# dxflib

Generated by Doxygen 1.9.5

# Chapter 1

# Todo List

**Member DL_Dxf::addAttribute (DL_CreationInterface *creationInterface)**

add attrib instead of normal text

**Member DL_Dxf::getStrippedLine (std::string &s, unsigned int size, FILE *stream, bool stripSpace=true)**

Change function to use safer FreeBSD strl* functions

Is it a problem if line is blank (i.e., newline only)? Then, when function returns, (s==NULL).

**Class DL_Writer**

Add error checking for string/entry length.

**Class DL_WriterA**

What if `fname` is NULL? Or `fname` can't be opened for another reason?

# Chapter 2

# Hierarchical Index

## 2.1 Class Hierarchy

This inheritance list is sorted roughly, but not completely, alphabetically:

# Chapter 3

# Class Index

## 3.1 Class List

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

# Chapter 4

# File Index

## 4.1 File List

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

# Chapter 5

# Class Documentation

## 5.1 DL_ArcAlignedTextData Struct Reference

Arc Aligned Text Data.

`#include <dl_entities.h>`

### Public Attributes

- std::string text
- std::string font
- std::string style
- double cx
- double cy
- double cz
- double radius
- double xScaleFactor
- double height
- double spacing
- double offset
- double rightOffset
- double leftOffset
- double startAngle
- double endAngle
- bool reversedCharacterOrder
- int direction
- int alignment
- int side
- bool bold
- bool italic
- bool underline
- int characerSet
- int pitch
- bool shxFont
- bool wizard
- int arcHandle

### 5.1.1 Detailed Description

Arc Aligned Text Data.

### 5.1.2 Member Data Documentation

#### 5.1.2.1 alignment

```
int DL_ArcAlignedTextData::alignment
```

Alignment: 1: fit 2: left 3: right 4: center

Referenced by DL_Dxf::addArcAlignedText().

#### 5.1.2.2 arcHandle

```
int DL_ArcAlignedTextData::arcHandle
```

Arc handle/ID

Referenced by DL_Dxf::addArcAlignedText().

#### 5.1.2.3 bold

```
bool DL_ArcAlignedTextData::bold
```

Bold flag

Referenced by DL_Dxf::addArcAlignedText().

#### 5.1.2.4 characerSet

```
int DL_ArcAlignedTextData::characerSet
```

Character set value. Windows character set identifier.

Referenced by DL_Dxf::addArcAlignedText().

#### 5.1.2.5 cx

```
double DL_ArcAlignedTextData::cx
```

X coordinate of arc center point.

Referenced by DL_Dxf::addArcAlignedText().

#### 5.1.2.6 cy

```
double DL_ArcAlignedTextData::cy
```

Y coordinate of arc center point.

Referenced by DL_Dxf::addArcAlignedText().

#### 5.1.2.7 cz

```
double DL_ArcAlignedTextData::cz
```

Z coordinate of arc center point.

Referenced by DL_Dxf::addArcAlignedText().

#### 5.1.2.8 direction

```
int DL_ArcAlignedTextData::direction
```

Direction 1: outward from center 2: inward from center

Referenced by DL_Dxf::addArcAlignedText().

#### 5.1.2.9 endAngle

```
double DL_ArcAlignedTextData::endAngle
```

End angle (radians)

Referenced by DL_Dxf::addArcAlignedText().

#### 5.1.2.10 font

```
std::string DL_ArcAlignedTextData::font
```

Font name

Referenced by DL_Dxf::addArcAlignedText().

#### 5.1.2.11 height

```
double DL_ArcAlignedTextData::height
```

Text height

Referenced by DL_Dxf::addArcAlignedText().

#### 5.1.2.12 italic

```
bool DL_ArcAlignedTextData::italic
```

Italic flag

Referenced by DL_Dxf::addArcAlignedText().

#### 5.1.2.13 leftOffset

```
double DL_ArcAlignedTextData::leftOffset
```

Left offset

Referenced by DL_Dxf::addArcAlignedText().

#### 5.1.2.14 offset

```
double DL_ArcAlignedTextData::offset
```

Offset from arc

Referenced by DL_Dxf::addArcAlignedText().

#### 5.1.2.15 pitch

```
int DL_ArcAlignedTextData::pitch
```

Pitch and family value. Windows pitch and character family identifier.

Referenced by DL_Dxf::addArcAlignedText().

#### 5.1.2.16 radius

```
double DL_ArcAlignedTextData::radius
```

Arc radius.

Referenced by DL_Dxf::addArcAlignedText().

#### 5.1.2.17 reversedCharacterOrder

```
bool DL_ArcAlignedTextData::reversedCharacterOrder
```

Reversed character order: false: normal true: reversed

Referenced by DL_Dxf::addArcAlignedText().

#### 5.1.2.18 rightOffset

```
double DL_ArcAlignedTextData::rightOffset
```

Right offset

Referenced by DL_Dxf::addArcAlignedText().

#### 5.1.2.19 shxFont

```
bool DL_ArcAlignedTextData::shxFont
```

Font type: false: TTF true: SHX

Referenced by DL_Dxf::addArcAlignedText().

#### 5.1.2.20 side

```
int DL_ArcAlignedTextData::side
```

Side 1: convex 2: concave

Referenced by DL_Dxf::addArcAlignedText().

#### 5.1.2.21 spacing

```
double DL_ArcAlignedTextData::spacing
```

Character spacing

Referenced by DL_Dxf::addArcAlignedText().

#### 5.1.2.22 startAngle

```
double DL_ArcAlignedTextData::startAngle
```

Start angle (radians)

Referenced by DL_Dxf::addArcAlignedText().

#### 5.1.2.23 style

```
std::string DL_ArcAlignedTextData::style
```

Style

Referenced by DL_Dxf::addArcAlignedText().

#### 5.1.2.24 text

```
std::string DL_ArcAlignedTextData::text
```

Text string

Referenced by DL_Dxf::addArcAlignedText().

#### 5.1.2.25 underline

```
bool DL_ArcAlignedTextData::underline
```

Underline flag

Referenced by DL_Dxf::addArcAlignedText().

#### 5.1.2.26 wizard

```
bool DL_ArcAlignedTextData::wizard
```

Wizard flag

Referenced by DL_Dxf::addArcAlignedText().

#### 5.1.2.27 xScaleFactor

```
double DL_ArcAlignedTextData::xScaleFactor
```

Relative X scale factor.

Referenced by DL_Dxf::addArcAlignedText().

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

- src/dl_entities.h

## 5.2 DL_ArcData Struct Reference

Arc Data.

```
#include <dl_entities.h>
```

### Public Member Functions

- DL_ArcData (double acx, double acy, double acz, double aRadius, double aAngle1, double aAngle2)
  *Constructor.*

### Public Attributes

- double cx
- double cy
- double cz
- double radius
- double angle1
- double angle2

### 5.2.1 Detailed Description

Arc Data.

### 5.2.2 Constructor & Destructor Documentation

#### 5.2.2.1 DL_ArcData()

```
DL_ArcData::DL_ArcData (
            double acx,
            double acy,
            double acz,
            double aRadius,
            double aAngle1,
            double aAngle2 )  [inline]
```

Constructor.

Parameters: see member variables.

### 5.2.3 Member Data Documentation

#### 5.2.3.1 angle1

```
double DL_ArcData::angle1
```

Startangle of arc in degrees.

Referenced by DL_Dxf::writeArc().

#### 5.2.3.2 angle2

```
double DL_ArcData::angle2
```

Endangle of arc in degrees.

Referenced by DL_Dxf::writeArc().

#### 5.2.3.3 cx

`double DL_ArcData::cx`

X Coordinate of center point.

Referenced by DL_Dxf::writeArc().

#### 5.2.3.4 cy

`double DL_ArcData::cy`

Y Coordinate of center point.

Referenced by DL_Dxf::writeArc().

#### 5.2.3.5 cz

`double DL_ArcData::cz`

Z Coordinate of center point.

Referenced by DL_Dxf::writeArc().

#### 5.2.3.6 radius

`double DL_ArcData::radius`

Radius of arc.

Referenced by DL_Dxf::writeArc().

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

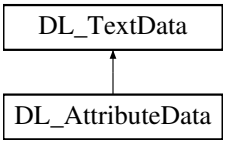
- src/dl_entities.h

## 5.3 DL_AttributeData Struct Reference

Block attribute data.

`#include <dl_entities.h>`

Inheritance diagram for DL_AttributeData:



## Public Member Functions

- **DL_AttributeData** (const DL_TextData &tData, const std::string &tag)
- DL_AttributeData (double ipx, double ipy, double ipz, double apx, double apy, double apz, double height, double xScaleFactor, int textGenerationFlags, int hJustification, int vJustification, const std::string &tag, const std::string &text, const std::string &style, double angle)

  *Constructor.*

## Public Attributes

- std::string tag

### 5.3.1 Detailed Description

Block attribute data.

### 5.3.2 Constructor & Destructor Documentation

#### 5.3.2.1 DL_AttributeData()

```
DL_AttributeData::DL_AttributeData (
            double ipx,
            double ipy,
            double ipz,
            double apx,
            double apy,
            double apz,
            double height,
            double xScaleFactor,
            int textGenerationFlags,
            int hJustification,
            int vJustification,
            const std::string & tag,
            const std::string & text,
            const std::string & style,
            double angle )  [inline]
```

Constructor.

Parameters: see member variables.

### 5.3.3 Member Data Documentation

#### 5.3.3.1 tag

```
std::string DL_AttributeData::tag
```

Tag.

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

- src/dl_entities.h

## 5.4 DL_Attributes Class Reference

Storing and passing around attributes.

```
#include <dl_attributes.h>
```

### Public Member Functions

- **DL_Attributes** ()
  *Default constructor.*
- DL_Attributes (const std::string &layer, int color, int width, const std::string &linetype, double linetypeScale)
  *Constructor for DXF attributes.*
- DL_Attributes (const std::string &layer, int color, int color24, int width, const std::string &linetype, int handle=-1)
  *Constructor for DXF attributes.*
- void setLayer (const std::string &layer)
  *Sets the layer.*
- std::string getLayer () const
- void setColor (int color)
  *Sets the color.*
- void setColor24 (int color)
  *Sets the 24bit color.*
- int getColor () const
- int getColor24 () const
- void **setWidth** (int width)
  *Sets the width.*
- int getWidth () const
- void setLinetype (const std::string &linetype)
  *Sets the line type.*
- void **setLinetypeScale** (double linetypeScale)
  *Sets the entity specific line type scale.*
- double **getLinetypeScale** () const
- std::string getLinetype () const
- void **setHandle** (int h)
- int **getHandle** () const
- void **setInPaperSpace** (bool on)
- bool **isInPaperSpace** () const

### 5.4.1 Detailed Description

Storing and passing around attributes.

Attributes are the layer name, color, width and line type.

**Author**

Andrew Mustun

### 5.4.2 Constructor & Destructor Documentation

#### 5.4.2.1 DL_Attributes() [1/2]

```
DL_Attributes::DL_Attributes (
            const std::string & layer,
            int color,
            int width,
            const std::string & linetype,
            double linetypeScale )  [inline]
```

Constructor for DXF attributes.

**Parameters**

| | |
|---|---|
| *layer* | Layer name for this entity or NULL for no layer (every entity should be on a named layer!). |
| *color* | Color number (0..256). 0 = BYBLOCK, 256 = BYLAYER. |
| *width* | Line thickness. Defaults to zero. -1 = BYLAYER, -2 = BYBLOCK, -3 = default width |
| *linetype* | Line type name or "BYLAYER" or "BYBLOCK". Defaults to "BYLAYER" |

#### 5.4.2.2 DL_Attributes() [2/2]

```
DL_Attributes::DL_Attributes (
            const std::string & layer,
            int color,
            int color24,
            int width,
            const std::string & linetype,
            int handle = -1 )  [inline]
```

Constructor for DXF attributes.

**Parameters**

| | |
|---|---|
| *layer* | Layer name for this entity or NULL for no layer (every entity should be on a named layer!). |
| *color* | Color number (0..256). 0 = BYBLOCK, 256 = BYLAYER. |

**Parameters**

| | |
|---|---|
| *color24* | 24 bit color (0x00RRGGBB, see DXF reference). |
| *width* | Line thickness. Defaults to zero. -1 = BYLAYER, -2 = BYBLOCK, -3 = default width |
| *linetype* | Line type name or "BYLAYER" or "BYBLOCK". Defaults to "BYLAYER" |

## 5.4.3 Member Function Documentation

### 5.4.3.1 getColor()

```
int DL_Attributes::getColor ( ) const  [inline]
```

**Returns**

Color.

**See also**

DL_Codes, dxfColors

Referenced by DL_Dxf::addLayer(), DL_Writer::entityAttributes(), and DL_Dxf::writeLayer().

### 5.4.3.2 getColor24()

```
int DL_Attributes::getColor24 ( ) const  [inline]
```

**Returns**

24 bit color or -1 if no 24bit color is defined.

**See also**

DL_Codes, dxfColors

Referenced by DL_Writer::entityAttributes(), and DL_Dxf::writeLayer().

#### 5.4.3.3 getLayer()

```
std::string DL_Attributes::getLayer ( ) const  [inline]
```

**Returns**

Layer name.

Referenced by DL_Writer::entityAttributes(), and DL_Dxf::writePolyline().

#### 5.4.3.4 getLinetype()

```
std::string DL_Attributes::getLinetype ( ) const  [inline]
```

**Returns**

Line type.

Referenced by DL_Dxf::addLayer(), DL_Writer::entityAttributes(), and DL_Dxf::writeLayer().

#### 5.4.3.5 getWidth()

```
int DL_Attributes::getWidth ( ) const  [inline]
```

**Returns**

Width.

Referenced by DL_Dxf::addLayer(), DL_Writer::entityAttributes(), and DL_Dxf::writeLayer().

#### 5.4.3.6 setColor()

```
void DL_Attributes::setColor (
            int color )  [inline]
```

Sets the color.

**See also**

DL_Codes, dxfColors

Referenced by DL_Dxf::addLayer().

#### 5.4.3.7 setColor24()

```
void DL_Attributes::setColor24 (
            int color )  [inline]
```

Sets the 24bit color.

**See also**

DL_Codes, dxfColors

#### 5.4.3.8 setLayer()

```
void DL_Attributes::setLayer (
            const std::string & layer )  [inline]
```

Sets the layer.

If the given pointer points to NULL, the new layer name will be an empty but valid string.

#### 5.4.3.9 setLinetype()

```
void DL_Attributes::setLinetype (
            const std::string & linetype )  [inline]
```

Sets the line type.

This can be any string and is not checked to be a valid line type.

Referenced by DL_Dxf::addLayer().

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

- src/dl_attributes.h

## 5.5 DL_BlockData Struct Reference

Block Data.

```
#include <dl_entities.h>
```

### Public Member Functions

- DL_BlockData (const std::string &bName, int bFlags, double bbpx, double bbpy, double bbpz)

  *Constructor.*

## Public Attributes

- std::string **name**

  *Block name.*
- int flags

  *Block flags.*
- double **bpx**

  *X Coordinate of base point.*
- double **bpy**

  *Y Coordinate of base point.*
- double **bpz**

  *Z Coordinate of base point.*

### 5.5.1 Detailed Description

Block Data.

### 5.5.2 Constructor & Destructor Documentation

#### 5.5.2.1 DL_BlockData()

```
DL_BlockData::DL_BlockData (
            const std::string & bName,
            int bFlags,
            double bbpx,
            double bbpy,
            double bbpz )  [inline]
```

Constructor.

Parameters: see member variables.

### 5.5.3 Member Data Documentation

#### 5.5.3.1 flags

```
int DL_BlockData::flags
```

Block flags.

(not used currently)

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

- src/dl_entities.h

## 5.6 DL_CircleData Struct Reference

Circle Data.

`#include <dl_entities.h>`

### Public Member Functions

- DL_CircleData (double acx, double acy, double acz, double aRadius)
  *Constructor.*

### Public Attributes

- double cx
- double cy
- double cz
- double radius

### 5.6.1 Detailed Description

Circle Data.

### 5.6.2 Constructor & Destructor Documentation

#### 5.6.2.1 DL_CircleData()

```
DL_CircleData::DL_CircleData (
            double acx,
            double acy,
            double acz,
            double aRadius )  [inline]
```

Constructor.

Parameters: see member variables.

### 5.6.3 Member Data Documentation

#### 5.6.3.1 cx

```
double DL_CircleData::cx
```

X Coordinate of center point.

Referenced by DL_Dxf::writeCircle().

#### 5.6.3.2 cy

```
double DL_CircleData::cy
```

Y Coordinate of center point.

Referenced by DL_Dxf::writeCircle().

#### 5.6.3.3 cz

```
double DL_CircleData::cz
```

Z Coordinate of center point.

Referenced by DL_Dxf::writeCircle().

#### 5.6.3.4 radius

```
double DL_CircleData::radius
```

Radius of arc.

Referenced by DL_Dxf::writeCircle().

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

- src/dl_entities.h

## 5.7 DL_Codes Class Reference

Codes for colors and DXF versions.

```
#include <dl_codes.h>
```

### Public Types

- enum color {
  **black** = 250 , **green** = 3 , **red** = 1 , **brown** = 15 ,
  **yellow** = 2 , **cyan** = 4 , **magenta** = 6 , **gray** = 8 ,
  **blue** = 5 , **l_blue** = 163 , **l_green** = 121 , **l_cyan** = 131 ,
  **l_red** = 23 , **l_magenta** = 221 , **l_gray** = 252 , **white** = 7 ,
  **bylayer** = 256 , **byblock** = 0 }

  *Standard DXF colors.*
- enum version {
  **AC1009_MIN** , **AC1009** , **AC1012** , **AC1014** ,
  **AC1015** }

  *Version numbers for the DXF Format.*

#### 5.7.1 Detailed Description

Codes for colors and DXF versions.

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

- src/dl_codes.h

## 5.8 DL_ControlPointData Struct Reference

Spline control point data.

`#include <dl_entities.h>`

### Public Member Functions

- DL_ControlPointData (double px, double py, double pz, double weight)

  *Constructor.*

### Public Attributes

- double x
- double y
- double z
- double w

#### 5.8.1 Detailed Description

Spline control point data.

#### 5.8.2 Constructor & Destructor Documentation

#### 5.8.2.1 DL_ControlPointData()

```
DL_ControlPointData::DL_ControlPointData (
            double px,
            double py,
            double pz,
            double weight )  [inline]
```

Constructor.

Parameters: see member variables.

### 5.8.3 Member Data Documentation

#### 5.8.3.1 w

```
double DL_ControlPointData::w
```

Weight of control point.

#### 5.8.3.2 x

```
double DL_ControlPointData::x
```

X coordinate of the control point.

Referenced by DL_Dxf::writeControlPoint().

#### 5.8.3.3 y

```
double DL_ControlPointData::y
```

Y coordinate of the control point.

Referenced by DL_Dxf::writeControlPoint().

#### 5.8.3.4 z

```
double DL_ControlPointData::z
```

Z coordinate of the control point.

Referenced by DL_Dxf::writeControlPoint().

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

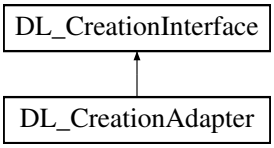
- src/dl_entities.h

## 5.9 DL_CreationAdapter Class Reference

An abstract adapter class for receiving DXF events when a DXF file is being read.

`#include <dl_creationadapter.h>`

Inheritance diagram for DL_CreationAdapter:




### Public Member Functions

- virtual void processCodeValuePair (unsigned int, const std::string &)
  *Called for every code / value tuple of the DXF file.*
- virtual void endSection ()
  *Called when a section (entity, table entry, etc.) is finished.*
- virtual void addLayer (const DL_LayerData &)
  *Called for every layer.*
- virtual void addLinetype (const DL_LinetypeData &)
  *Called for every linetype.*
- virtual void addLinetypeDash (double)
  *Called for every dash in linetype pattern.*
- virtual void addBlock (const DL_BlockData &)
  *Called for every block.*
- virtual void endBlock ()
  *Called to end the current block.*
- virtual void addTextStyle (const DL_StyleData &)
  *Called for every text style.*
- virtual void addPoint (const DL_PointData &)
  *Called for every point.*
- virtual void addLine (const DL_LineData &)
  *Called for every line.*
- virtual void addXLine (const DL_XLineData &)
  *Called for every xline.*
- virtual void addRay (const DL_RayData &)
  *Called for every ray.*
- virtual void addArc (const DL_ArcData &)
  *Called for every arc.*
- virtual void addCircle (const DL_CircleData &)
  *Called for every circle.*
- virtual void addEllipse (const DL_EllipseData &)
  *Called for every ellipse.*
- virtual void addPolyline (const DL_PolylineData &)
  *Called for every polyline start.*
- virtual void addVertex (const DL_VertexData &)
  *Called for every polyline vertex.*

- virtual void addSpline (const DL_SplineData &)

  *Called for every spline.*
- virtual void addControlPoint (const DL_ControlPointData &)

  *Called for every spline control point.*
- virtual void addFitPoint (const DL_FitPointData &)

  *Called for every spline fit point.*
- virtual void addKnot (const DL_KnotData &)

  *Called for every spline knot value.*
- virtual void addInsert (const DL_InsertData &)

  *Called for every insert.*
- virtual void addMText (const DL_MTextData &)

  *Called for every multi Text entity.*
- virtual void addMTextChunk (const std::string &)

  *Called for additional text chunks for MTEXT entities.*
- virtual void addText (const DL_TextData &)

  *Called for every text entity.*
- virtual void addArcAlignedText (const DL_ArcAlignedTextData &)

  *Called for every arc aligned text entity.*
- virtual void addAttribute (const DL_AttributeData &)

  *Called for every block Attribute entity.*
- virtual void addDimAlign (const DL_DimensionData &, const DL_DimAlignedData &)

  *Called for every aligned dimension entity.*
- virtual void addDimLinear (const DL_DimensionData &, const DL_DimLinearData &)

  *Called for every linear or rotated dimension entity.*
- virtual void addDimRadial (const DL_DimensionData &, const DL_DimRadialData &)

  *Called for every radial dimension entity.*
- virtual void addDimDiametric (const DL_DimensionData &, const DL_DimDiametricData &)

  *Called for every diametric dimension entity.*
- virtual void addDimAngular (const DL_DimensionData &, const DL_DimAngular2LData &)

  *Called for every angular dimension (2 lines version) entity.*
- virtual void addDimAngular3P (const DL_DimensionData &, const DL_DimAngular3PData &)

  *Called for every angular dimension (3 points version) entity.*
- virtual void addDimOrdinate (const DL_DimensionData &, const DL_DimOrdinateData &)

  *Called for every ordinate dimension entity.*
- virtual void addLeader (const DL_LeaderData &)

  *Called for every leader start.*
- virtual void addLeaderVertex (const DL_LeaderVertexData &)

  *Called for every leader vertex.*
- virtual void addHatch (const DL_HatchData &)

  *Called for every hatch entity.*
- virtual void addTrace (const DL_TraceData &)

  *Called for every trace start.*
- virtual void add3dFace (const DL_3dFaceData &)

  *Called for every 3dface start.*
- virtual void addSolid (const DL_SolidData &)

  *Called for every solid start.*
- virtual void addImage (const DL_ImageData &)

  *Called for every image entity.*
- virtual void linkImage (const DL_ImageDefData &)

  *Called for every image definition.*
- virtual void addHatchLoop (const DL_HatchLoopData &)

*Called for every hatch loop.*

- virtual void addHatchEdge (const DL_HatchEdgeData &)

  *Called for every hatch edge entity.*
- virtual void addXRecord (const std::string &)

  *Called for every XRecord with the given handle.*
- virtual void addXRecordString (int, const std::string &)

  *Called for XRecords of type string.*
- virtual void addXRecordReal (int, double)

  *Called for XRecords of type double.*
- virtual void addXRecordInt (int, int)

  *Called for XRecords of type int.*
- virtual void addXRecordBool (int, bool)

  *Called for XRecords of type bool.*
- virtual void addXDataApp (const std::string &)

  *Called for every beginning of an XData section of the given application.*
- virtual void addXDataString (int, const std::string &)

  *Called for XData tuples.*
- virtual void addXDataReal (int, double)

  *Called for XData tuples.*
- virtual void addXDataInt (int, int)

  *Called for XData tuples.*
- virtual void addDictionary (const DL_DictionaryData &)

  *Called for dictionary objects.*
- virtual void addDictionaryEntry (const DL_DictionaryEntryData &)

  *Called for dictionary entries.*
- virtual void endEntity ()

  *Called after an entity has been completed.*
- virtual void addComment (const std::string &)

  *Called for every comment in the DXF file (code 999).*
- virtual void setVariableVector (const std::string &, double, double, double, int)

  *Called for every vector variable in the DXF file (e.g.*
- virtual void setVariableString (const std::string &, const std::string &, int)

  *Called for every string variable in the DXF file (e.g.*
- virtual void setVariableInt (const std::string &, int, int)

  *Called for every int variable in the DXF file (e.g.*
- virtual void setVariableDouble (const std::string &, double, int)

  *Called for every double variable in the DXF file (e.g.*
- virtual void endSequence ()

  *Called when a SEQEND occurs (when a POLYLINE or ATTRIB is done)*

## Additional Inherited Members

### 5.9.1 Detailed Description

An abstract adapter class for receiving DXF events when a DXF file is being read.

The methods in this class are empty. This class exists as convenience for creating listener objects.

**Author**

Andrew Mustun

## 5.9.2 Member Function Documentation

### 5.9.2.1 add3dFace()

```
virtual void DL_CreationAdapter::add3dFace (
            const DL_3dFaceData & data )  [inline], [virtual]
```

Called for every 3dface start.

Implements DL_CreationInterface.

### 5.9.2.2 addArc()

```
virtual void DL_CreationAdapter::addArc (
            const DL_ArcData & data )  [inline], [virtual]
```

Called for every arc.

Implements DL_CreationInterface.

### 5.9.2.3 addArcAlignedText()

```
virtual void DL_CreationAdapter::addArcAlignedText (
            const DL_ArcAlignedTextData & data )  [inline], [virtual]
```

Called for every arc aligned text entity.

Implements DL_CreationInterface.

### 5.9.2.4 addAttribute()

```
virtual void DL_CreationAdapter::addAttribute (
            const DL_AttributeData & data )  [inline], [virtual]
```

Called for every block Attribute entity.

Implements DL_CreationInterface.

#### 5.9.2.5 addBlock()

```
virtual void DL_CreationAdapter::addBlock (
            const DL_BlockData & data )  [inline], [virtual]
```

Called for every block.

Note: all entities added after this command go into this block until endBlock() is called.

**See also**

endBlock()

Implements DL_CreationInterface.

#### 5.9.2.6 addCircle()

```
virtual void DL_CreationAdapter::addCircle (
            const DL_CircleData & data )  [inline], [virtual]
```

Called for every circle.

Implements DL_CreationInterface.

#### 5.9.2.7 addComment()

```
virtual void DL_CreationAdapter::addComment (
            const std::string & comment )  [inline], [virtual]
```

Called for every comment in the DXF file (code 999).

Implements DL_CreationInterface.

#### 5.9.2.8 addControlPoint()

```
virtual void DL_CreationAdapter::addControlPoint (
            const DL_ControlPointData & data )  [inline], [virtual]
```

Called for every spline control point.

Implements DL_CreationInterface.

#### 5.9.2.9 addDictionary()

```
virtual void DL_CreationAdapter::addDictionary (
            const DL_DictionaryData & data )  [inline], [virtual]
```

Called for dictionary objects.

Implements DL_CreationInterface.

#### 5.9.2.10 addDictionaryEntry()

```
virtual void DL_CreationAdapter::addDictionaryEntry (
            const DL_DictionaryEntryData & data )  [inline], [virtual]
```

Called for dictionary entries.

Implements DL_CreationInterface.

#### 5.9.2.11 addDimAlign()

```
virtual void DL_CreationAdapter::addDimAlign (
            const DL_DimensionData & data,
            const DL_DimAlignedData & edata )  [inline], [virtual]
```

Called for every aligned dimension entity.

Implements DL_CreationInterface.

#### 5.9.2.12 addDimAngular()

```
virtual void DL_CreationAdapter::addDimAngular (
            const DL_DimensionData & data,
            const DL_DimAngular2LData & edata )  [inline], [virtual]
```

Called for every angular dimension (2 lines version) entity.

Implements DL_CreationInterface.

#### 5.9.2.13 addDimAngular3P()

```
virtual void DL_CreationAdapter::addDimAngular3P (
            const DL_DimensionData & data,
            const DL_DimAngular3PData & edata )  [inline], [virtual]
```

Called for every angular dimension (3 points version) entity.

Implements DL_CreationInterface.

#### 5.9.2.14 addDimDiametric()

```
virtual void DL_CreationAdapter::addDimDiametric (
            const DL_DimensionData & data,
            const DL_DimDiametricData & edata )  [inline], [virtual]
```

Called for every diametric dimension entity.

Implements DL_CreationInterface.

#### 5.9.2.15 addDimLinear()

```
virtual void DL_CreationAdapter::addDimLinear (
            const DL_DimensionData & data,
            const DL_DimLinearData & edata )  [inline], [virtual]
```

Called for every linear or rotated dimension entity.

Implements DL_CreationInterface.

#### 5.9.2.16 addDimOrdinate()

```
virtual void DL_CreationAdapter::addDimOrdinate (
            const DL_DimensionData & data,
            const DL_DimOrdinateData & edata )  [inline], [virtual]
```

Called for every ordinate dimension entity.

Implements DL_CreationInterface.

#### 5.9.2.17 addDimRadial()

```
virtual void DL_CreationAdapter::addDimRadial (
            const DL_DimensionData & data,
            const DL_DimRadialData & edata )  [inline], [virtual]
```

Called for every radial dimension entity.

Implements DL_CreationInterface.

#### 5.9.2.18 addEllipse()

```
virtual void DL_CreationAdapter::addEllipse (
            const DL_EllipseData & data )  [inline], [virtual]
```

Called for every ellipse.

Implements DL_CreationInterface.

#### 5.9.2.19 addFitPoint()

```
virtual void DL_CreationAdapter::addFitPoint (
            const DL_FitPointData & data )  [inline], [virtual]
```

Called for every spline fit point.

Implements DL_CreationInterface.

#### 5.9.2.20 addHatch()

```
virtual void DL_CreationAdapter::addHatch (
            const DL_HatchData & data )  [inline], [virtual]
```

Called for every hatch entity.

Implements DL_CreationInterface.

#### 5.9.2.21 addHatchEdge()

```
virtual void DL_CreationAdapter::addHatchEdge (
            const DL_HatchEdgeData & data )  [inline], [virtual]
```

Called for every hatch edge entity.

Implements DL_CreationInterface.

#### 5.9.2.22 addHatchLoop()

```
virtual void DL_CreationAdapter::addHatchLoop (
            const DL_HatchLoopData & data )  [inline], [virtual]
```

Called for every hatch loop.

Implements DL_CreationInterface.

#### 5.9.2.23 addImage()

```
virtual void DL_CreationAdapter::addImage (
            const DL_ImageData & data )  [inline], [virtual]
```

Called for every image entity.

Implements DL_CreationInterface.

#### 5.9.2.24 addInsert()

```
virtual void DL_CreationAdapter::addInsert (
            const DL_InsertData & data )  [inline], [virtual]
```

Called for every insert.

Implements DL_CreationInterface.

#### 5.9.2.25 addKnot()

```
virtual void DL_CreationAdapter::addKnot (
            const DL_KnotData & data )  [inline], [virtual]
```

Called for every spline knot value.

Implements DL_CreationInterface.

#### 5.9.2.26 addLayer()

```
virtual void DL_CreationAdapter::addLayer (
            const DL_LayerData & data )  [inline], [virtual]
```

Called for every layer.

Implements DL_CreationInterface.

#### 5.9.2.27 addLeader()

```
virtual void DL_CreationAdapter::addLeader (
            const DL_LeaderData & data )  [inline], [virtual]
```

Called for every leader start.

Implements DL_CreationInterface.

#### 5.9.2.28 addLeaderVertex()

```
virtual void DL_CreationAdapter::addLeaderVertex (
            const DL_LeaderVertexData & data )  [inline], [virtual]
```

Called for every leader vertex.

Implements DL_CreationInterface.

#### 5.9.2.29 addLine()

```
virtual void DL_CreationAdapter::addLine (
            const DL_LineData & data )  [inline], [virtual]
```

Called for every line.

Implements DL_CreationInterface.

#### 5.9.2.30 addLinetype()

```
virtual void DL_CreationAdapter::addLinetype (
            const DL_LinetypeData & data )  [inline], [virtual]
```

Called for every linetype.

Implements DL_CreationInterface.

#### 5.9.2.31 addLinetypeDash()

```
virtual void DL_CreationAdapter::addLinetypeDash (
            double length )  [inline], [virtual]
```

Called for every dash in linetype pattern.

Implements DL_CreationInterface.

#### 5.9.2.32 addMText()

```
virtual void DL_CreationAdapter::addMText (
            const DL_MTextData & data )  [inline], [virtual]
```

Called for every multi Text entity.

Implements DL_CreationInterface.

#### 5.9.2.33 addMTextChunk()

```
virtual void DL_CreationAdapter::addMTextChunk (
            const std::string & text )  [inline], [virtual]
```

Called for additional text chunks for MTEXT entities.

The chunks come at 250 character in size each. Note that those chunks come **before** the actual MTEXT entity.

Implements DL_CreationInterface.

#### 5.9.2.34 addPoint()

```
virtual void DL_CreationAdapter::addPoint (
            const DL_PointData & data )  [inline], [virtual]
```

Called for every point.

Implements DL_CreationInterface.

#### 5.9.2.35 addPolyline()

```
virtual void DL_CreationAdapter::addPolyline (
            const DL_PolylineData & data )  [inline], [virtual]
```

Called for every polyline start.

Implements DL_CreationInterface.

#### 5.9.2.36 addRay()

```
virtual void DL_CreationAdapter::addRay (
            const DL_RayData & data )  [inline], [virtual]
```

Called for every ray.

Implements DL_CreationInterface.

#### 5.9.2.37 addSolid()

```
virtual void DL_CreationAdapter::addSolid (
            const DL_SolidData & data )  [inline], [virtual]
```

Called for every solid start.

Implements DL_CreationInterface.

#### 5.9.2.38 addSpline()

```
virtual void DL_CreationAdapter::addSpline (
            const DL_SplineData & data )  [inline], [virtual]
```

Called for every spline.

Implements DL_CreationInterface.

#### 5.9.2.39 addText()

```
virtual void DL_CreationAdapter::addText (
            const DL_TextData & data )  [inline], [virtual]
```

Called for every text entity.

Implements DL_CreationInterface.

#### 5.9.2.40 addTextStyle()

```
virtual void DL_CreationAdapter::addTextStyle (
            const DL_StyleData & data )  [inline], [virtual]
```

Called for every text style.

Implements DL_CreationInterface.

#### 5.9.2.41 addTrace()

```
virtual void DL_CreationAdapter::addTrace (
            const DL_TraceData & data )  [inline], [virtual]
```

Called for every trace start.

Implements DL_CreationInterface.

#### 5.9.2.42 addVertex()

```
virtual void DL_CreationAdapter::addVertex (
            const DL_VertexData & data )  [inline], [virtual]
```

Called for every polyline vertex.

Implements DL_CreationInterface.

#### 5.9.2.43 addXDataApp()

```
virtual void DL_CreationAdapter::addXDataApp (
            const std::string & appId )  [inline], [virtual]
```

Called for every beginning of an XData section of the given application.

Implements DL_CreationInterface.

#### 5.9.2.44 addXDataInt()

```
virtual void DL_CreationAdapter::addXDataInt (
            int code,
            int value )  [inline], [virtual]
```

Called for XData tuples.

Implements DL_CreationInterface.

#### 5.9.2.45 addXDataReal()

```
virtual void DL_CreationAdapter::addXDataReal (
            int code,
            double value )  [inline], [virtual]
```

Called for XData tuples.

Implements DL_CreationInterface.

#### 5.9.2.46 addXDataString()

```
virtual void DL_CreationAdapter::addXDataString (
            int code,
            const std::string & value )  [inline], [virtual]
```

Called for XData tuples.

Implements DL_CreationInterface.

#### 5.9.2.47 addXLine()

```
virtual void DL_CreationAdapter::addXLine (
            const DL_XLineData & data )  [inline], [virtual]
```

Called for every xline.

Implements DL_CreationInterface.

#### 5.9.2.48 addXRecord()

```
virtual void DL_CreationAdapter::addXRecord (
            const std::string & handle )  [inline], [virtual]
```

Called for every XRecord with the given handle.

Implements DL_CreationInterface.

#### 5.9.2.49 addXRecordBool()

```
virtual void DL_CreationAdapter::addXRecordBool (
            int code,
            bool value )  [inline], [virtual]
```

Called for XRecords of type bool.

Implements DL_CreationInterface.

#### 5.9.2.50 addXRecordInt()

```
virtual void DL_CreationAdapter::addXRecordInt (
            int code,
            int value )  [inline], [virtual]
```

Called for XRecords of type int.

Implements DL_CreationInterface.

#### 5.9.2.51 addXRecordReal()

```
virtual void DL_CreationAdapter::addXRecordReal (
            int code,
            double value )  [inline], [virtual]
```

Called for XRecords of type double.

Implements DL_CreationInterface.

#### 5.9.2.52 addXRecordString()

```
virtual void DL_CreationAdapter::addXRecordString (
            int code,
            const std::string & value )  [inline], [virtual]
```

Called for XRecords of type string.

Implements DL_CreationInterface.

#### 5.9.2.53 endBlock()

```
virtual void DL_CreationAdapter::endBlock ( )  [inline], [virtual]
```

Called to end the current block.

Implements DL_CreationInterface.

#### 5.9.2.54 endEntity()

```
virtual void DL_CreationAdapter::endEntity ( )  [inline], [virtual]
```

Called after an entity has been completed.

Implements DL_CreationInterface.

#### 5.9.2.55 endSection()

```
virtual void DL_CreationAdapter::endSection ( )  [inline], [virtual]
```

Called when a section (entity, table entry, etc.) is finished.

Implements DL_CreationInterface.

#### 5.9.2.56 endSequence()

```
virtual void DL_CreationAdapter::endSequence ( )  [inline], [virtual]
```

Called when a SEQEND occurs (when a POLYLINE or ATTRIB is done)

Implements DL_CreationInterface.

#### 5.9.2.57 linkImage()

```
virtual void DL_CreationAdapter::linkImage (
            const DL_ImageDefData & data )  [inline], [virtual]
```

Called for every image definition.

Implements DL_CreationInterface.

#### 5.9.2.58 processCodeValuePair()

```
virtual void DL_CreationAdapter::processCodeValuePair (
            unsigned int groupCode,
            const std::string & groupValue )  [inline], [virtual]
```

Called for every code / value tuple of the DXF file.

The complete DXF file contents can be handled by the implemetation of this function.

Implements DL_CreationInterface.

#### 5.9.2.59 setVariableDouble()

```
virtual void DL_CreationAdapter::setVariableDouble (
            const std::string & key,
            double value,
            int code )  [inline], [virtual]
```

Called for every double variable in the DXF file (e.g.

"$DIMEXO").

Implements DL_CreationInterface.

#### 5.9.2.60 setVariableInt()

```
virtual void DL_CreationAdapter::setVariableInt (
            const std::string & key,
            int value,
            int code )  [inline], [virtual]
```

Called for every int variable in the DXF file (e.g.

"$ACADMAINTVER").

Implements DL_CreationInterface.

#### 5.9.2.61 setVariableString()

```
virtual void DL_CreationAdapter::setVariableString (
            const std::string & key,
            const std::string & value,
            int code )  [inline], [virtual]
```

Called for every string variable in the DXF file (e.g.

"$ACADVER").

Implements DL_CreationInterface.

#### 5.9.2.62 setVariableVector()

```
virtual void DL_CreationAdapter::setVariableVector (
            const std::string & key,
            double v1,
            double v2,
            double v3,
            int code )  [inline], [virtual]
```

Called for every vector variable in the DXF file (e.g.

"$EXTMIN").

Implements DL_CreationInterface.

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

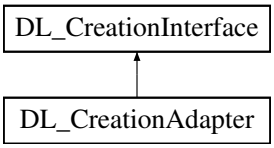
- src/dl_creationadapter.h

## 5.10 DL_CreationInterface Class Reference

Abstract class (interface) for the creation of new entities.

`#include <dl_creationinterface.h>`

Inheritance diagram for DL_CreationInterface:




### Public Member Functions

- virtual void processCodeValuePair (unsigned int groupCode, const std::string &groupValue)=0

  *Called for every code / value tuple of the DXF file.*
- virtual void endSection ()=0

  *Called when a section (entity, table entry, etc.) is finished.*
- virtual void addLayer (const DL_LayerData &data)=0

  *Called for every layer.*
- virtual void addLinetype (const DL_LinetypeData &data)=0

  *Called for every linetype.*
- virtual void addLinetypeDash (double length)=0

  *Called for every dash in linetype pattern.*
- virtual void addBlock (const DL_BlockData &data)=0

  *Called for every block.*
- virtual void endBlock ()=0

  *Called to end the current block.*
- virtual void addTextStyle (const DL_StyleData &data)=0

*Called for every text style.*

- virtual void addPoint (const DL_PointData &data)=0

  *Called for every point.*
- virtual void addLine (const DL_LineData &data)=0

  *Called for every line.*
- virtual void addXLine (const DL_XLineData &data)=0

  *Called for every xline.*
- virtual void addRay (const DL_RayData &data)=0

  *Called for every ray.*
- virtual void addArc (const DL_ArcData &data)=0

  *Called for every arc.*
- virtual void addCircle (const DL_CircleData &data)=0

  *Called for every circle.*
- virtual void addEllipse (const DL_EllipseData &data)=0

  *Called for every ellipse.*
- virtual void addPolyline (const DL_PolylineData &data)=0

  *Called for every polyline start.*
- virtual void addVertex (const DL_VertexData &data)=0

  *Called for every polyline vertex.*
- virtual void addSpline (const DL_SplineData &data)=0

  *Called for every spline.*
- virtual void addControlPoint (const DL_ControlPointData &data)=0

  *Called for every spline control point.*
- virtual void addFitPoint (const DL_FitPointData &data)=0

  *Called for every spline fit point.*
- virtual void addKnot (const DL_KnotData &data)=0

  *Called for every spline knot value.*
- virtual void addInsert (const DL_InsertData &data)=0

  *Called for every insert.*
- virtual void addTrace (const DL_TraceData &data)=0

  *Called for every trace start.*
- virtual void add3dFace (const DL_3dFaceData &data)=0

  *Called for every 3dface start.*
- virtual void addSolid (const DL_SolidData &data)=0

  *Called for every solid start.*
- virtual void addMText (const DL_MTextData &data)=0

  *Called for every multi Text entity.*
- virtual void addMTextChunk (const std::string &text)=0

  *Called for additional text chunks for MTEXT entities.*
- virtual void addText (const DL_TextData &data)=0

  *Called for every text entity.*
- virtual void addArcAlignedText (const DL_ArcAlignedTextData &data)=0

  *Called for every arc aligned text entity.*
- virtual void addAttribute (const DL_AttributeData &data)=0

  *Called for every block Attribute entity.*
- virtual void addDimAlign (const DL_DimensionData &data, const DL_DimAlignedData &edata)=0

  *Called for every aligned dimension entity.*
- virtual void addDimLinear (const DL_DimensionData &data, const DL_DimLinearData &edata)=0

  *Called for every linear or rotated dimension entity.*
- virtual void addDimRadial (const DL_DimensionData &data, const DL_DimRadialData &edata)=0

  *Called for every radial dimension entity.*

- virtual void addDimDiametric (const DL_DimensionData &data, const DL_DimDiametricData &edata)=0

  *Called for every diametric dimension entity.*
- virtual void addDimAngular (const DL_DimensionData &data, const DL_DimAngular2LData &edata)=0

  *Called for every angular dimension (2 lines version) entity.*
- virtual void addDimAngular3P (const DL_DimensionData &data, const DL_DimAngular3PData &edata)=0

  *Called for every angular dimension (3 points version) entity.*
- virtual void addDimOrdinate (const DL_DimensionData &data, const DL_DimOrdinateData &edata)=0

  *Called for every ordinate dimension entity.*
- virtual void addLeader (const DL_LeaderData &data)=0

  *Called for every leader start.*
- virtual void addLeaderVertex (const DL_LeaderVertexData &data)=0

  *Called for every leader vertex.*
- virtual void addHatch (const DL_HatchData &data)=0

  *Called for every hatch entity.*
- virtual void addImage (const DL_ImageData &data)=0

  *Called for every image entity.*
- virtual void linkImage (const DL_ImageDefData &data)=0

  *Called for every image definition.*
- virtual void addHatchLoop (const DL_HatchLoopData &data)=0

  *Called for every hatch loop.*
- virtual void addHatchEdge (const DL_HatchEdgeData &data)=0

  *Called for every hatch edge entity.*
- virtual void addXRecord (const std::string &handle)=0

  *Called for every XRecord with the given handle.*
- virtual void addXRecordString (int code, const std::string &value)=0

  *Called for XRecords of type string.*
- virtual void addXRecordReal (int code, double value)=0

  *Called for XRecords of type double.*
- virtual void addXRecordInt (int code, int value)=0

  *Called for XRecords of type int.*
- virtual void addXRecordBool (int code, bool value)=0

  *Called for XRecords of type bool.*
- virtual void addXDataApp (const std::string &appId)=0

  *Called for every beginning of an XData section of the given application.*
- virtual void addXDataString (int code, const std::string &value)=0

  *Called for XData tuples.*
- virtual void addXDataReal (int code, double value)=0

  *Called for XData tuples.*
- virtual void addXDataInt (int code, int value)=0

  *Called for XData tuples.*
- virtual void addDictionary (const DL_DictionaryData &data)=0

  *Called for dictionary objects.*
- virtual void addDictionaryEntry (const DL_DictionaryEntryData &data)=0

  *Called for dictionary entries.*
- virtual void endEntity ()=0

  *Called after an entity has been completed.*
- virtual void addComment (const std::string &comment)=0

  *Called for every comment in the DXF file (code 999).*
- virtual void setVariableVector (const std::string &key, double v1, double v2, double v3, int code)=0

  *Called for every vector variable in the DXF file (e.g.*
- virtual void setVariableString (const std::string &key, const std::string &value, int code)=0

*Called for every string variable in the DXF file (e.g.*

- virtual void setVariableInt (const std::string &key, int value, int code)=0

  *Called for every int variable in the DXF file (e.g.*
- virtual void setVariableDouble (const std::string &key, double value, int code)=0

  *Called for every double variable in the DXF file (e.g.*
- virtual void endSequence ()=0

  *Called when a SEQEND occurs (when a POLYLINE or ATTRIB is done)*
- void **setAttributes** (const DL_Attributes &attrib)

  *Sets the current attributes for entities.*
- DL_Attributes getAttributes ()
- void **setExtrusion** (double dx, double dy, double dz, double elevation)

  *Sets the current attributes for entities.*
- DL_Extrusion * getExtrusion ()

## Protected Attributes

- DL_Attributes **attributes**
- DL_Extrusion * **extrusion**

### 5.10.1 Detailed Description

Abstract class (interface) for the creation of new entities.

Inherit your class which takes care of the entities in the processed DXF file from this interface.

Double arrays passed to your implementation contain 3 double values for x, y, z coordinates unless stated differently.

**Author**

Andrew Mustun

### 5.10.2 Member Function Documentation

#### 5.10.2.1 add3dFace()

```
virtual void DL_CreationInterface::add3dFace (
            const DL_3dFaceData & data )  [pure virtual]
```

Called for every 3dface start.

Implemented in DL_CreationAdapter.

Referenced by DL_Dxf::add3dFace().

#### 5.10.2.2 addArc()

```
virtual void DL_CreationInterface::addArc (
            const DL_ArcData & data )  [pure virtual]
```

Called for every arc.

Implemented in DL_CreationAdapter.

Referenced by DL_Dxf::addArc().

#### 5.10.2.3 addArcAlignedText()

```
virtual void DL_CreationInterface::addArcAlignedText (
            const DL_ArcAlignedTextData & data )  [pure virtual]
```

Called for every arc aligned text entity.

Implemented in DL_CreationAdapter.

Referenced by DL_Dxf::addArcAlignedText().

#### 5.10.2.4 addAttribute()

```
virtual void DL_CreationInterface::addAttribute (
            const DL_AttributeData & data )  [pure virtual]
```

Called for every block Attribute entity.

Implemented in DL_CreationAdapter.

Referenced by DL_Dxf::addAttribute().

#### 5.10.2.5 addBlock()

```
virtual void DL_CreationInterface::addBlock (
            const DL_BlockData & data )  [pure virtual]
```

Called for every block.

Note: all entities added after this command go into this block until endBlock() is called.

**See also**

endBlock()

Implemented in DL_CreationAdapter.

Referenced by DL_Dxf::addBlock().

#### 5.10.2.6 addCircle()

```
virtual void DL_CreationInterface::addCircle (
            const DL_CircleData & data )  [pure virtual]
```

Called for every circle.

Implemented in DL_CreationAdapter.

Referenced by DL_Dxf::addCircle().

#### 5.10.2.7 addComment()

```
virtual void DL_CreationInterface::addComment (
            const std::string & comment )  [pure virtual]
```

Called for every comment in the DXF file (code 999).

Implemented in DL_CreationAdapter.

Referenced by DL_Dxf::addComment().

#### 5.10.2.8 addControlPoint()

```
virtual void DL_CreationInterface::addControlPoint (
            const DL_ControlPointData & data )  [pure virtual]
```

Called for every spline control point.

Implemented in DL_CreationAdapter.

Referenced by DL_Dxf::addSpline().

#### 5.10.2.9 addDictionary()

```
virtual void DL_CreationInterface::addDictionary (
            const DL_DictionaryData & data )  [pure virtual]
```

Called for dictionary objects.

Implemented in DL_CreationAdapter.

Referenced by DL_Dxf::handleDictionaryData().

#### 5.10.2.10 addDictionaryEntry()

```
virtual void DL_CreationInterface::addDictionaryEntry (
            const DL_DictionaryEntryData & data )  [pure virtual]
```

Called for dictionary entries.

Implemented in DL_CreationAdapter.

Referenced by DL_Dxf::handleDictionaryData().

#### 5.10.2.11 addDimAlign()

```
virtual void DL_CreationInterface::addDimAlign (
            const DL_DimensionData & data,
            const DL_DimAlignedData & edata )  [pure virtual]
```

Called for every aligned dimension entity.

Implemented in DL_CreationAdapter.

Referenced by DL_Dxf::addDimAligned().

#### 5.10.2.12 addDimAngular()

```
virtual void DL_CreationInterface::addDimAngular (
            const DL_DimensionData & data,
            const DL_DimAngular2LData & edata )  [pure virtual]
```

Called for every angular dimension (2 lines version) entity.

Implemented in DL_CreationAdapter.

Referenced by DL_Dxf::addDimAngular().

#### 5.10.2.13 addDimAngular3P()

```
virtual void DL_CreationInterface::addDimAngular3P (
            const DL_DimensionData & data,
            const DL_DimAngular3PData & edata )  [pure virtual]
```

Called for every angular dimension (3 points version) entity.

Implemented in DL_CreationAdapter.

Referenced by DL_Dxf::addDimAngular3P().

#### 5.10.2.14 addDimDiametric()

```
virtual void DL_CreationInterface::addDimDiametric (
            const DL_DimensionData & data,
            const DL_DimDiametricData & edata )  [pure virtual]
```

Called for every diametric dimension entity.

Implemented in DL_CreationAdapter.

Referenced by DL_Dxf::addDimDiametric().

#### 5.10.2.15 addDimLinear()

```
virtual void DL_CreationInterface::addDimLinear (
            const DL_DimensionData & data,
            const DL_DimLinearData & edata )  [pure virtual]
```

Called for every linear or rotated dimension entity.

Implemented in DL_CreationAdapter.

Referenced by DL_Dxf::addDimLinear().

#### 5.10.2.16 addDimOrdinate()

```
virtual void DL_CreationInterface::addDimOrdinate (
            const DL_DimensionData & data,
            const DL_DimOrdinateData & edata )  [pure virtual]
```

Called for every ordinate dimension entity.

Implemented in DL_CreationAdapter.

Referenced by DL_Dxf::addDimOrdinate().

#### 5.10.2.17 addDimRadial()

```
virtual void DL_CreationInterface::addDimRadial (
            const DL_DimensionData & data,
            const DL_DimRadialData & edata )  [pure virtual]
```

Called for every radial dimension entity.

Implemented in DL_CreationAdapter.

Referenced by DL_Dxf::addDimRadial().

#### 5.10.2.18 addEllipse()

```
virtual void DL_CreationInterface::addEllipse (
            const DL_EllipseData & data )  [pure virtual]
```

Called for every ellipse.

Implemented in DL_CreationAdapter.

Referenced by DL_Dxf::addEllipse().

#### 5.10.2.19 addFitPoint()

```
virtual void DL_CreationInterface::addFitPoint (
            const DL_FitPointData & data )  [pure virtual]
```

Called for every spline fit point.

Implemented in DL_CreationAdapter.

Referenced by DL_Dxf::addSpline().

#### 5.10.2.20 addHatch()

```
virtual void DL_CreationInterface::addHatch (
            const DL_HatchData & data )  [pure virtual]
```

Called for every hatch entity.

Implemented in DL_CreationAdapter.

Referenced by DL_Dxf::addHatch().

#### 5.10.2.21 addHatchEdge()

```
virtual void DL_CreationInterface::addHatchEdge (
            const DL_HatchEdgeData & data )  [pure virtual]
```

Called for every hatch edge entity.

Implemented in DL_CreationAdapter.

Referenced by DL_Dxf::addHatch().

#### 5.10.2.22 addHatchLoop()

```
virtual void DL_CreationInterface::addHatchLoop (
            const DL_HatchLoopData & data )  [pure virtual]
```

Called for every hatch loop.

Implemented in DL_CreationAdapter.

Referenced by DL_Dxf::addHatch().

#### 5.10.2.23 addImage()

```
virtual void DL_CreationInterface::addImage (
            const DL_ImageData & data )  [pure virtual]
```

Called for every image entity.

Implemented in DL_CreationAdapter.

Referenced by DL_Dxf::addImage().

#### 5.10.2.24 addInsert()

```
virtual void DL_CreationInterface::addInsert (
            const DL_InsertData & data )  [pure virtual]
```

Called for every insert.

Implemented in DL_CreationAdapter.

Referenced by DL_Dxf::addInsert().

#### 5.10.2.25 addKnot()

```
virtual void DL_CreationInterface::addKnot (
            const DL_KnotData & data )  [pure virtual]
```

Called for every spline knot value.

Implemented in DL_CreationAdapter.

Referenced by DL_Dxf::addSpline().

#### 5.10.2.26 addLayer()

```
virtual void DL_CreationInterface::addLayer (
            const DL_LayerData & data )  [pure virtual]
```

Called for every layer.

Implemented in DL_CreationAdapter.

Referenced by DL_Dxf::addLayer().

#### 5.10.2.27 addLeader()

```
virtual void DL_CreationInterface::addLeader (
            const DL_LeaderData & data )  [pure virtual]
```

Called for every leader start.

Implemented in DL_CreationAdapter.

Referenced by DL_Dxf::addLeader().

#### 5.10.2.28 addLeaderVertex()

```
virtual void DL_CreationInterface::addLeaderVertex (
            const DL_LeaderVertexData & data )  [pure virtual]
```

Called for every leader vertex.

Implemented in DL_CreationAdapter.

Referenced by DL_Dxf::addLeader().

#### 5.10.2.29 addLine()

```
virtual void DL_CreationInterface::addLine (
            const DL_LineData & data )  [pure virtual]
```

Called for every line.

Implemented in DL_CreationAdapter.

Referenced by DL_Dxf::addLine().

#### 5.10.2.30 addLinetype()

```
virtual void DL_CreationInterface::addLinetype (
            const DL_LinetypeData & data )  [pure virtual]
```

Called for every linetype.

Implemented in DL_CreationAdapter.

Referenced by DL_Dxf::addLinetype().

#### 5.10.2.31 addLinetypeDash()

```
virtual void DL_CreationInterface::addLinetypeDash (
            double length )  [pure virtual]
```

Called for every dash in linetype pattern.

Implemented in DL_CreationAdapter.

Referenced by DL_Dxf::handleLinetypeData().

#### 5.10.2.32 addMText()

```
virtual void DL_CreationInterface::addMText (
            const DL_MTextData & data )  [pure virtual]
```

Called for every multi Text entity.

Implemented in DL_CreationAdapter.

Referenced by DL_Dxf::addMText().

#### 5.10.2.33 addMTextChunk()

```
virtual void DL_CreationInterface::addMTextChunk (
            const std::string & text )  [pure virtual]
```

Called for additional text chunks for MTEXT entities.

The chunks come at 250 character in size each. Note that those chunks come **before** the actual MTEXT entity.

Implemented in DL_CreationAdapter.

Referenced by DL_Dxf::handleMTextData().

#### 5.10.2.34 addPoint()

```
virtual void DL_CreationInterface::addPoint (
            const DL_PointData & data )  [pure virtual]
```

Called for every point.

Implemented in DL_CreationAdapter.

Referenced by DL_Dxf::addPoint().

#### 5.10.2.35 addPolyline()

```
virtual void DL_CreationInterface::addPolyline (
            const DL_PolylineData & data )  [pure virtual]
```

Called for every polyline start.

Implemented in DL_CreationAdapter.

Referenced by DL_Dxf::addPolyline().

#### 5.10.2.36 addRay()

```
virtual void DL_CreationInterface::addRay (
            const DL_RayData & data )  [pure virtual]
```

Called for every ray.

Implemented in DL_CreationAdapter.

Referenced by DL_Dxf::addRay().

#### 5.10.2.37 addSolid()

```
virtual void DL_CreationInterface::addSolid (
            const DL_SolidData & data )  [pure virtual]
```

Called for every solid start.

Implemented in DL_CreationAdapter.

Referenced by DL_Dxf::addSolid().

#### 5.10.2.38 addSpline()

```
virtual void DL_CreationInterface::addSpline (
            const DL_SplineData & data )  [pure virtual]
```

Called for every spline.

Implemented in DL_CreationAdapter.

Referenced by DL_Dxf::addSpline().

#### 5.10.2.39 addText()

```
virtual void DL_CreationInterface::addText (
            const DL_TextData & data )  [pure virtual]
```

Called for every text entity.

Implemented in DL_CreationAdapter.

Referenced by DL_Dxf::addText().

#### 5.10.2.40 addTextStyle()

```
virtual void DL_CreationInterface::addTextStyle (
            const DL_StyleData & data )  [pure virtual]
```

Called for every text style.

Implemented in DL_CreationAdapter.

#### 5.10.2.41 addTrace()

```
virtual void DL_CreationInterface::addTrace (
            const DL_TraceData & data )  [pure virtual]
```

Called for every trace start.

Implemented in DL_CreationAdapter.

Referenced by DL_Dxf::addTrace().

#### 5.10.2.42 addVertex()

```
virtual void DL_CreationInterface::addVertex (
            const DL_VertexData & data )  [pure virtual]
```

Called for every polyline vertex.

Implemented in DL_CreationAdapter.

Referenced by DL_Dxf::addPolyline(), and DL_Dxf::addVertex().

#### 5.10.2.43 addXDataApp()

```
virtual void DL_CreationInterface::addXDataApp (
            const std::string & appId )  [pure virtual]
```

Called for every beginning of an XData section of the given application.

Implemented in DL_CreationAdapter.

Referenced by DL_Dxf::handleXData().

#### 5.10.2.44 addXDataInt()

```
virtual void DL_CreationInterface::addXDataInt (
            int code,
            int value )  [pure virtual]
```

Called for XData tuples.

Implemented in DL_CreationAdapter.

Referenced by DL_Dxf::handleXData().

#### 5.10.2.45 addXDataReal()

```
virtual void DL_CreationInterface::addXDataReal (
            int code,
            double value )  [pure virtual]
```

Called for XData tuples.

Implemented in DL_CreationAdapter.

Referenced by DL_Dxf::handleXData().

#### 5.10.2.46 addXDataString()

```
virtual void DL_CreationInterface::addXDataString (
            int code,
            const std::string & value )  [pure virtual]
```

Called for XData tuples.

Implemented in DL_CreationAdapter.

Referenced by DL_Dxf::handleXData().

#### 5.10.2.47 addXLine()

```
virtual void DL_CreationInterface::addXLine (
            const DL_XLineData & data )  [pure virtual]
```

Called for every xline.

Implemented in DL_CreationAdapter.

Referenced by DL_Dxf::addXLine().

#### 5.10.2.48 addXRecord()

```
virtual void DL_CreationInterface::addXRecord (
            const std::string & handle )  [pure virtual]
```

Called for every XRecord with the given handle.

Implemented in DL_CreationAdapter.

Referenced by DL_Dxf::handleXRecordData().

#### 5.10.2.49 addXRecordBool()

```
virtual void DL_CreationInterface::addXRecordBool (
            int code,
            bool value )  [pure virtual]
```

Called for XRecords of type bool.

Implemented in DL_CreationAdapter.

Referenced by DL_Dxf::handleXRecordData().

#### 5.10.2.50 addXRecordInt()

```
virtual void DL_CreationInterface::addXRecordInt (
            int code,
            int value )  [pure virtual]
```

Called for XRecords of type int.

Implemented in DL_CreationAdapter.

Referenced by DL_Dxf::handleXRecordData().

#### 5.10.2.51 addXRecordReal()

```
virtual void DL_CreationInterface::addXRecordReal (
            int code,
            double value )  [pure virtual]
```

Called for XRecords of type double.

Implemented in DL_CreationAdapter.

Referenced by DL_Dxf::handleXRecordData().

#### 5.10.2.52 addXRecordString()

```
virtual void DL_CreationInterface::addXRecordString (
            int code,
            const std::string & value )  [pure virtual]
```

Called for XRecords of type string.

Implemented in DL_CreationAdapter.

Referenced by DL_Dxf::handleXRecordData().

#### 5.10.2.53 endBlock()

```
virtual void DL_CreationInterface::endBlock ( )  [pure virtual]
```

Called to end the current block.

Implemented in DL_CreationAdapter.

Referenced by DL_Dxf::endBlock().

#### 5.10.2.54 endEntity()

```
virtual void DL_CreationInterface::endEntity ( )  [pure virtual]
```

Called after an entity has been completed.

Implemented in DL_CreationAdapter.

Referenced by DL_Dxf::addHatch(), DL_Dxf::addImage(), DL_Dxf::addImageDef(), DL_Dxf::addLeader(), DL_Dxf::addPolyline(), DL_Dxf::addSpline(), and DL_Dxf::endEntity().

#### 5.10.2.55 endSection()

```
virtual void DL_CreationInterface::endSection ( )  [pure virtual]
```

Called when a section (entity, table entry, etc.) is finished.

Implemented in DL_CreationAdapter.

Referenced by DL_Dxf::processDXFGroup().

#### 5.10.2.56 endSequence()

```
virtual void DL_CreationInterface::endSequence ( )  [pure virtual]
```

Called when a SEQEND occurs (when a POLYLINE or ATTRIB is done)

Implemented in DL_CreationAdapter.

Referenced by DL_Dxf::endSequence().

#### 5.10.2.57 getAttributes()

```
DL_Attributes DL_CreationInterface::getAttributes ( )  [inline]
```

**Returns**

the current attributes used for new entities.

Referenced by DL_Dxf::addLayer().

#### 5.10.2.58 getExtrusion()

```
DL_Extrusion * DL_CreationInterface::getExtrusion ( )  [inline]
```

**Returns**

the current attributes used for new entities.

#### 5.10.2.59 linkImage()

```
virtual void DL_CreationInterface::linkImage (
            const DL_ImageDefData & data )  [pure virtual]
```

Called for every image definition.

Implemented in DL_CreationAdapter.

Referenced by DL_Dxf::addImageDef().

#### 5.10.2.60 processCodeValuePair()

```
virtual void DL_CreationInterface::processCodeValuePair (
            unsigned int groupCode,
            const std::string & groupValue )  [pure virtual]
```

Called for every code / value tuple of the DXF file.

The complete DXF file contents can be handled by the implemetation of this function.

Implemented in DL_CreationAdapter.

Referenced by DL_Dxf::readDxfGroups().

#### 5.10.2.61 setVariableDouble()

```
virtual void DL_CreationInterface::setVariableDouble (
            const std::string & key,
            double value,
            int code )  [pure virtual]
```

Called for every double variable in the DXF file (e.g.

"$DIMEXO").

Implemented in DL_CreationAdapter.

Referenced by DL_Dxf::addSetting().

#### 5.10.2.62 setVariableInt()

```
virtual void DL_CreationInterface::setVariableInt (
            const std::string & key,
            int value,
            int code )  [pure virtual]
```

Called for every int variable in the DXF file (e.g.

"$ACADMAINTVER").

Implemented in DL_CreationAdapter.

Referenced by DL_Dxf::addSetting().

#### 5.10.2.63 setVariableString()

```
virtual void DL_CreationInterface::setVariableString (
            const std::string & key,
            const std::string & value,
            int code )  [pure virtual]
```

Called for every string variable in the DXF file (e.g.

"$ACADVER").

Implemented in DL_CreationAdapter.

Referenced by DL_Dxf::addSetting().

#### 5.10.2.64 setVariableVector()

```
virtual void DL_CreationInterface::setVariableVector (
            const std::string & key,
            double v1,
            double v2,
            double v3,
            int code )  [pure virtual]
```

Called for every vector variable in the DXF file (e.g.

"$EXTMIN").

Implemented in DL_CreationAdapter.

Referenced by DL_Dxf::addSetting().

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

- src/dl_creationinterface.h

## 5.11 DL_DictionaryData Struct Reference

Dictionary data.

`#include <dl_entities.h>`

### Public Member Functions

- **DL_DictionaryData** (const std::string &handle)

### Public Attributes

- std::string **handle**

### 5.11.1 Detailed Description

Dictionary data.

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

- src/dl_entities.h

## 5.12 DL_DictionaryEntryData Struct Reference

Dictionary entry data.

`#include <dl_entities.h>`

### Public Member Functions

- **DL_DictionaryEntryData** (const std::string &name, const std::string &handle)

### Public Attributes

- std::string **name**
- std::string **handle**

### 5.12.1 Detailed Description

Dictionary entry data.

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

- src/dl_entities.h

## 5.13 DL_DimAlignedData Struct Reference

Aligned Dimension Data.

`#include <dl_entities.h>`

### Public Member Functions

- DL_DimAlignedData (double depx1, double depy1, double depz1, double depx2, double depy2, double depz2)

  *Constructor.*

### Public Attributes

- double epx1
- double epy1
- double epz1
- double epx2
- double epy2
- double epz2

### 5.13.1 Detailed Description

Aligned Dimension Data.

### 5.13.2 Constructor & Destructor Documentation

#### 5.13.2.1 DL_DimAlignedData()

```
DL_DimAlignedData::DL_DimAlignedData (
            double depx1,
            double depy1,
            double depz1,
            double depx2,
            double depy2,
            double depz2 )  [inline]
```

Constructor.

Parameters: see member variables.

### 5.13.3 Member Data Documentation

#### 5.13.3.1 epx1

```
double DL_DimAlignedData::epx1
```

X Coordinate of Extension point 1.

Referenced by DL_Dxf::writeDimAligned().

#### 5.13.3.2 epx2

```
double DL_DimAlignedData::epx2
```

X Coordinate of Extension point 2.

Referenced by DL_Dxf::writeDimAligned().

#### 5.13.3.3 epy1

```
double DL_DimAlignedData::epy1
```

Y Coordinate of Extension point 1.

Referenced by DL_Dxf::writeDimAligned().

#### 5.13.3.4 epy2

```
double DL_DimAlignedData::epy2
```

Y Coordinate of Extension point 2.

Referenced by DL_Dxf::writeDimAligned().

#### 5.13.3.5 epz1

```
double DL_DimAlignedData::epz1
```

Z Coordinate of Extension point 1.

#### 5.13.3.6 epz2

```
double DL_DimAlignedData::epz2
```

Z Coordinate of Extension point 2.

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

- src/dl_entities.h

## 5.14 DL_DimAngular2LData Struct Reference

Angular Dimension Data.

```
#include <dl_entities.h>
```

### Public Member Functions

- DL_DimAngular2LData (double ddpx1, double ddpy1, double ddpz1, double ddpx2, double ddpy2, double ddpz2, double ddpx3, double ddpy3, double ddpz3, double ddpx4, double ddpy4, double ddpz4)

  *Constructor.*

### Public Attributes

- double dpx1
- double dpy1
- double dpz1
- double dpx2
- double dpy2
- double dpz2
- double dpx3
- double dpy3
- double dpz3
- double dpx4
- double dpy4
- double dpz4

### 5.14.1 Detailed Description

Angular Dimension Data.

### 5.14.2 Constructor & Destructor Documentation

#### 5.14.2.1 DL_DimAngular2LData()

```
DL_DimAngular2LData::DL_DimAngular2LData (
            double ddpx1,
            double ddpy1,
            double ddpz1,
            double ddpx2,
            double ddpy2,
            double ddpz2,
            double ddpx3,
            double ddpy3,
            double ddpz3,
            double ddpx4,
            double ddpy4,
            double ddpz4 )  [inline]
```

Constructor.

Parameters: see member variables.

### 5.14.3 Member Data Documentation

#### 5.14.3.1 dpx1

```
double DL_DimAngular2LData::dpx1
```

X Coordinate of definition point 1.

Referenced by DL_Dxf::writeDimAngular2L().

#### 5.14.3.2 dpx2

```
double DL_DimAngular2LData::dpx2
```

X Coordinate of definition point 2.

Referenced by DL_Dxf::writeDimAngular2L().

#### 5.14.3.3 dpx3

```
double DL_DimAngular2LData::dpx3
```

X Coordinate of definition point 3.

Referenced by DL_Dxf::writeDimAngular2L().

#### 5.14.3.4 dpx4

```
double DL_DimAngular2LData::dpx4
```

X Coordinate of definition point 4.

Referenced by DL_Dxf::writeDimAngular2L().

#### 5.14.3.5 dpy1

```
double DL_DimAngular2LData::dpy1
```

Y Coordinate of definition point 1.

Referenced by DL_Dxf::writeDimAngular2L().

#### 5.14.3.6 dpy2

```
double DL_DimAngular2LData::dpy2
```

Y Coordinate of definition point 2.

Referenced by DL_Dxf::writeDimAngular2L().

#### 5.14.3.7 dpy3

```
double DL_DimAngular2LData::dpy3
```

Y Coordinate of definition point 3.

Referenced by DL_Dxf::writeDimAngular2L().

#### 5.14.3.8 dpy4

```
double DL_DimAngular2LData::dpy4
```

Y Coordinate of definition point 4.

Referenced by DL_Dxf::writeDimAngular2L().

#### 5.14.3.9 dpz1

```
double DL_DimAngular2LData::dpz1
```

Z Coordinate of definition point 1.

#### 5.14.3.10 dpz2

```
double DL_DimAngular2LData::dpz2
```

Z Coordinate of definition point 2.

#### 5.14.3.11 dpz3

```
double DL_DimAngular2LData::dpz3
```

Z Coordinate of definition point 3.

#### 5.14.3.12 dpz4

```
double DL_DimAngular2LData::dpz4
```

Z Coordinate of definition point 4.

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

- src/dl_entities.h

## 5.15 DL_DimAngular3PData Struct Reference

Angular Dimension Data (3 points version).

```
#include <dl_entities.h>
```

### Public Member Functions

- DL_DimAngular3PData (double ddpx1, double ddpy1, double ddpz1, double ddpx2, double ddpy2, double ddpz2, double ddpx3, double ddpy3, double ddpz3)

  *Constructor.*

### Public Attributes

- double dpx1
- double dpy1
- double dpz1
- double dpx2
- double dpy2
- double dpz2
- double dpx3
- double dpy3
- double dpz3

### 5.15.1 Detailed Description

Angular Dimension Data (3 points version).

### 5.15.2 Constructor & Destructor Documentation

#### 5.15.2.1 DL_DimAngular3PData()

```
DL_DimAngular3PData::DL_DimAngular3PData (
            double ddpx1,
            double ddpy1,
            double ddpz1,
            double ddpx2,
            double ddpy2,
            double ddpz2,
            double ddpx3,
            double ddpy3,
            double ddpz3 )  [inline]
```

Constructor.

Parameters: see member variables.

### 5.15.3 Member Data Documentation

#### 5.15.3.1 dpx1

```
double DL_DimAngular3PData::dpx1
```

X Coordinate of definition point 1 (extension line 1 end).

Referenced by DL_Dxf::writeDimAngular3P().

#### 5.15.3.2 dpx2

```
double DL_DimAngular3PData::dpx2
```

X Coordinate of definition point 2 (extension line 2 end).

Referenced by DL_Dxf::writeDimAngular3P().

#### 5.15.3.3 dpx3

```
double DL_DimAngular3PData::dpx3
```

X Coordinate of definition point 3 (center).

Referenced by DL_Dxf::writeDimAngular3P().

#### 5.15.3.4 dpy1

```
double DL_DimAngular3PData::dpy1
```

Y Coordinate of definition point 1.

Referenced by DL_Dxf::writeDimAngular3P().

#### 5.15.3.5 dpy2

```
double DL_DimAngular3PData::dpy2
```

Y Coordinate of definition point 2.

Referenced by DL_Dxf::writeDimAngular3P().

#### 5.15.3.6 dpy3

```
double DL_DimAngular3PData::dpy3
```

Y Coordinate of definition point 3.

Referenced by DL_Dxf::writeDimAngular3P().

#### 5.15.3.7 dpz1

```
double DL_DimAngular3PData::dpz1
```

Z Coordinate of definition point 1.

#### 5.15.3.8 dpz2

```
double DL_DimAngular3PData::dpz2
```

Z Coordinate of definition point 2.

#### 5.15.3.9 dpz3

```
double DL_DimAngular3PData::dpz3
```

Z Coordinate of definition point 3.

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

- src/dl_entities.h

## 5.16 DL_DimDiametricData Struct Reference

Diametric Dimension Data.

```
#include <dl_entities.h>
```

### Public Member Functions

- DL_DimDiametricData (double ddpx, double ddpy, double ddpz, double dleader)
  *Constructor.*

### Public Attributes

- double dpx
- double dpy
- double dpz
- double leader

### 5.16.1 Detailed Description

Diametric Dimension Data.

### 5.16.2 Constructor & Destructor Documentation

#### 5.16.2.1 DL_DimDiametricData()

```
DL_DimDiametricData::DL_DimDiametricData (
            double ddpx,
            double ddpy,
            double ddpz,
            double dleader )  [inline]
```

Constructor.

Parameters: see member variables.

### 5.16.3 Member Data Documentation

#### 5.16.3.1 dpx

```
double DL_DimDiametricData::dpx
```

X Coordinate of definition point (DXF 15).

Referenced by DL_Dxf::writeDimDiametric().

#### 5.16.3.2 dpy

```
double DL_DimDiametricData::dpy
```

Y Coordinate of definition point (DXF 25).

Referenced by DL_Dxf::writeDimDiametric().

#### 5.16.3.3 dpz

```
double DL_DimDiametricData::dpz
```

Z Coordinate of definition point (DXF 35).

#### 5.16.3.4 leader

```
double DL_DimDiametricData::leader
```

Leader length

Referenced by DL_Dxf::writeDimDiametric().

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

- src/dl_entities.h

## 5.17 DL_DimensionData Struct Reference

Generic Dimension Data.

```
#include <dl_entities.h>
```

## Public Member Functions

- DL_DimensionData (double dpx, double dpy, double dpz, double mpx, double mpy, double mpz, int type, int attachmentPoint, int lineSpacingStyle, double lineSpacingFactor, const std::string &text, const std::string &style, double angle, double linearFactor=1.0, double dimScale=1.0)

  *Constructor.*

## Public Attributes

- double dpx
- double dpy
- double dpz
- double mpx
- double mpy
- double mpz
- int type

  *Dimension type.*
- int attachmentPoint

  *Attachment point.*
- int lineSpacingStyle

  *Line spacing style.*
- double lineSpacingFactor

  *Line spacing factor.*
- std::string text

  *Text string.*
- std::string style
- double **angle**

  *Rotation angle of dimension text away from default orientation.*
- double **linearFactor**

  *Linear factor style override.*
- double **dimScale**

  *Dimension scale (dimscale) style override.*
- bool **arrow1Flipped**
- bool **arrow2Flipped**

### 5.17.1 Detailed Description

Generic Dimension Data.

### 5.17.2 Constructor & Destructor Documentation

#### 5.17.2.1 DL_DimensionData()

```
DL_DimensionData::DL_DimensionData (
            double dpx,
            double dpy,
            double dpz,
            double mpx,
            double mpy,
            double mpz,
            int type,
            int attachmentPoint,
            int lineSpacingStyle,
            double lineSpacingFactor,
            const std::string & text,
            const std::string & style,
            double angle,
            double linearFactor = 1.0,
            double dimScale = 1.0 )  [inline]
```

Constructor.

Parameters: see member variables.

### 5.17.3 Member Data Documentation

#### 5.17.3.1 attachmentPoint

```
int DL_DimensionData::attachmentPoint
```

Attachment point.

1 = Top left, 2 = Top center, 3 = Top right, 4 = Middle left, 5 = Middle center, 6 = Middle right, 7 = Bottom left, 8 = Bottom center, 9 = Bottom right,

Referenced by DL_Dxf::writeDimAligned(), DL_Dxf::writeDimAngular2L(), DL_Dxf::writeDimAngular3P(), DL_Dxf::writeDimDiametric(), DL_Dxf::writeDimLinear(), DL_Dxf::writeDimOrdinate(), and DL_Dxf::writeDimRadial().

#### 5.17.3.2 dpx

```
double DL_DimensionData::dpx
```

X Coordinate of definition point.

Referenced by DL_Dxf::writeDimAligned(), DL_Dxf::writeDimAngular2L(), DL_Dxf::writeDimAngular3P(), DL_Dxf::writeDimDiametric(), DL_Dxf::writeDimLinear(), DL_Dxf::writeDimOrdinate(), and DL_Dxf::writeDimRadial().

#### 5.17.3.3 dpy

```
double DL_DimensionData::dpy
```

Y Coordinate of definition point.

Referenced by DL_Dxf::writeDimAligned(), DL_Dxf::writeDimAngular2L(), DL_Dxf::writeDimAngular3P(), DL_Dxf::writeDimDiametric(), DL_Dxf::writeDimLinear(), DL_Dxf::writeDimOrdinate(), and DL_Dxf::writeDimRadial().

#### 5.17.3.4 dpz

```
double DL_DimensionData::dpz
```

Z Coordinate of definition point.

Referenced by DL_Dxf::writeDimAligned(), DL_Dxf::writeDimAngular2L(), DL_Dxf::writeDimAngular3P(), DL_Dxf::writeDimDiametric(), DL_Dxf::writeDimLinear(), DL_Dxf::writeDimOrdinate(), and DL_Dxf::writeDimRadial().

#### 5.17.3.5 lineSpacingFactor

```
double DL_DimensionData::lineSpacingFactor
```

Line spacing factor.

0.25 .. 4.0

Referenced by DL_Dxf::writeDimAligned(), DL_Dxf::writeDimAngular2L(), DL_Dxf::writeDimAngular3P(), DL_Dxf::writeDimDiametric(), DL_Dxf::writeDimLinear(), DL_Dxf::writeDimOrdinate(), and DL_Dxf::writeDimRadial().

#### 5.17.3.6 lineSpacingStyle

```
int DL_DimensionData::lineSpacingStyle
```

Line spacing style.

1 = at least, 2 = exact

Referenced by DL_Dxf::writeDimAligned(), DL_Dxf::writeDimAngular2L(), DL_Dxf::writeDimAngular3P(), DL_Dxf::writeDimDiametric(), DL_Dxf::writeDimLinear(), DL_Dxf::writeDimOrdinate(), and DL_Dxf::writeDimRadial().

#### 5.17.3.7 mpx

```
double DL_DimensionData::mpx
```

X Coordinate of middle point of the text.

Referenced by DL_Dxf::writeDimAligned(), DL_Dxf::writeDimAngular2L(), DL_Dxf::writeDimAngular3P(), DL_Dxf::writeDimDiametric(), DL_Dxf::writeDimLinear(), DL_Dxf::writeDimOrdinate(), and DL_Dxf::writeDimRadial().

#### 5.17.3.8 mpy

```
double DL_DimensionData::mpy
```

Y Coordinate of middle point of the text.

Referenced by DL_Dxf::writeDimAligned(), DL_Dxf::writeDimAngular2L(), DL_Dxf::writeDimAngular3P(), DL_Dxf::writeDimDiametric(), DL_Dxf::writeDimLinear(), DL_Dxf::writeDimOrdinate(), and DL_Dxf::writeDimRadial().

#### 5.17.3.9 mpz

```
double DL_DimensionData::mpz
```

Z Coordinate of middle point of the text.

#### 5.17.3.10 style

```
std::string DL_DimensionData::style
```

Dimension style (font name).

#### 5.17.3.11 text

```
std::string DL_DimensionData::text
```

Text string.

Text string entered explicitly by user or null or "<>" for the actual measurement or " " (one blank space). for supressing the text.

Referenced by DL_Dxf::writeDimAligned(), DL_Dxf::writeDimAngular2L(), DL_Dxf::writeDimAngular3P(), DL_Dxf::writeDimDiametric(), DL_Dxf::writeDimLinear(), DL_Dxf::writeDimOrdinate(), and DL_Dxf::writeDimRadial().

#### 5.17.3.12 type

```
int DL_DimensionData::type
```

Dimension type.

0 Rotated, horizontal, or vertical
1 Aligned
2 Angular
3 Diametric
4 Radius
5 Angular 3-point
6 Ordinate
64 Ordinate type. This is a bit value (bit 7)
used only with integer value 6. If set,
ordinate is X-type; if not set, ordinate is Y-type
128 This is a bit value (bit 8) added to the
other group 70 values if the dimension text has been positioned at a user-defined
location rather than at the default location

Referenced by DL_Dxf::writeDimAligned(), DL_Dxf::writeDimAngular2L(), DL_Dxf::writeDimAngular3P(), DL_Dxf::writeDimDiametric(), DL_Dxf::writeDimLinear(), DL_Dxf::writeDimOrdinate(), and DL_Dxf::writeDimRadial().

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

- src/dl_entities.h

## 5.18 DL_DimLinearData Struct Reference

Linear (rotated) Dimension Data.

```
#include <dl_entities.h>
```

### Public Member Functions

- DL_DimLinearData (double ddpx1, double ddpy1, double ddpz1, double ddpx2, double ddpy2, double ddpz2, double dAngle, double dOblique)

  *Constructor.*

### Public Attributes

- double dpx1
- double dpy1
- double dpz1
- double dpx2
- double dpy2
- double dpz2
- double angle
- double oblique

### 5.18.1 Detailed Description

Linear (rotated) Dimension Data.

### 5.18.2 Constructor & Destructor Documentation

#### 5.18.2.1 DL_DimLinearData()

```
DL_DimLinearData::DL_DimLinearData (
            double ddpx1,
            double ddpy1,
            double ddpz1,
            double ddpx2,
            double ddpy2,
            double ddpz2,
            double dAngle,
            double dOblique )  [inline]
```

Constructor.

Parameters: see member variables.

### 5.18.3 Member Data Documentation

#### 5.18.3.1 angle

```
double DL_DimLinearData::angle
```

Rotation angle (angle of dimension line) in degrees.

Referenced by DL_Dxf::writeDimLinear().

#### 5.18.3.2 dpx1

```
double DL_DimLinearData::dpx1
```

X Coordinate of Extension point 1.

Referenced by DL_Dxf::writeDimLinear().

#### 5.18.3.3 dpx2

```
double DL_DimLinearData::dpx2
```

X Coordinate of Extension point 2.

Referenced by DL_Dxf::writeDimLinear().

#### 5.18.3.4 dpy1

```
double DL_DimLinearData::dpy1
```

Y Coordinate of Extension point 1.

Referenced by DL_Dxf::writeDimLinear().

#### 5.18.3.5 dpy2

```
double DL_DimLinearData::dpy2
```

Y Coordinate of Extension point 2.

Referenced by DL_Dxf::writeDimLinear().

#### 5.18.3.6 dpz1

```
double DL_DimLinearData::dpz1
```

Z Coordinate of Extension point 1.

#### 5.18.3.7 dpz2

```
double DL_DimLinearData::dpz2
```

Z Coordinate of Extension point 2.

#### 5.18.3.8 oblique

```
double DL_DimLinearData::oblique
```

Oblique angle in degrees.

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

- src/dl_entities.h

## 5.19 DL_DimOrdinateData Struct Reference

Ordinate Dimension Data.

`#include <dl_entities.h>`

### Public Member Functions

- DL_DimOrdinateData (double ddpx1, double ddpy1, double ddpz1, double ddpx2, double ddpy2, double ddpz2, bool dxtype)

  *Constructor.*

### Public Attributes

- double dpx1
- double dpy1
- double dpz1
- double dpx2
- double dpy2
- double dpz2
- bool xtype

### 5.19.1 Detailed Description

Ordinate Dimension Data.

### 5.19.2 Constructor & Destructor Documentation

#### 5.19.2.1 DL_DimOrdinateData()

```
DL_DimOrdinateData::DL_DimOrdinateData (
            double ddpx1,
            double ddpy1,
            double ddpz1,
            double ddpx2,
            double ddpy2,
            double ddpz2,
            bool dxtype )  [inline]
```

Constructor.

Parameters: see member variables.

### 5.19.3 Member Data Documentation

#### 5.19.3.1 dpx1

```
double DL_DimOrdinateData::dpx1
```

X Coordinate of definition point 1.

Referenced by DL_Dxf::writeDimOrdinate().

#### 5.19.3.2 dpx2

```
double DL_DimOrdinateData::dpx2
```

X Coordinate of definition point 2.

Referenced by DL_Dxf::writeDimOrdinate().

#### 5.19.3.3 dpy1

```
double DL_DimOrdinateData::dpy1
```

Y Coordinate of definition point 1.

Referenced by DL_Dxf::writeDimOrdinate().

#### 5.19.3.4 dpy2

```
double DL_DimOrdinateData::dpy2
```

Y Coordinate of definition point 2.

Referenced by DL_Dxf::writeDimOrdinate().

#### 5.19.3.5 dpz1

```
double DL_DimOrdinateData::dpz1
```

Z Coordinate of definition point 1.

#### 5.19.3.6 dpz2

```
double DL_DimOrdinateData::dpz2
```

Z Coordinate of definition point 2.

#### 5.19.3.7 xtype

```
bool DL_DimOrdinateData::xtype
```

True if the dimension indicates the X-value, false for Y-value

Referenced by DL_Dxf::writeDimOrdinate().

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

- src/dl_entities.h

## 5.20 DL_DimRadialData Struct Reference

Radial Dimension Data.

```
#include <dl_entities.h>
```

### Public Member Functions

- DL_DimRadialData (double ddpx, double ddpy, double ddpz, double dleader)
  *Constructor.*

### Public Attributes

- double dpx
- double dpy
- double dpz
- double leader

### 5.20.1 Detailed Description

Radial Dimension Data.

### 5.20.2 Constructor & Destructor Documentation

#### 5.20.2.1 DL_DimRadialData()

```
DL_DimRadialData::DL_DimRadialData (
            double ddpx,
            double ddpy,
            double ddpz,
            double dleader )  [inline]
```

Constructor.

Parameters: see member variables.

### 5.20.3 Member Data Documentation

#### 5.20.3.1 dpx

```
double DL_DimRadialData::dpx
```

X Coordinate of definition point.

Referenced by DL_Dxf::writeDimRadial().

#### 5.20.3.2 dpy

```
double DL_DimRadialData::dpy
```

Y Coordinate of definition point.

Referenced by DL_Dxf::writeDimRadial().

#### 5.20.3.3 dpz

```
double DL_DimRadialData::dpz
```

Z Coordinate of definition point.

#### 5.20.3.4 leader

```
double DL_DimRadialData::leader
```

Leader length

Referenced by DL_Dxf::writeDimRadial().

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

- src/dl_entities.h

## 5.21 DL_Dxf Class Reference

Reading and writing of DXF files.

```
#include <dl_dxf.h>
```

## Public Member Functions

- **DL_Dxf** ()

  *Default constructor.*
- ~**DL_Dxf** ()

  *Destructor.*
- bool in (const std::string &file, DL_CreationInterface *creationInterface)

  *Reads the given file and calls the appropriate functions in the given creation interface for every entity found in the file.*
- bool readDxfGroups (FILE *fp, DL_CreationInterface *creationInterface)

  *Reads a group couplet from a DXF file.*
- bool **readDxfGroups** (std::istream &stream, DL_CreationInterface *creationInterface)

  *Same as above but for input streams.*
- bool in (std::istream &stream, DL_CreationInterface *creationInterface)

  *Reads a DXF file from an existing stream.*
- bool processDXFGroup (DL_CreationInterface *creationInterface, int groupCode, const std::string &groupValue)

  *Processes a group (pair of group code and value).*
- void **addSetting** (DL_CreationInterface *creationInterface)

  *Adds a variable from the DXF file.*
- void **addLayer** (DL_CreationInterface *creationInterface)

  *Adds a layer that was read from the file via the creation interface.*
- void **addLinetype** (DL_CreationInterface *creationInterface)

  *Adds a linetype that was read from the file via the creation interface.*
- void **addBlock** (DL_CreationInterface *creationInterface)

  *Adds a block that was read from the file via the creation interface.*
- void **endBlock** (DL_CreationInterface *creationInterface)

  *Ends a block that was read from the file via the creation interface.*
- void **addTextStyle** (DL_CreationInterface *creationInterface)
- void **addPoint** (DL_CreationInterface *creationInterface)

  *Adds a point entity that was read from the file via the creation interface.*
- void **addLine** (DL_CreationInterface *creationInterface)

  *Adds a line entity that was read from the file via the creation interface.*
- void **addXLine** (DL_CreationInterface *creationInterface)

  *Adds an xline entity that was read from the file via the creation interface.*
- void **addRay** (DL_CreationInterface *creationInterface)

  *Adds a ray entity that was read from the file via the creation interface.*
- void **addPolyline** (DL_CreationInterface *creationInterface)

  *Adds a polyline entity that was read from the file via the creation interface.*
- void **addVertex** (DL_CreationInterface *creationInterface)

  *Adds a polyline vertex entity that was read from the file via the creation interface.*
- void **addSpline** (DL_CreationInterface *creationInterface)

  *Adds a spline entity that was read from the file via the creation interface.*
- void **addArc** (DL_CreationInterface *creationInterface)

  *Adds an arc entity that was read from the file via the creation interface.*
- void **addCircle** (DL_CreationInterface *creationInterface)

  *Adds a circle entity that was read from the file via the creation interface.*
- void **addEllipse** (DL_CreationInterface *creationInterface)

  *Adds an ellipse entity that was read from the file via the creation interface.*
- void **addInsert** (DL_CreationInterface *creationInterface)

  *Adds an insert entity that was read from the file via the creation interface.*
- void addTrace (DL_CreationInterface *creationInterface)

*Adds a trace entity (4 edge closed polyline) that was read from the file via the creation interface.*

- void **add3dFace** (DL_CreationInterface *creationInterface)

  *Adds a 3dface entity that was read from the file via the creation interface.*
- void addSolid (DL_CreationInterface *creationInterface)

  *Adds a solid entity (filled trace) that was read from the file via the creation interface.*
- void **addMText** (DL_CreationInterface *creationInterface)

  *Adds an MText entity that was read from the file via the creation interface.*
- void **addText** (DL_CreationInterface *creationInterface)

  *Adds an text entity that was read from the file via the creation interface.*
- void **addArcAlignedText** (DL_CreationInterface *creationInterface)

  *Adds an arc aligned text entity that was read from the file via the creation interface.*
- void addAttribute (DL_CreationInterface *creationInterface)

  *Adds an attrib entity that was read from the file via the creation interface.*
- DL_DimensionData getDimData ()
- void **addDimLinear** (DL_CreationInterface *creationInterface)

  *Adds a linear dimension entity that was read from the file via the creation interface.*
- void **addDimAligned** (DL_CreationInterface *creationInterface)

  *Adds an aligned dimension entity that was read from the file via the creation interface.*
- void **addDimRadial** (DL_CreationInterface *creationInterface)

  *Adds a radial dimension entity that was read from the file via the creation interface.*
- void **addDimDiametric** (DL_CreationInterface *creationInterface)

  *Adds a diametric dimension entity that was read from the file via the creation interface.*
- void **addDimAngular** (DL_CreationInterface *creationInterface)

  *Adds an angular dimension entity that was read from the file via the creation interface.*
- void **addDimAngular3P** (DL_CreationInterface *creationInterface)

  *Adds an angular dimension entity that was read from the file via the creation interface.*
- void **addDimOrdinate** (DL_CreationInterface *creationInterface)

  *Adds an ordinate dimension entity that was read from the file via the creation interface.*
- void **addLeader** (DL_CreationInterface *creationInterface)

  *Adds a leader entity that was read from the file via the creation interface.*
- void **addHatch** (DL_CreationInterface *creationInterface)

  *Adds a hatch entity that was read from the file via the creation interface.*
- void **addHatchLoop** ()
- void **addHatchEdge** ()
- bool **handleHatchData** (DL_CreationInterface *creationInterface)

  *Handles all hatch data.*
- void **addImage** (DL_CreationInterface *creationInterface)

  *Adds an image entity that was read from the file via the creation interface.*
- void **addImageDef** (DL_CreationInterface *creationInterface)

  *Adds an image definition that was read from the file via the creation interface.*
- void **addComment** (DL_CreationInterface *creationInterface, const std::string &comment)

  *Adds a comment from the DXF file.*
- void **addDictionary** (DL_CreationInterface *creationInterface)
- void **addDictionaryEntry** (DL_CreationInterface *creationInterface)
- bool **handleXRecordData** (DL_CreationInterface *creationInterface)

  *Handles all XRecord data.*
- bool **handleDictionaryData** (DL_CreationInterface *creationInterface)

  *Handles all dictionary data.*
- bool **handleXData** (DL_CreationInterface *creationInterface)

  *Handles XData for all object types.*
- bool **handleMTextData** (DL_CreationInterface *creationInterface)

*Handles additional MText data.*

- bool **handleLWPolylineData** (DL_CreationInterface *creationInterface)

  *Handles additional polyline data.*
- bool **handleSplineData** (DL_CreationInterface *creationInterface)

  *Handles additional spline data.*
- bool **handleLeaderData** (DL_CreationInterface *creationInterface)

  *Handles additional leader data.*
- bool **handleLinetypeData** (DL_CreationInterface *creationInterface)

  *Handles all dashes in linetype pattern.*
- void **endEntity** (DL_CreationInterface *creationInterface)

  *Ends some special entities like hatches or old style polylines.*
- void **endSequence** (DL_CreationInterface *creationInterface)

  *Ends a sequence and notifies the creation interface.*
- DL_WriterA * out (const char *file, DL_Codes::version version=DL_VERSION_2000)

  *Converts the given string into an int.*
- void **writeHeader** (DL_WriterA &dw)

  *Writes a DXF header to the file currently opened by the given DXF writer object.*
- void writePoint (DL_WriterA &dw, const DL_PointData &data, const DL_Attributes &attrib)

  *Writes a point entity to the file.*
- void writeLine (DL_WriterA &dw, const DL_LineData &data, const DL_Attributes &attrib)

  *Writes a line entity to the file.*
- void writeXLine (DL_WriterA &dw, const DL_XLineData &data, const DL_Attributes &attrib)

  *Writes an x line entity to the file.*
- void writeRay (DL_WriterA &dw, const DL_RayData &data, const DL_Attributes &attrib)

  *Writes a ray entity to the file.*
- void writePolyline (DL_WriterA &dw, const DL_PolylineData &data, const DL_Attributes &attrib)

  *Writes a polyline entity to the file.*
- void writeVertex (DL_WriterA &dw, const DL_VertexData &data)

  *Writes a single vertex of a polyline to the file.*
- void writePolylineEnd (DL_WriterA &dw)

  *Writes the polyline end.*
- void writeSpline (DL_WriterA &dw, const DL_SplineData &data, const DL_Attributes &attrib)

  *Writes a spline entity to the file.*
- void writeControlPoint (DL_WriterA &dw, const DL_ControlPointData &data)

  *Writes a single control point of a spline to the file.*
- void writeFitPoint (DL_WriterA &dw, const DL_FitPointData &data)

  *Writes a single fit point of a spline to the file.*
- void writeKnot (DL_WriterA &dw, const DL_KnotData &data)

  *Writes a single knot of a spline to the file.*
- void writeCircle (DL_WriterA &dw, const DL_CircleData &data, const DL_Attributes &attrib)

  *Writes a circle entity to the file.*
- void writeArc (DL_WriterA &dw, const DL_ArcData &data, const DL_Attributes &attrib)

  *Writes an arc entity to the file.*
- void writeEllipse (DL_WriterA &dw, const DL_EllipseData &data, const DL_Attributes &attrib)

  *Writes an ellipse entity to the file.*
- void writeSolid (DL_WriterA &dw, const DL_SolidData &data, const DL_Attributes &attrib)

  *Writes a solid entity to the file.*
- void writeTrace (DL_WriterA &dw, const DL_TraceData &data, const DL_Attributes &attrib)

  *Writes a trace entity to the file.*
- void write3dFace (DL_WriterA &dw, const DL_3dFaceData &data, const DL_Attributes &attrib)

  *Writes a 3d face entity to the file.*

- void writeInsert (DL_WriterA &dw, const DL_InsertData &data, const DL_Attributes &attrib)

  *Writes an insert to the file.*
- void writeMText (DL_WriterA &dw, const DL_MTextData &data, const DL_Attributes &attrib)

  *Writes a multi text entity to the file.*
- void writeText (DL_WriterA &dw, const DL_TextData &data, const DL_Attributes &attrib)

  *Writes a text entity to the file.*
- void **writeAttribute** (DL_WriterA &dw, const DL_AttributeData &data, const DL_Attributes &attrib)
- void **writeDimStyleOverrides** (DL_WriterA &dw, const DL_DimensionData &data)
- void writeDimAligned (DL_WriterA &dw, const DL_DimensionData &data, const DL_DimAlignedData &edata, const DL_Attributes &attrib)

  *Writes an aligned dimension entity to the file.*
- void writeDimLinear (DL_WriterA &dw, const DL_DimensionData &data, const DL_DimLinearData &edata, const DL_Attributes &attrib)

  *Writes a linear dimension entity to the file.*
- void writeDimRadial (DL_WriterA &dw, const DL_DimensionData &data, const DL_DimRadialData &edata, const DL_Attributes &attrib)

  *Writes a radial dimension entity to the file.*
- void writeDimDiametric (DL_WriterA &dw, const DL_DimensionData &data, const DL_DimDiametricData &edata, const DL_Attributes &attrib)

  *Writes a diametric dimension entity to the file.*
- void writeDimAngular2L (DL_WriterA &dw, const DL_DimensionData &data, const DL_DimAngular2LData &edata, const DL_Attributes &attrib)

  *Writes an angular dimension entity to the file.*
- void writeDimAngular3P (DL_WriterA &dw, const DL_DimensionData &data, const DL_DimAngular3PData &edata, const DL_Attributes &attrib)

  *Writes an angular dimension entity (3 points version) to the file.*
- void writeDimOrdinate (DL_WriterA &dw, const DL_DimensionData &data, const DL_DimOrdinateData &edata, const DL_Attributes &attrib)

  *Writes an ordinate dimension entity to the file.*
- void writeLeader (DL_WriterA &dw, const DL_LeaderData &data, const DL_Attributes &attrib)

  *Writes a leader entity to the file.*
- void writeLeaderVertex (DL_WriterA &dw, const DL_LeaderVertexData &data)

  *Writes a single vertex of a leader to the file.*
- void **writeLeaderEnd** (DL_WriterA &dw, const DL_LeaderData &data)
- void writeHatch1 (DL_WriterA &dw, const DL_HatchData &data, const DL_Attributes &attrib)

  *Writes the beginning of a hatch entity to the file.*
- void writeHatch2 (DL_WriterA &dw, const DL_HatchData &data, const DL_Attributes &attrib)

  *Writes the end of a hatch entity to the file.*
- void writeHatchLoop1 (DL_WriterA &dw, const DL_HatchLoopData &data)

  *Writes the beginning of a hatch loop to the file.*
- void writeHatchLoop2 (DL_WriterA &dw, const DL_HatchLoopData &data)

  *Writes the end of a hatch loop to the file.*
- void writeHatchEdge (DL_WriterA &dw, const DL_HatchEdgeData &data)

  *Writes the beginning of a hatch entity to the file.*
- unsigned long writeImage (DL_WriterA &dw, const DL_ImageData &data, const DL_Attributes &attrib)

  *Writes an image entity.*
- void **writeImageDef** (DL_WriterA &dw, int handle, const DL_ImageData &data)

  *Writes an image definiition entity.*
- void writeLayer (DL_WriterA &dw, const DL_LayerData &data, const DL_Attributes &attrib)

  *Writes a layer to the file.*
- void writeLinetype (DL_WriterA &dw, const DL_LinetypeData &data)

  *Writes a line type to the file.*

- void writeAppid (DL_WriterA &dw, const std::string &name)

  *Writes the APPID section to the DXF file.*
- void **writeBlock** (DL_WriterA &dw, const DL_BlockData &data)

  *Writes a block's definition (no entities) to the DXF file.*
- void writeEndBlock (DL_WriterA &dw, const std::string &name)

  *Writes a block end.*
- void writeVPort (DL_WriterA &dw)

  *Writes a viewport section.*
- void writeStyle (DL_WriterA &dw, const DL_StyleData &style)

  *Writes a style section.*
- void writeView (DL_WriterA &dw)

  *Writes a view section.*
- void writeUcs (DL_WriterA &dw)

  *Writes a ucs section.*
- void writeDimStyle (DL_WriterA &dw, double dimasz, double dimexe, double dimexo, double dimgap, double dimtxt)

  *Writes a dimstyle section.*
- void writeBlockRecord (DL_WriterA &dw)

  *Writes a blockrecord section.*
- void **writeBlockRecord** (DL_WriterA &dw, const std::string &name)

  *Writes a single block record with the given name.*
- void writeObjects (DL_WriterA &dw, const std::string &appDictionaryName="")

  *Writes a objects section.*
- void **writeAppDictionary** (DL_WriterA &dw)
- unsigned long **writeDictionaryEntry** (DL_WriterA &dw, const std::string &name)
- void **writeXRecord** (DL_WriterA &dw, int handle, int value)
- void **writeXRecord** (DL_WriterA &dw, int handle, double value)
- void **writeXRecord** (DL_WriterA &dw, int handle, bool value)
- void **writeXRecord** (DL_WriterA &dw, int handle, const std::string &value)
- void writeObjectsEnd (DL_WriterA &dw)

  *Writes the end of the objects section.*
- void **writeComment** (DL_WriterA &dw, const std::string &comment)

  *Writes a comment to the DXF file.*
- DL_Codes::version **getVersion** ()
- int getLibVersion (const std::string &str)
- bool **hasValue** (int code)
- int **getIntValue** (int code, int def)
- int **toInt** (const std::string &str)
- int **getInt16Value** (int code, int def)
- int **toInt16** (const std::string &str)
- bool **toBool** (const std::string &str)
- std::string **getStringValue** (int code, const std::string &def)
- double **getRealValue** (int code, double def)
- double **toReal** (const std::string &str)

## Static Public Member Functions

- static bool getStrippedLine (std::string &s, unsigned int size, FILE *stream, bool stripSpace=true)

  *Reads line from file & strips whitespace at start and newline at end.*
- static bool **getStrippedLine** (std::string &s, unsigned int size, std::istream &stream, bool stripSpace=true)

  *Same as above but for input streams.*
- static bool stripWhiteSpace (char **s, bool stripSpaces=true)

  *Strips leading whitespace and trailing Carriage Return (CR) and Line Feed (LF) from NULL terminated string.*
- static bool checkVariable (const char *var, DL_Codes::version version)

  *Converts the given string into a double or returns the given default valud (def) if value is NULL or empty.*
- static void test ()

  *Converts the given string into a double or returns the given default valud (def) if value is NULL or empty.*

### 5.21.1 Detailed Description

Reading and writing of DXF files.

This class can read in a DXF file and calls methods from the interface DL_EntityContainer to add the entities to the contianer provided by the user of the library.

It can also be used to write DXF files to a certain extent.

When saving entities, special values for colors and linetypes can be used:

Special colors are 0 (=BYBLOCK) and 256 (=BYLAYER). Special linetypes are "BYLAYER" and "BYBLOCK".

**Author**

Andrew Mustun

### 5.21.2 Member Function Documentation

#### 5.21.2.1 addAttribute()

```
void DL_Dxf::addAttribute (
            DL_CreationInterface * creationInterface )
```

Adds an attrib entity that was read from the file via the creation interface.

**Todo** add attrib instead of normal text

References DL_CreationInterface::addAttribute().

Referenced by processDXFGroup().

#### 5.21.2.2 addSolid()

```
void DL_Dxf::addSolid (
            DL_CreationInterface * creationInterface )
```

Adds a solid entity (filled trace) that was read from the file via the creation interface.

**Author**

AHM

References DL_CreationInterface::addSolid(), and DL_TraceData::x.

Referenced by processDXFGroup().

#### 5.21.2.3 addTrace()

```
void DL_Dxf::addTrace (
            DL_CreationInterface * creationInterface )
```

Adds a trace entity (4 edge closed polyline) that was read from the file via the creation interface.

**Author**

AHM

References DL_CreationInterface::addTrace(), and DL_TraceData::x.

Referenced by processDXFGroup().

#### 5.21.2.4 checkVariable()

```
bool DL_Dxf::checkVariable (
            const char * var,
            DL_Codes::version version )  [static]
```

Converts the given string into a double or returns the given default valud (def) if value is NULL or empty.

Checks if the given variable is known by the given DXF version.

Converts the given string into an int or returns the given default valud (def) if value is NULL or empty. Converts the given string into a string or returns the given default valud (def) if value is NULL or empty.

#### 5.21.2.5 getDimData()

```
DL_DimensionData DL_Dxf::getDimData ( )
```

**Returns**

dimension data from current values.

Referenced by addDimAligned(), addDimAngular(), addDimAngular3P(), addDimDiametric(), addDimLinear(), addDimOrdinate(), and addDimRadial().

#### 5.21.2.6 getLibVersion()

```
int DL_Dxf::getLibVersion (
            const std::string & str )
```

**Returns**

the library version as int (4 bytes, each byte one version number). e.g. if str = "2.0.2.0" getLibVersion returns 0x02000200

Referenced by processDXFGroup().

#### 5.21.2.7 getStrippedLine()

```
bool DL_Dxf::getStrippedLine (
            std::string & s,
            unsigned int size,
            FILE * fp,
            bool stripSpace = true )  [static]
```

Reads line from file & strips whitespace at start and newline at end.

**Parameters**

| | |
|---|---|
| *s* | Output<br>Pointer to character array that chopped line will be returned in. |
| *size* | Size of `s`. (Including space for NULL.) |
| *fp* | Input<br>Handle of input file. |

**Return values**

| | |
|---|---|
| *true* | if line could be read |
| *false* | if `fp` is already at end of file |

**Todo** Change function to use safer FreeBSD strl* functions

Is it a problem if line is blank (i.e., newline only)? Then, when function returns, (s==NULL).

References stripWhiteSpace().

Referenced by readDxfGroups().

#### 5.21.2.8 in() [1/2]

```
bool DL_Dxf::in (
            const std::string & file,
            DL_CreationInterface * creationInterface )
```

Reads the given file and calls the appropriate functions in the given creation interface for every entity found in the file.

**Parameters**

| | |
|---|---|
| *file* | Input Path and name of file to read |
| *creationInterface* | Pointer to the class which takes care of the entities in the file. |

**Return values**

| | |
|---|---|
| *true* | If `file` could be opened. |
| *false* | If `file` could not be opened. |

References readDxfGroups().

#### 5.21.2.9 in() [2/2]

```
bool DL_Dxf::in (
            std::istream & stream,
            DL_CreationInterface * creationInterface )
```

Reads a DXF file from an existing stream.

**Parameters**

| | |
|---|---|
| *stream* | The input stream. |
| *creationInterface* | Pointer to the class which takes care of the entities in the file. |

**Return values**

| | |
|---|---|
| *true* | If `file` could be opened. |
| *false* | If `file` could not be opened. |

References readDxfGroups().

#### 5.21.2.10 out()

```
DL_WriterA * DL_Dxf::out (
             const char * file,
             DL_Codes::version version = DL_VERSION_2000 )
```

Converts the given string into an int.

ok is set to false if there was an error.

Opens the given file for writing and returns a pointer to the dxf writer. This pointer needs to be passed on to other writing functions.

**Parameters**

| | |
|---|---|
| *file* | Full path of the file to open. |

**Returns**

Pointer to an ascii dxf writer object.

References DL_WriterA::openFailed().

#### 5.21.2.11 processDXFGroup()

```
bool DL_Dxf::processDXFGroup (
             DL_CreationInterface * creationInterface,
             int groupCode,
             const std::string & groupValue )
```

Processes a group (pair of group code and value).

**Parameters**

| | |
|---|---|
| *creationInterface* | Handle to class that creates entities and other CAD data from DXF group codes |
| *groupCode* | Constant indicating the data type of the group. |
| *groupValue* | The data value. |

**Return values**

| | |
|---|---|
| *true* | if done processing current entity and new entity begun |
| *false* | if not done processing current entity |

References add3dFace(), addArc(), addArcAlignedText(), addAttribute(), addBlock(), addCircle(), addComment(), addDimAligned(), addDimAngular(), addDimAngular3P(), addDimDiametric(), addDimLinear(), addDimOrdinate(), addDimRadial(), addEllipse(), addImage(), addImageDef(), addInsert(), addLayer(), addLeader(), addLine(), addLinetype(), addMText(), addPoint(), addPolyline(), addRay(), addSetting(), addSolid(), addSpline(), addText(), addTrace(), addVertex(), addXLine(), endBlock(), endEntity(), DL_CreationInterface::endSection(), endSequence(), getLibVersion(), handleDictionaryData(), handleHatchData(), handleLeaderData(), handleLinetypeData(), handleLWPolylineData(), handleMTextData(), handleSplineData(), handleXData(), handleXRecordData(), DL_CreationInterface::setAttributes(), DL_CreationInterface::setExtrusion(), and DL_Attributes::setLinetypeScale().

Referenced by readDxfGroups().

#### 5.21.2.12 readDxfGroups()

```
bool DL_Dxf::readDxfGroups (
            FILE * fp,
            DL_CreationInterface * creationInterface )
```

Reads a group couplet from a DXF file.

Calls another function to process it.

A group couplet consists of two lines that represent a single piece of data. An integer constant on the first line indicates the type of data. The value is on the next line.
This function reads a couplet, determines the type of data, and passes the value to the the appropriate handler function of `creationInterface`.
`fp` is advanced so that the next call to `readDXFGroups()` reads the next couplet in the file.

**Parameters**

| | |
|---|---|
| *fp* | Handle of input file |
| *creationInterface* | Handle of class which processes entities in the file |

**Return values**

| | |
|---|---|
| *true* | If EOF not reached. |
| *false* | If EOF reached. |

References getStrippedLine(), DL_CreationInterface::processCodeValuePair(), and processDXFGroup().

Referenced by in().

#### 5.21.2.13 stripWhiteSpace()

```
bool DL_Dxf::stripWhiteSpace (
            char ** s,
            bool stripSpace = true )  [static]
```

Strips leading whitespace and trailing Carriage Return (CR) and Line Feed (LF) from NULL terminated string.

**Parameters**

| | |
|---|---|
| *s* | Input and output. NULL terminates string. |

**Return values**

| | |
|---|---|
| *true* | if s is non-NULL |
| *false* | if s is NULL |

Referenced by getStrippedLine(), and test().

#### 5.21.2.14 test()

```
void DL_Dxf::test ( )  [static]
```

Converts the given string into a double or returns the given default valud (def) if value is NULL or empty.

Some test routines.

References stripWhiteSpace().

#### 5.21.2.15 write3dFace()

```
void DL_Dxf::write3dFace (
            DL_WriterA & dw,
            const DL_3dFaceData & data,
            const DL_Attributes & attrib )
```

Writes a 3d face entity to the file.

**Parameters**

| | |
|---|---|
| *dw* | DXF writer |
| *data* | Entity data from the file |
| *attrib* | Attributes |

References DL_WriterA::dxfString(), DL_Writer::entity(), DL_Writer::entityAttributes(), and DL_TraceData::x.

#### 5.21.2.16 writeAppid()

```
void DL_Dxf::writeAppid (
            DL_WriterA & dw,
            const std::string & name )
```

Writes the APPID section to the DXF file.

**Parameters**

| | |
|---|---|
| *name* | Application name |

References DL_WriterA::dxfInt(), DL_WriterA::dxfString(), and DL_Writer::tableAppidEntry().

#### 5.21.2.17 writeArc()

```
void DL_Dxf::writeArc (
            DL_WriterA & dw,
            const DL_ArcData & data,
            const DL_Attributes & attrib )
```

Writes an arc entity to the file.

**Parameters**

| | |
|---|---|
| *dw* | DXF writer |
| *data* | Entity data from the file |
| *attrib* | Attributes |

References DL_ArcData::angle1, DL_ArcData::angle2, DL_ArcData::cx, DL_ArcData::cy, DL_ArcData::cz, DL_WriterA::dxfReal(), DL_WriterA::dxfString(), DL_Writer::entity(), DL_Writer::entityAttributes(), and DL_ArcData::radius.

#### 5.21.2.18 writeBlockRecord()

```
void DL_Dxf::writeBlockRecord (
            DL_WriterA & dw )
```

Writes a blockrecord section.

This section is needed in DL_VERSION_R13. Note that this method currently only writes a faked BLOCKRECORD section to make the file readable by Aut∗cad.

References DL_WriterA::dxfHex(), DL_WriterA::dxfInt(), and DL_WriterA::dxfString().

#### 5.21.2.19 writeCircle()

```
void DL_Dxf::writeCircle (
            DL_WriterA & dw,
            const DL_CircleData & data,
            const DL_Attributes & attrib )
```

Writes a circle entity to the file.

**Parameters**

| | |
|---|---|
| *dw* | DXF writer |
| *data* | Entity data from the file |
| *attrib* | Attributes |

References DL_CircleData::cx, DL_CircleData::cy, DL_CircleData::cz, DL_WriterA::dxfReal(), DL_WriterA::dxfString(), DL_Writer::entity(), DL_Writer::entityAttributes(), and DL_CircleData::radius.

#### 5.21.2.20 writeControlPoint()

```
void DL_Dxf::writeControlPoint (
            DL_WriterA & dw,
            const DL_ControlPointData & data )
```

Writes a single control point of a spline to the file.

**Parameters**

| | |
|---|---|
| *dw* | DXF writer |
| *data* | Entity data from the file |
| *attrib* | Attributes |

References DL_WriterA::dxfReal(), DL_ControlPointData::x, DL_ControlPointData::y, and DL_ControlPointData::z.

#### 5.21.2.21 writeDimAligned()

```
void DL_Dxf::writeDimAligned (
            DL_WriterA & dw,
            const DL_DimensionData & data,
            const DL_DimAlignedData & edata,
            const DL_Attributes & attrib )
```

Writes an aligned dimension entity to the file.

**Parameters**

| | |
|---|---|
| *dw* | DXF writer |
| *data* | Generic dimension data for from the file |
| *data* | Specific aligned dimension data from the file |
| *attrib* | Attributes |

References DL_DimensionData::angle, DL_DimensionData::attachmentPoint, DL_DimensionData::dpx, DL_DimensionData::dpy, DL_DimensionData::dpz, DL_WriterA::dxfInt(), DL_WriterA::dxfReal(), DL_WriterA::dxfString(), DL_Writer::entity(), DL_Writer::entityAttributes(), DL_DimAlignedData::epx1, DL_DimAlignedData::epx2, DL_DimAlignedData::epy1,

DL_DimAlignedData::epy2, DL_DimensionData::lineSpacingFactor, DL_DimensionData::lineSpacingStyle, DL_DimensionData::mpx, DL_DimensionData::mpy, DL_DimensionData::text, and DL_DimensionData::type.

#### 5.21.2.22 writeDimAngular2L()

```
void DL_Dxf::writeDimAngular2L (
            DL_WriterA & dw,
            const DL_DimensionData & data,
            const DL_DimAngular2LData & edata,
            const DL_Attributes & attrib )
```

Writes an angular dimension entity to the file.

**Parameters**

| | |
|---|---|
| *dw* | DXF writer |
| *data* | Generic dimension data for from the file |
| *data* | Specific angular dimension data from the file |
| *attrib* | Attributes |

References DL_DimensionData::angle, DL_DimensionData::attachmentPoint, DL_DimensionData::dpx, DL_DimAngular2LData::dpx1, DL_DimAngular2LData::dpx2, DL_DimAngular2LData::dpx3, DL_DimAngular2LData::dpx4, DL_DimensionData::dpy, DL_DimAngular2LData::dpy1, DL_DimAngular2LData::dpy2, DL_DimAngular2LData::dpy3, DL_DimAngular2LData::dpy4, DL_DimensionData::dpz, DL_WriterA::dxfInt(), DL_WriterA::dxfReal(), DL_WriterA::dxfString(), DL_Writer::entity(), DL_Writer::entityAttributes(), DL_DimensionData::lineSpacingFactor, DL_DimensionData::lineSpacingStyle, DL_DimensionData::mpx, DL_DimensionData::mpy, DL_DimensionData::text, and DL_DimensionData::type.

#### 5.21.2.23 writeDimAngular3P()

```
void DL_Dxf::writeDimAngular3P (
            DL_WriterA & dw,
            const DL_DimensionData & data,
            const DL_DimAngular3PData & edata,
            const DL_Attributes & attrib )
```

Writes an angular dimension entity (3 points version) to the file.

**Parameters**

| | |
|---|---|
| *dw* | DXF writer |
| *data* | Generic dimension data for from the file |
| *data* | Specific angular dimension data from the file |
| *attrib* | Attributes |

References DL_DimensionData::angle, DL_DimensionData::attachmentPoint, DL_DimensionData::dpx, DL_DimAngular3PData::dpx1, DL_DimAngular3PData::dpx2, DL_DimAngular3PData::dpx3, DL_DimensionData::dpy, DL_DimAngular3PData::dpy1, DL_DimAngular3PData::dpy2, DL_DimAngular3PData::dpy3, DL_DimensionData::dpz, DL_WriterA::dxfInt(),

DL_WriterA::dxfReal(), DL_WriterA::dxfString(), DL_Writer::entity(), DL_Writer::entityAttributes(), DL_DimensionData::lineSpacingFac
DL_DimensionData::lineSpacingStyle, DL_DimensionData::mpx, DL_DimensionData::mpy, DL_DimensionData::text,
and DL_DimensionData::type.

#### 5.21.2.24 writeDimDiametric()

```
void DL_Dxf::writeDimDiametric (
            DL_WriterA & dw,
            const DL_DimensionData & data,
            const DL_DimDiametricData & edata,
            const DL_Attributes & attrib )
```

Writes a diametric dimension entity to the file.

**Parameters**

| | |
|---|---|
| *dw* | DXF writer |
| *data* | Generic dimension data for from the file |
| *data* | Specific diametric dimension data from the file |
| *attrib* | Attributes |

References DL_DimensionData::angle, DL_DimensionData::attachmentPoint, DL_DimensionData::dpx, DL_DimDiametricData::dpx,
DL_DimensionData::dpy, DL_DimDiametricData::dpy, DL_DimensionData::dpz, DL_WriterA::dxfInt(), DL_WriterA::dxfReal(),
DL_WriterA::dxfString(), DL_Writer::entity(), DL_Writer::entityAttributes(), DL_DimDiametricData::leader, DL_DimensionData::lineSpa
DL_DimensionData::lineSpacingStyle, DL_DimensionData::mpx, DL_DimensionData::mpy, DL_DimensionData::text,
and DL_DimensionData::type.

#### 5.21.2.25 writeDimLinear()

```
void DL_Dxf::writeDimLinear (
            DL_WriterA & dw,
            const DL_DimensionData & data,
            const DL_DimLinearData & edata,
            const DL_Attributes & attrib )
```

Writes a linear dimension entity to the file.

**Parameters**

| | |
|---|---|
| *dw* | DXF writer |
| *data* | Generic dimension data for from the file |
| *data* | Specific linear dimension data from the file |
| *attrib* | Attributes |

References DL_DimensionData::angle, DL_DimLinearData::angle, DL_DimensionData::attachmentPoint,
DL_DimensionData::dpx, DL_DimLinearData::dpx1, DL_DimLinearData::dpx2, DL_DimensionData::dpy, DL_DimLinearData::dpy1,
DL_DimLinearData::dpy2, DL_DimensionData::dpz, DL_WriterA::dxfInt(), DL_WriterA::dxfReal(), DL_WriterA::dxfString(),

DL_Writer::entity(), DL_Writer::entityAttributes(), DL_DimensionData::lineSpacingFactor, DL_DimensionData::lineSpacingStyle, DL_DimensionData::mpx, DL_DimensionData::mpy, DL_DimensionData::text, and DL_DimensionData::type.

#### 5.21.2.26 writeDimOrdinate()

```
void DL_Dxf::writeDimOrdinate (
            DL_WriterA & dw,
            const DL_DimensionData & data,
            const DL_DimOrdinateData & edata,
            const DL_Attributes & attrib )
```

Writes an ordinate dimension entity to the file.

**Parameters**

| | |
|---|---|
| *dw* | DXF writer |
| *data* | Generic dimension data for from the file |
| *data* | Specific ordinate dimension data from the file |
| *attrib* | Attributes |

References DL_DimensionData::attachmentPoint, DL_DimensionData::dpx, DL_DimOrdinateData::dpx1, DL_DimOrdinateData::dpx2, DL_DimensionData::dpy, DL_DimOrdinateData::dpy1, DL_DimOrdinateData::dpy2, DL_DimensionData::dpz, DL_WriterA::dxfInt(), DL_WriterA::dxfReal(), DL_WriterA::dxfString(), DL_Writer::entity(), DL_Writer::entityAttributes(), DL_DimensionData::lineSpacingFactor, DL_DimensionData::lineSpacingStyle, DL_DimensionData::mpx, DL_DimensionData::mpy, DL_DimensionData::text, DL_DimensionData::type, and DL_DimOrdinateData::xtype.

#### 5.21.2.27 writeDimRadial()

```
void DL_Dxf::writeDimRadial (
            DL_WriterA & dw,
            const DL_DimensionData & data,
            const DL_DimRadialData & edata,
            const DL_Attributes & attrib )
```

Writes a radial dimension entity to the file.

**Parameters**

| | |
|---|---|
| *dw* | DXF writer |
| *data* | Generic dimension data for from the file |
| *data* | Specific radial dimension data from the file |
| *attrib* | Attributes |

References DL_DimensionData::angle, DL_DimensionData::attachmentPoint, DL_DimensionData::dpx, DL_DimRadialData::dpx, DL_DimensionData::dpy, DL_DimRadialData::dpy, DL_DimensionData::dpz, DL_WriterA::dxfInt(), DL_WriterA::dxfReal(), DL_WriterA::dxfString(), DL_Writer::entity(), DL_Writer::entityAttributes(), DL_DimRadialData::leader, DL_DimensionData::lineSpacin

DL_DimensionData::lineSpacingStyle, DL_DimensionData::mpx, DL_DimensionData::mpy, DL_DimensionData::text, and DL_DimensionData::type.

#### 5.21.2.28 writeDimStyle()

```
void DL_Dxf::writeDimStyle (
            DL_WriterA & dw,
            double dimasz,
            double dimexe,
            double dimexo,
            double dimgap,
            double dimtxt )
```

Writes a dimstyle section.

This section is needed in DL_VERSION_R13. Note that this method currently only writes a faked DIMSTYLE section to make the file readable by Aut∗cad.

References DL_WriterA::dxfHex(), DL_WriterA::dxfInt(), DL_WriterA::dxfReal(), and DL_WriterA::dxfString().

#### 5.21.2.29 writeEllipse()

```
void DL_Dxf::writeEllipse (
            DL_WriterA & dw,
            const DL_EllipseData & data,
            const DL_Attributes & attrib )
```

Writes an ellipse entity to the file.

**Parameters**

| | |
|---|---|
| *dw* | DXF writer |
| *data* | Entity data from the file |
| *attrib* | Attributes |

References DL_EllipseData::angle1, DL_EllipseData::angle2, DL_EllipseData::cx, DL_EllipseData::cy, DL_EllipseData::cz, DL_WriterA::dxfReal(), DL_WriterA::dxfString(), DL_Writer::entity(), DL_Writer::entityAttributes(), DL_EllipseData::mx, DL_EllipseData::my, DL_EllipseData::mz, and DL_EllipseData::ratio.

#### 5.21.2.30 writeEndBlock()

```
void DL_Dxf::writeEndBlock (
            DL_WriterA & dw,
            const std::string & name )
```

Writes a block end.

**Parameters**

| | |
|---|---|
| *name* | Block name |

References DL_Writer::sectionBlockEntryEnd().

#### 5.21.2.31 writeFitPoint()

```
void DL_Dxf::writeFitPoint (
            DL_WriterA & dw,
            const DL_FitPointData & data )
```

Writes a single fit point of a spline to the file.

**Parameters**

| | |
|---|---|
| *dw* | DXF writer |
| *data* | Entity data from the file |
| *attrib* | Attributes |

References DL_WriterA::dxfReal(), DL_FitPointData::x, DL_FitPointData::y, and DL_FitPointData::z.

#### 5.21.2.32 writeHatch1()

```
void DL_Dxf::writeHatch1 (
            DL_WriterA & dw,
            const DL_HatchData & data,
            const DL_Attributes & attrib )
```

Writes the beginning of a hatch entity to the file.

This must be followed by one or more writeHatchLoop() calls and a writeHatch2() call.

**Parameters**

| | |
|---|---|
| *dw* | DXF writer |
| *data* | Entity data. |
| *attrib* | Attributes |

References DL_WriterA::dxfInt(), DL_WriterA::dxfReal(), DL_WriterA::dxfString(), DL_Writer::entity(), DL_Writer::entityAttributes(), DL_HatchData::numLoops, DL_HatchData::pattern, and DL_HatchData::solid.

#### 5.21.2.33 writeHatch2()

```
void DL_Dxf::writeHatch2 (
            DL_WriterA & dw,
            const DL_HatchData & data,
            const DL_Attributes & attrib )
```

Writes the end of a hatch entity to the file.

**Parameters**

| | |
|---|---|
| *dw* | DXF writer |
| *data* | Entity data. |
| *attrib* | Attributes |

References DL_HatchData::angle, DL_WriterA::dxfInt(), DL_WriterA::dxfReal(), DL_WriterA::dxfString(), DL_HatchData::originX, DL_HatchData::scale, and DL_HatchData::solid.

#### 5.21.2.34 writeHatchEdge()

```
void DL_Dxf::writeHatchEdge (
            DL_WriterA & dw,
            const DL_HatchEdgeData & data )
```

Writes the beginning of a hatch entity to the file.

**Parameters**

| | |
|---|---|
| *dw* | DXF writer |
| *data* | Entity data. |
| *attrib* | Attributes |

References DL_HatchEdgeData::angle1, DL_HatchEdgeData::angle2, DL_HatchEdgeData::ccw, DL_HatchEdgeData::cx, DL_HatchEdgeData::cy, DL_HatchEdgeData::degree, DL_Writer::dxfBool(), DL_WriterA::dxfInt(), DL_WriterA::dxfReal(), DL_HatchEdgeData::mx, DL_HatchEdgeData::my, DL_HatchEdgeData::nControl, DL_HatchEdgeData::nFit, DL_HatchEdgeData::nKnots, DL_HatchEdgeData::radius, DL_HatchEdgeData::ratio, DL_HatchEdgeData::type, DL_HatchEdgeData::x1, DL_HatchEdgeData::x2, DL_HatchEdgeData::y1, and DL_HatchEdgeData::y2.

#### 5.21.2.35 writeHatchLoop1()

```
void DL_Dxf::writeHatchLoop1 (
            DL_WriterA & dw,
            const DL_HatchLoopData & data )
```

Writes the beginning of a hatch loop to the file.

This must happen after writing the beginning of a hatch entity.

**Parameters**

| | |
|---|---|
| *dw* | DXF writer |
| *data* | Entity data. |
| *attrib* | Attributes |

References DL_WriterA::dxfInt(), and DL_HatchLoopData::numEdges.

#### 5.21.2.36 writeHatchLoop2()

```
void DL_Dxf::writeHatchLoop2 (
            DL_WriterA & dw,
            const DL_HatchLoopData & data )
```

Writes the end of a hatch loop to the file.

**Parameters**

| | |
|---|---|
| *dw* | DXF writer |
| *data* | Entity data. |
| *attrib* | Attributes |

References DL_WriterA::dxfInt().

#### 5.21.2.37 writeImage()

```
unsigned long DL_Dxf::writeImage (
            DL_WriterA & dw,
            const DL_ImageData & data,
            const DL_Attributes & attrib )
```

Writes an image entity.

**Returns**

IMAGEDEF handle. Needed for the IMAGEDEF counterpart.

References DL_ImageData::brightness, DL_ImageData::contrast, DL_WriterA::dxfInt(), DL_WriterA::dxfReal(), DL_WriterA::dxfString(), DL_Writer::entity(), DL_Writer::entityAttributes(), DL_ImageData::fade, DL_Writer::handle(), DL_ImageData::height, DL_ImageData::ipx, DL_ImageData::ipy, DL_ImageData::ipz, DL_ImageData::ux, DL_ImageData::uy, DL_ImageData::uz, DL_ImageData::vx, DL_ImageData::vy, DL_ImageData::vz, and DL_ImageData::width.

#### 5.21.2.38 writeInsert()

```
void DL_Dxf::writeInsert (
            DL_WriterA & dw,
            const DL_InsertData & data,
            const DL_Attributes & attrib )
```

Writes an insert to the file.

**Parameters**

| | |
|---|---|
| *dw* | DXF writer |
| *data* | Entity data from the file |
| *attrib* | Attributes |

References DL_InsertData::angle, DL_InsertData::cols, DL_InsertData::colSp, DL_WriterA::dxfInt(), DL_WriterA::dxfReal(), DL_WriterA::dxfString(), DL_Writer::entity(), DL_Writer::entityAttributes(), DL_InsertData::ipx, DL_InsertData::ipy, DL_InsertData::ipz, DL_InsertData::name, DL_InsertData::rows, DL_InsertData::rowSp, DL_InsertData::sx, and DL_InsertData::sy.

#### 5.21.2.39 writeKnot()

```
void DL_Dxf::writeKnot (
            DL_WriterA & dw,
            const DL_KnotData & data )
```

Writes a single knot of a spline to the file.

**Parameters**

| | |
|---|---|
| *dw* | DXF writer |
| *data* | Entity data from the file |
| *attrib* | Attributes |

References DL_WriterA::dxfReal(), and DL_KnotData::k.

#### 5.21.2.40 writeLayer()

```
void DL_Dxf::writeLayer (
            DL_WriterA & dw,
            const DL_LayerData & data,
            const DL_Attributes & attrib )
```

Writes a layer to the file.

Layers are stored in the tables section of a DXF file.

**Parameters**

| | |
|---|---|
| *dw* | DXF writer |
| *data* | Entity data from the file |
| *attrib* | Attributes |

References DL_WriterA::dxfHex(), DL_WriterA::dxfInt(), DL_WriterA::dxfString(), DL_LayerData::flags, DL_Attributes::getColor(),

DL_Attributes::getColor24(), DL_Attributes::getLinetype(), DL_Attributes::getWidth(), DL_LayerData::name, DL_LayerData::off, and DL_Writer::tableLayerEntry().

#### 5.21.2.41 writeLeader()

```
void DL_Dxf::writeLeader (
            DL_WriterA & dw,
            const DL_LeaderData & data,
            const DL_Attributes & attrib )
```

Writes a leader entity to the file.

**Parameters**

| | |
|---|---|
| *dw* | DXF writer |
| *data* | Entity data from the file |
| *attrib* | Attributes |

**See also**

writeVertex

References DL_LeaderData::arrowHeadFlag, DL_WriterA::dxfInt(), DL_WriterA::dxfReal(), DL_WriterA::dxfString(), DL_Writer::entity(), DL_Writer::entityAttributes(), DL_LeaderData::hooklineDirectionFlag, DL_LeaderData::hooklineFlag, DL_LeaderData::leaderCreationFlag, DL_LeaderData::leaderPathType, DL_LeaderData::number, DL_LeaderData::textAnnotationHei and DL_LeaderData::textAnnotationWidth.

#### 5.21.2.42 writeLeaderVertex()

```
void DL_Dxf::writeLeaderVertex (
            DL_WriterA & dw,
            const DL_LeaderVertexData & data )
```

Writes a single vertex of a leader to the file.

**Parameters**

| | |
|---|---|
| *dw* | DXF writer |
| *data* | Entity data |

References DL_WriterA::dxfReal(), DL_LeaderVertexData::x, and DL_LeaderVertexData::y.

#### 5.21.2.43 writeLine()

```
void DL_Dxf::writeLine (
```

```
            DL_WriterA & dw,
            const DL_LineData & data,
            const DL_Attributes & attrib )
```

Writes a line entity to the file.

**Parameters**

| | |
|---|---|
| *dw* | DXF writer |
| *data* | Entity data from the file |
| *attrib* | Attributes |

References DL_WriterA::dxfString(), DL_Writer::entity(), DL_Writer::entityAttributes(), DL_LineData::x1, DL_LineData::x2, DL_LineData::y1, DL_LineData::y2, DL_LineData::z1, and DL_LineData::z2.

#### 5.21.2.44 writeLinetype()

```
void DL_Dxf::writeLinetype (
            DL_WriterA & dw,
            const DL_LinetypeData & data )
```

Writes a line type to the file.

Line types are stored in the tables section of a DXF file.

References DL_LinetypeData::description, DL_WriterA::dxfInt(), DL_WriterA::dxfReal(), DL_WriterA::dxfString(), DL_LinetypeData::flags, DL_LinetypeData::name, DL_LinetypeData::numberOfDashes, DL_LinetypeData::pattern, DL_LinetypeData::patternLength, and DL_Writer::tableLinetypeEntry().

#### 5.21.2.45 writeMText()

```
void DL_Dxf::writeMText (
            DL_WriterA & dw,
            const DL_MTextData & data,
            const DL_Attributes & attrib )
```

Writes a multi text entity to the file.

**Parameters**

| | |
|---|---|
| *dw* | DXF writer |
| *data* | Entity data from the file |
| *attrib* | Attributes |

References DL_MTextData::angle, DL_MTextData::attachmentPoint, DL_MTextData::drawingDirection, DL_WriterA::dxfInt(), DL_WriterA::dxfReal(), DL_WriterA::dxfString(), DL_Writer::entity(), DL_Writer::entityAttributes(), DL_MTextData::height,

DL_MTextData::ipx, DL_MTextData::ipy, DL_MTextData::ipz, DL_MTextData::lineSpacingFactor, DL_MTextData::lineSpacingStyle, DL_MTextData::style, DL_MTextData::text, and DL_MTextData::width.

#### 5.21.2.46 writeObjects()

```
void DL_Dxf::writeObjects (
            DL_WriterA & dw,
            const std::string & appDictionaryName = "" )
```

Writes a objects section.

This section is needed in DL_VERSION_R13. Note that this method currently only writes a faked OBJECTS section to make the file readable by Aut∗cad.

References DL_WriterA::dxfHex(), DL_WriterA::dxfInt(), DL_WriterA::dxfReal(), DL_WriterA::dxfString(), DL_Writer::getNextHandle(), and DL_Writer::handle().

#### 5.21.2.47 writeObjectsEnd()

```
void DL_Dxf::writeObjectsEnd (
            DL_WriterA & dw )
```

Writes the end of the objects section.

This section is needed in DL_VERSION_R13. Note that this method currently only writes a faked OBJECTS section to make the file readable by Aut∗cad.

References DL_WriterA::dxfString().

#### 5.21.2.48 writePoint()

```
void DL_Dxf::writePoint (
            DL_WriterA & dw,
            const DL_PointData & data,
            const DL_Attributes & attrib )
```

Writes a point entity to the file.

**Parameters**

| | |
|---|---|
| *dw* | DXF writer |
| *data* | Entity data from the file |
| *attrib* | Attributes |

References DL_WriterA::dxfString(), DL_Writer::entity(), DL_Writer::entityAttributes(), DL_PointData::x, DL_PointData::y,

and DL_PointData::z.

#### 5.21.2.49 writePolyline()

```
void DL_Dxf::writePolyline (
            DL_WriterA & dw,
            const DL_PolylineData & data,
            const DL_Attributes & attrib )
```

Writes a polyline entity to the file.

**Parameters**

| | |
|---|---|
| *dw* | DXF writer |
| *data* | Entity data from the file |
| *attrib* | Attributes |

**See also**

writeVertex

References DL_WriterA::dxfInt(), DL_WriterA::dxfString(), DL_Writer::entity(), DL_Writer::entityAttributes(), DL_PolylineData::flags, DL_Attributes::getLayer(), and DL_PolylineData::number.

#### 5.21.2.50 writePolylineEnd()

```
void DL_Dxf::writePolylineEnd (
            DL_WriterA & dw )
```

Writes the polyline end.

Only needed for DXF R12.

References DL_Writer::entity().

#### 5.21.2.51 writeRay()

```
void DL_Dxf::writeRay (
            DL_WriterA & dw,
            const DL_RayData & data,
            const DL_Attributes & attrib )
```

Writes a ray entity to the file.

**Parameters**

| | |
|---|---|
| *dw* | DXF writer |
| *data* | Entity data from the file |
| *attrib* | Attributes |

References DL_RayData::bx, DL_RayData::by, DL_RayData::bz, DL_RayData::dx, DL_WriterA::dxfString(), DL_RayData::dy, DL_RayData::dz, DL_Writer::entity(), and DL_Writer::entityAttributes().

#### 5.21.2.52 writeSolid()

```
void DL_Dxf::writeSolid (
            DL_WriterA & dw,
            const DL_SolidData & data,
            const DL_Attributes & attrib )
```

Writes a solid entity to the file.

**Parameters**

| | |
|---|---|
| *dw* | DXF writer |
| *data* | Entity data from the file |
| *attrib* | Attributes |

References DL_WriterA::dxfReal(), DL_WriterA::dxfString(), DL_Writer::entity(), DL_Writer::entityAttributes(), DL_TraceData::thickness, and DL_TraceData::x.

#### 5.21.2.53 writeSpline()

```
void DL_Dxf::writeSpline (
            DL_WriterA & dw,
            const DL_SplineData & data,
            const DL_Attributes & attrib )
```

Writes a spline entity to the file.

**Parameters**

| | |
|---|---|
| *dw* | DXF writer |
| *data* | Entity data from the file |
| *attrib* | Attributes |

**See also**

writeControlPoint

References DL_SplineData::degree, DL_WriterA::dxfInt(), DL_WriterA::dxfString(), DL_Writer::entity(), DL_Writer::entityAttributes(), DL_SplineData::flags, DL_SplineData::nControl, DL_SplineData::nFit, and DL_SplineData::nKnots.

#### 5.21.2.54 writeStyle()

```
void DL_Dxf::writeStyle (
            DL_WriterA & dw,
            const DL_StyleData & style )
```

Writes a style section.

This section is needed in DL_VERSION_R13.

References DL_StyleData::bigFontFile, DL_WriterA::dxfInt(), DL_WriterA::dxfReal(), DL_WriterA::dxfString(), DL_StyleData::fixedTextHeight, DL_StyleData::flags, DL_Writer::handle(), DL_StyleData::lastHeightUsed, DL_StyleData::name, DL_StyleData::obliqueAngle, DL_StyleData::primaryFontFile, DL_StyleData::textGenerationFlags, and DL_StyleData::widthFactor.

#### 5.21.2.55 writeText()

```
void DL_Dxf::writeText (
            DL_WriterA & dw,
            const DL_TextData & data,
            const DL_Attributes & attrib )
```

Writes a text entity to the file.

**Parameters**

| | |
|---|---|
| *dw* | DXF writer |
| *data* | Entity data from the file |
| *attrib* | Attributes |

References DL_TextData::angle, DL_TextData::apx, DL_TextData::apy, DL_TextData::apz, DL_WriterA::dxfInt(), DL_WriterA::dxfReal(), DL_WriterA::dxfString(), DL_Writer::entity(), DL_Writer::entityAttributes(), DL_TextData::height, DL_TextData::hJustification, DL_TextData::ipx, DL_TextData::ipy, DL_TextData::ipz, DL_TextData::style, DL_TextData::text, DL_TextData::textGenerationFlags, DL_TextData::vJustification, and DL_TextData::xScaleFactor.

#### 5.21.2.56 writeTrace()

```
void DL_Dxf::writeTrace (
            DL_WriterA & dw,
            const DL_TraceData & data,
            const DL_Attributes & attrib )
```

Writes a trace entity to the file.

**Parameters**

| | |
|---|---|
| *dw* | DXF writer |
| *data* | Entity data from the file |
| *attrib* | Attributes |

References DL_WriterA::dxfReal(), DL_WriterA::dxfString(), DL_Writer::entity(), DL_Writer::entityAttributes(), DL_TraceData::thickness, and DL_TraceData::x.

#### 5.21.2.57 writeUcs()

```
void DL_Dxf::writeUcs (
            DL_WriterA & dw )
```

Writes a ucs section.

This section is needed in DL_VERSION_R13. Note that this method currently only writes a faked UCS section to make the file readable by Aut∗cad.

References DL_WriterA::dxfHex(), DL_WriterA::dxfInt(), and DL_WriterA::dxfString().

#### 5.21.2.58 writeVertex()

```
void DL_Dxf::writeVertex (
            DL_WriterA & dw,
            const DL_VertexData & data )
```

Writes a single vertex of a polyline to the file.

**Parameters**

| | |
|---|---|
| *dw* | DXF writer |
| *data* | Entity data from the file |
| *attrib* | Attributes |

References DL_VertexData::bulge, DL_WriterA::dxfReal(), DL_WriterA::dxfString(), DL_Writer::entity(), DL_VertexData::x, DL_VertexData::y, and DL_VertexData::z.

#### 5.21.2.59 writeView()

```
void DL_Dxf::writeView (
            DL_WriterA & dw )
```

Writes a view section.

This section is needed in DL_VERSION_R13. Note that this method currently only writes a faked VIEW section to make the file readable by Aut∗cad.

References DL_WriterA::dxfHex(), DL_WriterA::dxfInt(), and DL_WriterA::dxfString().

#### 5.21.2.60 writeVPort()

```
void DL_Dxf::writeVPort (
            DL_WriterA & dw )
```

Writes a viewport section.

This section is needed in DL_VERSION_R13. Note that this method currently only writes a faked VPORT section to make the file readable by Aut∗cad.

References DL_WriterA::dxfHex(), DL_WriterA::dxfInt(), DL_WriterA::dxfReal(), DL_WriterA::dxfString(), and DL_Writer::handle().

#### 5.21.2.61 writeXLine()

```
void DL_Dxf::writeXLine (
            DL_WriterA & dw,
            const DL_XLineData & data,
            const DL_Attributes & attrib )
```

Writes an x line entity to the file.

**Parameters**

| | |
|---|---|
| *dw* | DXF writer |
| *data* | Entity data from the file |
| *attrib* | Attributes |

References DL_XLineData::bx, DL_XLineData::by, DL_XLineData::bz, DL_XLineData::dx, DL_WriterA::dxfString(), DL_XLineData::dy, DL_XLineData::dz, DL_Writer::entity(), and DL_Writer::entityAttributes().

The documentation for this class was generated from the following files:

- src/dl_dxf.h
- src/dl_dxf.cpp

## 5.22 DL_EllipseData Struct Reference

Ellipse Data.

`#include <dl_entities.h>`

## Public Member Functions

- DL_EllipseData (double cx, double cy, double cz, double mx, double my, double mz, double ratio, double angle1, double angle2)

  *Constructor.*

## Public Attributes

- double cx
- double cy
- double cz
- double mx
- double my
- double mz
- double ratio
- double angle1
- double angle2

### 5.22.1 Detailed Description

Ellipse Data.

### 5.22.2 Constructor & Destructor Documentation

#### 5.22.2.1 DL_EllipseData()

```
DL_EllipseData::DL_EllipseData (
            double cx,
            double cy,
            double cz,
            double mx,
            double my,
            double mz,
            double ratio,
            double angle1,
            double angle2 )  [inline]
```

Constructor.

Parameters: see member variables.

### 5.22.3 Member Data Documentation

#### 5.22.3.1 angle1

```
double DL_EllipseData::angle1
```

Startangle of ellipse in rad.

Referenced by DL_Dxf::writeEllipse().

#### 5.22.3.2 angle2

```
double DL_EllipseData::angle2
```

Endangle of ellipse in rad.

Referenced by DL_Dxf::writeEllipse().

#### 5.22.3.3 cx

```
double DL_EllipseData::cx
```

X Coordinate of center point.

Referenced by DL_Dxf::writeEllipse().

#### 5.22.3.4 cy

```
double DL_EllipseData::cy
```

Y Coordinate of center point.

Referenced by DL_Dxf::writeEllipse().

#### 5.22.3.5 cz

```
double DL_EllipseData::cz
```

Z Coordinate of center point.

Referenced by DL_Dxf::writeEllipse().

#### 5.22.3.6 mx

`double DL_EllipseData::mx`

X coordinate of the endpoint of the major axis.

Referenced by DL_Dxf::writeEllipse().

#### 5.22.3.7 my

`double DL_EllipseData::my`

Y coordinate of the endpoint of the major axis.

Referenced by DL_Dxf::writeEllipse().

#### 5.22.3.8 mz

`double DL_EllipseData::mz`

Z coordinate of the endpoint of the major axis.

Referenced by DL_Dxf::writeEllipse().

#### 5.22.3.9 ratio

`double DL_EllipseData::ratio`

Ratio of minor axis to major axis..

Referenced by DL_Dxf::writeEllipse().

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

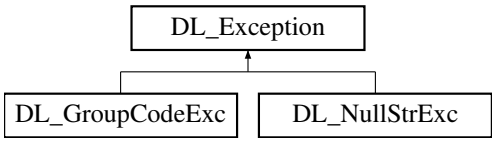
- src/dl_entities.h

## 5.23 DL_Exception Class Reference

Used for exception handling.

`#include <dl_exception.h>`

Inheritance diagram for DL_Exception:



### 5.23.1 Detailed Description

Used for exception handling.

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

- src/dl_exception.h

## 5.24 DL_Extrusion Class Reference

Extrusion direction.

`#include <dl_extrusion.h>`

### Public Member Functions

- **DL_Extrusion** ()
  *Default constructor.*
- ~**DL_Extrusion** ()
  *Destructor.*
- DL_Extrusion (double dx, double dy, double dz, double elevation)
  *Constructor for DXF extrusion.*
- void **setDirection** (double dx, double dy, double dz)
  *Sets the direction vector.*
- double * getDirection () const
- void getDirection (double dir[ ]) const
- void **setElevation** (double elevation)
  *Sets the elevation.*
- double getElevation () const
- DL_Extrusion **operator=** (const DL_Extrusion &extru)
  *Copies extrusion (deep copies) from another extrusion object.*

### 5.24.1 Detailed Description

Extrusion direction.

**Author**

Andrew Mustun

### 5.24.2 Constructor & Destructor Documentation

#### 5.24.2.1 DL_Extrusion()

```
DL_Extrusion::DL_Extrusion (
            double dx,
            double dy,
            double dz,
            double elevation )  [inline]
```

Constructor for DXF extrusion.

**Parameters**

| | |
|---|---|
| *direction* | Vector of axis along which the entity shall be extruded this is also the Z axis of the Entity coordinate system |
| *elevation* | Distance of the entities XY plane from the origin of the world coordinate system |

### 5.24.3 Member Function Documentation

#### 5.24.3.1 getDirection() [1/2]

```
double * DL_Extrusion::getDirection ( ) const  [inline]
```

**Returns**

direction vector.

#### 5.24.3.2 getDirection() [2/2]

```
void DL_Extrusion::getDirection (
            double dir[] ) const  [inline]
```

**Returns**

direction vector.

#### 5.24.3.3 getElevation()

```
double DL_Extrusion::getElevation ( ) const  [inline]
```

**Returns**

Elevation.

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

- src/dl_extrusion.h

## 5.25 DL_FitPointData Struct Reference

Spline fit point data.

`#include <dl_entities.h>`

### Public Member Functions

- DL_FitPointData (double x, double y, double z)
  *Constructor.*

### Public Attributes

- double x
- double y
- double z

### 5.25.1 Detailed Description

Spline fit point data.

### 5.25.2 Constructor & Destructor Documentation

#### 5.25.2.1 DL_FitPointData()

```
DL_FitPointData::DL_FitPointData (
            double x,
            double y,
            double z )  [inline]
```

Constructor.

Parameters: see member variables.

### 5.25.3 Member Data Documentation

#### 5.25.3.1 x

```
double DL_FitPointData::x
```

X coordinate of the fit point.

Referenced by DL_Dxf::writeFitPoint().

#### 5.25.3.2 y

```
double DL_FitPointData::y
```

Y coordinate of the fit point.

Referenced by DL_Dxf::writeFitPoint().

#### 5.25.3.3 z

```
double DL_FitPointData::z
```

Z coordinate of the fit point.

Referenced by DL_Dxf::writeFitPoint().

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

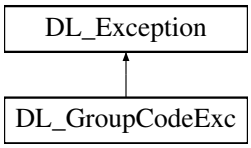
- src/dl_entities.h

## 5.26 DL_GroupCodeExc Class Reference

Used for exception handling.

```
#include <dl_exception.h>
```

Inheritance diagram for DL_GroupCodeExc:




### 5.26.1 Detailed Description

Used for exception handling.

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

- src/dl_exception.h

## 5.27 DL_HatchData Struct Reference

Hatch data.

```
#include <dl_entities.h>
```

## Public Member Functions

- **DL_HatchData** ()

  *Default constructor.*
- DL_HatchData (int numLoops, bool solid, double scale, double angle, const std::string &pattern, double originX=0.0, double originY=0.0)

  *Constructor.*

## Public Attributes

- int numLoops
- bool solid
- double scale
- double angle
- std::string pattern
- double originX
- double **originY**

### 5.27.1 Detailed Description

Hatch data.

### 5.27.2 Constructor & Destructor Documentation

#### 5.27.2.1 DL_HatchData()

```
DL_HatchData::DL_HatchData (
            int numLoops,
            bool solid,
            double scale,
            double angle,
            const std::string & pattern,
            double originX = 0.0,
            double originY = 0.0 )  [inline]
```

Constructor.

Parameters: see member variables.

### 5.27.3 Member Data Documentation

#### 5.27.3.1 angle

```
double DL_HatchData::angle
```

Pattern angle in degrees

Referenced by DL_Dxf::writeHatch2().

#### 5.27.3.2 numLoops

```
int DL_HatchData::numLoops
```

Number of boundary paths (loops).

Referenced by DL_Dxf::writeHatch1().

#### 5.27.3.3 originX

```
double DL_HatchData::originX
```

Pattern origin

Referenced by DL_Dxf::writeHatch2().

#### 5.27.3.4 pattern

```
std::string DL_HatchData::pattern
```

Pattern name.

Referenced by DL_Dxf::writeHatch1().

#### 5.27.3.5 scale

```
double DL_HatchData::scale
```

Pattern scale or spacing

Referenced by DL_Dxf::writeHatch2().

#### 5.27.3.6 solid

```
bool DL_HatchData::solid
```

Solid fill flag (true=solid, false=pattern).

Referenced by DL_Dxf::writeHatch1(), and DL_Dxf::writeHatch2().

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

- src/dl_entities.h

## 5.28 DL_HatchEdgeData Struct Reference

Hatch edge data.

```
#include <dl_entities.h>
```

### Public Member Functions

- **DL_HatchEdgeData** ()

  *Default constructor.*
- DL_HatchEdgeData (double x1, double y1, double x2, double y2)

  *Constructor for a line edge.*
- DL_HatchEdgeData (double cx, double cy, double radius, double angle1, double angle2, bool ccw)

  *Constructor for an arc edge.*
- DL_HatchEdgeData (double cx, double cy, double mx, double my, double ratio, double angle1, double angle2, bool ccw)

  *Constructor for an ellipse arc edge.*
- DL_HatchEdgeData (unsigned int degree, bool rational, bool periodic, unsigned int nKnots, unsigned int nControl, unsigned int nFit, const std::vector< double > &knots, const std::vector< std::vector< double > > &controlPoints, const std::vector< std::vector< double > > &fitPoints, const std::vector< double > &weights, double startTangentX, double startTangentY, double endTangentX, double endTangentY)

  *Constructor for a spline edge.*

### Public Attributes

- bool **defined**

  *Set to true if this edge is fully defined.*
- int type

  *Edge type.*
- double x1
- double y1
- double x2
- double y2
- double cx
- double cy
- double radius
- double angle1

- double angle2
- bool ccw
- double mx
- double my
- double ratio
- unsigned int degree
- bool **rational**
- bool **periodic**
- unsigned int nKnots
- unsigned int nControl
- unsigned int nFit
- std::vector< std::vector< double > > **controlPoints**
- std::vector< double > **knots**
- std::vector< double > **weights**
- std::vector< std::vector< double > > **fitPoints**
- double **startTangentX**
- double **startTangentY**
- double **endTangentX**
- double **endTangentY**
- std::vector< std::vector< double > > **vertices**

  *Polyline boundary vertices (x y [bulge])*

## 5.28.1 Detailed Description

Hatch edge data.

## 5.28.2 Constructor & Destructor Documentation

### 5.28.2.1 DL_HatchEdgeData() [1/4]

```
DL_HatchEdgeData::DL_HatchEdgeData (
            double x1,
            double y1,
            double x2,
            double y2 )  [inline]
```

Constructor for a line edge.

Parameters: see member variables.

### 5.28.2.2 DL_HatchEdgeData() [2/4]

```
DL_HatchEdgeData::DL_HatchEdgeData (
            double cx,
            double cy,
            double radius,
            double angle1,
            double angle2,
            bool ccw )  [inline]
```

Constructor for an arc edge.

Parameters: see member variables.

#### 5.28.2.3 DL_HatchEdgeData() [3/4]

```
DL_HatchEdgeData::DL_HatchEdgeData (
            double cx,
            double cy,
            double mx,
            double my,
            double ratio,
            double angle1,
            double angle2,
            bool ccw )  [inline]
```

Constructor for an ellipse arc edge.

Parameters: see member variables.

#### 5.28.2.4 DL_HatchEdgeData() [4/4]

```
DL_HatchEdgeData::DL_HatchEdgeData (
            unsigned int degree,
            bool rational,
            bool periodic,
            unsigned int nKnots,
            unsigned int nControl,
            unsigned int nFit,
            const std::vector< double > & knots,
            const std::vector< std::vector< double > > & controlPoints,
            const std::vector< std::vector< double > > & fitPoints,
            const std::vector< double > & weights,
            double startTangentX,
            double startTangentY,
            double endTangentX,
            double endTangentY )  [inline]
```

Constructor for a spline edge.

Parameters: see member variables.

### 5.28.3 Member Data Documentation

#### 5.28.3.1 angle1

```
double DL_HatchEdgeData::angle1
```

Start angle of arc or ellipse arc.

Referenced by DL_Dxf::handleHatchData(), and DL_Dxf::writeHatchEdge().

#### 5.28.3.2 angle2

```
double DL_HatchEdgeData::angle2
```

End angle of arc or ellipse arc.

Referenced by DL_Dxf::handleHatchData(), and DL_Dxf::writeHatchEdge().

#### 5.28.3.3 ccw

```
bool DL_HatchEdgeData::ccw
```

Counterclockwise flag for arc or ellipse arc.

Referenced by DL_Dxf::handleHatchData(), and DL_Dxf::writeHatchEdge().

#### 5.28.3.4 cx

```
double DL_HatchEdgeData::cx
```

Center point of arc or ellipse arc (X).

Referenced by DL_Dxf::handleHatchData(), and DL_Dxf::writeHatchEdge().

#### 5.28.3.5 cy

```
double DL_HatchEdgeData::cy
```

Center point of arc or ellipse arc (Y).

Referenced by DL_Dxf::handleHatchData(), and DL_Dxf::writeHatchEdge().

#### 5.28.3.6 degree

```
unsigned int DL_HatchEdgeData::degree
```

Spline degree

Referenced by DL_Dxf::handleHatchData(), and DL_Dxf::writeHatchEdge().

#### 5.28.3.7 mx

```
double DL_HatchEdgeData::mx
```

Major axis end point (X).

Referenced by DL_Dxf::handleHatchData(), and DL_Dxf::writeHatchEdge().

#### 5.28.3.8 my

```
double DL_HatchEdgeData::my
```

Major axis end point (Y).

Referenced by DL_Dxf::handleHatchData(), and DL_Dxf::writeHatchEdge().

#### 5.28.3.9 nControl

```
unsigned int DL_HatchEdgeData::nControl
```

Number of control points.

Referenced by DL_Dxf::handleHatchData(), and DL_Dxf::writeHatchEdge().

#### 5.28.3.10 nFit

```
unsigned int DL_HatchEdgeData::nFit
```

Number of fit points.

Referenced by DL_Dxf::handleHatchData(), and DL_Dxf::writeHatchEdge().

#### 5.28.3.11 nKnots

```
unsigned int DL_HatchEdgeData::nKnots
```

Number of knots.

Referenced by DL_Dxf::handleHatchData(), and DL_Dxf::writeHatchEdge().

#### 5.28.3.12 radius

```
double DL_HatchEdgeData::radius
```

Arc radius.

Referenced by DL_Dxf::handleHatchData(), and DL_Dxf::writeHatchEdge().

#### 5.28.3.13 ratio

```
double DL_HatchEdgeData::ratio
```

Axis ratio

Referenced by DL_Dxf::handleHatchData(), and DL_Dxf::writeHatchEdge().

#### 5.28.3.14 type

```
int DL_HatchEdgeData::type
```

Edge type.

1=line, 2=arc, 3=elliptic arc, 4=spline.

Referenced by DL_Dxf::handleHatchData(), and DL_Dxf::writeHatchEdge().

#### 5.28.3.15 x1

```
double DL_HatchEdgeData::x1
```

Start point (X).

Referenced by DL_Dxf::handleHatchData(), and DL_Dxf::writeHatchEdge().

#### 5.28.3.16 x2

```
double DL_HatchEdgeData::x2
```

End point (X).

Referenced by DL_Dxf::handleHatchData(), and DL_Dxf::writeHatchEdge().

#### 5.28.3.17 y1

```
double DL_HatchEdgeData::y1
```

Start point (Y).

Referenced by DL_Dxf::handleHatchData(), and DL_Dxf::writeHatchEdge().

#### 5.28.3.18 y2

```
double DL_HatchEdgeData::y2
```

End point (Y).

Referenced by DL_Dxf::handleHatchData(), and DL_Dxf::writeHatchEdge().

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

- src/dl_entities.h

## 5.29 DL_HatchLoopData Struct Reference

Hatch boundary path (loop) data.

```
#include <dl_entities.h>
```

### Public Member Functions

- **DL_HatchLoopData** ()
  *Default constructor.*
- DL_HatchLoopData (int hNumEdges)
  *Constructor.*

### Public Attributes

- int numEdges

### 5.29.1 Detailed Description

Hatch boundary path (loop) data.

### 5.29.2 Constructor & Destructor Documentation

#### 5.29.2.1 DL_HatchLoopData()

```
DL_HatchLoopData::DL_HatchLoopData (
            int hNumEdges ) [inline]
```

Constructor.

Parameters: see member variables.

### 5.29.3 Member Data Documentation

#### 5.29.3.1 numEdges

```
int DL_HatchLoopData::numEdges
```

Number of edges in this loop.

Referenced by DL_Dxf::writeHatchLoop1().

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

- src/dl_entities.h

## 5.30 DL_ImageData Struct Reference

Image Data.

```
#include <dl_entities.h>
```

### Public Member Functions

- DL_ImageData (const std::string &iref, double iipx, double iipy, double iipz, double iux, double iuy, double iuz, double ivx, double ivy, double ivz, int iwidth, int iheight, int ibrightness, int icontrast, int ifade)

  *Constructor.*

### Public Attributes

- std::string ref
- double ipx
- double ipy
- double ipz
- double ux
- double uy
- double uz
- double vx
- double vy
- double vz
- int width
- int height
- int brightness
- int contrast
- int fade

### 5.30.1 Detailed Description

Image Data.

### 5.30.2 Constructor & Destructor Documentation

#### 5.30.2.1 DL_ImageData()

```
DL_ImageData::DL_ImageData (
            const std::string & iref,
            double iipx,
            double iipy,
            double iipz,
            double iux,
            double iuy,
            double iuz,
            double ivx,
            double ivy,
            double ivz,
            int iwidth,
            int iheight,
            int ibrightness,
            int icontrast,
            int ifade )  [inline]
```

Constructor.

Parameters: see member variables.

### 5.30.3 Member Data Documentation

#### 5.30.3.1 brightness

```
int DL_ImageData::brightness
```

Brightness (0..100, default = 50).

Referenced by DL_Dxf::writeImage().

#### 5.30.3.2 contrast

```
int DL_ImageData::contrast
```

Contrast (0..100, default = 50).

Referenced by DL_Dxf::writeImage().

#### 5.30.3.3 fade

```
int DL_ImageData::fade
```

Fade (0..100, default = 0).

Referenced by DL_Dxf::writeImage().

#### 5.30.3.4 height

```
int DL_ImageData::height
```

Height of image in pixel.

Referenced by DL_Dxf::writeImage(), and DL_Dxf::writeImageDef().

#### 5.30.3.5 ipx

```
double DL_ImageData::ipx
```

X Coordinate of insertion point.

Referenced by DL_Dxf::writeImage().

#### 5.30.3.6 ipy

```
double DL_ImageData::ipy
```

Y Coordinate of insertion point.

Referenced by DL_Dxf::writeImage().

#### 5.30.3.7 ipz

```
double DL_ImageData::ipz
```

Z Coordinate of insertion point.

Referenced by DL_Dxf::writeImage().

#### 5.30.3.8 ref

```
std::string DL_ImageData::ref
```

Reference to the image file (unique, used to refer to the image def object).

Referenced by DL_Dxf::writeImageDef().

#### 5.30.3.9 ux

```
double DL_ImageData::ux
```

X Coordinate of u vector along bottom of image.

Referenced by DL_Dxf::writeImage().

#### 5.30.3.10 uy

```
double DL_ImageData::uy
```

Y Coordinate of u vector along bottom of image.

Referenced by DL_Dxf::writeImage().

#### 5.30.3.11 uz

```
double DL_ImageData::uz
```

Z Coordinate of u vector along bottom of image.

Referenced by DL_Dxf::writeImage().

#### 5.30.3.12 vx

```
double DL_ImageData::vx
```

X Coordinate of v vector along left side of image.

Referenced by DL_Dxf::writeImage().

#### 5.30.3.13 vy

```
double DL_ImageData::vy
```

Y Coordinate of v vector along left side of image.

Referenced by DL_Dxf::writeImage().

#### 5.30.3.14 vz

```
double DL_ImageData::vz
```

Z Coordinate of v vector along left side of image.

Referenced by DL_Dxf::writeImage().

#### 5.30.3.15 width

```
int DL_ImageData::width
```

Width of image in pixel.

Referenced by DL_Dxf::writeImage(), and DL_Dxf::writeImageDef().

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

- src/dl_entities.h

## 5.31 DL_ImageDefData Struct Reference

Image Definition Data.

```
#include <dl_entities.h>
```

## Public Member Functions

- DL_ImageDefData (const std::string &iref, const std::string &ifile)
  *Constructor.*

## Public Attributes

- std::string ref
- std::string file

### 5.31.1 Detailed Description

Image Definition Data.

### 5.31.2 Constructor & Destructor Documentation

#### 5.31.2.1 DL_ImageDefData()

```
DL_ImageDefData::DL_ImageDefData (
            const std::string & iref,
            const std::string & ifile )  [inline]
```

Constructor.

Parameters: see member variables.

### 5.31.3 Member Data Documentation

#### 5.31.3.1 file

```
std::string DL_ImageDefData::file
```

Image file

#### 5.31.3.2 ref

```
std::string DL_ImageDefData::ref
```

Reference to the image file (unique, used to refer to the image def object).

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

- src/dl_entities.h

## 5.32 DL_InsertData Struct Reference

Insert Data.

`#include <dl_entities.h>`

### Public Member Functions

- DL_InsertData (const std::string &name, double ipx, double ipy, double ipz, double sx, double sy, double sz, double angle, int cols, int rows, double colSp, double rowSp)

  *Constructor.*

### Public Attributes

- std::string name
- double ipx
- double ipy
- double ipz
- double sx
- double sy
- double sz
- double angle
- int cols
- int rows
- double colSp
- double rowSp

### 5.32.1 Detailed Description

Insert Data.

### 5.32.2 Constructor & Destructor Documentation

#### 5.32.2.1 DL_InsertData()

```
DL_InsertData::DL_InsertData (
            const std::string & name,
            double ipx,
            double ipy,
            double ipz,
            double sx,
            double sy,
            double sz,
            double angle,
            int cols,
            int rows,
            double colSp,
            double rowSp )  [inline]
```

Constructor.

Parameters: see member variables.

### 5.32.3 Member Data Documentation

#### 5.32.3.1 angle

```
double DL_InsertData::angle
```

Rotation angle in degrees.

Referenced by DL_Dxf::writeInsert().

#### 5.32.3.2 cols

```
int DL_InsertData::cols
```

Number of colums if we insert an array of the block or 1.

Referenced by DL_Dxf::writeInsert().

#### 5.32.3.3 colSp

```
double DL_InsertData::colSp
```

Values for the spacing between cols.

Referenced by DL_Dxf::writeInsert().

#### 5.32.3.4 ipx

```
double DL_InsertData::ipx
```

X Coordinate of insertion point.

Referenced by DL_Dxf::writeInsert().

#### 5.32.3.5 ipy

```
double DL_InsertData::ipy
```

Y Coordinate of insertion point.

Referenced by DL_Dxf::writeInsert().

#### 5.32.3.6 ipz

```
double DL_InsertData::ipz
```

Z Coordinate of insertion point.

Referenced by DL_Dxf::writeInsert().

#### 5.32.3.7 name

```
std::string DL_InsertData::name
```

Name of the referred block.

Referenced by DL_Dxf::writeInsert().

#### 5.32.3.8 rows

```
int DL_InsertData::rows
```

Number of rows if we insert an array of the block or 1.

Referenced by DL_Dxf::writeInsert().

#### 5.32.3.9 rowSp

```
double DL_InsertData::rowSp
```

Values for the spacing between rows.

Referenced by DL_Dxf::writeInsert().

#### 5.32.3.10 sx

```
double DL_InsertData::sx
```

X Scale factor.

Referenced by DL_Dxf::writeInsert().

#### 5.32.3.11 sy

```
double DL_InsertData::sy
```

Y Scale factor.

Referenced by DL_Dxf::writeInsert().

#### 5.32.3.12 sz

```
double DL_InsertData::sz
```

Z Scale factor.

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

- src/dl_entities.h

## 5.33 DL_KnotData Struct Reference

Spline knot data.

```
#include <dl_entities.h>
```

### Public Member Functions

- DL_KnotData (double pk)
  *Constructor.*

### Public Attributes

- double k

### 5.33.1 Detailed Description

Spline knot data.

### 5.33.2 Constructor & Destructor Documentation

#### 5.33.2.1 DL_KnotData()

```
DL_KnotData::DL_KnotData (
            double pk )  [inline]
```

Constructor.

Parameters: see member variables.

### 5.33.3 Member Data Documentation

#### 5.33.3.1 k

```
double DL_KnotData::k
```

Knot value.

Referenced by DL_Dxf::writeKnot().

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

- src/dl_entities.h

## 5.34 DL_LayerData Struct Reference

Layer Data.

```
#include <dl_entities.h>
```

### Public Member Functions

- DL_LayerData (const std::string &name, int flags, bool off=false)
  *Constructor.*

### Public Attributes

- std::string **name**
  *Layer name.*
- int flags
  *Layer flags.*
- bool **off**
  *Layer is off.*

### 5.34.1 Detailed Description

Layer Data.

### 5.34.2 Constructor & Destructor Documentation

#### 5.34.2.1 DL_LayerData()

```
DL_LayerData::DL_LayerData (
            const std::string & name,
            int flags,
            bool off = false )  [inline]
```

Constructor.

Parameters: see member variables.

### 5.34.3 Member Data Documentation

#### 5.34.3.1 flags

```
int DL_LayerData::flags
```

Layer flags.

(1 = frozen, 2 = frozen by default, 4 = locked)

Referenced by DL_Dxf::writeLayer().

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

- src/dl_entities.h

## 5.35 DL_LeaderData Struct Reference

Leader (arrow).

```
#include <dl_entities.h>
```

## Public Member Functions

- DL_LeaderData (int arrowHeadFlag, int leaderPathType, int leaderCreationFlag, int hooklineDirectionFlag, int hooklineFlag, double textAnnotationHeight, double textAnnotationWidth, int number, double dimScale=1.0)

  *Constructor.*

## Public Attributes

- int arrowHeadFlag
- int leaderPathType
- int leaderCreationFlag
- int hooklineDirectionFlag
- int hooklineFlag
- double textAnnotationHeight
- double textAnnotationWidth
- int number
- double dimScale

### 5.35.1 Detailed Description

Leader (arrow).

### 5.35.2 Constructor & Destructor Documentation

#### 5.35.2.1 DL_LeaderData()

```
DL_LeaderData::DL_LeaderData (
              int arrowHeadFlag,
              int leaderPathType,
              int leaderCreationFlag,
              int hooklineDirectionFlag,
              int hooklineFlag,
              double textAnnotationHeight,
              double textAnnotationWidth,
              int number,
              double dimScale = 1.0 )  [inline]
```

Constructor.

Parameters: see member variables.

### 5.35.3 Member Data Documentation

#### 5.35.3.1 arrowHeadFlag

```
int DL_LeaderData::arrowHeadFlag
```

Arrow head flag (71).

Referenced by DL_Dxf::writeLeader().

#### 5.35.3.2 dimScale

```
double DL_LeaderData::dimScale
```

Dimension scale (dimscale) style override.

#### 5.35.3.3 hooklineDirectionFlag

```
int DL_LeaderData::hooklineDirectionFlag
```

Hookline direction flag (74).

Referenced by DL_Dxf::writeLeader().

#### 5.35.3.4 hooklineFlag

```
int DL_LeaderData::hooklineFlag
```

Hookline flag (75)

Referenced by DL_Dxf::writeLeader().

#### 5.35.3.5 leaderCreationFlag

```
int DL_LeaderData::leaderCreationFlag
```

Leader creation flag (73).

Referenced by DL_Dxf::writeLeader().

#### 5.35.3.6 leaderPathType

```
int DL_LeaderData::leaderPathType
```

Leader path type (72).

Referenced by DL_Dxf::writeLeader().

#### 5.35.3.7 number

```
int DL_LeaderData::number
```

Number of vertices in leader (76).

Referenced by DL_Dxf::writeLeader().

#### 5.35.3.8 textAnnotationHeight

```
double DL_LeaderData::textAnnotationHeight
```

Text annotation height (40).

Referenced by DL_Dxf::writeLeader().

#### 5.35.3.9 textAnnotationWidth

```
double DL_LeaderData::textAnnotationWidth
```

Text annotation width (41)

Referenced by DL_Dxf::writeLeader().

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

- src/dl_entities.h

## 5.36 DL_LeaderVertexData Struct Reference

Leader Vertex Data.

```
#include <dl_entities.h>
```

## Public Member Functions

- DL_LeaderVertexData (double px=0.0, double py=0.0, double pz=0.0)
  *Constructor.*

## Public Attributes

- double x
- double y
- double z

### 5.36.1 Detailed Description

Leader Vertex Data.

### 5.36.2 Constructor & Destructor Documentation

#### 5.36.2.1 DL_LeaderVertexData()

```
DL_LeaderVertexData::DL_LeaderVertexData (
            double px = 0.0,
            double py = 0.0,
            double pz = 0.0 )  [inline]
```

Constructor.

Parameters: see member variables.

### 5.36.3 Member Data Documentation

#### 5.36.3.1 x

```
double DL_LeaderVertexData::x
```

X Coordinate of the vertex.

Referenced by DL_Dxf::writeLeaderVertex().

#### 5.36.3.2 y

```
double DL_LeaderVertexData::y
```

Y Coordinate of the vertex.

Referenced by DL_Dxf::writeLeaderVertex().

#### 5.36.3.3 z

```
double DL_LeaderVertexData::z
```

Z Coordinate of the vertex.

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

- src/dl_entities.h

## 5.37 DL_LineData Struct Reference

Line Data.

```
#include <dl_entities.h>
```

### Public Member Functions

- DL_LineData (double lx1, double ly1, double lz1, double lx2, double ly2, double lz2)
  *Constructor.*

### Public Attributes

- double x1
- double y1
- double z1
- double x2
- double y2
- double z2

### 5.37.1 Detailed Description

Line Data.

### 5.37.2 Constructor & Destructor Documentation

#### 5.37.2.1 DL_LineData()

```
DL_LineData::DL_LineData (
              double lx1,
              double ly1,
              double lz1,
              double lx2,
              double ly2,
              double lz2 )  [inline]
```

Constructor.

Parameters: see member variables.

### 5.37.3 Member Data Documentation

#### 5.37.3.1 x1

```
double DL_LineData::x1
```

X Start coordinate of the point.

Referenced by DL_Dxf::writeLine().

#### 5.37.3.2 x2

```
double DL_LineData::x2
```

X End coordinate of the point.

Referenced by DL_Dxf::writeLine().

#### 5.37.3.3 y1

```
double DL_LineData::y1
```

Y Start coordinate of the point.

Referenced by DL_Dxf::writeLine().

#### 5.37.3.4 y2

```
double DL_LineData::y2
```

Y End coordinate of the point.

Referenced by DL_Dxf::writeLine().

#### 5.37.3.5 z1

```
double DL_LineData::z1
```

Z Start coordinate of the point.

Referenced by DL_Dxf::writeLine().

#### 5.37.3.6 z2

```
double DL_LineData::z2
```

Z End coordinate of the point.

Referenced by DL_Dxf::writeLine().

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

- src/dl_entities.h

## 5.38 DL_LinetypeData Struct Reference

Line Type Data.

```
#include <dl_entities.h>
```

### Public Member Functions

- DL_LinetypeData (const std::string &name, const std::string &description, int flags, int numberOfDashes, double patternLength, double *pattern=NULL)

  *Constructor.*

### Public Attributes

- std::string **name**

  *Linetype name.*
- std::string **description**

  *Linetype description.*
- int **flags**

  *Linetype flags.*
- int **numberOfDashes**

  *Number of dashes.*
- double **patternLength**

  *Pattern length.*
- double * **pattern**

  *Pattern.*

## 5.38.1 Detailed Description

Line Type Data.

## 5.38.2 Constructor & Destructor Documentation

#### 5.38.2.1 DL_LinetypeData()

```
DL_LinetypeData::DL_LinetypeData (
            const std::string & name,
            const std::string & description,
            int flags,
            int numberOfDashes,
            double patternLength,
            double * pattern = NULL )  [inline]
```

Constructor.

Parameters: see member variables.

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

- src/dl_entities.h

# 5.39 DL_MTextData Struct Reference

MText Data.

```
#include <dl_entities.h>
```

## Public Member Functions

- DL_MTextData (double ipx, double ipy, double ipz, double dirx, double diry, double dirz, double height, double width, int attachmentPoint, int drawingDirection, int lineSpacingStyle, double lineSpacingFactor, const std↩ ::string &text, const std::string &style, double angle)

  *Constructor.*

## Public Attributes

- double ipx
- double ipy
- double ipz
- double dirx
- double diry
- double dirz
- double height
- double width
- int attachmentPoint

  *Attachment point.*
- int drawingDirection

  *Drawing direction.*
- int lineSpacingStyle

  *Line spacing style.*
- double lineSpacingFactor

  *Line spacing factor.*
- std::string text
- std::string style
- double angle

### 5.39.1 Detailed Description

MText Data.

### 5.39.2 Constructor & Destructor Documentation

#### 5.39.2.1 DL_MTextData()

```
DL_MTextData::DL_MTextData (
            double ipx,
            double ipy,
            double ipz,
            double dirx,
            double diry,
            double dirz,
            double height,
            double width,
            int attachmentPoint,
```

```
            int drawingDirection,
            int lineSpacingStyle,
            double lineSpacingFactor,
            const std::string & text,
            const std::string & style,
            double angle )  [inline]
```

Constructor.

Parameters: see member variables.

## 5.39.3 Member Data Documentation

### 5.39.3.1 angle

```
double DL_MTextData::angle
```

Rotation angle.

Referenced by DL_Dxf::writeMText().

### 5.39.3.2 attachmentPoint

```
int DL_MTextData::attachmentPoint
```

Attachment point.

1 = Top left, 2 = Top center, 3 = Top right, 4 = Middle left, 5 = Middle center, 6 = Middle right, 7 = Bottom left, 8 = Bottom center, 9 = Bottom right

Referenced by DL_Dxf::writeMText().

### 5.39.3.3 dirx

```
double DL_MTextData::dirx
```

X Coordinate of X direction vector.

### 5.39.3.4 diry

```
double DL_MTextData::diry
```

Y Coordinate of X direction vector.

#### 5.39.3.5 dirz

```
double DL_MTextData::dirz
```

Z Coordinate of X direction vector.

#### 5.39.3.6 drawingDirection

```
int DL_MTextData::drawingDirection
```

Drawing direction.

1 = left to right, 3 = top to bottom, 5 = by style

Referenced by DL_Dxf::writeMText().

#### 5.39.3.7 height

```
double DL_MTextData::height
```

Text height

Referenced by DL_Dxf::writeMText().

#### 5.39.3.8 ipx

```
double DL_MTextData::ipx
```

X Coordinate of insertion point.

Referenced by DL_Dxf::writeMText().

#### 5.39.3.9 ipy

```
double DL_MTextData::ipy
```

Y Coordinate of insertion point.

Referenced by DL_Dxf::writeMText().

#### 5.39.3.10 ipz

```
double DL_MTextData::ipz
```

Z Coordinate of insertion point.

Referenced by DL_Dxf::writeMText().

#### 5.39.3.11 lineSpacingFactor

```
double DL_MTextData::lineSpacingFactor
```

Line spacing factor.

0.25 .. 4.0

Referenced by DL_Dxf::writeMText().

#### 5.39.3.12 lineSpacingStyle

```
int DL_MTextData::lineSpacingStyle
```

Line spacing style.

1 = at least, 2 = exact

Referenced by DL_Dxf::writeMText().

#### 5.39.3.13 style

```
std::string DL_MTextData::style
```

Style string.

Referenced by DL_Dxf::writeMText().

#### 5.39.3.14 text

```
std::string DL_MTextData::text
```

Text string.

Referenced by DL_Dxf::writeMText().

#### 5.39.3.15 width

```
double DL_MTextData::width
```

Width of the text box.

Referenced by DL_Dxf::writeMText().

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

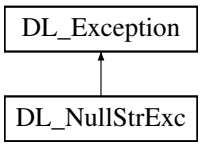
- src/dl_entities.h

## 5.40 DL_NullStrExc Class Reference

Used for exception handling.

```
#include <dl_exception.h>
```

Inheritance diagram for DL_NullStrExc:




### 5.40.1 Detailed Description

Used for exception handling.

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

- src/dl_exception.h

## 5.41 DL_PointData Struct Reference

Point Data.

```
#include <dl_entities.h>
```

### Public Member Functions

- DL_PointData (double px=0.0, double py=0.0, double pz=0.0)
  *Constructor.*

## Public Attributes

- double x
- double y
- double z

### 5.41.1 Detailed Description

Point Data.

### 5.41.2 Constructor & Destructor Documentation

#### 5.41.2.1 DL_PointData()

```
DL_PointData::DL_PointData (
            double px = 0.0,
            double py = 0.0,
            double pz = 0.0 )  [inline]
```

Constructor.

Parameters: see member variables.

### 5.41.3 Member Data Documentation

#### 5.41.3.1 x

```
double DL_PointData::x
```

X Coordinate of the point.

Referenced by DL_Dxf::writePoint().

#### 5.41.3.2 y

```
double DL_PointData::y
```

Y Coordinate of the point.

Referenced by DL_Dxf::writePoint().

#### 5.41.3.3 z

```
double DL_PointData::z
```

Z Coordinate of the point.

Referenced by DL_Dxf::writePoint().

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

- src/dl_entities.h

## 5.42 DL_PolylineData Struct Reference

Polyline Data.

```
#include <dl_entities.h>
```

### Public Member Functions

- DL_PolylineData (int pNumber, int pMVerteces, int pNVerteces, int pFlags, double pElevation=0.0)
  *Constructor.*

### Public Attributes

- unsigned int number
- unsigned int m
- unsigned int n
- double elevation
- int flags

### 5.42.1 Detailed Description

Polyline Data.

### 5.42.2 Constructor & Destructor Documentation

#### 5.42.2.1 DL_PolylineData()

```
DL_PolylineData::DL_PolylineData (
            int pNumber,
            int pMVerteces,
            int pNVerteces,
            int pFlags,
            double pElevation = 0.0 )  [inline]
```

Constructor.

Parameters: see member variables.

### 5.42.3 Member Data Documentation

#### 5.42.3.1 elevation

```
double DL_PolylineData::elevation
```

elevation of the polyline.

#### 5.42.3.2 flags

```
int DL_PolylineData::flags
```

Flags

Referenced by DL_Dxf::writePolyline().

#### 5.42.3.3 m

```
unsigned int DL_PolylineData::m
```

Number of vertices in m direction if polyline is a polygon mesh.

#### 5.42.3.4 n

```
unsigned int DL_PolylineData::n
```

Number of vertices in n direction if polyline is a polygon mesh.

#### 5.42.3.5 number

```
unsigned int DL_PolylineData::number
```

Number of vertices in this polyline.

Referenced by DL_Dxf::writePolyline().

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

- src/dl_entities.h

## 5.43 DL_RayData Struct Reference

Ray Data.

`#include <dl_entities.h>`

### Public Member Functions

- DL_RayData (double bx, double by, double bz, double dx, double dy, double dz)
  *Constructor.*

### Public Attributes

- double bx
- double by
- double bz
- double dx
- double dy
- double dz

### 5.43.1 Detailed Description

Ray Data.

### 5.43.2 Constructor & Destructor Documentation

#### 5.43.2.1 DL_RayData()

```
DL_RayData::DL_RayData (
            double bx,
            double by,
            double bz,
            double dx,
            double dy,
            double dz )  [inline]
```

Constructor.

Parameters: see member variables.

### 5.43.3 Member Data Documentation

#### 5.43.3.1 bx

```
double DL_RayData::bx
```

X base point.

Referenced by DL_Dxf::writeRay().

#### 5.43.3.2 by

```
double DL_RayData::by
```

Y base point.

Referenced by DL_Dxf::writeRay().

#### 5.43.3.3 bz

```
double DL_RayData::bz
```

Z base point.

Referenced by DL_Dxf::writeRay().

#### 5.43.3.4 dx

```
double DL_RayData::dx
```

X direction vector.

Referenced by DL_Dxf::writeRay().

#### 5.43.3.5 dy

```
double DL_RayData::dy
```

Y direction vector.

Referenced by DL_Dxf::writeRay().

#### 5.43.3.6 dz

```
double DL_RayData::dz
```

Z direction vector.

Referenced by DL_Dxf::writeRay().

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

- src/dl_entities.h

## 5.44 DL_SplineData Struct Reference

Spline Data.

```
#include <dl_entities.h>
```

### Public Member Functions

- DL_SplineData (int degree, int nKnots, int nControl, int nFit, int flags)
  *Constructor.*

### Public Attributes

- unsigned int degree
- unsigned int nKnots
- unsigned int nControl
- unsigned int nFit
- int flags
- double **tangentStartX**
- double **tangentStartY**
- double **tangentStartZ**
- double **tangentEndX**
- double **tangentEndY**
- double **tangentEndZ**

### 5.44.1 Detailed Description

Spline Data.

### 5.44.2 Constructor & Destructor Documentation

#### 5.44.2.1 DL_SplineData()

```
DL_SplineData::DL_SplineData (
            int degree,
            int nKnots,
            int nControl,
            int nFit,
            int flags )  [inline]
```

Constructor.

Parameters: see member variables.

### 5.44.3 Member Data Documentation

#### 5.44.3.1 degree

```
unsigned int DL_SplineData::degree
```

Degree of the spline curve.

Referenced by DL_Dxf::writeSpline().

#### 5.44.3.2 flags

```
int DL_SplineData::flags
```

Flags

Referenced by DL_Dxf::writeSpline().

#### 5.44.3.3 nControl

```
unsigned int DL_SplineData::nControl
```

Number of control points.

Referenced by DL_Dxf::writeSpline().

#### 5.44.3.4 nFit

```
unsigned int DL_SplineData::nFit
```

Number of fit points.

Referenced by DL_Dxf::writeSpline().

#### 5.44.3.5 nKnots

```
unsigned int DL_SplineData::nKnots
```

Number of knots.

Referenced by DL_Dxf::writeSpline().

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

- src/dl_entities.h

## 5.45 DL_StyleData Struct Reference

Text style data.

```
#include <dl_entities.h>
```

### Public Member Functions

- **DL_StyleData** (const std::string &name, int flags, double fixedTextHeight, double widthFactor, double obliqueAngle, int textGenerationFlags, double lastHeightUsed, const std::string &primaryFontFile, const std::string &bigFontFile)

  *Constructor Parameters: see member variables.*
- bool **operator==** (const DL_StyleData &other)

### Public Attributes

- std::string **name**

  *Style name.*
- int **flags**

  *Style flags.*
- double **fixedTextHeight**

  *Fixed text height or 0 for not fixed.*
- double **widthFactor**

  *Width factor.*
- double **obliqueAngle**

  *Oblique angle.*
- int **textGenerationFlags**

  *Text generation flags.*
- double **lastHeightUsed**

  *Last height used.*
- std::string **primaryFontFile**

  *Primary font file name.*
- std::string **bigFontFile**

  *Big font file name.*
- bool **bold**
- bool **italic**

### 5.45.1 Detailed Description

Text style data.

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

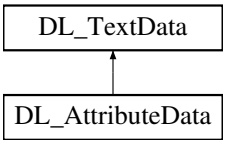
- src/dl_entities.h

## 5.46 DL_TextData Struct Reference

Text Data.

`#include <dl_entities.h>`

Inheritance diagram for DL_TextData:




### Public Member Functions

- DL_TextData (double ipx, double ipy, double ipz, double apx, double apy, double apz, double height, double xScaleFactor, int textGenerationFlags, int hJustification, int vJustification, const std::string &text, const std↩ ::string &style, double angle)

  *Constructor.*

### Public Attributes

- double ipx
- double ipy
- double ipz
- double apx
- double apy
- double apz
- double height
- double xScaleFactor
- int textGenerationFlags
- int hJustification

  *Horizontal justification.*
- int vJustification

  *Vertical justification.*
- std::string text
- std::string style
- double angle

### 5.46.1 Detailed Description

Text Data.

### 5.46.2 Constructor & Destructor Documentation

#### 5.46.2.1 DL_TextData()

```
DL_TextData::DL_TextData (
            double ipx,
            double ipy,
            double ipz,
            double apx,
            double apy,
            double apz,
            double height,
            double xScaleFactor,
            int textGenerationFlags,
            int hJustification,
            int vJustification,
            const std::string & text,
            const std::string & style,
            double angle )  [inline]
```

Constructor.

Parameters: see member variables.

### 5.46.3 Member Data Documentation

#### 5.46.3.1 angle

```
double DL_TextData::angle
```

Rotation angle of dimension text away from default orientation.

Referenced by DL_Dxf::writeText().

#### 5.46.3.2 apx

```
double DL_TextData::apx
```

X Coordinate of alignment point.

Referenced by DL_Dxf::writeText().

#### 5.46.3.3 apy

```
double DL_TextData::apy
```

Y Coordinate of alignment point.

Referenced by DL_Dxf::writeText().

#### 5.46.3.4 apz

```
double DL_TextData::apz
```

Z Coordinate of alignment point.

Referenced by DL_Dxf::writeText().

#### 5.46.3.5 height

```
double DL_TextData::height
```

Text height

Referenced by DL_Dxf::writeText().

#### 5.46.3.6 hJustification

```
int DL_TextData::hJustification
```

Horizontal justification.

0 = Left (default), 1 = Center, 2 = Right, 3 = Aligned, 4 = Middle, 5 = Fit For 3, 4, 5 the vertical alignment has to be 0.

Referenced by DL_Dxf::writeText().

#### 5.46.3.7 ipx

```
double DL_TextData::ipx
```

X Coordinate of insertion point.

Referenced by DL_Dxf::writeText().

#### 5.46.3.8 ipy

```
double DL_TextData::ipy
```

Y Coordinate of insertion point.

Referenced by DL_Dxf::writeText().

#### 5.46.3.9 ipz

```
double DL_TextData::ipz
```

Z Coordinate of insertion point.

Referenced by DL_Dxf::writeText().

#### 5.46.3.10 style

```
std::string DL_TextData::style
```

Style (font).

Referenced by DL_Dxf::writeText().

#### 5.46.3.11 text

```
std::string DL_TextData::text
```

Text string.

Referenced by DL_Dxf::writeText().

#### 5.46.3.12 textGenerationFlags

```
int DL_TextData::textGenerationFlags
```

0 = default, 2 = Backwards, 4 = Upside down

Referenced by DL_Dxf::writeText().

#### 5.46.3.13 vJustification

```
int DL_TextData::vJustification
```

Vertical justification.

0 = Baseline (default), 1 = Bottom, 2 = Middle, 3= Top

Referenced by DL_Dxf::writeText().

#### 5.46.3.14 xScaleFactor

```
double DL_TextData::xScaleFactor
```

Relative X scale factor.

Referenced by DL_Dxf::writeText().

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

- src/dl_entities.h

## 5.47 DL_TraceData Struct Reference

Trace Data / solid data / 3d face data.

```
#include <dl_entities.h>
```

### Public Member Functions

- DL_TraceData (double sx1, double sy1, double sz1, double sx2, double sy2, double sz2, double sx3, double sy3, double sz3, double sx4, double sy4, double sz4, double sthickness=0.0)

  *Constructor.*

### Public Attributes

- double thickness
- double x [4]
- double **y** [4]
- double **z** [4]

### 5.47.1 Detailed Description

Trace Data / solid data / 3d face data.

## 5.47.2 Constructor & Destructor Documentation

### 5.47.2.1 DL_TraceData()

```
DL_TraceData::DL_TraceData (
            double sx1,
            double sy1,
            double sz1,
            double sx2,
            double sy2,
            double sz2,
            double sx3,
            double sy3,
            double sz3,
            double sx4,
            double sy4,
            double sz4,
            double sthickness = 0.0 )  [inline]
```

Constructor.

Parameters: see member variables.

## 5.47.3 Member Data Documentation

### 5.47.3.1 thickness

```
double DL_TraceData::thickness
```

Thickness

Referenced by DL_Dxf::writeSolid(), and DL_Dxf::writeTrace().

### 5.47.3.2 x

```
double DL_TraceData::x[4]
```

Points

Referenced by DL_Dxf::add3dFace(), DL_Dxf::addSolid(), DL_Dxf::addTrace(), DL_Dxf::write3dFace(), DL_Dxf::writeSolid(), and DL_Dxf::writeTrace().

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

- src/dl_entities.h

## 5.48 DL_VertexData Struct Reference

Vertex Data.

`#include <dl_entities.h>`

### Public Member Functions

- DL_VertexData (double px=0.0, double py=0.0, double pz=0.0, double pBulge=0.0)
  *Constructor.*

### Public Attributes

- double x
- double y
- double z
- double bulge

### 5.48.1 Detailed Description

Vertex Data.

### 5.48.2 Constructor & Destructor Documentation

#### 5.48.2.1 DL_VertexData()

```
DL_VertexData::DL_VertexData (
            double px = 0.0,
            double py = 0.0,
            double pz = 0.0,
            double pBulge = 0.0 )  [inline]
```

Constructor.

Parameters: see member variables.

### 5.48.3 Member Data Documentation

#### 5.48.3.1 bulge

```
double DL_VertexData::bulge
```

Bulge of vertex. (The tangent of 1/4 of the arc angle or 0 for lines)

Referenced by DL_Dxf::writeVertex().

#### 5.48.3.2 x

```
double DL_VertexData::x
```

X Coordinate of the vertex.

Referenced by DL_Dxf::writeVertex().

#### 5.48.3.3 y

```
double DL_VertexData::y
```

Y Coordinate of the vertex.

Referenced by DL_Dxf::writeVertex().

#### 5.48.3.4 z

```
double DL_VertexData::z
```

Z Coordinate of the vertex.

Referenced by DL_Dxf::writeVertex().

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

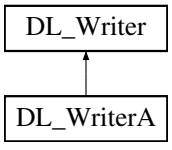
- src/dl_entities.h

## 5.49 DL_Writer Class Reference

Defines interface for writing low level DXF constructs to a file.

```
#include <dl_writer.h>
```

Inheritance diagram for DL_Writer:



## Public Member Functions

- DL_Writer (DL_Codes::version version)
- void section (const char *name) const

  *Generic section for section 'name'.*
- void sectionHeader () const

  *Section HEADER.*
- void sectionTables () const

  *Section TABLES.*
- void sectionBlocks () const

  *Section BLOCKS.*
- void sectionEntities () const

  *Section ENTITIES.*
- void sectionClasses () const

  *Section CLASSES.*
- void sectionObjects () const

  *Section OBJECTS.*
- void sectionEnd () const

  *End of a section.*
- void table (const char *name, int num, int h=0) const

  *Generic table for table 'name' with 'num' entries:*
- void tableLayers (int num) const

  *Table for layers.*
- void tableLinetypes (int num) const

  *Table for line types.*
- void tableAppid (int num) const

  *Table for application id.*
- void tableStyle (int num) const

  *Table for text style.*
- void tableEnd () const

  *End of a table.*
- void dxfEOF () const

  *End of the DXF file.*
- void comment (const char *text) const

  *Comment.*
- void entity (const char *entTypeName) const

  *Entity.*
- void entityAttributes (const DL_Attributes &attrib) const

  *Attributes of an entity.*
- void **subClass** (const char *sub) const

  *Subclass.*
- void tableLayerEntry (unsigned long int h=0) const

  *Layer (must be in the TABLES section LAYER).*
- void tableLinetypeEntry (unsigned long int h=0) const

  *Line type (must be in the TABLES section LTYPE).*
- void tableAppidEntry (unsigned long int h=0) const

  *Appid (must be in the TABLES section APPID).*
- void sectionBlockEntry (unsigned long int h=0) const

  *Block (must be in the section BLOCKS).*
- void sectionBlockEntryEnd (unsigned long int h=0) const

  *End of Block (must be in the section BLOCKS).*

- void **color** (int col=256) const
- void **linetype** (const char *lt) const
- void **linetypeScale** (double scale) const
- void **lineWeight** (int lw) const
- void **coord** (int gc, double x, double y, double z=0) const
- void **coordTriplet** (int gc, const double *value) const
- void **resetHandle** () const
- unsigned long **handle** (int gc=5) const

  *Writes a unique handle and returns it.*
- unsigned long getNextHandle () const
- virtual void dxfReal (int gc, double value) const =0

  *Must be overwritten by the implementing class to write a real value to the file.*
- virtual void dxfInt (int gc, int value) const =0

  *Must be overwritten by the implementing class to write an int value to the file.*
- virtual void dxfBool (int gc, bool value) const

  *Can be overwritten by the implementing class to write a bool value to the file.*
- virtual void dxfHex (int gc, int value) const =0

  *Must be overwritten by the implementing class to write an int value (hex) to the file.*
- virtual void dxfString (int gc, const char *value) const =0

  *Must be overwritten by the implementing class to write a string to the file.*
- virtual void dxfString (int gc, const std::string &value) const =0

  *Must be overwritten by the implementing class to write a string to the file.*

## Protected Attributes

- unsigned long **m_handle**
- unsigned long **modelSpaceHandle**
- unsigned long **paperSpaceHandle**
- unsigned long **paperSpace0Handle**
- DL_Codes::version **version**

  *DXF version to be created.*

### 5.49.1 Detailed Description

Defines interface for writing low level DXF constructs to a file.

Implementation is defined in derived classes that write to binary or ASCII files.

Implements functions that write higher level constructs in terms of the low level ones.

**Todo** Add error checking for string/entry length.

### 5.49.2 Constructor & Destructor Documentation

#### 5.49.2.1 DL_Writer()

```
DL_Writer::DL_Writer (
            DL_Codes::version version )  [inline]
```

**Parameters**

| | |
|---|---|
| *version* | DXF version. Defaults to DL_VERSION_2002. |

## 5.49.3 Member Function Documentation

### 5.49.3.1 comment()

```
void DL_Writer::comment (
            const char * text ) const  [inline]
```

Comment.

```
 999
 text
```

Referenced by DL_Dxf::writeHeader().

### 5.49.3.2 dxfBool()

```
virtual void DL_Writer::dxfBool (
            int gc,
            bool value ) const  [inline], [virtual]
```

Can be overwritten by the implementing class to write a bool value to the file.

**Parameters**

| | |
|---|---|
| *gc* | Group code. |
| *value* | The bool value. |

Referenced by DL_Dxf::writeHatchEdge().

### 5.49.3.3 dxfEOF()

```
void DL_Writer::dxfEOF ( ) const  [inline]
```

End of the DXF file.

```
  0
EOF
```

#### 5.49.3.4 dxfHex()

```
virtual void DL_Writer::dxfHex (
            int gc,
            int value ) const  [pure virtual]
```

Must be overwritten by the implementing class to write an int value (hex) to the file.

**Parameters**

| | |
|---|---|
| *gc* | Group code. |
| *value* | The int value. |

Implemented in DL_WriterA.

#### 5.49.3.5 dxfInt()

```
virtual void DL_Writer::dxfInt (
            int gc,
            int value ) const  [pure virtual]
```

Must be overwritten by the implementing class to write an int value to the file.

**Parameters**

| | |
|---|---|
| *gc* | Group code. |
| *value* | The int value. |

Implemented in DL_WriterA.

#### 5.49.3.6 dxfReal()

```
virtual void DL_Writer::dxfReal (
            int gc,
            double value ) const  [pure virtual]
```

Must be overwritten by the implementing class to write a real value to the file.

**Parameters**

| | |
|---|---|
| *gc* | Group code. |
| *value* | The real value. |

Implemented in DL_WriterA.

#### 5.49.3.7 dxfString() [1/2]

```
virtual void DL_Writer::dxfString (
            int gc,
            const char * value ) const  [pure virtual]
```

Must be overwritten by the implementing class to write a string to the file.

**Parameters**

| | |
|---|---|
| *gc* | Group code. |
| *value* | The string. |

Implemented in DL_WriterA.

#### 5.49.3.8 dxfString() [2/2]

```
virtual void DL_Writer::dxfString (
            int gc,
            const std::string & value ) const  [pure virtual]
```

Must be overwritten by the implementing class to write a string to the file.

**Parameters**

| | |
|---|---|
| *gc* | Group code. |
| *value* | The string. |

Implemented in DL_WriterA.

#### 5.49.3.9 entity()

```
void DL_Writer::entity (
            const char * entTypeName ) const  [inline]
```

Entity.

```
  0
entTypeName
```

**Returns**

Unique handle or 0.

Referenced by DL_Dxf::write3dFace(), DL_Dxf::writeArc(), DL_Dxf::writeCircle(), DL_Dxf::writeDimAligned(), DL_Dxf::writeDimAngular2L(), DL_Dxf::writeDimAngular3P(), DL_Dxf::writeDimDiametric(), DL_Dxf::writeDimLinear(), DL_Dxf::writeDimOrdinate(), DL_Dxf::writeDimRadial(), DL_Dxf::writeEllipse(), DL_Dxf::writeHatch1(), DL_Dxf::writeImage(), DL_Dxf::writeInsert(), DL_Dxf::writeLeader(), DL_Dxf::writeLine(), DL_Dxf::writeMText(), DL_Dxf::writePoint(), DL_Dxf::writePolyline(), DL_Dxf::writePolylineEnd(), DL_Dxf::writeRay(), DL_Dxf::writeSolid(), DL_Dxf::writeSpline(), DL_Dxf::writeText(), DL_Dxf::writeTrace(), DL_Dxf::writeVertex(), and DL_Dxf::writeXLine().

#### 5.49.3.10 entityAttributes()

```
void DL_Writer::entityAttributes (
            const DL_Attributes & attrib ) const  [inline]
```

Attributes of an entity.

```
  8
layer
62
color
39
width
 6
linetype
```

References DL_Attributes::getColor(), DL_Attributes::getColor24(), DL_Attributes::getLayer(), DL_Attributes::getLinetype(), and DL_Attributes::getWidth().

Referenced by DL_Dxf::write3dFace(), DL_Dxf::writeArc(), DL_Dxf::writeCircle(), DL_Dxf::writeDimAligned(), DL_Dxf::writeDimAngular2L(), DL_Dxf::writeDimAngular3P(), DL_Dxf::writeDimDiametric(), DL_Dxf::writeDimLinear(), DL_Dxf::writeDimOrdinate(), DL_Dxf::writeDimRadial(), DL_Dxf::writeEllipse(), DL_Dxf::writeHatch1(), DL_Dxf::writeImage(), DL_Dxf::writeInsert(), DL_Dxf::writeLeader(), DL_Dxf::writeLine(), DL_Dxf::writeMText(), DL_Dxf::writePoint(), DL_Dxf::writePolyline(), DL_Dxf::writeRay(), DL_Dxf::writeSolid(), DL_Dxf::writeSpline(), DL_Dxf::writeText(), DL_Dxf::writeTrace(), and DL_Dxf::writeXLine().

#### 5.49.3.11 getNextHandle()

```
unsigned long DL_Writer::getNextHandle ( ) const  [inline]
```

**Returns**

Next handle that will be written.

Referenced by DL_Dxf::writeObjects().

#### 5.49.3.12 section()

```
void DL_Writer::section (
            const char * name ) const  [inline]
```

Generic section for section 'name'.

```
  0
SECTION
  2
name
```

#### 5.49.3.13 sectionBlockEntry()

```
void DL_Writer::sectionBlockEntry (
            unsigned long int h = 0 ) const  [inline]
```

Block (must be in the section BLOCKS).

```
  0
BLOCK
```

Referenced by DL_Dxf::writeBlock().

#### 5.49.3.14 sectionBlockEntryEnd()

```
void DL_Writer::sectionBlockEntryEnd (
            unsigned long int h = 0 ) const  [inline]
```

End of Block (must be in the section BLOCKS).

```
  0
ENDBLK
```

Referenced by DL_Dxf::writeEndBlock().

#### 5.49.3.15 sectionBlocks()

```
void DL_Writer::sectionBlocks ( ) const  [inline]
```

Section BLOCKS.

```
  0
SECTION
  2
BLOCKS
```

#### 5.49.3.16 sectionClasses()

```
void DL_Writer::sectionClasses ( ) const  [inline]
```

Section CLASSES.

```
  0
SECTION
  2
CLASSES
```

#### 5.49.3.17 sectionEnd()

```
void DL_Writer::sectionEnd ( ) const  [inline]
```

End of a section.

```
  0
ENDSEC
```

#### 5.49.3.18 sectionEntities()

```
void DL_Writer::sectionEntities ( ) const  [inline]
```

Section ENTITIES.

```
  0
SECTION
  2
ENTITIES
```

#### 5.49.3.19 sectionHeader()

```
void DL_Writer::sectionHeader ( ) const  [inline]
```

Section HEADER.

```
  0
SECTION
  2
HEADER
```

Referenced by DL_Dxf::writeHeader().

#### 5.49.3.20 sectionObjects()

```
void DL_Writer::sectionObjects ( ) const  [inline]
```

Section OBJECTS.

```
  0
SECTION
  2
OBJECTS
```

#### 5.49.3.21 sectionTables()

```
void DL_Writer::sectionTables ( ) const  [inline]
```

Section TABLES.

```
  0
SECTION
  2
TABLES
```

#### 5.49.3.22 table()

```
void DL_Writer::table (
            const char * name,
            int num,
            int h = 0 ) const  [inline]
```

Generic table for table 'name' with 'num' entries:

```
  0
TABLE
  2
name
70
 num
```

#### 5.49.3.23 tableAppid()

```
void DL_Writer::tableAppid (
            int num ) const  [inline]
```

Table for application id.

**Parameters**

| | |
|---|---|
| *num* | Number of registered applications in total. |

```
  0
TABLE
  2
APPID
  70
     num
```

#### 5.49.3.24 tableAppidEntry()

```
void DL_Writer::tableAppidEntry (
            unsigned long int h = 0 ) const  [inline]
```

Appid (must be in the TABLES section APPID).

```
  0
APPID
```

Referenced by DL_Dxf::writeAppid().

#### 5.49.3.25 tableEnd()

```
void DL_Writer::tableEnd ( ) const  [inline]
```

End of a table.

```
  0
ENDTAB
```

#### 5.49.3.26 tableLayerEntry()

```
void DL_Writer::tableLayerEntry (
            unsigned long int h = 0 ) const  [inline]
```

Layer (must be in the TABLES section LAYER).

```
  0
LAYER
```

Referenced by DL_Dxf::writeLayer().

#### 5.49.3.27 tableLayers()

```
void DL_Writer::tableLayers (
            int num ) const  [inline]
```

Table for layers.

**Parameters**

| | |
|---|---|
| *num* | Number of layers in total. |

```
  0
TABLE
  2
LAYER
  70
     num
```

#### 5.49.3.28 tableLinetypeEntry()

```
void DL_Writer::tableLinetypeEntry (
             unsigned long int h = 0 ) const  [inline]
```

Line type (must be in the TABLES section LTYPE).

```
  0
LTYPE
```

Referenced by DL_Dxf::writeLinetype().

#### 5.49.3.29 tableLinetypes()

```
void DL_Writer::tableLinetypes (
             int num ) const  [inline]
```

Table for line types.

**Parameters**

| | |
|---|---|
| *num* | Number of line types in total. |

```
  0
TABLE
  2
LTYPE
  70
     num
```

#### 5.49.3.30 tableStyle()

```
void DL_Writer::tableStyle (
            int num ) const  [inline]
```

Table for text style.

**Parameters**

| | |
|---|---|
| *num* | Number of text styles. |

```
  0
TABLE
  2
STYLE
  70
     num
```

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

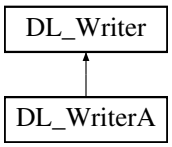
- src/dl_writer.h

## 5.50 DL_WriterA Class Reference

Implements functions defined in DL_Writer for writing low level DXF constructs to an ASCII format DXF file.

`#include <dl_writer_ascii.h>`

Inheritance diagram for DL_WriterA:




### Public Member Functions

- **DL_WriterA** (const char *fname, DL_Codes::version version=DL_VERSION_2000)
- bool openFailed () const
- void **close** () const

  *Closes the output file.*
- void dxfReal (int gc, double value) const

  *Writes a real (double) variable to the DXF file.*
- void dxfInt (int gc, int value) const

  *Writes an int variable to the DXF file.*
- void dxfHex (int gc, int value) const

  *Writes a hex int variable to the DXF file.*
- void dxfString (int gc, const char *value) const

  *Writes a string variable to the DXF file.*
- void dxfString (int gc, const std::string &value) const

  *Must be overwritten by the implementing class to write a string to the file.*

## Static Public Member Functions

- static void **strReplace** (char *str, char src, char dest)

  *Replaces every occurence of src with dest in the null terminated str.*

## Additional Inherited Members

## 5.50.1 Detailed Description

Implements functions defined in DL_Writer for writing low level DXF constructs to an ASCII format DXF file.

@para fname File name of the file to be created. @para version DXF version. Defaults to DL_VERSION_2002.

**Todo** What if `fname` is NULL? Or `fname` can't be opened for another reason?

## 5.50.2 Member Function Documentation

### 5.50.2.1 dxfHex()

```
void DL_WriterA::dxfHex (
            int gc,
            int value ) const  [virtual]
```

Writes a hex int variable to the DXF file.

**Parameters**

| | |
|---|---|
| *gc* | Group code. |
| *value* | Int value |

Implements DL_Writer.

References dxfString().

Referenced by DL_Dxf::writeBlockRecord(), DL_Dxf::writeDimStyle(), DL_Dxf::writeHeader(), DL_Dxf::writeImageDef(), DL_Dxf::writeLayer(), DL_Dxf::writeObjects(), DL_Dxf::writeUcs(), DL_Dxf::writeView(), and DL_Dxf::writeVPort().

### 5.50.2.2 dxfInt()

```
void DL_WriterA::dxfInt (
            int gc,
            int value ) const  [virtual]
```

Writes an int variable to the DXF file.

**Parameters**

| | |
|---|---|
| *gc* | Group code. |
| *value* | Int value |

Implements DL_Writer.

Referenced by DL_Dxf::writeAppid(), DL_Dxf::writeBlock(), DL_Dxf::writeBlockRecord(), DL_Dxf::writeDimAligned(), DL_Dxf::writeDimAngular2L(), DL_Dxf::writeDimAngular3P(), DL_Dxf::writeDimDiametric(), DL_Dxf::writeDimLinear(), DL_Dxf::writeDimOrdinate(), DL_Dxf::writeDimRadial(), DL_Dxf::writeDimStyle(), DL_Dxf::writeHatch1(), DL_Dxf::writeHatch2(), DL_Dxf::writeHatchEdge(), DL_Dxf::writeHatchLoop1(), DL_Dxf::writeHatchLoop2(), DL_Dxf::writeImage(), DL_Dxf::writeImageDef(), DL_Dxf::writeInsert(), DL_Dxf::writeLayer(), DL_Dxf::writeLeader(), DL_Dxf::writeLinetype(), DL_Dxf::writeMText(), DL_Dxf::writeObjects(), DL_Dxf::writePolyline(), DL_Dxf::writeSpline(), DL_Dxf::writeStyle(), DL_Dxf::writeText(), DL_Dxf::writeUcs(), DL_Dxf::writeView(), and DL_Dxf::writeVPort().

#### 5.50.2.3 dxfReal()

```
void DL_WriterA::dxfReal (
            int gc,
            double value ) const  [virtual]
```

Writes a real (double) variable to the DXF file.

**Parameters**

| | |
|---|---|
| *gc* | Group code. |
| *value* | Double value |

Implements DL_Writer.

References dxfString(), strReplace(), and DL_Writer::version.

Referenced by DL_Dxf::writeArc(), DL_Dxf::writeCircle(), DL_Dxf::writeControlPoint(), DL_Dxf::writeDimAligned(), DL_Dxf::writeDimAngular2L(), DL_Dxf::writeDimAngular3P(), DL_Dxf::writeDimDiametric(), DL_Dxf::writeDimLinear(), DL_Dxf::writeDimOrdinate(), DL_Dxf::writeDimRadial(), DL_Dxf::writeDimStyle(), DL_Dxf::writeEllipse(), DL_Dxf::writeFitPoint(), DL_Dxf::writeHatch1(), DL_Dxf::writeHatch2(), DL_Dxf::writeHatchEdge(), DL_Dxf::writeImage(), DL_Dxf::writeImageDef(), DL_Dxf::writeInsert(), DL_Dxf::writeKnot(), DL_Dxf::writeLeader(), DL_Dxf::writeLeaderVertex(), DL_Dxf::writeLinetype(), DL_Dxf::writeMText(), DL_Dxf::writeObjects(), DL_Dxf::writeSolid(), DL_Dxf::writeStyle(), DL_Dxf::writeText(), DL_Dxf::writeTrace(), DL_Dxf::writeVertex(), and DL_Dxf::writeVPort().

#### 5.50.2.4 dxfString() [1/2]

```
void DL_WriterA::dxfString (
            int gc,
            const char * value ) const  [virtual]
```

Writes a string variable to the DXF file.

**Parameters**

| | |
|---|---|
| *gc* | Group code. |
| *value* | String |

Implements DL_Writer.

Referenced by dxfHex(), dxfReal(), DL_Dxf::write3dFace(), DL_Dxf::writeAppid(), DL_Dxf::writeArc(), DL_Dxf::writeBlock(), DL_Dxf::writeBlockRecord(), DL_Dxf::writeCircle(), DL_Dxf::writeComment(), DL_Dxf::writeDimAligned(), DL_Dxf::writeDimAngular2L(), DL_Dxf::writeDimAngular3P(), DL_Dxf::writeDimDiametric(), DL_Dxf::writeDimLinear(), DL_Dxf::writeDimOrdinate(), DL_Dxf::writeDimRadial(), DL_Dxf::writeDimStyle(), DL_Dxf::writeEllipse(), DL_Dxf::writeHatch1(), DL_Dxf::writeHatch2(), DL_Dxf::writeHeader(), DL_Dxf::writeImage(), DL_Dxf::writeImageDef(), DL_Dxf::writeInsert(), DL_Dxf::writeLayer(), DL_Dxf::writeLeader(), DL_Dxf::writeLine(), DL_Dxf::writeLinetype(), DL_Dxf::writeMText(), DL_Dxf::writeObjects(), DL_Dxf::writeObjectsEnd(), DL_Dxf::writePoint(), DL_Dxf::writePolyline(), DL_Dxf::writeRay(), DL_Dxf::writeSolid(), DL_Dxf::writeSpline(), DL_Dxf::writeStyle(), DL_Dxf::writeText(), DL_Dxf::writeTrace(), DL_Dxf::writeUcs(), DL_Dxf::writeVertex(), DL_Dxf::writeView(), DL_Dxf::writeVPort(), and DL_Dxf::writeXLine().

#### 5.50.2.5 dxfString() [2/2]

```
void DL_WriterA::dxfString (
            int gc,
            const std::string & value ) const  [virtual]
```

Must be overwritten by the implementing class to write a string to the file.

**Parameters**

| | |
|---|---|
| *gc* | Group code. |
| *value* | The string. |

Implements DL_Writer.

#### 5.50.2.6 openFailed()

```
bool DL_WriterA::openFailed ( ) const
```

**Return values**

| | |
|---|---|
| *true* | Opening file has failed. |
| *false* | Otherwise. |

Referenced by DL_Dxf::out().

The documentation for this class was generated from the following files:

- src/dl_writer_ascii.h
- src/dl_writer_ascii.cpp

## 5.51 DL_XLineData Struct Reference

XLine Data.

`#include <dl_entities.h>`

### Public Member Functions

- DL_XLineData (double bx, double by, double bz, double dx, double dy, double dz)
  *Constructor.*

### Public Attributes

- double bx
- double by
- double bz
- double dx
- double dy
- double dz

### 5.51.1 Detailed Description

XLine Data.

### 5.51.2 Constructor & Destructor Documentation

#### 5.51.2.1 DL_XLineData()

```
DL_XLineData::DL_XLineData (
            double bx,
            double by,
            double bz,
            double dx,
            double dy,
            double dz )  [inline]
```

Constructor.

Parameters: see member variables.

### 5.51.3 Member Data Documentation

#### 5.51.3.1 bx

```
double DL_XLineData::bx
```

X base point.

Referenced by DL_Dxf::writeXLine().

#### 5.51.3.2 by

```
double DL_XLineData::by
```

Y base point.

Referenced by DL_Dxf::writeXLine().

#### 5.51.3.3 bz

```
double DL_XLineData::bz
```

Z base point.

Referenced by DL_Dxf::writeXLine().

#### 5.51.3.4 dx

```
double DL_XLineData::dx
```

X direction vector.

Referenced by DL_Dxf::writeXLine().

#### 5.51.3.5 dy

```
double DL_XLineData::dy
```

Y direction vector.

Referenced by DL_Dxf::writeXLine().

#### 5.51.3.6 dz

```
double DL_XLineData::dz
```

Z direction vector.

Referenced by DL_Dxf::writeXLine().

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

- src/dl_entities.h

# Chapter 6

# File Documentation

## 6.1 dl_attributes.h

```
1 /**********************************************************************
2 ** Copyright (C) 2001-2013 RibbonSoft, GmbH. All rights reserved.
3 **
4 ** This file is part of the dxflib project.
5 **
6 ** This file is free software; you can redistribute it and/or modify
7 ** it under the terms of the GNU General Public License as published by
8 ** the Free Software Foundation; either version 2 of the License, or
9 ** (at your option) any later version.
10 **
11 ** Licensees holding valid dxflib Professional Edition licenses may use
12 ** this file in accordance with the dxflib Commercial License
13 ** Agreement provided with the Software.
14 **
15 ** This file is provided AS IS with NO WARRANTY OF ANY KIND, INCLUDING THE
16 ** WARRANTY OF DESIGN, MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE.
17 **
18 ** See http://www.ribbonsoft.com for further details.
19 **
20 ** Contact info@ribbonsoft.com if any conditions of this licensing are
21 ** not clear to you.
22 **
23 **********************************************************************/
24
25 #ifndef DL_ATTRIBUTES_H
26 #define DL_ATTRIBUTES_H
27
28 #include "dl_global.h"
29
30 #include <string>
31 #include <vector>
32
33 #include "dl_codes.h"
34
41 class DXFLIB_EXPORT DL_Attributes {
42
43 public:
44
48     DL_Attributes() :
49         layer(""),
50         color(0),
51         color24(-1),
52         width(0),
53         linetype("BYLAYER"),
54         linetypeScale(1.0),
55         handle(-1),
56         inPaperSpace(false) {
57     }
58
70     DL_Attributes(const std::string& layer,
71                   int color, int width,
72                   const std::string& linetype,
73                   double linetypeScale) :
74         layer(layer),
75         color(color),
76         color24(-1),
77         width(width),
78         linetype(linetype),
```

```
79          linetypeScale(linetypeScale),
80          handle(-1),
81          inPaperSpace(false) {
82
83      }
84
97      DL_Attributes(const std::string& layer,
98                    int color, int color24, int width,
99                    const std::string& linetype,
100                    int handle=-1)  :
101         layer(layer),
102         color(color),
103         color24(color24),
104         width(width),
105         linetype(linetype),
106         linetypeScale(1.0),
107         handle(handle),
108         inPaperSpace(false) {
109     }
110
115     void setLayer(const std::string& layer) {
116         this->layer = layer;
117     }
118
122     std::string getLayer() const {
123         return layer;
124     }
125
131     void setColor(int color) {
132         this->color = color;
133     }
134
140     void setColor24(int color) {
141         this->color24 = color;
142     }
143
149     int getColor() const {
150         return color;
151     }
152
158     int getColor24() const {
159         return color24;
160     }
161
165     void setWidth(int width) {
166         this->width = width;
167     }
168
172     int getWidth() const {
173         return width;
174     }
175
180     void setLinetype(const std::string& linetype) {
181         this->linetype = linetype;
182     }
183
187     void setLinetypeScale(double linetypeScale) {
188         this->linetypeScale = linetypeScale;
189     }
190
191     double getLinetypeScale() const {
192         return linetypeScale;
193     }
194
198     std::string getLinetype() const {
199         if (linetype.length()==0) {
200             return "BYLAYER";
201         } else {
202             return linetype;
203         }
204     }
205
206     void setHandle(int h) {
207         handle = h;
208     }
209
210     int getHandle() const {
211         return handle;
212     }
213
214     void setInPaperSpace(bool on) {
215         inPaperSpace = on;
216     }
217
218     bool isInPaperSpace() const {
219         return inPaperSpace;
220     }
```

```
221
222 private:
223     std::string layer;
224     int color;
225     int color24;
226     int width;
227     std::string linetype;
228     double linetypeScale;
229     int handle;
230
231     // DXF code 67 (true: entity in paper space, false: entity in model space (default):
232     bool inPaperSpace;
233 };
234
235 #endif
236
237 // EOF
```

## 6.2 dl_codes.h

```
1 /**********************************************************************
2 ** Copyright (C) 2001-2013 RibbonSoft, GmbH. All rights reserved.
3 ** Copyright (C) 2001 Robert J. Campbell Jr.
4 **
5 ** This file is part of the dxflib project.
6 **
7 ** This file is free software; you can redistribute it and/or modify
8 ** it under the terms of the GNU General Public License as published by
9 ** the Free Software Foundation; either version 2 of the License, or
10 ** (at your option) any later version.
11 **
12 ** Licensees holding valid dxflib Professional Edition licenses may use
13 ** this file in accordance with the dxflib Commercial License
14 ** Agreement provided with the Software.
15 **
16 ** This file is provided AS IS with NO WARRANTY OF ANY KIND, INCLUDING THE
17 ** WARRANTY OF DESIGN, MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE.
18 **
19 ** See http://www.ribbonsoft.com for further details.
20 **
21 ** Contact info@ribbonsoft.com if any conditions of this licensing are
22 ** not clear to you.
23 **
24 **********************************************************************/
25
30 #ifndef DXF_CODES_H
31 #define DXF_CODES_H
32
33 #include "dl_global.h"
34
35 #if _MSC_VER > 1000
36 #pragma once
37 #endif // _MSC_VER > 1000
38
39 #if defined(__OS2__)||defined(__EMX__)
40 #define strcasecmp(s,t) stricmp(s,t)
41 #endif
42
43 #if defined(_WIN32)
44 #define strcasecmp(s,t) _stricmp(s,t)
45 #endif
46
47
48 #ifdef _WIN32
49 #undef M_PI
50 #define M_PI   3.14159265358979323846
51 #pragma warning(disable : 4800)
52 #endif
53
54 #ifndef M_PI
55 #define M_PI 3.14159265358979323846264338327795
56 #endif
57
58 #define DL_DXF_MAXLINE 1024
59 #define DL_DXF_MAXGROUPCODE 1100
60
61 // used to mark invalid vectors:
62 //#define DL_DXF_MAXDOUBLE 1.0E+10
63
67 class DXFLIB_EXPORT DL_Codes {
68 public:
72     enum color {
73         black = 250,
```

```
74          green = 3,
75          red = 1,
76          brown = 15,
77          yellow = 2,
78          cyan = 4,
79          magenta = 6,
80          gray = 8,
81          blue = 5,
82          l_blue = 163,
83          l_green = 121,
84          l_cyan = 131,
85          l_red = 23,
86          l_magenta = 221,
87          l_gray = 252,
88          white = 7,
89          bylayer = 256,
90          byblock = 0
91      };
92
96      enum version {
97          AC1009_MIN,     // R12, minimalistic
98          AC1009,         // R12
99          AC1012,
100          AC1014,
101          AC1015          // R2000
102      };
103 };
104
105
106 // Extended color palette:
107 // The first entry is only for direct indexing starting with [1]
108 // Color 1 is red (1,0,0)
109 const double dxfColors[][3] = {
110                                 {0,0,0},                // unused
111                                 {1,0,0},                // 1
112                                 {1,1,0},
113                                 {0,1,0},
114                                 {0,1,1},
115                                 {0,0,1},
116                                 {1,0,1},
117                                 {1,1,1},                // black or white
118                                 {0.5,0.5,0.5},
119                                 {0.75,0.75,0.75},
120                                 {1,0,0},                // 10
121                                 {1,0.5,0.5},
122                                 {0.65,0,0},
123                                 {0.65,0.325,0.325},
124                                 {0.5,0,0},
125                                 {0.5,0.25,0.25},
126                                 {0.3,0,0},
127                                 {0.3,0.15,0.15},
128                                 {0.15,0,0},
129                                 {0.15,0.075,0.075},
130                                 {1,0.25,0},             // 20
131                                 {1,0.625,0.5},
132                                 {0.65,0.1625,0},
133                                 {0.65,0.4063,0.325},
134                                 {0.5,0.125,0},
135                                 {0.5,0.3125,0.25},
136                                 {0.3,0.075,0},
137                                 {0.3,0.1875,0.15},
138                                 {0.15,0.0375,0},
139                                 {0.15,0.0938,0.075},
140                                 {1,0.5,0},              // 30
141                                 {1,0.75,0.5},
142                                 {0.65,0.325,0},
143                                 {0.65,0.4875,0.325},
144                                 {0.5,0.25,0},
145                                 {0.5,0.375,0.25},
146                                 {0.3,0.15,0},
147                                 {0.3,0.225,0.15},
148                                 {0.15,0.075,0},
149                                 {0.15,0.1125,0.075},
150                                 {1,0.75,0},             // 40
151                                 {1,0.875,0.5},
152                                 {0.65,0.4875,0},
153                                 {0.65,0.5688,0.325},
154                                 {0.5,0.375,0},
155                                 {0.5,0.4375,0.25},
156                                 {0.3,0.225,0},
157                                 {0.3,0.2625,0.15},
158                                 {0.15,0.1125,0},
159                                 {0.15,0.1313,0.075},
160                                 {1,1,0},                // 50
161                                 {1,1,0.5},
162                                 {0.65,0.65,0},
163                                 {0.65,0.65,0.325},
```

```
164                                 {0.5,0.5,0},
165                                 {0.5,0.5,0.25},
166                                 {0.3,0.3,0},
167                                 {0.3,0.3,0.15},
168                                 {0.15,0.15,0},
169                                 {0.15,0.15,0.075},
170                                 {0.75,1,0},             // 60
171                                 {0.875,1,0.5},
172                                 {0.4875,0.65,0},
173                                 {0.5688,0.65,0.325},
174                                 {0.375,0.5,0},
175                                 {0.4375,0.5,0.25},
176                                 {0.225,0.3,0},
177                                 {0.2625,0.3,0.15},
178                                 {0.1125,0.15,0},
179                                 {0.1313,0.15,0.075},
180                                 {0.5,1,0},              // 70
181                                 {0.75,1,0.5},
182                                 {0.325,0.65,0},
183                                 {0.4875,0.65,0.325},
184                                 {0.25,0.5,0},
185                                 {0.375,0.5,0.25},
186                                 {0.15,0.3,0},
187                                 {0.225,0.3,0.15},
188                                 {0.075,0.15,0},
189                                 {0.1125,0.15,0.075},
190                                 {0.25,1,0},             // 80
191                                 {0.625,1,0.5},
192                                 {0.1625,0.65,0},
193                                 {0.4063,0.65,0.325},
194                                 {0.125,0.5,0},
195                                 {0.3125,0.5,0.25},
196                                 {0.075,0.3,0},
197                                 {0.1875,0.3,0.15},
198                                 {0.0375,0.15,0},
199                                 {0.0938,0.15,0.075},
200                                 {0,1,0},                // 90
201                                 {0.5,1,0.5},
202                                 {0,0.65,0},
203                                 {0.325,0.65,0.325},
204                                 {0,0.5,0},
205                                 {0.25,0.5,0.25},
206                                 {0,0.3,0},
207                                 {0.15,0.3,0.15},
208                                 {0,0.15,0},
209                                 {0.075,0.15,0.075},
210                                 {0,1,0.25},             // 100
211                                 {0.5,1,0.625},
212                                 {0,0.65,0.1625},
213                                 {0.325,0.65,0.4063},
214                                 {0,0.5,0.125},
215                                 {0.25,0.5,0.3125},
216                                 {0,0.3,0.075},
217                                 {0.15,0.3,0.1875},
218                                 {0,0.15,0.0375},
219                                 {0.075,0.15,0.0938},
220                                 {0,1,0.5},              // 110
221                                 {0.5,1,0.75},
222                                 {0,0.65,0.325},
223                                 {0.325,0.65,0.4875},
224                                 {0,0.5,0.25},
225                                 {0.25,0.5,0.375},
226                                 {0,0.3,0.15},
227                                 {0.15,0.3,0.225},
228                                 {0,0.15,0.075},
229                                 {0.075,0.15,0.1125},
230                                 {0,1,0.75},             // 120
231                                 {0.5,1,0.875},
232                                 {0,0.65,0.4875},
233                                 {0.325,0.65,0.5688},
234                                 {0,0.5,0.375},
235                                 {0.25,0.5,0.4375},
236                                 {0,0.3,0.225},
237                                 {0.15,0.3,0.2625},
238                                 {0,0.15,0.1125},
239                                 {0.075,0.15,0.1313},
240                                 {0,1,1},                // 130
241                                 {0.5,1,1},
242                                 {0,0.65,0.65},
243                                 {0.325,0.65,0.65},
244                                 {0,0.5,0.5},
245                                 {0.25,0.5,0.5},
246                                 {0,0.3,0.3},
247                                 {0.15,0.3,0.3},
248                                 {0,0.15,0.15},
249                                 {0.075,0.15,0.15},
250                                 {0,0.75,1},             // 140
```

```
251                             {0.5,0.875,1},
252                             {0,0.4875,0.65},
253                             {0.325,0.5688,0.65},
254                             {0,0.375,0.5},
255                             {0.25,0.4375,0.5},
256                             {0,0.225,0.3},
257                             {0.15,0.2625,0.3},
258                             {0,0.1125,0.15},
259                             {0.075,0.1313,0.15},
260                             {0,0.5,1},              // 150
261                             {0.5,0.75,1},
262                             {0,0.325,0.65},
263                             {0.325,0.4875,0.65},
264                             {0,0.25,0.5},
265                             {0.25,0.375,0.5},
266                             {0,0.15,0.3},
267                             {0.15,0.225,0.3},
268                             {0,0.075,0.15},
269                             {0.075,0.1125,0.15},
270                             {0,0.25,1},             // 160
271                             {0.5,0.625,1},
272                             {0,0.1625,0.65},
273                             {0.325,0.4063,0.65},
274                             {0,0.125,0.5},
275                             {0.25,0.3125,0.5},
276                             {0,0.075,0.3},
277                             {0.15,0.1875,0.3},
278                             {0,0.0375,0.15},
279                             {0.075,0.0938,0.15},
280                             {0,0,1},                // 170
281                             {0.5,0.5,1},
282                             {0,0,0.65},
283                             {0.325,0.325,0.65},
284                             {0,0,0.5},
285                             {0.25,0.25,0.5},
286                             {0,0,0.3},
287                             {0.15,0.15,0.3},
288                             {0,0,0.15},
289                             {0.075,0.075,0.15},
290                             {0.25,0,1},             // 180
291                             {0.625,0.5,1},
292                             {0.1625,0,0.65},
293                             {0.4063,0.325,0.65},
294                             {0.125,0,0.5},
295                             {0.3125,0.25,0.5},
296                             {0.075,0,0.3},
297                             {0.1875,0.15,0.3},
298                             {0.0375,0,0.15},
299                             {0.0938,0.075,0.15},
300                             {0.5,0,1},              // 190
301                             {0.75,0.5,1},
302                             {0.325,0,0.65},
303                             {0.4875,0.325,0.65},
304                             {0.25,0,0.5},
305                             {0.375,0.25,0.5},
306                             {0.15,0,0.3},
307                             {0.225,0.15,0.3},
308                             {0.075,0,0.15},
309                             {0.1125,0.075,0.15},
310                             {0.75,0,1},             // 200
311                             {0.875,0.5,1},
312                             {0.4875,0,0.65},
313                             {0.5688,0.325,0.65},
314                             {0.375,0,0.5},
315                             {0.4375,0.25,0.5},
316                             {0.225,0,0.3},
317                             {0.2625,0.15,0.3},
318                             {0.1125,0,0.15},
319                             {0.1313,0.075,0.15},
320                             {1,0,1},                // 210
321                             {1,0.5,1},
322                             {0.65,0,0.65},
323                             {0.65,0.325,0.65},
324                             {0.5,0,0.5},
325                             {0.5,0.25,0.5},
326                             {0.3,0,0.3},
327                             {0.3,0.15,0.3},
328                             {0.15,0,0.15},
329                             {0.15,0.075,0.15},
330                             {1,0,0.75},             // 220
331                             {1,0.5,0.875},
332                             {0.65,0,0.4875},
333                             {0.65,0.325,0.5688},
334                             {0.5,0,0.375},
335                             {0.5,0.25,0.4375},
336                             {0.3,0,0.225},
337                             {0.3,0.15,0.2625},
```

```
338                                 {0.15,0,0.1125},
339                                 {0.15,0.075,0.1313},
340                                 {1,0,0.5},              // 230
341                                 {1,0.5,0.75},
342                                 {0.65,0,0.325},
343                                 {0.65,0.325,0.4875},
344                                 {0.5,0,0.25},
345                                 {0.5,0.25,0.375},
346                                 {0.3,0,0.15},
347                                 {0.3,0.15,0.225},
348                                 {0.15,0,0.075},
349                                 {0.15,0.075,0.1125},
350                                 {1,0,0.25},             // 240
351                                 {1,0.5,0.625},
352                                 {0.65,0,0.1625},
353                                 {0.65,0.325,0.4063},
354                                 {0.5,0,0.125},
355                                 {0.5,0.25,0.3125},
356                                 {0.3,0,0.075},
357                                 {0.3,0.15,0.1875},
358                                 {0.15,0,0.0375},
359                                 {0.15,0.075,0.0938},
360                                 {0.33,0.33,0.33},       // 250
361                                 {0.464,0.464,0.464},
362                                 {0.598,0.598,0.598},
363                                 {0.732,0.732,0.732},
364                                 {0.866,0.866,0.866},
365                                 {1,1,1}                 // 255
366                             }
367                             ;
368
369
370 // AutoCAD VERSION aliases
371 #define DL_VERSION_R12    DL_Codes::AC1009
372 #define DL_VERSION_LT2    DL_Codes::AC1009
373 #define DL_VERSION_R13    DL_Codes::AC1012   // not supported yet
374 #define DL_VERSION_LT95   DL_Codes::AC1012   // not supported yet
375 #define DL_VERSION_R14    DL_Codes::AC1014   // not supported yet
376 #define DL_VERSION_LT97   DL_Codes::AC1014   // not supported yet
377 #define DL_VERSION_LT98   DL_Codes::AC1014   // not supported yet
378 #define DL_VERSION_2000   DL_Codes::AC1015
379 #define DL_VERSION_2002   DL_Codes::AC1015
380
381
382 // DXF Group Codes:
383
384 // Strings
385 #define DL_STRGRP_START      0
386 #define DL_STRGRP_END        9
387
388 // Coordinates
389 #define DL_CRDGRP_START     10
390 #define DL_CRDGRP_END       19
391
392 // Real values
393 #define DL_RLGRP_START      38
394 #define DL_RLGRP_END        59
395
396 // Short integer values
397 #define DL_SHOGRP_START     60
398 #define DL_SHOGRP_END       79
399
400 // New in Release 13,
401 #define DL_SUBCLASS        100
402
403 // More coordinates
404 #define DL_CRD2GRP_START   210
405 #define DL_CRD2GRP_END     239
406
407 // Extended data strings
408 #define DL_ESTRGRP_START  1000
409 #define DL_ESTRGRP_END    1009
410
411 // Extended data reals
412 #define DL_ERLGRP_START   1010
413 #define DL_ERLGRP_END     1059
414
415
416 #define DL_Y8_COORD_CODE       28
417 #define DL_Z0_COORD_CODE       30
418 #define DL_Z8_COORD_CODE       38
419
420 #define DL_POINT_COORD_CODE    10
421 #define DL_INSERT_COORD_CODE   10
422
423 #define DL_CRD2GRP_START      210
424 #define DL_CRD2GRP_END        239
```

```
425
426 #define DL_THICKNESS            39
427 #define DL_FIRST_REAL_CODE      THICKNESS
428 #define DL_LAST_REAL_CODE       59
429 #define DL_FIRST_INT_CODE       60
430 #define DL_ATTFLAGS_CODE        70
431 #define DL_PLINE_FLAGS_CODE     70
432 #define DL_LAYER_FLAGS_CODE     70
433 #define DL_FLD_LEN_CODE         73 // Inside ATTRIB resbuf
434 #define DL_LAST_INT_CODE        79
435 #define DL_X_EXTRU_CODE        210
436 #define DL_Y_EXTRU_CODE        220
437 #define DL_Z_EXTRU_CODE        230
438 #define DL_COMMENT_CODE        999
439
440 // Start and endpoints of a line
441 #define DL_LINE_START_CODE      10  // Followed by x coord
442 #define DL_LINE_END_CODE        11  // Followed by x coord
443
444 // Some codes used by blocks
445 #define DL_BLOCK_FLAGS_CODE     70  // An int containing flags
446 #define DL_BLOCK_BASE_CODE      10  // Origin of block definition
447 #define DL_XREF_DEPENDENT       16  // If a block contains an XREF
448 #define DL_XREF_RESOLVED        32  // If a XREF resolved ok
449 #define DL_REFERENCED           64  // If a block is ref'd in DWG
450
451 #define DL_XSCALE_CODE          41
452 #define DL_YSCALE_CODE          42
453 #define DL_ANGLE_CODE           50
454 #define DL_INS_POINT_CODE       10  // Followed by x of ins pnt
455 #define DL_NAME2_CODE            3  // Second appearance of name
456
457 // Some codes used by circle entities
458 #define DL_CENTER_CODE          10  // Followed by x of center
459 #define DL_RADIUS_CODE          40  // Followd by radius of circle
460
461 #define DL_COND_OP_CODE         -4  // Conditional op,ads_ssget
462
463 // When using ads_buildlist you MUST use RTDXF0 instead of these
464 #define DL_ENTITY_TYPE_CODE      0  // Then there is LINE, 3DFACE..
465 #define DL_SES_CODE              0  // Start End String Code
466 #define DL_FILE_SEP_CODE         0  // File separator
467 #define DL_SOT_CODE              0  // Start Of Table
468 #define DL_TEXTVAL_CODE          1
469 #define DL_NAME_CODE             2
470 #define DL_BLOCK_NAME_CODE       2
471 #define DL_SECTION_NAME_CODE     2
472 #define DL_ENT_HAND_CODE         5  // What follows is hexa string
473 #define DL_TXT_STYLE_CODE        7  // Inside attributes
474 #define DL_LAYER_NAME_CODE       8  // What follows is layer name
475 #define DL_FIRST_XCOORD_CODE    10  // Group code x of 1st coord
476 #define DL_FIRST_YCOORD_CODE    20  // Group code y of 1st coord
477 #define DL_FIRST_ZCOORD_CODE    30  // Group code z of 1st coord
478 #define DL_L_START_CODE         10
479 #define DL_L_END_CODE           11
480 #define DL_TXTHI_CODE           40
481 #define DL_SCALE_X_CODE         41
482 #define DL_SCALE_Y_CODE         42
483 #define DL_SCALE_Z_CODE         43
484 #define DL_BULGE_CODE           42  // Used in PLINE verts for arcs
485 #define DL_ROTATION_CODE        50
486 #define DL_COLOUR_CODE          62  // What follows is a color int
487 #define DL_LTYPE_CODE            6  // What follows is a linetype
488
489
490 // Attribute flags
491 #define DL_ATTS_FOLLOW_CODE     66
492 #define DL_ATT_TAG_CODE          2
493 #define DL_ATT_VAL_CODE          1
494 #define DL_ATT_FLAGS_CODE       70  // 4 1 bit flags as follows...
495 #define DL_ATT_INVIS_FLAG        1
496 #define DL_ATT_CONST_FLAG        2
497 #define DL_ATT_VERIFY_FLAG       4 // Prompt and verify
498 #define DL_ATT_PRESET_FLAG       8 // No prompt and no verify
499
500 // PLINE defines
501 // Flags
502 #define DL_OPEN_PLINE       0x00
503 #define DL_CLOSED_PLINE     0x01
504 #define DL_POLYLINE3D       0x08
505 #define DL_PFACE_MESH       0x40
506 #define DL_PGON_MESH        0x10
507 // Vertices follow entity, required in POLYLINES
508 #define DL_VERTS_FOLLOW_CODE   66 // Value should always be 1
509 #define DL_VERTEX_COORD_CODE   10
510
511
```

```
512 // LAYER flags
513 #define DL_FROZEN           1
514 #define DL_FROZEN_BY_DEF    2
515 #define DL_LOCKED           4
516 #define DL_OBJECT_USED     64   // Object is ref'd in the dwg
517
518 #define DL_BLOCK_EN_CODE   -2   // Block entity definition
519 #define DL_E_NAME          -1   // Entity name
520
521 // Extended data codes
522 #define DL_EXTD_SENTINEL    (-3)
523 #define DL_EXTD_STR         1000
524 #define DL_EXTD_APP_NAME    1001
525 #define DL_EXTD_CTL_STR     1002
526 #define DL_EXTD_LYR_STR     1003
527 #define DL_EXTD_CHUNK       1004
528 #define DL_EXTD_HANDLE      1005
529 #define DL_EXTD_POINT       1010
530 #define DL_EXTD_POS         1011
531 #define DL_EXTD_DISP        1012
532 #define DL_EXTD_DIR         1013
533 #define DL_EXTD_FLOAT       1040
534 #define DL_EXTD_DIST        1041
535 #define DL_EXTD_SCALE       1042
536 #define DL_EXTD_INT16       1070
537 #define DL_EXTD_INT32       1071
538
539 // UCS codes for use in ads_trans
540 #define DL_WCS_TRANS_CODE      0
541 #define DL_UCS_TRANS_CODE      1
542 #define DL_DCS_TRANS_CODE      2
543 #define DL_PCS_TRANS_CODE      3
544
545 #endif
546
```

## 6.3 dl_creationadapter.h

```
1 /**********************************************************************
2 ** Copyright (C) 2001-2013 RibbonSoft, GmbH. All rights reserved.
3 **
4 ** This file is part of the dxflib project.
5 **
6 ** This file is free software; you can redistribute it and/or modify
7 ** it under the terms of the GNU General Public License as published by
8 ** the Free Software Foundation; either version 2 of the License, or
9 ** (at your option) any later version.
10 **
11 ** Licensees holding valid dxflib Professional Edition licenses may use
12 ** this file in accordance with the dxflib Commercial License
13 ** Agreement provided with the Software.
14 **
15 ** This file is provided AS IS with NO WARRANTY OF ANY KIND, INCLUDING THE
16 ** WARRANTY OF DESIGN, MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE.
17 **
18 ** See http://www.ribbonsoft.com for further details.
19 **
20 ** Contact info@ribbonsoft.com if any conditions of this licensing are
21 ** not clear to you.
22 **
23 **********************************************************************/
24
25 #ifndef DL_CREATIONADAPTER_H
26 #define DL_CREATIONADAPTER_H
27
28 #include "dl_global.h"
29
30 #include "dl_creationinterface.h"
31
39 class DXFLIB_EXPORT DL_CreationAdapter : public DL_CreationInterface {
40 public:
41     DL_CreationAdapter() {}
42     virtual ~DL_CreationAdapter() {}
43     virtual void processCodeValuePair(unsigned int, const std::string&) {}
44     virtual void endSection() {}
45     virtual void addLayer(const DL_LayerData&) {}
46     virtual void addLinetype(const DL_LinetypeData&) {}
47     virtual void addLinetypeDash(double) {}
48     virtual void addBlock(const DL_BlockData&) {}
49     virtual void endBlock() {}
50     virtual void addTextStyle(const DL_StyleData&) {}
51     virtual void addPoint(const DL_PointData&) {}
52     virtual void addLine(const DL_LineData&) {}
```

```
53      virtual void addXLine(const DL_XLineData&) {}
54      virtual void addRay(const DL_RayData&) {}
55
56      virtual void addArc(const DL_ArcData&) {}
57      virtual void addCircle(const DL_CircleData&) {}
58      virtual void addEllipse(const DL_EllipseData&) {}
59
60      virtual void addPolyline(const DL_PolylineData&) {}
61      virtual void addVertex(const DL_VertexData&) {}
62
63      virtual void addSpline(const DL_SplineData&) {}
64      virtual void addControlPoint(const DL_ControlPointData&) {}
65      virtual void addFitPoint(const DL_FitPointData&) {}
66      virtual void addKnot(const DL_KnotData&) {}
67
68      virtual void addInsert(const DL_InsertData&) {}
69
70      virtual void addMText(const DL_MTextData&) {}
71      virtual void addMTextChunk(const std::string&) {}
72      virtual void addText(const DL_TextData&) {}
73      virtual void addArcAlignedText(const DL_ArcAlignedTextData&) {}
74      virtual void addAttribute(const DL_AttributeData&) {}
75
76      virtual void addDimAlign(const DL_DimensionData&,
77                               const DL_DimAlignedData&) {}
78      virtual void addDimLinear(const DL_DimensionData&,
79                                const DL_DimLinearData&) {}
80      virtual void addDimRadial(const DL_DimensionData&,
81                                const DL_DimRadialData&) {}
82      virtual void addDimDiametric(const DL_DimensionData&,
83                                const DL_DimDiametricData&) {}
84      virtual void addDimAngular(const DL_DimensionData&,
85                                const DL_DimAngular2LData&) {}
86      virtual void addDimAngular3P(const DL_DimensionData&,
87                                const DL_DimAngular3PData&) {}
88      virtual void addDimOrdinate(const DL_DimensionData&,
89                               const DL_DimOrdinateData&) {}
90      virtual void addLeader(const DL_LeaderData&) {}
91      virtual void addLeaderVertex(const DL_LeaderVertexData&) {}
92
93      virtual void addHatch(const DL_HatchData&) {}
94
95      virtual void addTrace(const DL_TraceData&) {}
96      virtual void add3dFace(const DL_3dFaceData&) {}
97      virtual void addSolid(const DL_SolidData&) {}
98
99      virtual void addImage(const DL_ImageData&) {}
100     virtual void linkImage(const DL_ImageDefData&) {}
101     virtual void addHatchLoop(const DL_HatchLoopData&) {}
102     virtual void addHatchEdge(const DL_HatchEdgeData&) {}
103
104     virtual void addXRecord(const std::string&) {}
105     virtual void addXRecordString(int, const std::string&) {}
106     virtual void addXRecordReal(int, double) {}
107     virtual void addXRecordInt(int, int) {}
108     virtual void addXRecordBool(int, bool) {}
109
110     virtual void addXDataApp(const std::string&) {}
111     virtual void addXDataString(int, const std::string&) {}
112     virtual void addXDataReal(int, double) {}
113     virtual void addXDataInt(int, int) {}
114
115     virtual void addDictionary(const DL_DictionaryData&) {}
116     virtual void addDictionaryEntry(const DL_DictionaryEntryData&) {}
117
118     virtual void endEntity() {}
119
120     virtual void addComment(const std::string&) {}
121
122     virtual void setVariableVector(const std::string&,  double, double, double, int) {}
123     virtual void setVariableString(const std::string&, const std::string&, int) {}
124     virtual void setVariableInt(const std::string&, int, int) {}
125     virtual void setVariableDouble(const std::string&, double, int) {}
126 #ifdef DL_COMPAT
127     virtual void setVariableVector(const char*,  double, double, double, int) {}
128     virtual void setVariableString(const char*, const char*, int) {}
129     virtual void setVariableInt(const char*, int, int) {}
130     virtual void setVariableDouble(const char*, double, int) {}
131     virtual void processCodeValuePair(unsigned int, char*) {}
132     virtual void addComment(const char*) {}
133     virtual void addMTextChunk(const char*) {}
134 #endif
135     virtual void endSequence() {}
136 };
137
138 #endif
```

## 6.4 dl_creationinterface.h

```
1 /**********************************************************************
2 ** Copyright (C) 2001-2013 RibbonSoft, GmbH. All rights reserved.
3 **
4 ** This file is part of the dxflib project.
5 **
6 ** This file is free software; you can redistribute it and/or modify
7 ** it under the terms of the GNU General Public License as published by
8 ** the Free Software Foundation; either version 2 of the License, or
9 ** (at your option) any later version.
10 **
11 ** Licensees holding valid dxflib Professional Edition licenses may use
12 ** this file in accordance with the dxflib Commercial License
13 ** Agreement provided with the Software.
14 **
15 ** This file is provided AS IS with NO WARRANTY OF ANY KIND, INCLUDING THE
16 ** WARRANTY OF DESIGN, MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE.
17 **
18 ** See http://www.ribbonsoft.com for further details.
19 **
20 ** Contact info@ribbonsoft.com if any conditions of this licensing are
21 ** not clear to you.
22 **
23 **********************************************************************/
24
25 #ifndef DL_CREATIONINTERFACE_H
26 #define DL_CREATIONINTERFACE_H
27
28 #include "dl_global.h"
29
30 #include <string.h>
31
32 #include "dl_attributes.h"
33 #include "dl_codes.h"
34 #include "dl_entities.h"
35 #include "dl_extrusion.h"
36
47 class DXFLIB_EXPORT DL_CreationInterface {
48 public:
49     DL_CreationInterface() {
50         extrusion = new DL_Extrusion;
51     }
52     virtual ~DL_CreationInterface() {
53         delete extrusion;
54     }
55
60     virtual void processCodeValuePair(unsigned int groupCode, const std::string& groupValue) = 0;
61
65     virtual void endSection() = 0;
66
70     virtual void addLayer(const DL_LayerData& data) = 0;
71
75     virtual void addLinetype(const DL_LinetypeData& data) = 0;
76
80     virtual void addLinetypeDash(double length) = 0;
81
88     virtual void addBlock(const DL_BlockData& data) = 0;
89
91     virtual void endBlock() = 0;
92
94     virtual void addTextStyle(const DL_StyleData& data) = 0;
95
97     virtual void addPoint(const DL_PointData& data) = 0;
98
100     virtual void addLine(const DL_LineData& data) = 0;
101
103     virtual void addXLine(const DL_XLineData& data) = 0;
104
106     virtual void addRay(const DL_RayData& data) = 0;
107
109     virtual void addArc(const DL_ArcData& data) = 0;
110
112     virtual void addCircle(const DL_CircleData& data) = 0;
113
115     virtual void addEllipse(const DL_EllipseData& data) = 0;
116
118     virtual void addPolyline(const DL_PolylineData& data) = 0;
119
121     virtual void addVertex(const DL_VertexData& data) = 0;
122
124     virtual void addSpline(const DL_SplineData& data) = 0;
125
127     virtual void addControlPoint(const DL_ControlPointData& data) = 0;
128
130     virtual void addFitPoint(const DL_FitPointData& data) = 0;
131
```

```
133     virtual void addKnot(const DL_KnotData& data) = 0;
134
136     virtual void addInsert(const DL_InsertData& data) = 0;
137
139     virtual void addTrace(const DL_TraceData& data) = 0;
140
142     virtual void add3dFace(const DL_3dFaceData& data) = 0;
143
145     virtual void addSolid(const DL_SolidData& data) = 0;
146
147
149     virtual void addMText(const DL_MTextData& data) = 0;
150
156     virtual void addMTextChunk(const std::string& text) = 0;
157
159     virtual void addText(const DL_TextData& data) = 0;
160
162     virtual void addArcAlignedText(const DL_ArcAlignedTextData& data) = 0;
163
165     virtual void addAttribute(const DL_AttributeData& data) = 0;
166
170     virtual void addDimAlign(const DL_DimensionData& data,
171                              const DL_DimAlignedData& edata) = 0;
175     virtual void addDimLinear(const DL_DimensionData& data,
176                               const DL_DimLinearData& edata) = 0;
177
181     virtual void addDimRadial(const DL_DimensionData& data,
182                               const DL_DimRadialData& edata) = 0;
183
187     virtual void addDimDiametric(const DL_DimensionData& data,
188                               const DL_DimDiametricData& edata) = 0;
189
193     virtual void addDimAngular(const DL_DimensionData& data,
194                               const DL_DimAngular2LData& edata) = 0;
195
199     virtual void addDimAngular3P(const DL_DimensionData& data,
200                               const DL_DimAngular3PData& edata) = 0;
201
205     virtual void addDimOrdinate(const DL_DimensionData& data,
206                              const DL_DimOrdinateData& edata) = 0;
207
211     virtual void addLeader(const DL_LeaderData& data) = 0;
212
216     virtual void addLeaderVertex(const DL_LeaderVertexData& data) = 0;
217
221     virtual void addHatch(const DL_HatchData& data) = 0;
222
226     virtual void addImage(const DL_ImageData& data) = 0;
227
231     virtual void linkImage(const DL_ImageDefData& data) = 0;
232
236     virtual void addHatchLoop(const DL_HatchLoopData& data) = 0;
237
241     virtual void addHatchEdge(const DL_HatchEdgeData& data) = 0;
242
246     virtual void addXRecord(const std::string& handle) = 0;
247
251     virtual void addXRecordString(int code, const std::string& value) = 0;
252
256     virtual void addXRecordReal(int code, double value) = 0;
257
261     virtual void addXRecordInt(int code, int value) = 0;
262
266     virtual void addXRecordBool(int code, bool value) = 0;
267
271     virtual void addXDataApp(const std::string& appId) = 0;
272
276     virtual void addXDataString(int code, const std::string& value) = 0;
277
281     virtual void addXDataReal(int code, double value) = 0;
282
286     virtual void addXDataInt(int code, int value) = 0;
287
291     virtual void addDictionary(const DL_DictionaryData& data) = 0;
292
296     virtual void addDictionaryEntry(const DL_DictionaryEntryData& data) = 0;
297
301     virtual void endEntity() = 0;
302
306     virtual void addComment(const std::string& comment) = 0;
307
311     virtual void setVariableVector(const std::string& key,  double v1, double v2, double v3, int code) =
    0;
312
316     virtual void setVariableString(const std::string& key, const std::string& value, int code) = 0;
317
321     virtual void setVariableInt(const std::string& key, int value, int code) = 0;
```

```
322
326     virtual void setVariableDouble(const std::string& key, double value, int code) = 0;
327
328 #ifdef DL_COMPAT
329     virtual void setVariableVector(const char* key,  double v1, double v2, double v3, int code) = 0;
330     virtual void setVariableString(const char* key, const char* value, int code) = 0;
331     virtual void setVariableInt(const char* key, int value, int code) = 0;
332     virtual void setVariableDouble(const char* key, double value, int code) = 0;
333     virtual void processCodeValuePair(unsigned int groupCode, char* groupValue) = 0;
334     virtual void addComment(const char* comment) = 0;
335     virtual void addMTextChunk(const char* text) = 0;
336 #endif
337
341     virtual void endSequence() = 0;
342
344     void setAttributes(const DL_Attributes& attrib) {
345         attributes = attrib;
346     }
347
349     DL_Attributes getAttributes() {
350         return attributes;
351     }
352
354     void setExtrusion(double dx, double dy, double dz, double elevation) {
355         extrusion->setDirection(dx, dy, dz);
356         extrusion->setElevation(elevation);
357     }
358
360     DL_Extrusion* getExtrusion() {
361         return extrusion;
362     }
363
364 protected:
365     DL_Attributes attributes;
366     DL_Extrusion *extrusion;
367 };
368
369 #endif
```

## 6.5 dl_dxf.h

```
1 /**********************************************************************
2 ** Copyright (C) 2001-2013 RibbonSoft, GmbH. All rights reserved.
3 **
4 ** This file is part of the dxflib project.
5 **
6 ** This file is free software; you can redistribute it and/or modify
7 ** it under the terms of the GNU General Public License as published by
8 ** the Free Software Foundation; either version 2 of the License, or
9 ** (at your option) any later version.
10 **
11 ** Licensees holding valid dxflib Professional Edition licenses may use
12 ** this file in accordance with the dxflib Commercial License
13 ** Agreement provided with the Software.
14 **
15 ** This file is provided AS IS with NO WARRANTY OF ANY KIND, INCLUDING THE
16 ** WARRANTY OF DESIGN, MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE.
17 **
18 ** See http://www.ribbonsoft.com for further details.
19 **
20 ** Contact info@ribbonsoft.com if any conditions of this licensing are
21 ** not clear to you.
22 **
23 **********************************************************************/
24
25 #ifndef DL_DXF_H
26 #define DL_DXF_H
27
28 #include "dl_global.h"
29
30 #include <limits>
31 #include <stdio.h>
32 #include <stdlib.h>
33 #include <string>
34 #include <sstream>
35 #include <map>
36
37 #include "dl_attributes.h"
38 #include "dl_codes.h"
39 #include "dl_entities.h"
40 #include "dl_writer_ascii.h"
41
42 #ifdef _WIN32
```

```
43 #undef M_PI
44 #define M_PI   3.14159265358979323846
45 #pragma warning(disable : 4800)
46 #endif
47
48 #ifndef M_PI
49 #define M_PI 3.14159265358979323846264338327795
50 #endif
51
52 #ifndef DL_NANDOUBLE
53 #define DL_NANDOUBLE std::numeric_limits<double>::quiet_NaN()
54 #endif
55
56 class DL_CreationInterface;
57 class DL_WriterA;
58
59
60 #define DL_VERSION "3.26.4.0"
61
62 #define DL_VERSION_MAJOR    3
63 #define DL_VERSION_MINOR    26
64 #define DL_VERSION_REV      4
65 #define DL_VERSION_BUILD    0
66
67 #define DL_UNKNOWN               0
68 #define DL_LAYER                10
69 #define DL_BLOCK                11
70 #define DL_ENDBLK               12
71 #define DL_LINETYPE             13
72 #define DL_STYLE                20
73 #define DL_SETTING              50
74 #define DL_ENTITY_POINT        100
75 #define DL_ENTITY_LINE         101
76 #define DL_ENTITY_POLYLINE     102
77 #define DL_ENTITY_LWPOLYLINE   103
78 #define DL_ENTITY_VERTEX       104
79 #define DL_ENTITY_SPLINE       105
80 #define DL_ENTITY_KNOT         106
81 #define DL_ENTITY_CONTROLPOINT 107
82 #define DL_ENTITY_ARC          108
83 #define DL_ENTITY_CIRCLE       109
84 #define DL_ENTITY_ELLIPSE      110
85 #define DL_ENTITY_INSERT       111
86 #define DL_ENTITY_TEXT         112
87 #define DL_ENTITY_MTEXT        113
88 #define DL_ENTITY_DIMENSION    114
89 #define DL_ENTITY_LEADER       115
90 #define DL_ENTITY_HATCH        116
91 #define DL_ENTITY_ATTRIB       117
92 #define DL_ENTITY_IMAGE        118
93 #define DL_ENTITY_IMAGEDEF     119
94 #define DL_ENTITY_TRACE        120
95 #define DL_ENTITY_SOLID        121
96 #define DL_ENTITY_3DFACE       122
97 #define DL_ENTITY_XLINE        123
98 #define DL_ENTITY_RAY          124
99 #define DL_ENTITY_ARCALIGNEDTEXT 125
100 #define DL_ENTITY_SEQEND       126
101 #define DL_XRECORD             200
102 #define DL_DICTIONARY          210
103
104
122 class DXFLIB_EXPORT DL_Dxf {
123 public:
124     DL_Dxf();
125     ~DL_Dxf();
126
127     bool in(const std::string& file,
128             DL_CreationInterface* creationInterface);
129     bool readDxfGroups(FILE* fp,
130                        DL_CreationInterface* creationInterface);
131     static bool getStrippedLine(std::string& s, unsigned int size,
132                                 FILE* stream, bool stripSpace = true);
133
134     bool readDxfGroups(std::istream& stream,
135                        DL_CreationInterface* creationInterface);
136     bool in(std::istream &stream,
137             DL_CreationInterface* creationInterface);
138     static bool getStrippedLine(std::string& s, unsigned int size,
139                                 std::istream& stream, bool stripSpace = true);
140
141     static bool stripWhiteSpace(char** s, bool stripSpaces = true);
142
143     bool processDXFGroup(DL_CreationInterface* creationInterface,
144                          int groupCode, const std::string& groupValue);
145     void addSetting(DL_CreationInterface* creationInterface);
146     void addLayer(DL_CreationInterface* creationInterface);
```

```
147     void addLinetype(DL_CreationInterface *creationInterface);
148     void addBlock(DL_CreationInterface* creationInterface);
149     void endBlock(DL_CreationInterface* creationInterface);
150     void addTextStyle(DL_CreationInterface* creationInterface);
151
152     void addPoint(DL_CreationInterface* creationInterface);
153     void addLine(DL_CreationInterface* creationInterface);
154     void addXLine(DL_CreationInterface* creationInterface);
155     void addRay(DL_CreationInterface* creationInterface);
156
157     void addPolyline(DL_CreationInterface* creationInterface);
158     void addVertex(DL_CreationInterface* creationInterface);
159
160     void addSpline(DL_CreationInterface* creationInterface);
161
162     void addArc(DL_CreationInterface* creationInterface);
163     void addCircle(DL_CreationInterface* creationInterface);
164     void addEllipse(DL_CreationInterface* creationInterface);
165     void addInsert(DL_CreationInterface* creationInterface);
166
167     void addTrace(DL_CreationInterface* creationInterface);
168     void add3dFace(DL_CreationInterface* creationInterface);
169     void addSolid(DL_CreationInterface* creationInterface);
170
171     void addMText(DL_CreationInterface* creationInterface);
172     void addText(DL_CreationInterface* creationInterface);
173     void addArcAlignedText(DL_CreationInterface* creationInterface);
174
175     void addAttribute(DL_CreationInterface* creationInterface);
176
177     DL_DimensionData getDimData();
178     void addDimLinear(DL_CreationInterface* creationInterface);
179     void addDimAligned(DL_CreationInterface* creationInterface);
180     void addDimRadial(DL_CreationInterface* creationInterface);
181     void addDimDiametric(DL_CreationInterface* creationInterface);
182     void addDimAngular(DL_CreationInterface* creationInterface);
183     void addDimAngular3P(DL_CreationInterface* creationInterface);
184     void addDimOrdinate(DL_CreationInterface* creationInterface);
185
186     void addLeader(DL_CreationInterface* creationInterface);
187
188     void addHatch(DL_CreationInterface* creationInterface);
189     void addHatchLoop();
190     void addHatchEdge();
191     bool handleHatchData(DL_CreationInterface* creationInterface);
192
193     void addImage(DL_CreationInterface* creationInterface);
194     void addImageDef(DL_CreationInterface* creationInterface);
195
196     void addComment(DL_CreationInterface* creationInterface, const std::string& comment);
197
198     void addDictionary(DL_CreationInterface* creationInterface);
199     void addDictionaryEntry(DL_CreationInterface* creationInterface);
200
201     bool handleXRecordData(DL_CreationInterface* creationInterface);
202     bool handleDictionaryData(DL_CreationInterface* creationInterface);
203
204     bool handleXData(DL_CreationInterface *creationInterface);
205     bool handleMTextData(DL_CreationInterface* creationInterface);
206     bool handleLWPolylineData(DL_CreationInterface* creationInterface);
207     bool handleSplineData(DL_CreationInterface* creationInterface);
208     bool handleLeaderData(DL_CreationInterface* creationInterface);
209     bool handleLinetypeData(DL_CreationInterface* creationInterface);
210
211     void endEntity(DL_CreationInterface* creationInterface);
212
213     void endSequence(DL_CreationInterface* creationInterface);
214
215     //int  stringToInt(const char* s, bool* ok=NULL);
216
217     DL_WriterA* out(const char* file,
218                     DL_Codes::version version=DL_VERSION_2000);
219
220     void writeHeader(DL_WriterA& dw);
221
222     void writePoint(DL_WriterA& dw,
223                     const DL_PointData& data,
224                     const DL_Attributes& attrib);
225     void writeLine(DL_WriterA& dw,
226                    const DL_LineData& data,
227                    const DL_Attributes& attrib);
228     void writeXLine(DL_WriterA& dw,
229                    const DL_XLineData& data,
230                    const DL_Attributes& attrib);
231     void writeRay(DL_WriterA& dw,
232                     const DL_RayData& data,
233                     const DL_Attributes& attrib);
```

```
234     void writePolyline(DL_WriterA& dw,
235                        const DL_PolylineData& data,
236                        const DL_Attributes& attrib);
237     void writeVertex(DL_WriterA& dw,
238                      const DL_VertexData& data);
239     void writePolylineEnd(DL_WriterA& dw);
240     void writeSpline(DL_WriterA& dw,
241                      const DL_SplineData& data,
242                      const DL_Attributes& attrib);
243     void writeControlPoint(DL_WriterA& dw,
244                    const DL_ControlPointData& data);
245     void writeFitPoint(DL_WriterA& dw,
246                      const DL_FitPointData& data);
247     void writeKnot(DL_WriterA& dw,
248                    const DL_KnotData& data);
249     void writeCircle(DL_WriterA& dw,
250                      const DL_CircleData& data,
251                      const DL_Attributes& attrib);
252     void writeArc(DL_WriterA& dw,
253                   const DL_ArcData& data,
254                   const DL_Attributes& attrib);
255     void writeEllipse(DL_WriterA& dw,
256                       const DL_EllipseData& data,
257                       const DL_Attributes& attrib);
258     void writeSolid(DL_WriterA& dw,
259                     const DL_SolidData& data,
260                     const DL_Attributes& attrib);
261     void writeTrace(DL_WriterA& dw,
262                     const DL_TraceData& data,
263                     const DL_Attributes& attrib);
264     void write3dFace(DL_WriterA& dw,
265                      const DL_3dFaceData& data,
266                      const DL_Attributes& attrib);
267     void writeInsert(DL_WriterA& dw,
268                      const DL_InsertData& data,
269                      const DL_Attributes& attrib);
270     void writeMText(DL_WriterA& dw,
271                     const DL_MTextData& data,
272                     const DL_Attributes& attrib);
273     void writeText(DL_WriterA& dw,
274                    const DL_TextData& data,
275                    const DL_Attributes& attrib);
276     void writeAttribute(DL_WriterA& dw,
277                    const DL_AttributeData& data,
278                    const DL_Attributes& attrib);
279     void writeDimStyleOverrides(DL_WriterA& dw,
280                                 const DL_DimensionData& data);
281     void writeDimAligned(DL_WriterA& dw,
282                          const DL_DimensionData& data,
283                          const DL_DimAlignedData& edata,
284                          const DL_Attributes& attrib);
285     void writeDimLinear(DL_WriterA& dw,
286                         const DL_DimensionData& data,
287                         const DL_DimLinearData& edata,
288                         const DL_Attributes& attrib);
289     void writeDimRadial(DL_WriterA& dw,
290                         const DL_DimensionData& data,
291                         const DL_DimRadialData& edata,
292                         const DL_Attributes& attrib);
293     void writeDimDiametric(DL_WriterA& dw,
294                            const DL_DimensionData& data,
295                            const DL_DimDiametricData& edata,
296                            const DL_Attributes& attrib);
297     void writeDimAngular2L(DL_WriterA& dw,
298                            const DL_DimensionData& data,
299                            const DL_DimAngular2LData& edata,
300                            const DL_Attributes& attrib);
301     void writeDimAngular3P(DL_WriterA& dw,
302                            const DL_DimensionData& data,
303                            const DL_DimAngular3PData& edata,
304                            const DL_Attributes& attrib);
305     void writeDimOrdinate(DL_WriterA& dw,
306                           const DL_DimensionData& data,
307                           const DL_DimOrdinateData& edata,
308                           const DL_Attributes& attrib);
309     void writeLeader(DL_WriterA& dw,
310                      const DL_LeaderData& data,
311                      const DL_Attributes& attrib);
312     void writeLeaderVertex(DL_WriterA& dw,
313                            const DL_LeaderVertexData& data);
314     void writeLeaderEnd(DL_WriterA& dw,
315                         const DL_LeaderData& data);
316     void writeHatch1(DL_WriterA& dw,
317                      const DL_HatchData& data,
318                      const DL_Attributes& attrib);
319     void writeHatch2(DL_WriterA& dw,
320                      const DL_HatchData& data,
```

```
321                     const DL_Attributes& attrib);
322     void writeHatchLoop1(DL_WriterA& dw,
323                          const DL_HatchLoopData& data);
324     void writeHatchLoop2(DL_WriterA& dw,
325                          const DL_HatchLoopData& data);
326     void writeHatchEdge(DL_WriterA& dw,
327                         const DL_HatchEdgeData& data);
328
329     unsigned long writeImage(DL_WriterA& dw,
330                              const DL_ImageData& data,
331                              const DL_Attributes& attrib);
332
333     void writeImageDef(DL_WriterA& dw, int handle,
334                        const DL_ImageData& data);
335
336     void writeLayer(DL_WriterA& dw,
337                     const DL_LayerData& data,
338                     const DL_Attributes& attrib);
339
340     void writeLinetype(DL_WriterA& dw,
341                        const DL_LinetypeData& data);
342
343     void writeAppid(DL_WriterA& dw, const std::string& name);
344
345     void writeBlock(DL_WriterA& dw,
346                     const DL_BlockData& data);
347     void writeEndBlock(DL_WriterA& dw, const std::string& name);
348
349     void writeVPort(DL_WriterA& dw);
350     void writeStyle(DL_WriterA& dw, const DL_StyleData& style);
351     void writeView(DL_WriterA& dw);
352     void writeUcs(DL_WriterA& dw);
353     void writeDimStyle(DL_WriterA& dw,
354                        double dimasz, double dimexe, double dimexo,
355                        double dimgap, double dimtxt);
356     void writeBlockRecord(DL_WriterA& dw);
357     void writeBlockRecord(DL_WriterA& dw, const std::string& name);
358     void writeObjects(DL_WriterA& dw, const std::string& appDictionaryName = "");
359     void writeAppDictionary(DL_WriterA& dw);
360     unsigned long writeDictionaryEntry(DL_WriterA& dw, const std::string& name);
361     void writeXRecord(DL_WriterA& dw, int handle, int value);
362     void writeXRecord(DL_WriterA& dw, int handle, double value);
363     void writeXRecord(DL_WriterA& dw, int handle, bool value);
364     void writeXRecord(DL_WriterA& dw, int handle, const std::string& value);
365     void writeObjectsEnd(DL_WriterA& dw);
366
367     void writeComment(DL_WriterA& dw, const std::string& comment);
368
373     //static double toReal(const char* value, double def=0.0);
374
379 //    static int toInt(const char* value, int def=0) {
380 //        if (value!=NULL && value[0] != '\0') {
381 //            return atoi(value);
382 //        }
383
384 //        return def;
385 //    }
386
391 //    static const char* toString(const char* value, const char* def="") {
392 //        if (value!=NULL && value[0] != '\0') {
393 //            return value;
394 //        } else {
395 //            return def;
396 //        }
397 //    }
398
399     static bool checkVariable(const char* var, DL_Codes::version version);
400
401     DL_Codes::version getVersion() {
402         return version;
403     }
404
405     int getLibVersion(const std::string &str);
406
407     static void test();
408
409     bool hasValue(int code) {
410         return values.count(code)==1;
411     }
412
413     int getIntValue(int code, int def) {
414         if (!hasValue(code)) {
415             return def;
416         }
417         return toInt(values[code]);
418     }
419
```

```
420     int toInt(const std::string& str) {
421         char* p;
422         return strtol(str.c_str(), &p, 10);
423     }
424 
425     int getInt16Value(int code, int def) {
426         if (!hasValue(code)) {
427             return def;
428         }
429         return toInt16(values[code]);
430     }
431 
432     int toInt16(const std::string& str) {
433         char* p;
434         return strtol(str.c_str(), &p, 16);
435     }
436 
437     bool toBool(const std::string& str) {
438         char* p;
439         return (bool)strtol(str.c_str(), &p, 10);
440     }
441 
442     std::string getStringValue(int code, const std::string& def) {
443         if (!hasValue(code)) {
444             return def;
445         }
446         return values[code];
447     }
448 
449     double getRealValue(int code, double def) {
450         if (!hasValue(code)) {
451             return def;
452         }
453         return toReal(values[code]);
454     }
455 
456     double toReal(const std::string& str) {
457         double ret;
458         // make sure the real value uses '.' not ',':
459         std::string str2 = str;
460         std::replace(str2.begin(), str2.end(), ',', '.');
461         // make sure c++ expects '.' not ',':
462         std::istringstream istr(str2);
463         //istr.imbue(std::locale("C"));
464         istr >> ret;
465         return ret;
466     }
467 
468 private:
469     DL_Codes::version version;
470 
471     std::string polylineLayer;
472     double* vertices;
473     int maxVertices;
474     int vertexIndex;
475 
476     double* knots;
477     int maxKnots;
478     int knotIndex;
479 
480     double* weights;
481     int weightIndex;
482 
483     double* controlPoints;
484     int maxControlPoints;
485     int controlPointIndex;
486 
487     double* fitPoints;
488     int maxFitPoints;
489     int fitPointIndex;
490 
491     double* leaderVertices;
492     int maxLeaderVertices;
493     int leaderVertexIndex;
494 
495     bool firstHatchLoop;
496     DL_HatchEdgeData hatchEdge;
497     std::vector<std::vector<DL_HatchEdgeData> > hatchEdges;
498 
499     std::string xRecordHandle;
500     bool xRecordValues;
501 
502     // Only the useful part of the group code
503     std::string groupCodeTmp;
504     // ...same as integer
505     unsigned int groupCode;
506     // Only the useful part of the group value
```

```
507     std::string groupValue;
508     // Current entity type
509     int currentObjectType;
510     // Value of the current setting
511     char settingValue[DL_DXF_MAXLINE+1];
512     // Key of the current setting (e.g. "$ACADVER")
513     std::string settingKey;
514     // Stores the group codes
515     std::map<int, std::string> values;
516     // First call of this method. We initialize all group values in
517     //  the first call.
518     bool firstCall;
519     // Attributes of the current entity (layer, color, width, line type)
520     DL_Attributes attrib;
521     // library version. hex: 0x20003001 = 2.0.3.1
522     int libVersion;
523     // app specific dictionary handle:
524     unsigned long appDictionaryHandle;
525     // handle of standard text style, referenced by dimstyle:
526     unsigned long styleHandleStd;
527 };
528
529 #endif
530
531 // EOF
```

## 6.6 dl_entities.h

```
1  /**********************************************************************
2  ** Copyright (C) 2001-2013 RibbonSoft, GmbH. All rights reserved.
3  **
4  ** This file is part of the dxflib project.
5  **
6  ** This file is free software; you can redistribute it and/or modify
7  ** it under the terms of the GNU General Public License as published by
8  ** the Free Software Foundation; either version 2 of the License, or
9  ** (at your option) any later version.
10 **
11 ** Licensees holding valid dxflib Professional Edition licenses may use
12 ** this file in accordance with the dxflib Commercial License
13 ** Agreement provided with the Software.
14 **
15 ** This file is provided AS IS with NO WARRANTY OF ANY KIND, INCLUDING THE
16 ** WARRANTY OF DESIGN, MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE.
17 **
18 ** See http://www.ribbonsoft.com for further details.
19 **
20 ** Contact info@ribbonsoft.com if any conditions of this licensing are
21 ** not clear to you.
22 **
23 **********************************************************************/
24
25 #ifndef DL_ENTITIES_H
26 #define DL_ENTITIES_H
27
28 #include "dl_global.h"
29
30 #include <string>
31 #include <vector>
32
36 struct DXFLIB_EXPORT DL_LayerData {
41     DL_LayerData(const std::string& name,
42                  int flags, bool off = false) :
43         name(name), flags(flags), off(off) {
44     }
45
47     std::string name;
49     int flags;
51     bool off;
52 };
53
54
55
59 struct DXFLIB_EXPORT DL_BlockData {
64     DL_BlockData(const std::string& bName,
65                  int bFlags,
66                  double bbpx, double bbpy, double bbpz) {
67         name = bName;
68         flags = bFlags;
69         bpx = bbpx;
70         bpy = bbpy;
71         bpz = bbpz;
72     }
```

```
73
75     std::string name;
77     int flags;
79     double bpx;
81     double bpy;
83     double bpz;
84 };
85
86
90 struct DXFLIB_EXPORT DL_LinetypeData {
95     DL_LinetypeData(
96         const std::string& name,
97         const std::string& description,
98         int flags,
99         int numberOfDashes,
100         double patternLength,
101         double* pattern = NULL
102         )
103         : name(name),
104         description(description),
105         flags(flags),
106         numberOfDashes(numberOfDashes),
107         patternLength(patternLength),
108         pattern(pattern)
109     {}
110
112     std::string name;
114     std::string description;
116     int flags;
118     int numberOfDashes;
120     double patternLength;
122     double* pattern;
123 };
124
125
126
130 struct DXFLIB_EXPORT DL_StyleData {
135     DL_StyleData(
136         const std::string& name,
137         int flags,
138         double fixedTextHeight,
139         double widthFactor,
140         double obliqueAngle,
141         int textGenerationFlags,
142         double lastHeightUsed,
143         const std::string& primaryFontFile,
144         const std::string& bigFontFile
145         )
146         : name(name),
147         flags(flags),
148         fixedTextHeight(fixedTextHeight),
149         widthFactor(widthFactor),
150         obliqueAngle(obliqueAngle),
151         textGenerationFlags(textGenerationFlags),
152         lastHeightUsed(lastHeightUsed),
153         primaryFontFile(primaryFontFile),
154         bigFontFile(bigFontFile),
155         bold(false),
156         italic(false) {
157     }
158
159     bool operator==(const DL_StyleData& other) {
160         // ignore lastHeightUsed:
161         return (name==other.name &&
162             flags==other.flags &&
163             fixedTextHeight==other.fixedTextHeight &&
164             widthFactor==other.widthFactor &&
165             obliqueAngle==other.obliqueAngle &&
166             textGenerationFlags==other.textGenerationFlags &&
167             primaryFontFile==other.primaryFontFile &&
168             bigFontFile==other.bigFontFile);
169     }
170
172     std::string name;
174     int flags;
176     double fixedTextHeight;
178     double widthFactor;
180     double obliqueAngle;
182     int textGenerationFlags;
184     double lastHeightUsed;
186     std::string primaryFontFile;
188     std::string bigFontFile;
189
190     bool bold;
191     bool italic;
192 };
193
```

```
197 struct DXFLIB_EXPORT DL_PointData {
202     DL_PointData(double px=0.0, double py=0.0, double pz=0.0) {
203         x = px;
204         y = py;
205         z = pz;
206     }
207
209     double x;
211     double y;
213     double z;
214 };
215
216
217
221 struct DXFLIB_EXPORT DL_LineData {
226     DL_LineData(double lx1, double ly1, double lz1,
227                 double lx2, double ly2, double lz2) {
228         x1 = lx1;
229         y1 = ly1;
230         z1 = lz1;
231
232         x2 = lx2;
233         y2 = ly2;
234         z2 = lz2;
235     }
236
238     double x1;
240     double y1;
242     double z1;
243
245     double x2;
247     double y2;
249     double z2;
250 };
251
255 struct DXFLIB_EXPORT DL_XLineData {
260     DL_XLineData(double bx, double by, double bz,
261                  double dx, double dy, double dz) :
262         bx(bx), by(by), bz(bz),
263         dx(dx), dy(dy), dz(dz) {
264     }
265
267     double bx;
269     double by;
271     double bz;
272
274     double dx;
276     double dy;
278     double dz;
279 };
280
284 struct DXFLIB_EXPORT DL_RayData {
289     DL_RayData(double bx, double by, double bz,
290                double dx, double dy, double dz) :
291         bx(bx), by(by), bz(bz),
292         dx(dx), dy(dy), dz(dz) {
293     }
294
296     double bx;
298     double by;
300     double bz;
301
303     double dx;
305     double dy;
307     double dz;
308 };
309
310
311
315 struct DXFLIB_EXPORT DL_ArcData {
320     DL_ArcData(double acx, double acy, double acz,
321                double aRadius,
322                double aAngle1, double aAngle2) {
323
324         cx = acx;
325         cy = acy;
326         cz = acz;
327         radius = aRadius;
328         angle1 = aAngle1;
329         angle2 = aAngle2;
330     }
331
333     double cx;
335     double cy;
337     double cz;
338
340     double radius;
```

```
342     double angle1;
344     double angle2;
345 };
346
347
348
352 struct DXFLIB_EXPORT DL_CircleData {
357     DL_CircleData(double acx, double acy, double acz,
358                   double aRadius) {
359
360         cx = acx;
361         cy = acy;
362         cz = acz;
363         radius = aRadius;
364     }
365
367     double cx;
369     double cy;
371     double cz;
372
374     double radius;
375 };
376
377
378
382 struct DXFLIB_EXPORT DL_PolylineData {
387     DL_PolylineData(int pNumber, int pMVerteces, int pNVerteces, int pFlags, double pElevation = 0.0) {
388         number = pNumber;
389         m = pMVerteces;
390         n = pNVerteces;
391         elevation = pElevation;
392         flags = pFlags;
393     }
394
396     unsigned int number;
397
399     unsigned int m;
400
402     unsigned int n;
403
405     double elevation;
406
408     int flags;
409 };
410
411
412
416 struct DXFLIB_EXPORT DL_VertexData {
421     DL_VertexData(double px=0.0, double py=0.0, double pz=0.0,
422                   double pBulge=0.0) {
423         x = px;
424         y = py;
425         z = pz;
426         bulge = pBulge;
427     }
428
430     double x;
432     double y;
434     double z;
437     double bulge;
438 };
439
440
444 struct DXFLIB_EXPORT DL_TraceData {
445     DL_TraceData() {
446         thickness = 0.0;
447         for (int i=0; i<4; i++) {
448             x[i] = 0.0;
449             y[i] = 0.0;
450             z[i] = 0.0;
451         }
452     }
453
458     DL_TraceData(double sx1, double sy1, double sz1,
459                  double sx2, double sy2, double sz2,
460                  double sx3, double sy3, double sz3,
461                  double sx4, double sy4, double sz4,
462                  double sthickness=0.0) {
463
464         thickness = sthickness;
465
466         x[0] = sx1;
467         y[0] = sy1;
468         z[0] = sz1;
469
470         x[1] = sx2;
471         y[1] = sy2;
```

```
472         z[1] = sz2;
473
474         x[2] = sx3;
475         y[2] = sy3;
476         z[2] = sz3;
477
478         x[3] = sx4;
479         y[3] = sy4;
480         z[3] = sz4;
481     }
482
484     double thickness;
485
487     double x[4];
488     double y[4];
489     double z[4];
490 };
491
492
493
494
495
499 typedef DL_TraceData DL_SolidData;
500
501
505 typedef DL_TraceData DL_3dFaceData;
506
507
511 struct DXFLIB_EXPORT DL_SplineData {
516     DL_SplineData(int degree,
517                   int nKnots,
518                   int nControl,
519                   int nFit,
520                   int flags) :
521         degree(degree),
522         nKnots(nKnots),
523         nControl(nControl),
524         nFit(nFit),
525         flags(flags) {
526     }
527
529     unsigned int degree;
530
532     unsigned int nKnots;
533
535     unsigned int nControl;
536
538     unsigned int nFit;
539
541     int flags;
542
543     double tangentStartX;
544     double tangentStartY;
545     double tangentStartZ;
546     double tangentEndX;
547     double tangentEndY;
548     double tangentEndZ;
549 };
550
551
552
556 struct DXFLIB_EXPORT DL_KnotData {
557     DL_KnotData() {}
562     DL_KnotData(double pk) {
563         k = pk;
564     }
565
567     double k;
568 };
569
570
571
575 struct DXFLIB_EXPORT DL_ControlPointData {
580     DL_ControlPointData(double px, double py, double pz, double weight) {
581         x = px;
582         y = py;
583         z = pz;
584         w = weight;
585     }
586
588     double x;
590     double y;
592     double z;
594     double w;
595 };
596
597
```

```
598
602 struct DXFLIB_EXPORT DL_FitPointData {
607     DL_FitPointData(double x, double y, double z) : x(x), y(y), z(z) {}
608
610     double x;
612     double y;
614     double z;
615 };
616
617
618
622 struct DXFLIB_EXPORT DL_EllipseData {
627     DL_EllipseData(double cx, double cy, double cz,
628                    double mx, double my, double mz,
629                    double ratio,
630                    double angle1, double angle2)
631         : cx(cx),
632           cy(cy),
633           cz(cz),
634           mx(mx),
635           my(my),
636           mz(mz),
637           ratio(ratio),
638           angle1(angle1),
639           angle2(angle2) {
640     }
641
643     double cx;
645     double cy;
647     double cz;
648
650     double mx;
652     double my;
654     double mz;
655
657     double ratio;
659     double angle1;
661     double angle2;
662 };
663
664
665
669 struct DXFLIB_EXPORT DL_InsertData {
674     DL_InsertData(const std::string& name,
675                   double ipx, double ipy, double ipz,
676                   double sx, double sy, double sz,
677                   double angle,
678                   int cols, int rows,
679                   double colSp, double rowSp) :
680           name(name),
681           ipx(ipx), ipy(ipy), ipz(ipz),
682           sx(sx), sy(sy), sz(sz),
683           angle(angle),
684           cols(cols), rows(rows),
685           colSp(colSp), rowSp(rowSp) {
686     }
687
689     std::string name;
691     double ipx;
693     double ipy;
695     double ipz;
697     double sx;
699     double sy;
701     double sz;
703     double angle;
705     int cols;
707     int rows;
709     double colSp;
711     double rowSp;
712 };
713
714
715
719 struct DXFLIB_EXPORT DL_MTextData {
724     DL_MTextData(double ipx, double ipy, double ipz,
725                  double dirx, double diry, double dirz,
726                  double height, double width,
727                  int attachmentPoint,
728                  int drawingDirection,
729                  int lineSpacingStyle,
730                  double lineSpacingFactor,
731                  const std::string& text,
732                  const std::string& style,
733                  double angle) :
734         ipx(ipx), ipy(ipy), ipz(ipz),
735         dirx(dirx), diry(diry), dirz(dirz),
736         height(height), width(width),
```

```
737             attachmentPoint(attachmentPoint),
738             drawingDirection(drawingDirection),
739             lineSpacingStyle(lineSpacingStyle),
740             lineSpacingFactor(lineSpacingFactor),
741             text(text),
742             style(style),
743             angle(angle) {
744
745         }
746
748         double ipx;
750         double ipy;
752         double ipz;
754         double dirx;
756         double diry;
758         double dirz;
760         double height;
762         double width;
770         int attachmentPoint;
776         int drawingDirection;
782         int lineSpacingStyle;
786         double lineSpacingFactor;
788         std::string text;
790         std::string style;
792         double angle;
793 };
794
795
796
800 struct DXFLIB_EXPORT DL_TextData {
805     DL_TextData(double ipx, double ipy, double ipz,
806                 double apx, double apy, double apz,
807                 double height, double xScaleFactor,
808                 int textGenerationFlags,
809                 int hJustification,
810                 int vJustification,
811                 const std::string& text,
812                 const std::string& style,
813                 double angle)
814         : ipx(ipx), ipy(ipy), ipz(ipz),
815           apx(apx), apy(apy), apz(apz),
816           height(height), xScaleFactor(xScaleFactor),
817           textGenerationFlags(textGenerationFlags),
818           hJustification(hJustification),
819           vJustification(vJustification),
820           text(text),
821           style(style),
822           angle(angle) {
823     }
824
826     double ipx;
828     double ipy;
830     double ipz;
831
833     double apx;
835     double apy;
837     double apz;
838
840     double height;
842     double xScaleFactor;
844     int textGenerationFlags;
852     int hJustification;
858     int vJustification;
860     std::string text;
862     std::string style;
864     double angle;
865 };
866
870 struct DXFLIB_EXPORT DL_ArcAlignedTextData {
871
873     std::string text;
875     std::string font;
877     std::string style;
878
880     double cx;
882     double cy;
884     double cz;
886     double radius;
887
889     double xScaleFactor;
891     double height;
893     double spacing;
895     double offset;
897     double rightOffset;
899     double leftOffset;
901     double startAngle;
903     double endAngle;
```

```
908     bool reversedCharacterOrder;
913     int direction;
920     int alignment;
925     int side;
927     bool bold;
929     bool italic;
931     bool underline;
933     int characerSet;
935     int pitch;
940     bool shxFont;
942     bool wizard;
944     int arcHandle;
945 };
946
950 struct DXFLIB_EXPORT DL_AttributeData : public DL_TextData {
951     DL_AttributeData(const DL_TextData& tData, const std::string& tag)
952         : DL_TextData(tData), tag(tag) {
953
954     }
955
960     DL_AttributeData(double ipx, double ipy, double ipz,
961                 double apx, double apy, double apz,
962                 double height, double xScaleFactor,
963                 int textGenerationFlags,
964                 int hJustification,
965                 int vJustification,
966                 const std::string& tag,
967                 const std::string& text,
968                 const std::string& style,
969                 double angle)
970         : DL_TextData(ipx, ipy, ipz,
971                 apx, apy, apz,
972                 height, xScaleFactor,
973                 textGenerationFlags,
974                 hJustification,
975                 vJustification,
976                 text,
977                 style,
978                 angle),
979           tag(tag) {
980     }
981
983     std::string tag;
984 };
985
986
990 struct DXFLIB_EXPORT DL_DimensionData {
995     DL_DimensionData(double dpx, double dpy, double dpz,
996                      double mpx, double mpy, double mpz,
997                      int type,
998                      int attachmentPoint,
999                      int lineSpacingStyle,
1000                      double lineSpacingFactor,
1001                      const std::string& text,
1002                      const std::string& style,
1003                      double angle,
1004                      double linearFactor = 1.0,
1005                      double dimScale = 1.0) :
1006         dpx(dpx), dpy(dpy), dpz(dpz),
1007         mpx(mpx), mpy(mpy), mpz(mpz),
1008         type(type),
1009         attachmentPoint(attachmentPoint),
1010         lineSpacingStyle(lineSpacingStyle),
1011         lineSpacingFactor(lineSpacingFactor),
1012         text(text),
1013         style(style),
1014         angle(angle),
1015         linearFactor(linearFactor),
1016         dimScale(dimScale) {
1017
1018     }
1019
1021     double dpx;
1023     double dpy;
1025     double dpz;
1027     double mpx;
1029     double mpy;
1031     double mpz;
1051     int type;
1059     int attachmentPoint;
1065     int lineSpacingStyle;
1069     double lineSpacingFactor;
1077     std::string text;
1079     std::string style;
1084     double angle;
1088     double linearFactor;
1092     double dimScale;
```

```
1093     bool arrow1Flipped;
1094     bool arrow2Flipped;
1095 };
1096
1097
1098
1102 struct DXFLIB_EXPORT DL_DimAlignedData {
1107     DL_DimAlignedData(double depx1, double depy1, double depz1,
1108                       double depx2, double depy2, double depz2) {
1109
1110         epx1 = depx1;
1111         epy1 = depy1;
1112         epz1 = depz1;
1113
1114         epx2 = depx2;
1115         epy2 = depy2;
1116         epz2 = depz2;
1117     }
1118
1120     double epx1;
1122     double epy1;
1124     double epz1;
1125
1127     double epx2;
1129     double epy2;
1131     double epz2;
1132 };
1133
1134
1135
1139 struct DXFLIB_EXPORT DL_DimLinearData {
1144     DL_DimLinearData(double ddpx1, double ddpy1, double ddpz1,
1145                      double ddpx2, double ddpy2, double ddpz2,
1146                      double dAngle, double dOblique) {
1147
1148         dpx1 = ddpx1;
1149         dpy1 = ddpy1;
1150         dpz1 = ddpz1;
1151
1152         dpx2 = ddpx2;
1153         dpy2 = ddpy2;
1154         dpz2 = ddpz2;
1155
1156         angle = dAngle;
1157         oblique = dOblique;
1158     }
1159
1161     double dpx1;
1163     double dpy1;
1165     double dpz1;
1166
1168     double dpx2;
1170     double dpy2;
1172     double dpz2;
1173
1175     double angle;
1177     double oblique;
1178 };
1179
1180
1181
1185 struct DXFLIB_EXPORT DL_DimRadialData {
1190     DL_DimRadialData(double ddpx, double ddpy, double ddpz, double dleader) {
1191         dpx = ddpx;
1192         dpy = ddpy;
1193         dpz = ddpz;
1194
1195         leader = dleader;
1196     }
1197
1199     double dpx;
1201     double dpy;
1203     double dpz;
1204
1206     double leader;
1207 };
1208
1209
1210
1214 struct DXFLIB_EXPORT DL_DimDiametricData {
1219     DL_DimDiametricData(double ddpx, double ddpy, double ddpz, double dleader) {
1220         dpx = ddpx;
1221         dpy = ddpy;
1222         dpz = ddpz;
1223
1224         leader = dleader;
1225     }
```

```
1226
1228     double dpx;
1230     double dpy;
1232     double dpz;
1233
1235     double leader;
1236 };
1237
1238
1239
1243 struct DXFLIB_EXPORT DL_DimAngular2LData {
1248     DL_DimAngular2LData(double ddpx1, double ddpy1, double ddpz1,
1249                         double ddpx2, double ddpy2, double ddpz2,
1250                         double ddpx3, double ddpy3, double ddpz3,
1251                         double ddpx4, double ddpy4, double ddpz4) {
1252
1253         dpx1 = ddpx1;
1254         dpy1 = ddpy1;
1255         dpz1 = ddpz1;
1256
1257         dpx2 = ddpx2;
1258         dpy2 = ddpy2;
1259         dpz2 = ddpz2;
1260
1261         dpx3 = ddpx3;
1262         dpy3 = ddpy3;
1263         dpz3 = ddpz3;
1264
1265         dpx4 = ddpx4;
1266         dpy4 = ddpy4;
1267         dpz4 = ddpz4;
1268     }
1269
1271     double dpx1;
1273     double dpy1;
1275     double dpz1;
1276
1278     double dpx2;
1280     double dpy2;
1282     double dpz2;
1283
1285     double dpx3;
1287     double dpy3;
1289     double dpz3;
1290
1292     double dpx4;
1294     double dpy4;
1296     double dpz4;
1297 };
1298
1299
1303 struct DXFLIB_EXPORT DL_DimAngular3PData {
1308     DL_DimAngular3PData(double ddpx1, double ddpy1, double ddpz1,
1309                         double ddpx2, double ddpy2, double ddpz2,
1310                         double ddpx3, double ddpy3, double ddpz3) {
1311
1312         dpx1 = ddpx1;
1313         dpy1 = ddpy1;
1314         dpz1 = ddpz1;
1315
1316         dpx2 = ddpx2;
1317         dpy2 = ddpy2;
1318         dpz2 = ddpz2;
1319
1320         dpx3 = ddpx3;
1321         dpy3 = ddpy3;
1322         dpz3 = ddpz3;
1323     }
1324
1326     double dpx1;
1328     double dpy1;
1330     double dpz1;
1331
1333     double dpx2;
1335     double dpy2;
1337     double dpz2;
1338
1340     double dpx3;
1342     double dpy3;
1344     double dpz3;
1345 };
1346
1347
1348
1352 struct DXFLIB_EXPORT DL_DimOrdinateData {
1357     DL_DimOrdinateData(double ddpx1, double ddpy1, double ddpz1,
1358                        double ddpx2, double ddpy2, double ddpz2,
```

```
1359                        bool dxtype) {
1360
1361         dpx1 = ddpx1;
1362         dpy1 = ddpy1;
1363         dpz1 = ddpz1;
1364
1365         dpx2 = ddpx2;
1366         dpy2 = ddpy2;
1367         dpz2 = ddpz2;
1368
1369         xtype = dxtype;
1370     }
1371
1373     double dpx1;
1375     double dpy1;
1377     double dpz1;
1378
1380     double dpx2;
1382     double dpy2;
1384     double dpz2;
1385
1387     bool xtype;
1388 };
1389
1390
1391
1395 struct DXFLIB_EXPORT DL_LeaderData {
1400     DL_LeaderData(int arrowHeadFlag,
1401                   int leaderPathType,
1402                   int leaderCreationFlag,
1403                   int hooklineDirectionFlag,
1404                   int hooklineFlag,
1405                   double textAnnotationHeight,
1406                   double textAnnotationWidth,
1407                   int number,
1408                   double dimScale = 1.0) :
1409         arrowHeadFlag(arrowHeadFlag),
1410         leaderPathType(leaderPathType),
1411         leaderCreationFlag(leaderCreationFlag),
1412         hooklineDirectionFlag(hooklineDirectionFlag),
1413         hooklineFlag(hooklineFlag),
1414         textAnnotationHeight(textAnnotationHeight),
1415         textAnnotationWidth(textAnnotationWidth),
1416         number(number),
1417         dimScale(dimScale) {
1418
1419     }
1420
1422     int arrowHeadFlag;
1424     int leaderPathType;
1426     int leaderCreationFlag;
1428     int hooklineDirectionFlag;
1430     int hooklineFlag;
1432     double textAnnotationHeight;
1434     double textAnnotationWidth;
1436     int number;
1438     double dimScale;
1439 };
1440
1441
1442
1446 struct DXFLIB_EXPORT DL_LeaderVertexData {
1451     DL_LeaderVertexData(double px=0.0, double py=0.0, double pz=0.0) {
1452         x = px;
1453         y = py;
1454         z = pz;
1455     }
1456
1458     double x;
1460     double y;
1462     double z;
1463 };
1464
1465
1466
1470 struct DXFLIB_EXPORT DL_HatchData {
1474     DL_HatchData() {}
1475
1480     DL_HatchData(int numLoops,
1481                  bool solid,
1482                  double scale,
1483                  double angle,
1484                  const std::string& pattern,
1485                  double originX = 0.0,
1486                  double originY = 0.0) :
1487         numLoops(numLoops),
1488         solid(solid),
```

```
1489         scale(scale),
1490         angle(angle),
1491         pattern(pattern),
1492         originX(originX),
1493         originY(originY) {
1494
1495     }
1496
1498     int numLoops;
1500     bool solid;
1502     double scale;
1504     double angle;
1506     std::string pattern;
1508     double originX;
1509     double originY;
1510 };
1511
1512
1513
1517 struct DXFLIB_EXPORT DL_HatchLoopData {
1521     DL_HatchLoopData() {}
1526     DL_HatchLoopData(int hNumEdges) {
1527         numEdges = hNumEdges;
1528     }
1529
1531     int numEdges;
1532 };
1533
1534
1535
1539 struct DXFLIB_EXPORT DL_HatchEdgeData {
1543     DL_HatchEdgeData() : defined(false), x1(0.0), y1(0.0), x2(0.0), y2(0.0) {
1544     }
1545
1550     DL_HatchEdgeData(double x1, double y1,
1551                      double x2, double y2) :
1552         defined(true),
1553         type(1),
1554         x1(x1),
1555         y1(y1),
1556         x2(x2),
1557         y2(y2) {
1558     }
1559
1564     DL_HatchEdgeData(double cx, double cy,
1565                      double radius,
1566                      double angle1, double angle2,
1567                      bool ccw) :
1568         defined(true),
1569         type(2),
1570         cx(cx),
1571         cy(cy),
1572         radius(radius),
1573         angle1(angle1),
1574         angle2(angle2),
1575         ccw(ccw) {
1576     }
1577
1582     DL_HatchEdgeData(double cx, double cy,
1583                      double mx, double my,
1584                      double ratio,
1585                      double angle1, double angle2,
1586                      bool ccw) :
1587         defined(true),
1588         type(3),
1589         cx(cx),
1590         cy(cy),
1591         angle1(angle1),
1592         angle2(angle2),
1593         ccw(ccw),
1594         mx(mx),
1595         my(my),
1596         ratio(ratio) {
1597     }
1598
1603     DL_HatchEdgeData(unsigned int degree,
1604                      bool rational,
1605                      bool periodic,
1606                      unsigned int nKnots,
1607                      unsigned int nControl,
1608                      unsigned int nFit,
1609                      const std::vector<double>& knots,
1610                      const std::vector<std::vector<double> >& controlPoints,
1611                      const std::vector<std::vector<double> >& fitPoints,
1612                      const std::vector<double>& weights,
1613                      double startTangentX,
1614                      double startTangentY,
```

```
1615                    double endTangentX,
1616                    double endTangentY) :
1617         defined(true),
1618         type(4),
1619         degree(degree),
1620         rational(rational),
1621         periodic(periodic),
1622         nKnots(nKnots),
1623         nControl(nControl),
1624         nFit(nFit),
1625         controlPoints(controlPoints),
1626         knots(knots),
1627         weights(weights),
1628         fitPoints(fitPoints),
1629         startTangentX(startTangentX),
1630         startTangentY(startTangentY),
1631         endTangentX(endTangentX),
1632         endTangentY(endTangentY) {
1633     }
1634
1638     bool defined;
1639
1643     int type;
1644
1645     // line edges:
1646
1648     double x1;
1650     double y1;
1652     double x2;
1654     double y2;
1655
1657     double cx;
1659     double cy;
1661     double radius;
1663     double angle1;
1665     double angle2;
1667     bool ccw;
1668
1670     double mx;
1672     double my;
1674     double ratio;
1675
1676
1678     unsigned int degree;
1679     bool rational;
1680     bool periodic;
1682     unsigned int nKnots;
1684     unsigned int nControl;
1686     unsigned int nFit;
1687
1688     std::vector<std::vector<double> > controlPoints;
1689     std::vector<double> knots;
1690     std::vector<double> weights;
1691     std::vector<std::vector<double> > fitPoints;
1692
1693     double startTangentX;
1694     double startTangentY;
1695
1696     double endTangentX;
1697     double endTangentY;
1698
1700     std::vector<std::vector<double> > vertices;
1701     //bool closed;
1702 };
1703
1704
1705
1709 struct DXFLIB_EXPORT DL_ImageData {
1714     DL_ImageData(const std::string& iref,
1715                  double iipx, double iipy, double iipz,
1716                  double iux, double iuy, double iuz,
1717                  double ivx, double ivy, double ivz,
1718                  int iwidth, int iheight,
1719                  int ibrightness, int icontrast, int ifade) {
1720         ref = iref;
1721         ipx = iipx;
1722         ipy = iipy;
1723         ipz = iipz;
1724         ux = iux;
1725         uy = iuy;
1726         uz = iuz;
1727         vx = ivx;
1728         vy = ivy;
1729         vz = ivz;
1730         width = iwidth;
1731         height = iheight;
1732         brightness = ibrightness;
```

```
1733         contrast = icontrast;
1734         fade = ifade;
1735     }
1736
1739     std::string ref;
1741     double ipx;
1743     double ipy;
1745     double ipz;
1747     double ux;
1749     double uy;
1751     double uz;
1753     double vx;
1755     double vy;
1757     double vz;
1759     int width;
1761     int height;
1763     int brightness;
1765     int contrast;
1767     int fade;
1768 };
1769
1770
1771
1775 struct DXFLIB_EXPORT DL_ImageDefData {
1780     DL_ImageDefData(const std::string& iref,
1781                     const std::string& ifile) {
1782         ref = iref;
1783         file = ifile;
1784     }
1785
1788     std::string ref;
1789
1791     std::string file;
1792 };
1793
1794
1795
1799 struct DXFLIB_EXPORT DL_DictionaryData {
1800     DL_DictionaryData(const std::string& handle) : handle(handle) {}
1801     std::string handle;
1802 };
1803
1804
1805
1809 struct DXFLIB_EXPORT DL_DictionaryEntryData {
1810     DL_DictionaryEntryData(const std::string& name, const std::string& handle) :
1811         name(name), handle(handle) {}
1812
1813     std::string name;
1814     std::string handle;
1815 };
1816
1817 #endif
1818
1819 // EOF
```

## 6.7 dl_exception.h

```
1 /**********************************************************************
2 ** Copyright (C) 2001-2013 RibbonSoft, GmbH. All rights reserved.
3 ** Copyright (C) 2001 Robert J. Campbell Jr.
4 **
5 ** This file is part of the dxflib project.
6 **
7 ** This file is free software; you can redistribute it and/or modify
8 ** it under the terms of the GNU General Public License as published by
9 ** the Free Software Foundation; either version 2 of the License, or
10 ** (at your option) any later version.
11 **
12 ** Licensees holding valid dxflib Professional Edition licenses may use
13 ** this file in accordance with the dxflib Commercial License
14 ** Agreement provided with the Software.
15 **
16 ** This file is provided AS IS with NO WARRANTY OF ANY KIND, INCLUDING THE
17 ** WARRANTY OF DESIGN, MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE.
18 **
19 ** See http://www.ribbonsoft.com for further details.
20 **
21 ** Contact info@ribbonsoft.com if any conditions of this licensing are
22 ** not clear to you.
23 **
24 **********************************************************************/
25
```

```
26 #ifndef DL_EXCEPTION_H
27 #define DL_EXCEPTION_H
28
29 #include "dl_global.h"
30
31 #if _MSC_VER > 1000
32 #pragma once
33 #endif // _MSC_VER > 1000
34
38 class DXFLIB_EXPORT DL_Exception {}
39 ;
40
44 class DXFLIB_EXPORT DL_NullStrExc : public DL_Exception {}
45 ;
46
50 class DXFLIB_EXPORT DL_GroupCodeExc : public DL_Exception {
51     DL_GroupCodeExc(int gc=0) : groupCode(gc) {}
52     int groupCode;
53 };
54 #endif
55
```

## 6.8 dl_extrusion.h

```
1 /**********************************************************************
2 ** Copyright (C) 2001-2013 RibbonSoft, GmbH. All rights reserved.
3 **
4 ** This file is part of the dxflib project.
5 **
6 ** This file is free software; you can redistribute it and/or modify
7 ** it under the terms of the GNU General Public License as published by
8 ** the Free Software Foundation; either version 2 of the License, or
9 ** (at your option) any later version.
10 **
11 ** Licensees holding valid dxflib Professional Edition licenses may use
12 ** this file in accordance with the dxflib Commercial License
13 ** Agreement provided with the Software.
14 **
15 ** This file is provided AS IS with NO WARRANTY OF ANY KIND, INCLUDING THE
16 ** WARRANTY OF DESIGN, MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE.
17 **
18 ** See http://www.ribbonsoft.com for further details.
19 **
20 ** Contact info@ribbonsoft.com if any conditions of this licensing are
21 ** not clear to you.
22 **
23 **********************************************************************/
24
25 #ifndef DL_EXTRUSION_H
26 #define DL_EXTRUSION_H
27
28 #include "dl_global.h"
29
30 #include <math.h>
31
32
38 class DXFLIB_EXPORT DL_Extrusion {
39
40 public:
41
45     DL_Extrusion() {
46         direction = new double[3];
47         setDirection(0.0, 0.0, 1.0);
48         setElevation(0.0);
49     }
50
51
55     ~DL_Extrusion() {
56         delete[] direction ;
57     }
58
59
68     DL_Extrusion(double dx, double dy, double dz, double elevation) {
69         direction = new double[3];
70         setDirection(dx, dy, dz);
71         setElevation(elevation);
72     }
73
74
75
79     void setDirection(double dx, double dy, double dz) {
80         direction[0]=dx;
81         direction[1]=dy;
```

```
82          direction[2]=dz;
83      }
84
85
86
90      double* getDirection() const {
91          return direction;
92      }
93
94
95
99      void getDirection(double dir[]) const {
100         dir[0]=direction[0];
101         dir[1]=direction[1];
102         dir[2]=direction[2];
103     }
104
105
106
110     void setElevation(double elevation) {
111         this->elevation = elevation;
112     }
113
114
115
119     double getElevation() const {
120         return elevation;
121     }
122
123
124
128     DL_Extrusion operator = (const DL_Extrusion& extru) {
129         setDirection(extru.direction[0], extru.direction[1], extru.direction[2]);
130         setElevation(extru.elevation);
131
132         return *this;
133     }
134
135
136
137 private:
138     double *direction;
139     double elevation;
140 };
141
142 #endif
143
```

## 6.9 dl_global.h

```
1 #if defined(DXFLIB_DLL)
2 #   ifdef _WIN32
3 #       if defined(DXFLIB_LIBRARY)
4 #           define DXFLIB_EXPORT __declspec(dllexport)
5 #       else
6 #           define DXFLIB_EXPORT __declspec(dllimport)
7 #       endif
8 #   else
9 #       define DXFLIB_EXPORT
10 #   endif
11 #else
12 #   define DXFLIB_EXPORT
13 #endif
```

## 6.10 dl_writer.h

```
1 /**********************************************************************
2 ** Copyright (C) 2001-2013 RibbonSoft, GmbH. All rights reserved.
3 ** Copyright (C) 2001 Robert J. Campbell Jr.
4 **
5 ** This file is part of the dxflib project.
6 **
7 ** This file is free software; you can redistribute it and/or modify
8 ** it under the terms of the GNU General Public License as published by
9 ** the Free Software Foundation; either version 2 of the License, or
10 ** (at your option) any later version.
11 **
12 ** Licensees holding valid dxflib Professional Edition licenses may use
13 ** this file in accordance with the dxflib Commercial License
```

```
14 ** Agreement provided with the Software.
15 **
16 ** This file is provided AS IS with NO WARRANTY OF ANY KIND, INCLUDING THE
17 ** WARRANTY OF DESIGN, MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE.
18 **
19 ** See http://www.ribbonsoft.com for further details.
20 **
21 ** Contact info@ribbonsoft.com if any conditions of this licensing are
22 ** not clear to you.
23 **
24 **********************************************************************/
25
26 #ifndef DL_WRITER_H
27 #define DL_WRITER_H
28
29 #include "dl_global.h"
30
31 #ifndef _WIN32
32 #include <strings.h>
33 #endif
34
35 #if _MSC_VER > 1000
36 #pragma once
37 #endif // _MSC_VER > 1000
38
39 #include <cstring>
40 #include <iostream>
41 #include <algorithm>
42
43 #include "dl_attributes.h"
44 #include "dl_codes.h"
45
46
47
58 class DXFLIB_EXPORT DL_Writer {
59 public:
63     DL_Writer(DL_Codes::version version) : m_handle(0x30) {
64         this->version = version;
65         modelSpaceHandle = 0;
66         paperSpaceHandle = 0;
67         paperSpace0Handle = 0;
68     }
69
70     virtual ~DL_Writer() {}
71     ;
72
82     void section(const char* name) const {
83         dxfString(0, "SECTION");
84         dxfString(2, name);
85     }
86
97     void sectionHeader() const {
98         section("HEADER");
99     }
100
111     void sectionTables() const {
112         section("TABLES");
113     }
114
125     void sectionBlocks() const {
126         section("BLOCKS");
127     }
128
139     void sectionEntities() const {
140         section("ENTITIES");
141     }
142
153     void sectionClasses() const {
154         section("CLASSES");
155     }
156
167     void sectionObjects() const {
168         section("OBJECTS");
169     }
170
179     void sectionEnd() const {
180         dxfString(0, "ENDSEC");
181     }
182
195     void table(const char* name, int num, int h=0) const {
196         dxfString(0, "TABLE");
197         dxfString(2, name);
198         if (version>=DL_VERSION_2000) {
199             if (h==0) {
200                 handle();
201             }
202             else {
```

```
203                 dxfHex(5, h);
204             }
205             dxfString(100, "AcDbSymbolTable");
206         }
207         dxfInt(70, num);
208     }
209
223     void tableLayers(int num) const {
224         table("LAYER", num, 2);
225     }
226
240     void tableLinetypes(int num) const {
241         //linetypeHandle = 5;
242         table("LTYPE", num, 5);
243     }
244
258     void tableAppid(int num) const {
259         table("APPID", num, 9);
260     }
261
275     void tableStyle(int num) const {
276         table("STYLE", num, 3);
277     }
278
287     void tableEnd() const {
288         dxfString(0, "ENDTAB");
289     }
290
299     void dxfEOF() const {
300         dxfString(0, "EOF");
301     }
302
311     void comment(const char* text) const {
312         dxfString(999, text);
313     }
314
325     void entity(const char* entTypeName) const {
326         dxfString(0, entTypeName);
327         if (version>=DL_VERSION_2000) {
328             handle();
329         }
330     }
331
346     void entityAttributes(const DL_Attributes& attrib) const {
347
348         // layer name:
349         dxfString(8, attrib.getLayer());
350
351         // R12 doesn't accept BYLAYER values. The value has to be missing
352         //   in that case.
353         if (version>=DL_VERSION_2000 || attrib.getColor()!=256) {
354             dxfInt(62, attrib.getColor());
355         }
356         if (version>=DL_VERSION_2000 && attrib.getColor24()!=-1) {
357             dxfInt(420, attrib.getColor24());
358         }
359         if (version>=DL_VERSION_2000) {
360             dxfInt(370, attrib.getWidth());
361         }
362         if (version>=DL_VERSION_2000) {
363             dxfReal(48, attrib.getLinetypeScale());
364         }
365         std::string linetype = attrib.getLinetype();
366         std::transform(linetype.begin(), linetype.end(), linetype.begin(), ::toupper);
367         if (version>=DL_VERSION_2000 || linetype=="BYLAYER") {
368             dxfString(6, attrib.getLinetype());
369         }
370     }
371
375     void subClass(const char* sub) const {
376         dxfString(100, sub);
377     }
378
387     void tableLayerEntry(unsigned long int h=0)  const {
388         dxfString(0, "LAYER");
389         if (version>=DL_VERSION_2000) {
390             if (h==0) {
391                 handle();
392             } else {
393                 dxfHex(5, h);
394             }
395             dxfString(100, "AcDbSymbolTableRecord");
396             dxfString(100, "AcDbLayerTableRecord");
397         }
398     }
399
408     void tableLinetypeEntry(unsigned long int h=0)  const {
```

```
409         dxfString(0, "LTYPE");
410         if (version>=DL_VERSION_2000) {
411             if (h==0) {
412                 handle();
413             } else {
414                 dxfHex(5, h);
415             }
416             //dxfHex(330, 0x5);
417             dxfString(100, "AcDbSymbolTableRecord");
418             dxfString(100, "AcDbLinetypeTableRecord");
419         }
420     }
421
430     void tableAppidEntry(unsigned long int h=0)  const {
431         dxfString(0, "APPID");
432         if (version>=DL_VERSION_2000) {
433             if (h==0) {
434                 handle();
435             } else {
436                 dxfHex(5, h);
437             }
438             //dxfHex(330, 0x9);
439             dxfString(100, "AcDbSymbolTableRecord");
440             dxfString(100, "AcDbRegAppTableRecord");
441         }
442     }
443
452     void sectionBlockEntry(unsigned long int h=0)  const {
453         dxfString(0, "BLOCK");
454         if (version>=DL_VERSION_2000) {
455             if (h==0) {
456                 handle();
457             } else {
458                 dxfHex(5, h);
459             }
460             //dxfHex(330, blockHandle);
461             dxfString(100, "AcDbEntity");
462             if (h==0x1C) {
463                 dxfInt(67, 1);
464             }
465             dxfString(8, "0");                  // TODO: Layer for block
466             dxfString(100, "AcDbBlockBegin");
467         }
468     }
469
478     void sectionBlockEntryEnd(unsigned long int h=0)  const {
479         dxfString(0, "ENDBLK");
480         if (version>=DL_VERSION_2000) {
481             if (h==0) {
482                 handle();
483             } else {
484                 dxfHex(5, h);
485             }
486             //dxfHex(330, blockHandle);
487             dxfString(100, "AcDbEntity");
488             if (h==0x1D) {
489                 dxfInt(67, 1);
490             }
491             dxfString(8, "0");                  // TODO: Layer for block
492             dxfString(100, "AcDbBlockEnd");
493         }
494     }
495
496     void color(int col=256) const {
497         dxfInt(62, col);
498     }
499     void linetype(const char *lt) const {
500         dxfString(6, lt);
501     }
502     void linetypeScale(double scale) const {
503         dxfReal(48, scale);
504     }
505     void lineWeight(int lw) const {
506         dxfInt(370, lw);
507     }
508
509     void coord(int gc, double x, double y, double z=0) const {
510         dxfReal(gc, x);
511         dxfReal(gc+10, y);
512         dxfReal(gc+20, z);
513     }
514
515     void coordTriplet(int gc, const double* value) const {
516         if (value) {
517             dxfReal(gc, *value++);
518             dxfReal(gc+10, *value++);
519             dxfReal(gc+20, *value++);
```

```
520         }
521     }
522
523     void resetHandle() const {
524         m_handle = 1;
525     }
526
530     unsigned long handle(int gc=5) const {
531         // handle has to be hex
532         dxfHex(gc, m_handle);
533         return m_handle++;
534     }
535
539     unsigned long getNextHandle() const {
540         return m_handle;
541     }
542
550     virtual void dxfReal(int gc, double value) const = 0;
551
559     virtual void dxfInt(int gc, int value) const = 0;
560
568     virtual void dxfBool(int gc, bool value) const {
569         dxfInt(gc, (int)value);
570     }
571
579     virtual void dxfHex(int gc, int value) const = 0;
580
588     virtual void dxfString(int gc, const char* value) const = 0;
589
597     virtual void dxfString(int gc, const std::string& value) const = 0;
598
599 protected:
600     mutable unsigned long m_handle;
601     mutable unsigned long modelSpaceHandle;
602     mutable unsigned long paperSpaceHandle;
603     mutable unsigned long paperSpace0Handle;
604
608     DL_Codes::version version;
609 private:
610 };
611
612 #endif
```

## 6.11 dl_writer_ascii.h

```
1 /**********************************************************************
2 ** Copyright (C) 2001-2013 RibbonSoft, GmbH. All rights reserved.
3 ** Copyright (C) 2001 Robert J. Campbell Jr.
4 **
5 ** This file is part of the dxflib project.
6 **
7 ** This file is free software; you can redistribute it and/or modify
8 ** it under the terms of the GNU General Public License as published by
9 ** the Free Software Foundation; either version 2 of the License, or
10 ** (at your option) any later version.
11 **
12 ** Licensees holding valid dxflib Professional Edition licenses may use
13 ** this file in accordance with the dxflib Commercial License
14 ** Agreement provided with the Software.
15 **
16 ** This file is provided AS IS with NO WARRANTY OF ANY KIND, INCLUDING THE
17 ** WARRANTY OF DESIGN, MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE.
18 **
19 ** See http://www.ribbonsoft.com for further details.
20 **
21 ** Contact info@ribbonsoft.com if any conditions of this licensing are
22 ** not clear to you.
23 **
24 **********************************************************************/
25
26 #ifndef DL_WRITER_ASCII_H
27 #define DL_WRITER_ASCII_H
28
29 #include "dl_global.h"
30
31 #if _MSC_VER > 1000
32 #pragma once
33 #endif // _MSC_VER > 1000
34
35 #include "dl_writer.h"
36 #include <fstream>
37 #include <string>
38
```

```
49 class DXFLIB_EXPORT DL_WriterA : public DL_Writer {
50 public:
51     DL_WriterA(const char* fname, DL_Codes::version version=DL_VERSION_2000)
52             : DL_Writer(version), m_ofile(fname) {}
53     virtual ~DL_WriterA() {}
54 
55     bool openFailed() const;
56     void close() const;
57     void dxfReal(int gc, double value) const;
58     void dxfInt(int gc, int value) const;
59     void dxfHex(int gc, int value) const;
60     void dxfString(int gc, const char* value) const;
61     void dxfString(int gc, const std::string& value) const;
62 
63     static void strReplace(char* str, char src, char dest);
64 
65 private:
69     mutable std::ofstream m_ofile;
70 
71 };
72 
73 #endif
74 
```

# Index