what Microstation uses as a color table when there isn't one in a DGN file but I am not absolutely convinced it is appropriate.

Parameters

hDGN	the file.
color_index	the color index to lookup.
red	location to put red component.
green	location to put green component.
blue	location to put blue component.

Returns

TRUE on success or FALSE on failure. May fail if color_index is out of range.

4.1.4.29 DGNOpen()

```
DGNHandle CPL_DLL DGNOpen (
            const char * pszFilename,
            int bUpdate )
```

Open a DGN file.

The file is opened, and minimally verified to ensure it is a DGN (ISFF) file. If the file cannot be opened for read access an error with code CPLE_OpenFailed with be reported via CPLError() and NULL returned. If the file header

does not appear to be a DGN file, an error with code CPLE_AppDefined will be reported via CPLError(), and NULL returned.

If successful a handle for further access is returned. This should be closed with DGNClose() when no longer needed.

DGNOpen() does not scan the file on open, and should be very fast even for large files.

Parameters

pszFilename	name of file to try opening.
bUpdate	should the file be opened with read+update (r+) mode?

Returns

handle to use for further access to file using DGN API, or NULL if open fails.

4.1.4.30 DGNReadElement()

```
DGNElemCore CPL_DLL* DGNReadElement (
            DGNHandle hDGN )
```

Read a DGN element.

This function will return the next element in the file, starting with the first. It is affected by DGNGotoElement() calls.

The element is read into a structure which includes the DGNElemCore structure. It is expected that applications will inspect the stype field of the returned DGNElemCore and use it to cast the pointer to the appropriate element structure type such as DGNElemMultiPoint.

Parameters

hDGN	the handle of the file to read from.

Returns

pointer to element structure, or NULL on EOF or processing error. The structure should be freed with DGN↩ FreeElement() when no longer needed.

4.1.4.31 DGNResizeElement()

```
int CPL_DLL DGNResizeElement (
            DGNHandle hDGN,
            DGNElemCore * psElement,
            int nNewSize )
```

Resize an existing element.

If the new size is the same as the old nothing happens.

Otherwise, the old element in the file is marked as deleted, and the DGNElemCore.offset and element_id are set to -1 indicating that the element should be written to the end of file when next written by DGNWriteElement(). The internal raw data buffer is updated to the new size.

Only elements with "raw_data" loaded may be moved.

In normal use the DGNResizeElement() call would be called on a previously loaded element, and afterwards the raw_data would be updated before calling DGNWriteElement(). If DGNWriteElement() isn't called after DGNResizeElement() then the element will be lost having been marked as deleted in its old position but never written at the new location.

Parameters

hDGN	the DGN file on which the element lives.
psElement	the element to alter.
nNewSize	the desired new size of the element in bytes. Must be a multiple of 2.

Returns

TRUE on success, or FALSE on error.

4.1.4.32 DGNRewind()

```
void CPL_DLL DGNRewind (
            DGNHandle hDGN )
```

Rewind element reading.

Rewind the indicated DGN file, so the next element read with DGNReadElement() will be the first. Does not require indexing like the more general DGNReadElement() function.

Parameters

hDGN	handle to file.

4.1.4.33 DGNSetOptions()

```
void CPL_DLL DGNSetOptions (
            DGNHandle hDGN,
            int nOptions )
```

Set file access options.

Sets a flag affecting how the file is accessed. Currently there is only one support flag:

DGNO_CAPTURE_RAW_DATA: If this is enabled (it is off by default), then the raw binary data associated with elements will be kept in the raw_data field within the DGNElemCore when they are read. This is required if the application needs to interpret the raw data itself. It is also necessary if the element is to be written back to this file, or another file using DGNWriteElement(). Off by default (to conserve memory).

Parameters

hDGN	handle to file returned by DGNOpen().
nOptions	ORed option flags.

4.1.4.34 DGNSetSpatialFilter()

```
void CPL_DLL DGNSetSpatialFilter (
            DGNHandle hDGN,
            double dfXMin,
            double dfYMin,
            double dfXMax,
            double dfYMax )
```

Set rectangle for which features are desired.

If a spatial filter is set with this function, DGNReadElement() will only return spatial elements (elements with a known bounding box) and only those elements for which this bounding box overlaps the requested region.

If all four values (dfXMin, dfXMax, dfYMin and dfYMax) are zero, the spatial filter is disabled. Note that installing a spatial filter won't reduce the amount of data read from disk. All elements are still scanned, but the amount of processing work for elements outside the spatial filter is minimized.

Parameters

hDGN	Handle from DGNOpen() for file to update.
dfXMin	minimum x coordinate for extents (georeferenced coordinates).
dfYMin	minimum y coordinate for extents (georeferenced coordinates).
dfXMax	maximum x coordinate for extents (georeferenced coordinates).
dfYMax	maximum y coordinate for extents (georeferenced coordinates).

4.1.4.35 DGNTestOpen()

```
int CPL_DLL DGNTestOpen (
            GByte * pabyHeader,
            int nByteCount )
```

Test if header is DGN.

Parameters

pabyHeader	block of header data from beginning of file.
nByteCount	number of bytes in pabyHeader.

Returns

TRUE if the header appears to be from a DGN file, otherwise FALSE.

4.1.4.36 DGNTypeToName()

```
const char CPL_DLL* DGNTypeToName (
            int nType )
```

Convert type to name.

Returns a human readable name for an element type such as DGNT_LINE.

Parameters

nType	the DGNT_* type code to translate.

Returns

a pointer to an internal string with the translation. This string should not be modified or freed.

4.1.4.37 DGNUpdateElemCore()

```
int CPL_DLL DGNUpdateElemCore (
            DGNHandle hDGN,
            DGNElemCore * psElement,
            int nLevel,
            int nGraphicGroup,
            int nColor,
            int nWeight,
            int nStyle )
```

Change element core values.

The indicated values in the element are updated in the structure, as well as in the raw data. The updated element is not written to disk. That must be done with DGNWriteElement(). The element must have raw_data loaded.

Parameters

hDGN	the file on which the element belongs.
psElement	the element to modify.
nLevel	the new level value.
nGraphicGroup	the new graphic group value.
nColor	the new color index.
nWeight	the new element weight.
nStyle	the new style value for the element.

Returns

Returns TRUE on success or FALSE on failure.

4.1.4.38 DGNWriteElement()

```
int CPL_DLL DGNWriteElement (
            DGNHandle hDGN,
            DGNElemCore * psElement )
```

Write element to file.

Only elements with "raw_data" loaded may be written. This should include elements created with the various DGNCreate*() functions, and those read from the file with the DGNO_CAPTURE_RAW_DATA flag turned on with DGNSetOptions().

The passed element is written to the indicated file. If the DGNElemCore.offset field is -1 then the element is written at the end of the file (and offset/element are reset properly) otherwise the element is written back to the location indicated by DGNElemCore.offset.

If the element is added at the end of the file, and if an element index has already been built, it will be updated to reference the new element.

This function takes care of ensuring that the end-of-file marker is maintained after the last element.

Parameters

hDGN	the file to write the element to.
psElement	the element to write.

Returns

TRUE on success or FALSE in case of failure.

Index