dxflib

Generated by Doxygen 1.10.0

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:

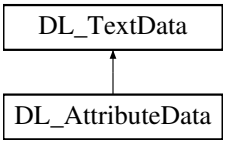
- src/dl_entities.h

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 Member Functions inherited from 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)

  Constructor.

Public Attributes

- std::string tag

Public Attributes inherited from DL_TextData

- 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

5.3.1 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.
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:

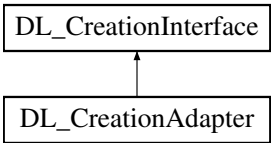
- src/dl_entities.h

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)

Public Member Functions inherited from DL_CreationInterface

- 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 ()

Additional Inherited Members

Protected Attributes inherited from DL_CreationInterface

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

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:

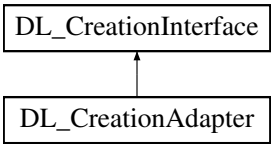
- src/dl_creationadapter.h

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 Pointer to character array that chopped line will be returned in.
size	Size of `s`. (Including space for NULL.)
fp	Input 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(), and 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(), and 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(), and 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(), 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:

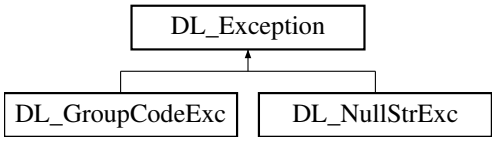
- src/dl_entities.h

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:

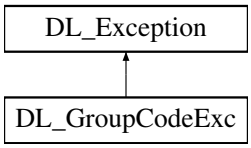
- src/dl_entities.h

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:

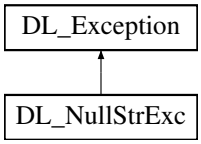
- src/dl_entities.h

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:

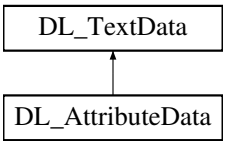
- src/dl_entities.h

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:

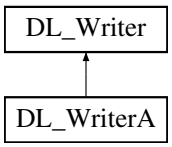
- src/dl_entities.h

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:

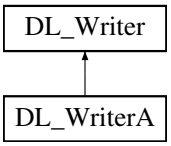
- src/dl_writer.h

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.

Public Member Functions inherited from DL_Writer

- 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 dxfBool (int gc, bool value) const

  Can be overwritten by the implementing class to write a bool value 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

Protected Attributes inherited from DL_Writer

- unsigned long **m_handle**
- unsigned long **modelSpaceHandle**
- unsigned long **paperSpaceHandle**
- unsigned long **paperSpace0Handle**
- DL_Codes::version **version**

  DXF version to be created.

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::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::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

```
00001 /**********************************************************************
00002 ** Copyright (C) 2001-2013 RibbonSoft, GmbH. All rights reserved.
00003 **
00004 ** This file is part of the dxflib project.
00005 **
00006 ** This file is free software; you can redistribute it and/or modify
00007 ** it under the terms of the GNU General Public License as published by
00008 ** the Free Software Foundation; either version 2 of the License, or
00009 ** (at your option) any later version.
00010 **
00011 ** Licensees holding valid dxflib Professional Edition licenses may use
00012 ** this file in accordance with the dxflib Commercial License
00013 ** Agreement provided with the Software.
00014 **
00015 ** This file is provided AS IS with NO WARRANTY OF ANY KIND, INCLUDING THE
00016 ** WARRANTY OF DESIGN, MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE.
00017 **
00018 ** See http://www.ribbonsoft.com for further details.
00019 **
00020 ** Contact info@ribbonsoft.com if any conditions of this licensing are
00021 ** not clear to you.
00022 **
00023 **********************************************************************/
00024
00025 #ifndef DL_ATTRIBUTES_H
00026 #define DL_ATTRIBUTES_H
00027
00028 #include "dl_global.h"
00029
00030 #include <string>
00031 #include <vector>
00032
00033 #include "dl_codes.h"
00034
00041 class DXFLIB_EXPORT DL_Attributes {
00042
00043 public:
00044
00048     DL_Attributes() :
00049         layer(""),
00050         color(0),
00051         color24(-1),
00052         width(0),
00053         linetype("BYLAYER"),
00054         linetypeScale(1.0),
00055         handle(-1),
00056         inPaperSpace(false) {
00057     }
00058
00070     DL_Attributes(const std::string& layer,
00071                   int color, int width,
00072                   const std::string& linetype,
00073                   double linetypeScale) :
00074         layer(layer),
00075         color(color),
00076         color24(-1),
00077         width(width),
00078         linetype(linetype),
```

```
00079         linetypeScale(linetypeScale),
00080         handle(-1),
00081         inPaperSpace(false) {
00082
00083     }
00084
00097     DL_Attributes(const std::string& layer,
00098                   int color, int color24, int width,
00099                   const std::string& linetype,
00100                   int handle=-1)  :
00101         layer(layer),
00102         color(color),
00103         color24(color24),
00104         width(width),
00105         linetype(linetype),
00106         linetypeScale(1.0),
00107         handle(handle),
00108         inPaperSpace(false) {
00109     }
00110
00115     void setLayer(const std::string& layer) {
00116         this->layer = layer;
00117     }
00118
00122     std::string getLayer() const {
00123         return layer;
00124     }
00125
00131     void setColor(int color) {
00132         this->color = color;
00133     }
00134
00140     void setColor24(int color) {
00141         this->color24 = color;
00142     }
00143
00149     int getColor() const {
00150         return color;
00151     }
00152
00158     int getColor24() const {
00159         return color24;
00160     }
00161
00165     void setWidth(int width) {
00166         this->width = width;
00167     }
00168
00172     int getWidth() const {
00173         return width;
00174     }
00175
00180     void setLinetype(const std::string& linetype) {
00181         this->linetype = linetype;
00182     }
00183
00187     void setLinetypeScale(double linetypeScale) {
00188         this->linetypeScale = linetypeScale;
00189     }
00190
00191     double getLinetypeScale() const {
00192         return linetypeScale;
00193     }
00194
00198     std::string getLinetype() const {
00199         if (linetype.length()==0) {
00200             return "BYLAYER";
00201         } else {
00202             return linetype;
00203         }
00204     }
00205
00206     void setHandle(int h) {
00207         handle = h;
00208     }
00209
00210     int getHandle() const {
00211         return handle;
00212     }
00213
00214     void setInPaperSpace(bool on) {
00215         inPaperSpace = on;
00216     }
00217
00218     bool isInPaperSpace() const {
00219         return inPaperSpace;
00220     }
```

```
00221
00222 private:
00223     std::string layer;
00224     int color;
00225     int color24;
00226     int width;
00227     std::string linetype;
00228     double linetypeScale;
00229     int handle;
00230
00231     // DXF code 67 (true: entity in paper space, false: entity in model space (default):
00232     bool inPaperSpace;
00233 };
00234
00235 #endif
00236
00237 // EOF
```

6.2 dl_codes.h

```
00001 /**********************************************************************
00002 ** Copyright (C) 2001-2013 RibbonSoft, GmbH. All rights reserved.
00003 ** Copyright (C) 2001 Robert J. Campbell Jr.
00004 **
00005 ** This file is part of the dxflib project.
00006 **
00007 ** This file is free software; you can redistribute it and/or modify
00008 ** it under the terms of the GNU General Public License as published by
00009 ** the Free Software Foundation; either version 2 of the License, or
00010 ** (at your option) any later version.
00011 **
00012 ** Licensees holding valid dxflib Professional Edition licenses may use
00013 ** this file in accordance with the dxflib Commercial License
00014 ** Agreement provided with the Software.
00015 **
00016 ** This file is provided AS IS with NO WARRANTY OF ANY KIND, INCLUDING THE
00017 ** WARRANTY OF DESIGN, MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE.
00018 **
00019 ** See http://www.ribbonsoft.com for further details.
00020 **
00021 ** Contact info@ribbonsoft.com if any conditions of this licensing are
00022 ** not clear to you.
00023 **
00024 **********************************************************************/
00025
00030 #ifndef DXF_CODES_H
00031 #define DXF_CODES_H
00032
00033 #include "dl_global.h"
00034
00035 #if _MSC_VER > 1000
00036 #pragma once
00037 #endif // _MSC_VER > 1000
00038
00039 #if defined(__OS2__)||defined(__EMX__)
00040 #define strcasecmp(s,t) stricmp(s,t)
00041 #endif
00042
00043 #if defined(_WIN32)
00044 #define strcasecmp(s,t) _stricmp(s,t)
00045 #endif
00046
00047
00048 #ifdef _WIN32
00049 #undef M_PI
00050 #define M_PI   3.14159265358979323846
00051 #pragma warning(disable : 4800)
00052 #endif
00053
00054 #ifndef M_PI
00055 #define M_PI 3.14159265358979323846264338327795
00056 #endif
00057
00058 #define DL_DXF_MAXLINE 1024
00059 #define DL_DXF_MAXGROUPCODE 1100
00060
00061 // used to mark invalid vectors:
00062 //#define DL_DXF_MAXDOUBLE 1.0E+10
00063
00067 class DXFLIB_EXPORT DL_Codes {
00068 public:
00072     enum color {
00073         black = 250,
```

```
00074         green = 3,
00075         red = 1,
00076         brown = 15,
00077         yellow = 2,
00078         cyan = 4,
00079         magenta = 6,
00080         gray = 8,
00081         blue = 5,
00082         l_blue = 163,
00083         l_green = 121,
00084         l_cyan = 131,
00085         l_red = 23,
00086         l_magenta = 221,
00087         l_gray = 252,
00088         white = 7,
00089         bylayer = 256,
00090         byblock = 0
00091     };
00092
00096     enum version {
00097         AC1009_MIN,     // R12, minimalistic
00098         AC1009,         // R12
00099         AC1012,
00100         AC1014,
00101         AC1015          // R2000
00102     };
00103 };
00104
00105
00106 // Extended color palette:
00107 // The first entry is only for direct indexing starting with [1]
00108 // Color 1 is red (1,0,0)
00109 const double dxfColors[][3] = {
00110                                     {0,0,0},                // unused
00111                                     {1,0,0},                // 1
00112                                     {1,1,0},
00113                                     {0,1,0},
00114                                     {0,1,1},
00115                                     {0,0,1},
00116                                     {1,0,1},
00117                                     {1,1,1},                // black or white
00118                                     {0.5,0.5,0.5},
00119                                     {0.75,0.75,0.75},
00120                                     {1,0,0},                // 10
00121                                     {1,0.5,0.5},
00122                                     {0.65,0,0},
00123                                     {0.65,0.325,0.325},
00124                                     {0.5,0,0},
00125                                     {0.5,0.25,0.25},
00126                                     {0.3,0,0},
00127                                     {0.3,0.15,0.15},
00128                                     {0.15,0,0},
00129                                     {0.15,0.075,0.075},
00130                                     {1,0.25,0},             // 20
00131                                     {1,0.625,0.5},
00132                                     {0.65,0.1625,0},
00133                                     {0.65,0.4063,0.325},
00134                                     {0.5,0.125,0},
00135                                     {0.5,0.3125,0.25},
00136                                     {0.3,0.075,0},
00137                                     {0.3,0.1875,0.15},
00138                                     {0.15,0.0375,0},
00139                                     {0.15,0.0938,0.075},
00140                                     {1,0.5,0},              // 30
00141                                     {1,0.75,0.5},
00142                                     {0.65,0.325,0},
00143                                     {0.65,0.4875,0.325},
00144                                     {0.5,0.25,0},
00145                                     {0.5,0.375,0.25},
00146                                     {0.3,0.15,0},
00147                                     {0.3,0.225,0.15},
00148                                     {0.15,0.075,0},
00149                                     {0.15,0.1125,0.075},
00150                                     {1,0.75,0},             // 40
00151                                     {1,0.875,0.5},
00152                                     {0.65,0.4875,0},
00153                                     {0.65,0.5688,0.325},
00154                                     {0.5,0.375,0},
00155                                     {0.5,0.4375,0.25},
00156                                     {0.3,0.225,0},
00157                                     {0.3,0.2625,0.15},
00158                                     {0.15,0.1125,0},
00159                                     {0.15,0.1313,0.075},
00160                                     {1,1,0},                // 50
00161                                     {1,1,0.5},
00162                                     {0.65,0.65,0},
00163                                     {0.65,0.65,0.325},
```

```
00164                                   {0.5,0.5,0},
00165                                   {0.5,0.5,0.25},
00166                                   {0.3,0.3,0},
00167                                   {0.3,0.3,0.15},
00168                                   {0.15,0.15,0},
00169                                   {0.15,0.15,0.075},
00170                                   {0.75,1,0},             // 60
00171                                   {0.875,1,0.5},
00172                                   {0.4875,0.65,0},
00173                                   {0.5688,0.65,0.325},
00174                                   {0.375,0.5,0},
00175                                   {0.4375,0.5,0.25},
00176                                   {0.225,0.3,0},
00177                                   {0.2625,0.3,0.15},
00178                                   {0.1125,0.15,0},
00179                                   {0.1313,0.15,0.075},
00180                                   {0.5,1,0},              // 70
00181                                   {0.75,1,0.5},
00182                                   {0.325,0.65,0},
00183                                   {0.4875,0.65,0.325},
00184                                   {0.25,0.5,0},
00185                                   {0.375,0.5,0.25},
00186                                   {0.15,0.3,0},
00187                                   {0.225,0.3,0.15},
00188                                   {0.075,0.15,0},
00189                                   {0.1125,0.15,0.075},
00190                                   {0.25,1,0},             // 80
00191                                   {0.625,1,0.5},
00192                                   {0.1625,0.65,0},
00193                                   {0.4063,0.65,0.325},
00194                                   {0.125,0.5,0},
00195                                   {0.3125,0.5,0.25},
00196                                   {0.075,0.3,0},
00197                                   {0.1875,0.3,0.15},
00198                                   {0.0375,0.15,0},
00199                                   {0.0938,0.15,0.075},
00200                                   {0,1,0},                // 90
00201                                   {0.5,1,0.5},
00202                                   {0,0.65,0},
00203                                   {0.325,0.65,0.325},
00204                                   {0,0.5,0},
00205                                   {0.25,0.5,0.25},
00206                                   {0,0.3,0},
00207                                   {0.15,0.3,0.15},
00208                                   {0,0.15,0},
00209                                   {0.075,0.15,0.075},
00210                                   {0,1,0.25},             // 100
00211                                   {0.5,1,0.625},
00212                                   {0,0.65,0.1625},
00213                                   {0.325,0.65,0.4063},
00214                                   {0,0.5,0.125},
00215                                   {0.25,0.5,0.3125},
00216                                   {0,0.3,0.075},
00217                                   {0.15,0.3,0.1875},
00218                                   {0,0.15,0.0375},
00219                                   {0.075,0.15,0.0938},
00220                                   {0,1,0.5},              // 110
00221                                   {0.5,1,0.75},
00222                                   {0,0.65,0.325},
00223                                   {0.325,0.65,0.4875},
00224                                   {0,0.5,0.25},
00225                                   {0.25,0.5,0.375},
00226                                   {0,0.3,0.15},
00227                                   {0.15,0.3,0.225},
00228                                   {0,0.15,0.075},
00229                                   {0.075,0.15,0.1125},
00230                                   {0,1,0.75},             // 120
00231                                   {0.5,1,0.875},
00232                                   {0,0.65,0.4875},
00233                                   {0.325,0.65,0.5688},
00234                                   {0,0.5,0.375},
00235                                   {0.25,0.5,0.4375},
00236                                   {0,0.3,0.225},
00237                                   {0.15,0.3,0.2625},
00238                                   {0,0.15,0.1125},
00239                                   {0.075,0.15,0.1313},
00240                                   {0,1,1},                // 130
00241                                   {0.5,1,1},
00242                                   {0,0.65,0.65},
00243                                   {0.325,0.65,0.65},
00244                                   {0,0.5,0.5},
00245                                   {0.25,0.5,0.5},
00246                                   {0,0.3,0.3},
00247                                   {0.15,0.3,0.3},
00248                                   {0,0.15,0.15},
00249                                   {0.075,0.15,0.15},
00250                                   {0,0.75,1},             // 140
```

```
00251                                   {0.5,0.875,1},
00252                                   {0,0.4875,0.65},
00253                                   {0.325,0.5688,0.65},
00254                                   {0,0.375,0.5},
00255                                   {0.25,0.4375,0.5},
00256                                   {0,0.225,0.3},
00257                                   {0.15,0.2625,0.3},
00258                                   {0,0.1125,0.15},
00259                                   {0.075,0.1313,0.15},
00260                                   {0,0.5,1},              // 150
00261                                   {0.5,0.75,1},
00262                                   {0,0.325,0.65},
00263                                   {0.325,0.4875,0.65},
00264                                   {0,0.25,0.5},
00265                                   {0.25,0.375,0.5},
00266                                   {0,0.15,0.3},
00267                                   {0.15,0.225,0.3},
00268                                   {0,0.075,0.15},
00269                                   {0.075,0.1125,0.15},
00270                                   {0,0.25,1},             // 160
00271                                   {0.5,0.625,1},
00272                                   {0,0.1625,0.65},
00273                                   {0.325,0.4063,0.65},
00274                                   {0,0.125,0.5},
00275                                   {0.25,0.3125,0.5},
00276                                   {0,0.075,0.3},
00277                                   {0.15,0.1875,0.3},
00278                                   {0,0.0375,0.15},
00279                                   {0.075,0.0938,0.15},
00280                                   {0,0,1},                // 170
00281                                   {0.5,0.5,1},
00282                                   {0,0,0.65},
00283                                   {0.325,0.325,0.65},
00284                                   {0,0,0.5},
00285                                   {0.25,0.25,0.5},
00286                                   {0,0,0.3},
00287                                   {0.15,0.15,0.3},
00288                                   {0,0,0.15},
00289                                   {0.075,0.075,0.15},
00290                                   {0.25,0,1},             // 180
00291                                   {0.625,0.5,1},
00292                                   {0.1625,0,0.65},
00293                                   {0.4063,0.325,0.65},
00294                                   {0.125,0,0.5},
00295                                   {0.3125,0.25,0.5},
00296                                   {0.075,0,0.3},
00297                                   {0.1875,0.15,0.3},
00298                                   {0.0375,0,0.15},
00299                                   {0.0938,0.075,0.15},
00300                                   {0.5,0,1},              // 190
00301                                   {0.75,0.5,1},
00302                                   {0.325,0,0.65},
00303                                   {0.4875,0.325,0.65},
00304                                   {0.25,0,0.5},
00305                                   {0.375,0.25,0.5},
00306                                   {0.15,0,0.3},
00307                                   {0.225,0.15,0.3},
00308                                   {0.075,0,0.15},
00309                                   {0.1125,0.075,0.15},
00310                                   {0.75,0,1},             // 200
00311                                   {0.875,0.5,1},
00312                                   {0.4875,0,0.65},
00313                                   {0.5688,0.325,0.65},
00314                                   {0.375,0,0.5},
00315                                   {0.4375,0.25,0.5},
00316                                   {0.225,0,0.3},
00317                                   {0.2625,0.15,0.3},
00318                                   {0.1125,0,0.15},
00319                                   {0.1313,0.075,0.15},
00320                                   {1,0,1},                // 210
00321                                   {1,0.5,1},
00322                                   {0.65,0,0.65},
00323                                   {0.65,0.325,0.65},
00324                                   {0.5,0,0.5},
00325                                   {0.5,0.25,0.5},
00326                                   {0.3,0,0.3},
00327                                   {0.3,0.15,0.3},
00328                                   {0.15,0,0.15},
00329                                   {0.15,0.075,0.15},
00330                                   {1,0,0.75},             // 220
00331                                   {1,0.5,0.875},
00332                                   {0.65,0,0.4875},
00333                                   {0.65,0.325,0.5688},
00334                                   {0.5,0,0.375},
00335                                   {0.5,0.25,0.4375},
00336                                   {0.3,0,0.225},
00337                                   {0.3,0.15,0.2625},
```

```
00338                                   {0.15,0,0.1125},
00339                                   {0.15,0.075,0.1313},
00340                                   {1,0,0.5},              // 230
00341                                   {1,0.5,0.75},
00342                                   {0.65,0,0.325},
00343                                   {0.65,0.325,0.4875},
00344                                   {0.5,0,0.25},
00345                                   {0.5,0.25,0.375},
00346                                   {0.3,0,0.15},
00347                                   {0.3,0.15,0.225},
00348                                   {0.15,0,0.075},
00349                                   {0.15,0.075,0.1125},
00350                                   {1,0,0.25},             // 240
00351                                   {1,0.5,0.625},
00352                                   {0.65,0,0.1625},
00353                                   {0.65,0.325,0.4063},
00354                                   {0.5,0,0.125},
00355                                   {0.5,0.25,0.3125},
00356                                   {0.3,0,0.075},
00357                                   {0.3,0.15,0.1875},
00358                                   {0.15,0,0.0375},
00359                                   {0.15,0.075,0.0938},
00360                                   {0.33,0.33,0.33},       // 250
00361                                   {0.464,0.464,0.464},
00362                                   {0.598,0.598,0.598},
00363                                   {0.732,0.732,0.732},
00364                                   {0.866,0.866,0.866},
00365                                   {1,1,1}                 // 255
00366                                 }
00367                                 ;
00368
00369
00370 // AutoCAD VERSION aliases
00371 #define DL_VERSION_R12    DL_Codes::AC1009
00372 #define DL_VERSION_LT2    DL_Codes::AC1009
00373 #define DL_VERSION_R13    DL_Codes::AC1012   // not supported yet
00374 #define DL_VERSION_LT95   DL_Codes::AC1012   // not supported yet
00375 #define DL_VERSION_R14    DL_Codes::AC1014   // not supported yet
00376 #define DL_VERSION_LT97   DL_Codes::AC1014   // not supported yet
00377 #define DL_VERSION_LT98   DL_Codes::AC1014   // not supported yet
00378 #define DL_VERSION_2000   DL_Codes::AC1015
00379 #define DL_VERSION_2002   DL_Codes::AC1015
00380
00381
00382 // DXF Group Codes:
00383
00384 // Strings
00385 #define DL_STRGRP_START      0
00386 #define DL_STRGRP_END        9
00387
00388 // Coordinates
00389 #define DL_CRDGRP_START     10
00390 #define DL_CRDGRP_END       19
00391
00392 // Real values
00393 #define DL_RLGRP_START      38
00394 #define DL_RLGRP_END        59
00395
00396 // Short integer values
00397 #define DL_SHOGRP_START     60
00398 #define DL_SHOGRP_END       79
00399
00400 // New in Release 13,
00401 #define DL_SUBCLASS        100
00402
00403 // More coordinates
00404 #define DL_CRD2GRP_START   210
00405 #define DL_CRD2GRP_END     239
00406
00407 // Extended data strings
00408 #define DL_ESTRGRP_START  1000
00409 #define DL_ESTRGRP_END    1009
00410
00411 // Extended data reals
00412 #define DL_ERLGRP_START   1010
00413 #define DL_ERLGRP_END     1059
00414
00415
00416 #define DL_Y8_COORD_CODE       28
00417 #define DL_Z0_COORD_CODE       30
00418 #define DL_Z8_COORD_CODE       38
00419
00420 #define DL_POINT_COORD_CODE    10
00421 #define DL_INSERT_COORD_CODE   10
00422
00423 #define DL_CRD2GRP_START      210
00424 #define DL_CRD2GRP_END        239
```

```
00425
00426 #define DL_THICKNESS            39
00427 #define DL_FIRST_REAL_CODE      THICKNESS
00428 #define DL_LAST_REAL_CODE       59
00429 #define DL_FIRST_INT_CODE       60
00430 #define DL_ATTFLAGS_CODE        70
00431 #define DL_PLINE_FLAGS_CODE     70
00432 #define DL_LAYER_FLAGS_CODE     70
00433 #define DL_FLD_LEN_CODE         73 // Inside ATTRIB resbuf
00434 #define DL_LAST_INT_CODE        79
00435 #define DL_X_EXTRU_CODE        210
00436 #define DL_Y_EXTRU_CODE        220
00437 #define DL_Z_EXTRU_CODE        230
00438 #define DL_COMMENT_CODE        999
00439
00440 // Start and endpoints of a line
00441 #define DL_LINE_START_CODE      10  // Followed by x coord
00442 #define DL_LINE_END_CODE        11  // Followed by x coord
00443
00444 // Some codes used by blocks
00445 #define DL_BLOCK_FLAGS_CODE     70  // An int containing flags
00446 #define DL_BLOCK_BASE_CODE      10  // Origin of block definition
00447 #define DL_XREF_DEPENDENT       16  // If a block contains an XREF
00448 #define DL_XREF_RESOLVED        32  // If a XREF resolved ok
00449 #define DL_REFERENCED           64  // If a block is ref'd in DWG
00450
00451 #define DL_XSCALE_CODE          41
00452 #define DL_YSCALE_CODE          42
00453 #define DL_ANGLE_CODE           50
00454 #define DL_INS_POINT_CODE       10  // Followed by x of ins pnt
00455 #define DL_NAME2_CODE            3  // Second appearance of name
00456
00457 // Some codes used by circle entities
00458 #define DL_CENTER_CODE          10  // Followed by x of center
00459 #define DL_RADIUS_CODE          40  // Followd by radius of circle
00460
00461 #define DL_COND_OP_CODE         -4  // Conditional op,ads_ssget
00462
00463 // When using ads_buildlist you MUST use RTDXF0 instead of these
00464 #define DL_ENTITY_TYPE_CODE      0  // Then there is LINE, 3DFACE..
00465 #define DL_SES_CODE              0  // Start End String Code
00466 #define DL_FILE_SEP_CODE         0  // File separator
00467 #define DL_SOT_CODE              0  // Start Of Table
00468 #define DL_TEXTVAL_CODE          1
00469 #define DL_NAME_CODE             2
00470 #define DL_BLOCK_NAME_CODE       2
00471 #define DL_SECTION_NAME_CODE     2
00472 #define DL_ENT_HAND_CODE         5  // What follows is hexa string
00473 #define DL_TXT_STYLE_CODE        7  // Inside attributes
00474 #define DL_LAYER_NAME_CODE       8  // What follows is layer name
00475 #define DL_FIRST_XCOORD_CODE    10  // Group code x of 1st coord
00476 #define DL_FIRST_YCOORD_CODE    20  // Group code y of 1st coord
00477 #define DL_FIRST_ZCOORD_CODE    30  // Group code z of 1st coord
00478 #define DL_L_START_CODE         10
00479 #define DL_L_END_CODE           11
00480 #define DL_TXTHI_CODE           40
00481 #define DL_SCALE_X_CODE         41
00482 #define DL_SCALE_Y_CODE         42
00483 #define DL_SCALE_Z_CODE         43
00484 #define DL_BULGE_CODE           42  // Used in PLINE verts for arcs
00485 #define DL_ROTATION_CODE        50
00486 #define DL_COLOUR_CODE          62  // What follows is a color int
00487 #define DL_LTYPE_CODE            6  // What follows is a linetype
00488
00489
00490 // Attribute flags
00491 #define DL_ATTS_FOLLOW_CODE     66
00492 #define DL_ATT_TAG_CODE          2
00493 #define DL_ATT_VAL_CODE          1
00494 #define DL_ATT_FLAGS_CODE       70  // 4 1 bit flags as follows...
00495 #define DL_ATT_INVIS_FLAG        1
00496 #define DL_ATT_CONST_FLAG        2
00497 #define DL_ATT_VERIFY_FLAG       4 // Prompt and verify
00498 #define DL_ATT_PRESET_FLAG       8 // No prompt and no verify
00499
00500 // PLINE defines
00501 // Flags
00502 #define DL_OPEN_PLINE       0x00
00503 #define DL_CLOSED_PLINE     0x01
00504 #define DL_POLYLINE3D       0x08
00505 #define DL_PFACE_MESH       0x40
00506 #define DL_PGON_MESH        0x10
00507 // Vertices follow entity, required in POLYLINES
00508 #define DL_VERTS_FOLLOW_CODE   66 // Value should always be 1
00509 #define DL_VERTEX_COORD_CODE   10
00510
00511
```

```
00512 // LAYER flags
00513 #define DL_FROZEN           1
00514 #define DL_FROZEN_BY_DEF    2
00515 #define DL_LOCKED           4
00516 #define DL_OBJECT_USED     64   // Object is ref'd in the dwg
00517
00518 #define DL_BLOCK_EN_CODE   -2   // Block entity definition
00519 #define DL_E_NAME          -1   // Entity name
00520
00521 // Extended data codes
00522 #define DL_EXTD_SENTINEL    (-3)
00523 #define DL_EXTD_STR         1000
00524 #define DL_EXTD_APP_NAME    1001
00525 #define DL_EXTD_CTL_STR     1002
00526 #define DL_EXTD_LYR_STR     1003
00527 #define DL_EXTD_CHUNK       1004
00528 #define DL_EXTD_HANDLE      1005
00529 #define DL_EXTD_POINT       1010
00530 #define DL_EXTD_POS         1011
00531 #define DL_EXTD_DISP        1012
00532 #define DL_EXTD_DIR         1013
00533 #define DL_EXTD_FLOAT       1040
00534 #define DL_EXTD_DIST        1041
00535 #define DL_EXTD_SCALE       1042
00536 #define DL_EXTD_INT16       1070
00537 #define DL_EXTD_INT32       1071
00538
00539 // UCS codes for use in ads_trans
00540 #define DL_WCS_TRANS_CODE      0
00541 #define DL_UCS_TRANS_CODE      1
00542 #define DL_DCS_TRANS_CODE      2
00543 #define DL_PCS_TRANS_CODE      3
00544
00545 #endif
00546
```

6.3 dl_creationadapter.h

```
00001 /****************************************************************************
00002 ** Copyright (C) 2001-2013 RibbonSoft, GmbH. All rights reserved.
00003 **
00004 ** This file is part of the dxflib project.
00005 **
00006 ** This file is free software; you can redistribute it and/or modify
00007 ** it under the terms of the GNU General Public License as published by
00008 ** the Free Software Foundation; either version 2 of the License, or
00009 ** (at your option) any later version.
00010 **
00011 ** Licensees holding valid dxflib Professional Edition licenses may use
00012 ** this file in accordance with the dxflib Commercial License
00013 ** Agreement provided with the Software.
00014 **
00015 ** This file is provided AS IS with NO WARRANTY OF ANY KIND, INCLUDING THE
00016 ** WARRANTY OF DESIGN, MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE.
00017 **
00018 ** See http://www.ribbonsoft.com for further details.
00019 **
00020 ** Contact info@ribbonsoft.com if any conditions of this licensing are
00021 ** not clear to you.
00022 **
00023 **********************************************************************/
00024
00025 #ifndef DL_CREATIONADAPTER_H
00026 #define DL_CREATIONADAPTER_H
00027
00028 #include "dl_global.h"
00029
00030 #include "dl_creationinterface.h"
00031
00039 class DXFLIB_EXPORT DL_CreationAdapter : public DL_CreationInterface {
00040 public:
00041     DL_CreationAdapter() {}
00042     virtual ~DL_CreationAdapter() {}
00043     virtual void processCodeValuePair(unsigned int, const std::string&) {}
00044     virtual void endSection() {}
00045     virtual void addLayer(const DL_LayerData&) {}
00046     virtual void addLinetype(const DL_LinetypeData&) {}
00047     virtual void addLinetypeDash(double) {}
00048     virtual void addBlock(const DL_BlockData&) {}
00049     virtual void endBlock() {}
00050     virtual void addTextStyle(const DL_StyleData&) {}
00051     virtual void addPoint(const DL_PointData&) {}
00052     virtual void addLine(const DL_LineData&) {}
```

```
00053     virtual void addXLine(const DL_XLineData&) {}
00054     virtual void addRay(const DL_RayData&) {}
00055
00056     virtual void addArc(const DL_ArcData&) {}
00057     virtual void addCircle(const DL_CircleData&) {}
00058     virtual void addEllipse(const DL_EllipseData&) {}
00059
00060     virtual void addPolyline(const DL_PolylineData&) {}
00061     virtual void addVertex(const DL_VertexData&) {}
00062
00063     virtual void addSpline(const DL_SplineData&) {}
00064     virtual void addControlPoint(const DL_ControlPointData&) {}
00065     virtual void addFitPoint(const DL_FitPointData&) {}
00066     virtual void addKnot(const DL_KnotData&) {}
00067
00068     virtual void addInsert(const DL_InsertData&) {}
00069
00070     virtual void addMText(const DL_MTextData&) {}
00071     virtual void addMTextChunk(const std::string&) {}
00072     virtual void addText(const DL_TextData&) {}
00073     virtual void addArcAlignedText(const DL_ArcAlignedTextData&) {}
00074     virtual void addAttribute(const DL_AttributeData&) {}
00075
00076     virtual void addDimAlign(const DL_DimensionData&,
00077                              const DL_DimAlignedData&) {}
00078     virtual void addDimLinear(const DL_DimensionData&,
00079                               const DL_DimLinearData&) {}
00080     virtual void addDimRadial(const DL_DimensionData&,
00081                               const DL_DimRadialData&) {}
00082     virtual void addDimDiametric(const DL_DimensionData&,
00083                               const DL_DimDiametricData&) {}
00084     virtual void addDimAngular(const DL_DimensionData&,
00085                               const DL_DimAngular2LData&) {}
00086     virtual void addDimAngular3P(const DL_DimensionData&,
00087                               const DL_DimAngular3PData&) {}
00088     virtual void addDimOrdinate(const DL_DimensionData&,
00089                              const DL_DimOrdinateData&) {}
00090     virtual void addLeader(const DL_LeaderData&) {}
00091     virtual void addLeaderVertex(const DL_LeaderVertexData&) {}
00092
00093     virtual void addHatch(const DL_HatchData&) {}
00094
00095     virtual void addTrace(const DL_TraceData&) {}
00096     virtual void add3dFace(const DL_3dFaceData&) {}
00097     virtual void addSolid(const DL_SolidData&) {}
00098
00099     virtual void addImage(const DL_ImageData&) {}
00100     virtual void linkImage(const DL_ImageDefData&) {}
00101     virtual void addHatchLoop(const DL_HatchLoopData&) {}
00102     virtual void addHatchEdge(const DL_HatchEdgeData&) {}
00103
00104     virtual void addXRecord(const std::string&) {}
00105     virtual void addXRecordString(int, const std::string&) {}
00106     virtual void addXRecordReal(int, double) {}
00107     virtual void addXRecordInt(int, int) {}
00108     virtual void addXRecordBool(int, bool) {}
00109
00110     virtual void addXDataApp(const std::string&) {}
00111     virtual void addXDataString(int, const std::string&) {}
00112     virtual void addXDataReal(int, double) {}
00113     virtual void addXDataInt(int, int) {}
00114
00115     virtual void addDictionary(const DL_DictionaryData&) {}
00116     virtual void addDictionaryEntry(const DL_DictionaryEntryData&) {}
00117
00118     virtual void endEntity() {}
00119
00120     virtual void addComment(const std::string&) {}
00121
00122     virtual void setVariableVector(const std::string&,  double, double, double, int) {}
00123     virtual void setVariableString(const std::string&, const std::string&, int) {}
00124     virtual void setVariableInt(const std::string&, int, int) {}
00125     virtual void setVariableDouble(const std::string&, double, int) {}
00126 #ifdef DL_COMPAT
00127     virtual void setVariableVector(const char*,  double, double, double, int) {}
00128     virtual void setVariableString(const char*, const char*, int) {}
00129     virtual void setVariableInt(const char*, int, int) {}
00130     virtual void setVariableDouble(const char*, double, int) {}
00131     virtual void processCodeValuePair(unsigned int, char*) {}
00132     virtual void addComment(const char*) {}
00133     virtual void addMTextChunk(const char*) {}
00134 #endif
00135     virtual void endSequence() {}
00136 };
00137
00138 #endif
```

6.4 dl_creationinterface.h

```
00001 /**********************************************************************
00002 ** Copyright (C) 2001-2013 RibbonSoft, GmbH. All rights reserved.
00003 **
00004 ** This file is part of the dxflib project.
00005 **
00006 ** This file is free software; you can redistribute it and/or modify
00007 ** it under the terms of the GNU General Public License as published by
00008 ** the Free Software Foundation; either version 2 of the License, or
00009 ** (at your option) any later version.
00010 **
00011 ** Licensees holding valid dxflib Professional Edition licenses may use
00012 ** this file in accordance with the dxflib Commercial License
00013 ** Agreement provided with the Software.
00014 **
00015 ** This file is provided AS IS with NO WARRANTY OF ANY KIND, INCLUDING THE
00016 ** WARRANTY OF DESIGN, MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE.
00017 **
00018 ** See http://www.ribbonsoft.com for further details.
00019 **
00020 ** Contact info@ribbonsoft.com if any conditions of this licensing are
00021 ** not clear to you.
00022 **
00023 **********************************************************************/
00024
00025 #ifndef DL_CREATIONINTERFACE_H
00026 #define DL_CREATIONINTERFACE_H
00027
00028 #include "dl_global.h"
00029
00030 #include <string.h>
00031
00032 #include "dl_attributes.h"
00033 #include "dl_codes.h"
00034 #include "dl_entities.h"
00035 #include "dl_extrusion.h"
00036
00047 class DXFLIB_EXPORT DL_CreationInterface {
00048 public:
00049     DL_CreationInterface() {
00050         extrusion = new DL_Extrusion;
00051     }
00052     virtual ~DL_CreationInterface() {
00053         delete extrusion;
00054     }
00055
00060     virtual void processCodeValuePair(unsigned int groupCode, const std::string& groupValue) = 0;
00061
00065     virtual void endSection() = 0;
00066
00070     virtual void addLayer(const DL_LayerData& data) = 0;
00071
00075     virtual void addLinetype(const DL_LinetypeData& data) = 0;
00076
00080     virtual void addLinetypeDash(double length) = 0;
00081
00088     virtual void addBlock(const DL_BlockData& data) = 0;
00089
00091     virtual void endBlock() = 0;
00092
00094     virtual void addTextStyle(const DL_StyleData& data) = 0;
00095
00097     virtual void addPoint(const DL_PointData& data) = 0;
00098
00100     virtual void addLine(const DL_LineData& data) = 0;
00101
00103     virtual void addXLine(const DL_XLineData& data) = 0;
00104
00106     virtual void addRay(const DL_RayData& data) = 0;
00107
00109     virtual void addArc(const DL_ArcData& data) = 0;
00110
00112     virtual void addCircle(const DL_CircleData& data) = 0;
00113
00115     virtual void addEllipse(const DL_EllipseData& data) = 0;
00116
00118     virtual void addPolyline(const DL_PolylineData& data) = 0;
00119
00121     virtual void addVertex(const DL_VertexData& data) = 0;
00122
00124     virtual void addSpline(const DL_SplineData& data) = 0;
00125
00127     virtual void addControlPoint(const DL_ControlPointData& data) = 0;
00128
00130     virtual void addFitPoint(const DL_FitPointData& data) = 0;
00131
```

```
00133     virtual void addKnot(const DL_KnotData& data) = 0;
00134
00136     virtual void addInsert(const DL_InsertData& data) = 0;
00137
00139     virtual void addTrace(const DL_TraceData& data) = 0;
00140
00142     virtual void add3dFace(const DL_3dFaceData& data) = 0;
00143
00145     virtual void addSolid(const DL_SolidData& data) = 0;
00146
00147
00149     virtual void addMText(const DL_MTextData& data) = 0;
00150
00156     virtual void addMTextChunk(const std::string& text) = 0;
00157
00159     virtual void addText(const DL_TextData& data) = 0;
00160
00162     virtual void addArcAlignedText(const DL_ArcAlignedTextData& data) = 0;
00163
00165     virtual void addAttribute(const DL_AttributeData& data) = 0;
00166
00170     virtual void addDimAlign(const DL_DimensionData& data,
00171                              const DL_DimAlignedData& edata) = 0;
00175     virtual void addDimLinear(const DL_DimensionData& data,
00176                               const DL_DimLinearData& edata) = 0;
00177
00181     virtual void addDimRadial(const DL_DimensionData& data,
00182                               const DL_DimRadialData& edata) = 0;
00183
00187     virtual void addDimDiametric(const DL_DimensionData& data,
00188                               const DL_DimDiametricData& edata) = 0;
00189
00193     virtual void addDimAngular(const DL_DimensionData& data,
00194                               const DL_DimAngular2LData& edata) = 0;
00195
00199     virtual void addDimAngular3P(const DL_DimensionData& data,
00200                               const DL_DimAngular3PData& edata) = 0;
00201
00205     virtual void addDimOrdinate(const DL_DimensionData& data,
00206                              const DL_DimOrdinateData& edata) = 0;
00207
00211     virtual void addLeader(const DL_LeaderData& data) = 0;
00212
00216     virtual void addLeaderVertex(const DL_LeaderVertexData& data) = 0;
00217
00221     virtual void addHatch(const DL_HatchData& data) = 0;
00222
00226     virtual void addImage(const DL_ImageData& data) = 0;
00227
00231     virtual void linkImage(const DL_ImageDefData& data) = 0;
00232
00236     virtual void addHatchLoop(const DL_HatchLoopData& data) = 0;
00237
00241     virtual void addHatchEdge(const DL_HatchEdgeData& data) = 0;
00242
00246     virtual void addXRecord(const std::string& handle) = 0;
00247
00251     virtual void addXRecordString(int code, const std::string& value) = 0;
00252
00256     virtual void addXRecordReal(int code, double value) = 0;
00257
00261     virtual void addXRecordInt(int code, int value) = 0;
00262
00266     virtual void addXRecordBool(int code, bool value) = 0;
00267
00271     virtual void addXDataApp(const std::string& appId) = 0;
00272
00276     virtual void addXDataString(int code, const std::string& value) = 0;
00277
00281     virtual void addXDataReal(int code, double value) = 0;
00282
00286     virtual void addXDataInt(int code, int value) = 0;
00287
00291     virtual void addDictionary(const DL_DictionaryData& data) = 0;
00292
00296     virtual void addDictionaryEntry(const DL_DictionaryEntryData& data) = 0;
00297
00301     virtual void endEntity() = 0;
00302
00306     virtual void addComment(const std::string& comment) = 0;
00307
00311     virtual void setVariableVector(const std::string& key,  double v1, double v2, double v3, int code)
      = 0;
00312
00316     virtual void setVariableString(const std::string& key, const std::string& value, int code) = 0;
00317
00321     virtual void setVariableInt(const std::string& key, int value, int code) = 0;
```

```
00322
00326     virtual void setVariableDouble(const std::string& key, double value, int code) = 0;
00327
00328 #ifdef DL_COMPAT
00329     virtual void setVariableVector(const char* key,  double v1, double v2, double v3, int code) = 0;
00330     virtual void setVariableString(const char* key, const char* value, int code) = 0;
00331     virtual void setVariableInt(const char* key, int value, int code) = 0;
00332     virtual void setVariableDouble(const char* key, double value, int code) = 0;
00333     virtual void processCodeValuePair(unsigned int groupCode, char* groupValue) = 0;
00334     virtual void addComment(const char* comment) = 0;
00335     virtual void addMTextChunk(const char* text) = 0;
00336 #endif
00337
00341     virtual void endSequence() = 0;
00342
00344     void setAttributes(const DL_Attributes& attrib) {
00345         attributes = attrib;
00346     }
00347
00349     DL_Attributes getAttributes() {
00350         return attributes;
00351     }
00352
00354     void setExtrusion(double dx, double dy, double dz, double elevation) {
00355         extrusion->setDirection(dx, dy, dz);
00356         extrusion->setElevation(elevation);
00357     }
00358
00360     DL_Extrusion* getExtrusion() {
00361         return extrusion;
00362     }
00363
00364 protected:
00365     DL_Attributes attributes;
00366     DL_Extrusion *extrusion;
00367 };
00368
00369 #endif
```

6.5 dl_dxf.h

```
00001 /**********************************************************************
00002 ** Copyright (C) 2001-2013 RibbonSoft, GmbH. All rights reserved.
00003 **
00004 ** This file is part of the dxflib project.
00005 **
00006 ** This file is free software; you can redistribute it and/or modify
00007 ** it under the terms of the GNU General Public License as published by
00008 ** the Free Software Foundation; either version 2 of the License, or
00009 ** (at your option) any later version.
00010 **
00011 ** Licensees holding valid dxflib Professional Edition licenses may use
00012 ** this file in accordance with the dxflib Commercial License
00013 ** Agreement provided with the Software.
00014 **
00015 ** This file is provided AS IS with NO WARRANTY OF ANY KIND, INCLUDING THE
00016 ** WARRANTY OF DESIGN, MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE.
00017 **
00018 ** See http://www.ribbonsoft.com for further details.
00019 **
00020 ** Contact info@ribbonsoft.com if any conditions of this licensing are
00021 ** not clear to you.
00022 **
00023 **********************************************************************/
00024
00025 #ifndef DL_DXF_H
00026 #define DL_DXF_H
00027
00028 #include "dl_global.h"
00029
00030 #include <limits>
00031 #include <stdio.h>
00032 #include <stdlib.h>
00033 #include <string>
00034 #include <sstream>
00035 #include <map>
00036
00037 #include "dl_attributes.h"
00038 #include "dl_codes.h"
00039 #include "dl_entities.h"
00040 #include "dl_writer_ascii.h"
00041
00042 #ifdef _WIN32
```

```
00043 #undef M_PI
00044 #define M_PI   3.14159265358979323846
00045 #pragma warning(disable : 4800)
00046 #endif
00047
00048 #ifndef M_PI
00049 #define M_PI 3.14159265358979323846264338327795
00050 #endif
00051
00052 #ifndef DL_NANDOUBLE
00053 #define DL_NANDOUBLE std::numeric_limits<double>::quiet_NaN()
00054 #endif
00055
00056 class DL_CreationInterface;
00057 class DL_WriterA;
00058
00059
00060 #define DL_VERSION "3.26.4.0"
00061
00062 #define DL_VERSION_MAJOR    3
00063 #define DL_VERSION_MINOR    26
00064 #define DL_VERSION_REV      4
00065 #define DL_VERSION_BUILD    0
00066
00067 #define DL_UNKNOWN               0
00068 #define DL_LAYER                10
00069 #define DL_BLOCK                11
00070 #define DL_ENDBLK               12
00071 #define DL_LINETYPE             13
00072 #define DL_STYLE                20
00073 #define DL_SETTING              50
00074 #define DL_ENTITY_POINT        100
00075 #define DL_ENTITY_LINE         101
00076 #define DL_ENTITY_POLYLINE     102
00077 #define DL_ENTITY_LWPOLYLINE   103
00078 #define DL_ENTITY_VERTEX       104
00079 #define DL_ENTITY_SPLINE       105
00080 #define DL_ENTITY_KNOT         106
00081 #define DL_ENTITY_CONTROLPOINT 107
00082 #define DL_ENTITY_ARC          108
00083 #define DL_ENTITY_CIRCLE       109
00084 #define DL_ENTITY_ELLIPSE      110
00085 #define DL_ENTITY_INSERT       111
00086 #define DL_ENTITY_TEXT         112
00087 #define DL_ENTITY_MTEXT        113
00088 #define DL_ENTITY_DIMENSION    114
00089 #define DL_ENTITY_LEADER       115
00090 #define DL_ENTITY_HATCH        116
00091 #define DL_ENTITY_ATTRIB       117
00092 #define DL_ENTITY_IMAGE        118
00093 #define DL_ENTITY_IMAGEDEF     119
00094 #define DL_ENTITY_TRACE        120
00095 #define DL_ENTITY_SOLID        121
00096 #define DL_ENTITY_3DFACE       122
00097 #define DL_ENTITY_XLINE        123
00098 #define DL_ENTITY_RAY          124
00099 #define DL_ENTITY_ARCALIGNEDTEXT 125
00100 #define DL_ENTITY_SEQEND       126
00101 #define DL_XRECORD             200
00102 #define DL_DICTIONARY          210
00103
00104
00122 class DXFLIB_EXPORT DL_Dxf {
00123 public:
00124     DL_Dxf();
00125     ~DL_Dxf();
00126
00127     bool in(const std::string& file,
00128             DL_CreationInterface* creationInterface);
00129     bool readDxfGroups(FILE* fp,
00130                        DL_CreationInterface* creationInterface);
00131     static bool getStrippedLine(std::string& s, unsigned int size,
00132                                 FILE* stream, bool stripSpace = true);
00133
00134     bool readDxfGroups(std::istream& stream,
00135                        DL_CreationInterface* creationInterface);
00136     bool in(std::istream &stream,
00137             DL_CreationInterface* creationInterface);
00138     static bool getStrippedLine(std::string& s, unsigned int size,
00139                                 std::istream& stream, bool stripSpace = true);
00140
00141     static bool stripWhiteSpace(char** s, bool stripSpaces = true);
00142
00143     bool processDXFGroup(DL_CreationInterface* creationInterface,
00144                          int groupCode, const std::string& groupValue);
00145     void addSetting(DL_CreationInterface* creationInterface);
00146     void addLayer(DL_CreationInterface* creationInterface);
```

```
00147     void addLinetype(DL_CreationInterface *creationInterface);
00148     void addBlock(DL_CreationInterface* creationInterface);
00149     void endBlock(DL_CreationInterface* creationInterface);
00150     void addTextStyle(DL_CreationInterface* creationInterface);
00151
00152     void addPoint(DL_CreationInterface* creationInterface);
00153     void addLine(DL_CreationInterface* creationInterface);
00154     void addXLine(DL_CreationInterface* creationInterface);
00155     void addRay(DL_CreationInterface* creationInterface);
00156
00157     void addPolyline(DL_CreationInterface* creationInterface);
00158     void addVertex(DL_CreationInterface* creationInterface);
00159
00160     void addSpline(DL_CreationInterface* creationInterface);
00161
00162     void addArc(DL_CreationInterface* creationInterface);
00163     void addCircle(DL_CreationInterface* creationInterface);
00164     void addEllipse(DL_CreationInterface* creationInterface);
00165     void addInsert(DL_CreationInterface* creationInterface);
00166
00167     void addTrace(DL_CreationInterface* creationInterface);
00168     void add3dFace(DL_CreationInterface* creationInterface);
00169     void addSolid(DL_CreationInterface* creationInterface);
00170
00171     void addMText(DL_CreationInterface* creationInterface);
00172     void addText(DL_CreationInterface* creationInterface);
00173     void addArcAlignedText(DL_CreationInterface* creationInterface);
00174
00175     void addAttribute(DL_CreationInterface* creationInterface);
00176
00177     DL_DimensionData getDimData();
00178     void addDimLinear(DL_CreationInterface* creationInterface);
00179     void addDimAligned(DL_CreationInterface* creationInterface);
00180     void addDimRadial(DL_CreationInterface* creationInterface);
00181     void addDimDiametric(DL_CreationInterface* creationInterface);
00182     void addDimAngular(DL_CreationInterface* creationInterface);
00183     void addDimAngular3P(DL_CreationInterface* creationInterface);
00184     void addDimOrdinate(DL_CreationInterface* creationInterface);
00185
00186     void addLeader(DL_CreationInterface* creationInterface);
00187
00188     void addHatch(DL_CreationInterface* creationInterface);
00189     void addHatchLoop();
00190     void addHatchEdge();
00191     bool handleHatchData(DL_CreationInterface* creationInterface);
00192
00193     void addImage(DL_CreationInterface* creationInterface);
00194     void addImageDef(DL_CreationInterface* creationInterface);
00195
00196     void addComment(DL_CreationInterface* creationInterface, const std::string& comment);
00197
00198     void addDictionary(DL_CreationInterface* creationInterface);
00199     void addDictionaryEntry(DL_CreationInterface* creationInterface);
00200
00201     bool handleXRecordData(DL_CreationInterface* creationInterface);
00202     bool handleDictionaryData(DL_CreationInterface* creationInterface);
00203
00204     bool handleXData(DL_CreationInterface *creationInterface);
00205     bool handleMTextData(DL_CreationInterface* creationInterface);
00206     bool handleLWPolylineData(DL_CreationInterface* creationInterface);
00207     bool handleSplineData(DL_CreationInterface* creationInterface);
00208     bool handleLeaderData(DL_CreationInterface* creationInterface);
00209     bool handleLinetypeData(DL_CreationInterface* creationInterface);
00210
00211     void endEntity(DL_CreationInterface* creationInterface);
00212
00213     void endSequence(DL_CreationInterface* creationInterface);
00214
00215     //int  stringToInt(const char* s, bool* ok=NULL);
00216
00217     DL_WriterA* out(const char* file,
00218                     DL_Codes::version version=DL_VERSION_2000);
00219
00220     void writeHeader(DL_WriterA& dw);
00221
00222     void writePoint(DL_WriterA& dw,
00223                     const DL_PointData& data,
00224                     const DL_Attributes& attrib);
00225     void writeLine(DL_WriterA& dw,
00226                    const DL_LineData& data,
00227                    const DL_Attributes& attrib);
00228     void writeXLine(DL_WriterA& dw,
00229                     const DL_XLineData& data,
00230                     const DL_Attributes& attrib);
00231     void writeRay(DL_WriterA& dw,
00232                     const DL_RayData& data,
00233                     const DL_Attributes& attrib);
```

```
00234     void writePolyline(DL_WriterA& dw,
00235                        const DL_PolylineData& data,
00236                        const DL_Attributes& attrib);
00237     void writeVertex(DL_WriterA& dw,
00238                      const DL_VertexData& data);
00239     void writePolylineEnd(DL_WriterA& dw);
00240     void writeSpline(DL_WriterA& dw,
00241                      const DL_SplineData& data,
00242                      const DL_Attributes& attrib);
00243     void writeControlPoint(DL_WriterA& dw,
00244                      const DL_ControlPointData& data);
00245     void writeFitPoint(DL_WriterA& dw,
00246                      const DL_FitPointData& data);
00247     void writeKnot(DL_WriterA& dw,
00248                      const DL_KnotData& data);
00249     void writeCircle(DL_WriterA& dw,
00250                      const DL_CircleData& data,
00251                      const DL_Attributes& attrib);
00252     void writeArc(DL_WriterA& dw,
00253                   const DL_ArcData& data,
00254                   const DL_Attributes& attrib);
00255     void writeEllipse(DL_WriterA& dw,
00256                       const DL_EllipseData& data,
00257                       const DL_Attributes& attrib);
00258     void writeSolid(DL_WriterA& dw,
00259                     const DL_SolidData& data,
00260                     const DL_Attributes& attrib);
00261     void writeTrace(DL_WriterA& dw,
00262                     const DL_TraceData& data,
00263                     const DL_Attributes& attrib);
00264     void write3dFace(DL_WriterA& dw,
00265                      const DL_3dFaceData& data,
00266                      const DL_Attributes& attrib);
00267     void writeInsert(DL_WriterA& dw,
00268                      const DL_InsertData& data,
00269                      const DL_Attributes& attrib);
00270     void writeMText(DL_WriterA& dw,
00271                     const DL_MTextData& data,
00272                     const DL_Attributes& attrib);
00273     void writeText(DL_WriterA& dw,
00274                    const DL_TextData& data,
00275                    const DL_Attributes& attrib);
00276     void writeAttribute(DL_WriterA& dw,
00277                         const DL_AttributeData& data,
00278                         const DL_Attributes& attrib);
00279     void writeDimStyleOverrides(DL_WriterA& dw,
00280                                 const DL_DimensionData& data);
00281     void writeDimAligned(DL_WriterA& dw,
00282                          const DL_DimensionData& data,
00283                          const DL_DimAlignedData& edata,
00284                          const DL_Attributes& attrib);
00285     void writeDimLinear(DL_WriterA& dw,
00286                         const DL_DimensionData& data,
00287                         const DL_DimLinearData& edata,
00288                         const DL_Attributes& attrib);
00289     void writeDimRadial(DL_WriterA& dw,
00290                         const DL_DimensionData& data,
00291                         const DL_DimRadialData& edata,
00292                         const DL_Attributes& attrib);
00293     void writeDimDiametric(DL_WriterA& dw,
00294                            const DL_DimensionData& data,
00295                            const DL_DimDiametricData& edata,
00296                            const DL_Attributes& attrib);
00297     void writeDimAngular2L(DL_WriterA& dw,
00298                            const DL_DimensionData& data,
00299                            const DL_DimAngular2LData& edata,
00300                            const DL_Attributes& attrib);
00301     void writeDimAngular3P(DL_WriterA& dw,
00302                            const DL_DimensionData& data,
00303                            const DL_DimAngular3PData& edata,
00304                            const DL_Attributes& attrib);
00305     void writeDimOrdinate(DL_WriterA& dw,
00306                           const DL_DimensionData& data,
00307                           const DL_DimOrdinateData& edata,
00308                           const DL_Attributes& attrib);
00309     void writeLeader(DL_WriterA& dw,
00310                      const DL_LeaderData& data,
00311                      const DL_Attributes& attrib);
00312     void writeLeaderVertex(DL_WriterA& dw,
00313                            const DL_LeaderVertexData& data);
00314     void writeLeaderEnd(DL_WriterA& dw,
00315                         const DL_LeaderData& data);
00316     void writeHatch1(DL_WriterA& dw,
00317                      const DL_HatchData& data,
00318                      const DL_Attributes& attrib);
00319     void writeHatch2(DL_WriterA& dw,
00320                      const DL_HatchData& data,
```

```
00321                     const DL_Attributes& attrib);
00322     void writeHatchLoop1(DL_WriterA& dw,
00323                          const DL_HatchLoopData& data);
00324     void writeHatchLoop2(DL_WriterA& dw,
00325                          const DL_HatchLoopData& data);
00326     void writeHatchEdge(DL_WriterA& dw,
00327                         const DL_HatchEdgeData& data);
00328
00329     unsigned long writeImage(DL_WriterA& dw,
00330                   const DL_ImageData& data,
00331                   const DL_Attributes& attrib);
00332
00333     void writeImageDef(DL_WriterA& dw, int handle,
00334                        const DL_ImageData& data);
00335
00336     void writeLayer(DL_WriterA& dw,
00337                     const DL_LayerData& data,
00338                     const DL_Attributes& attrib);
00339
00340     void writeLinetype(DL_WriterA& dw,
00341                        const DL_LinetypeData& data);
00342
00343     void writeAppid(DL_WriterA& dw, const std::string& name);
00344
00345     void writeBlock(DL_WriterA& dw,
00346                     const DL_BlockData& data);
00347     void writeEndBlock(DL_WriterA& dw, const std::string& name);
00348
00349     void writeVPort(DL_WriterA& dw);
00350     void writeStyle(DL_WriterA& dw, const DL_StyleData& style);
00351     void writeView(DL_WriterA& dw);
00352     void writeUcs(DL_WriterA& dw);
00353     void writeDimStyle(DL_WriterA& dw,
00354                        double dimasz, double dimexe, double dimexo,
00355                        double dimgap, double dimtxt);
00356     void writeBlockRecord(DL_WriterA& dw);
00357     void writeBlockRecord(DL_WriterA& dw, const std::string& name);
00358     void writeObjects(DL_WriterA& dw, const std::string& appDictionaryName = "");
00359     void writeAppDictionary(DL_WriterA& dw);
00360     unsigned long writeDictionaryEntry(DL_WriterA& dw, const std::string& name);
00361     void writeXRecord(DL_WriterA& dw, int handle, int value);
00362     void writeXRecord(DL_WriterA& dw, int handle, double value);
00363     void writeXRecord(DL_WriterA& dw, int handle, bool value);
00364     void writeXRecord(DL_WriterA& dw, int handle, const std::string& value);
00365     void writeObjectsEnd(DL_WriterA& dw);
00366
00367     void writeComment(DL_WriterA& dw, const std::string& comment);
00368
00373     //static double toReal(const char* value, double def=0.0);
00374
00379 //    static int toInt(const char* value, int def=0) {
00380 //        if (value!=NULL && value[0] != '\0') {
00381 //            return atoi(value);
00382 //        }
00383
00384 //        return def;
00385 //    }
00386
00391 //    static const char* toString(const char* value, const char* def="") {
00392 //        if (value!=NULL && value[0] != '\0') {
00393 //            return value;
00394 //        } else {
00395 //            return def;
00396 //        }
00397 //    }
00398
00399     static bool checkVariable(const char* var, DL_Codes::version version);
00400
00401     DL_Codes::version getVersion() {
00402         return version;
00403     }
00404
00405     int getLibVersion(const std::string &str);
00406
00407     static void test();
00408
00409     bool hasValue(int code) {
00410         return values.count(code)==1;
00411     }
00412
00413     int getIntValue(int code, int def) {
00414         if (!hasValue(code)) {
00415             return def;
00416         }
00417         return toInt(values[code]);
00418     }
00419
```

```
00420     int toInt(const std::string& str) {
00421         char* p;
00422         return strtol(str.c_str(), &p, 10);
00423     }
00424
00425     int getInt16Value(int code, int def) {
00426         if (!hasValue(code)) {
00427             return def;
00428         }
00429         return toInt16(values[code]);
00430     }
00431
00432     int toInt16(const std::string& str) {
00433         char* p;
00434         return strtol(str.c_str(), &p, 16);
00435     }
00436
00437     bool toBool(const std::string& str) {
00438         char* p;
00439         return (bool)strtol(str.c_str(), &p, 10);
00440     }
00441
00442     std::string getStringValue(int code, const std::string& def) {
00443         if (!hasValue(code)) {
00444             return def;
00445         }
00446         return values[code];
00447     }
00448
00449     double getRealValue(int code, double def) {
00450         if (!hasValue(code)) {
00451             return def;
00452         }
00453         return toReal(values[code]);
00454     }
00455
00456     double toReal(const std::string& str) {
00457         double ret;
00458         // make sure the real value uses '.' not ',':
00459         std::string str2 = str;
00460         std::replace(str2.begin(), str2.end(), ',', '.');
00461         // make sure c++ expects '.' not ',':
00462         std::istringstream istr(str2);
00463         //istr.imbue(std::locale("C"));
00464         istr » ret;
00465         return ret;
00466     }
00467
00468 private:
00469     DL_Codes::version version;
00470
00471     std::string polylineLayer;
00472     double* vertices;
00473     int maxVertices;
00474     int vertexIndex;
00475
00476     double* knots;
00477     int maxKnots;
00478     int knotIndex;
00479
00480     double* weights;
00481     int weightIndex;
00482
00483     double* controlPoints;
00484     int maxControlPoints;
00485     int controlPointIndex;
00486
00487     double* fitPoints;
00488     int maxFitPoints;
00489     int fitPointIndex;
00490
00491     double* leaderVertices;
00492     int maxLeaderVertices;
00493     int leaderVertexIndex;
00494
00495     bool firstHatchLoop;
00496     DL_HatchEdgeData hatchEdge;
00497     std::vector<std::vector<DL_HatchEdgeData> > hatchEdges;
00498
00499     std::string xRecordHandle;
00500     bool xRecordValues;
00501
00502     // Only the useful part of the group code
00503     std::string groupCodeTmp;
00504     // ...same as integer
00505     unsigned int groupCode;
00506     // Only the useful part of the group value
```

```
00507     std::string groupValue;
00508     // Current entity type
00509     int currentObjectType;
00510     // Value of the current setting
00511     char settingValue[DL_DXF_MAXLINE+1];
00512     // Key of the current setting (e.g. "$ACADVER")
00513     std::string settingKey;
00514     // Stores the group codes
00515     std::map<int, std::string> values;
00516     // First call of this method. We initialize all group values in
00517     //  the first call.
00518     bool firstCall;
00519     // Attributes of the current entity (layer, color, width, line type)
00520     DL_Attributes attrib;
00521     // library version. hex: 0x20003001 = 2.0.3.1
00522     int libVersion;
00523     // app specific dictionary handle:
00524     unsigned long appDictionaryHandle;
00525     // handle of standard text style, referenced by dimstyle:
00526     unsigned long styleHandleStd;
00527 };
00528
00529 #endif
00530
00531 // EOF
```

6.6 dl_entities.h

```
00001 /**********************************************************************
00002 ** Copyright (C) 2001-2013 RibbonSoft, GmbH. All rights reserved.
00003 **
00004 ** This file is part of the dxflib project.
00005 **
00006 ** This file is free software; you can redistribute it and/or modify
00007 ** it under the terms of the GNU General Public License as published by
00008 ** the Free Software Foundation; either version 2 of the License, or
00009 ** (at your option) any later version.
00010 **
00011 ** Licensees holding valid dxflib Professional Edition licenses may use
00012 ** this file in accordance with the dxflib Commercial License
00013 ** Agreement provided with the Software.
00014 **
00015 ** This file is provided AS IS with NO WARRANTY OF ANY KIND, INCLUDING THE
00016 ** WARRANTY OF DESIGN, MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE.
00017 **
00018 ** See http://www.ribbonsoft.com for further details.
00019 **
00020 ** Contact info@ribbonsoft.com if any conditions of this licensing are
00021 ** not clear to you.
00022 **
00023 **********************************************************************/
00024
00025 #ifndef DL_ENTITIES_H
00026 #define DL_ENTITIES_H
00027
00028 #include "dl_global.h"
00029
00030 #include <string>
00031 #include <vector>
00032
00036 struct DXFLIB_EXPORT DL_LayerData {
00041     DL_LayerData(const std::string& name,
00042                  int flags, bool off = false) :
00043         name(name), flags(flags), off(off) {
00044     }
00045
00047     std::string name;
00049     int flags;
00051     bool off;
00052 };
00053
00054
00055
00059 struct DXFLIB_EXPORT DL_BlockData {
00064     DL_BlockData(const std::string& bName,
00065                  int bFlags,
00066                  double bbpx, double bbpy, double bbpz) {
00067         name = bName;
00068         flags = bFlags;
00069         bpx = bbpx;
00070         bpy = bbpy;
00071         bpz = bbpz;
00072     }
```

```
00073
00075     std::string name;
00077     int flags;
00079     double bpx;
00081     double bpy;
00083     double bpz;
00084 };
00085
00086
00090 struct DXFLIB_EXPORT DL_LinetypeData {
00095     DL_LinetypeData(
00096         const std::string& name,
00097         const std::string& description,
00098         int flags,
00099         int numberOfDashes,
00100         double patternLength,
00101         double* pattern = NULL
00102         )
00103         : name(name),
00104         description(description),
00105         flags(flags),
00106         numberOfDashes(numberOfDashes),
00107         patternLength(patternLength),
00108         pattern(pattern)
00109     {}
00110
00112     std::string name;
00114     std::string description;
00116     int flags;
00118     int numberOfDashes;
00120     double patternLength;
00122     double* pattern;
00123 };
00124
00125
00126
00130 struct DXFLIB_EXPORT DL_StyleData {
00135     DL_StyleData(
00136         const std::string& name,
00137         int flags,
00138         double fixedTextHeight,
00139         double widthFactor,
00140         double obliqueAngle,
00141         int textGenerationFlags,
00142         double lastHeightUsed,
00143         const std::string& primaryFontFile,
00144         const std::string& bigFontFile
00145         )
00146         : name(name),
00147         flags(flags),
00148         fixedTextHeight(fixedTextHeight),
00149         widthFactor(widthFactor),
00150         obliqueAngle(obliqueAngle),
00151         textGenerationFlags(textGenerationFlags),
00152         lastHeightUsed(lastHeightUsed),
00153         primaryFontFile(primaryFontFile),
00154         bigFontFile(bigFontFile),
00155         bold(false),
00156         italic(false) {
00157     }
00158
00159     bool operator==(const DL_StyleData& other) {
00160         // ignore lastHeightUsed:
00161         return (name==other.name &&
00162             flags==other.flags &&
00163             fixedTextHeight==other.fixedTextHeight &&
00164             widthFactor==other.widthFactor &&
00165             obliqueAngle==other.obliqueAngle &&
00166             textGenerationFlags==other.textGenerationFlags &&
00167             primaryFontFile==other.primaryFontFile &&
00168             bigFontFile==other.bigFontFile);
00169     }
00170
00172     std::string name;
00174     int flags;
00176     double fixedTextHeight;
00178     double widthFactor;
00180     double obliqueAngle;
00182     int textGenerationFlags;
00184     double lastHeightUsed;
00186     std::string primaryFontFile;
00188     std::string bigFontFile;
00189
00190     bool bold;
00191     bool italic;
00192 };
00193
```

```
00197 struct DXFLIB_EXPORT DL_PointData {
00202     DL_PointData(double px=0.0, double py=0.0, double pz=0.0) {
00203         x = px;
00204         y = py;
00205         z = pz;
00206     }
00207
00209     double x;
00211     double y;
00213     double z;
00214 };
00215
00216
00217
00221 struct DXFLIB_EXPORT DL_LineData {
00226     DL_LineData(double lx1, double ly1, double lz1,
00227                 double lx2, double ly2, double lz2) {
00228         x1 = lx1;
00229         y1 = ly1;
00230         z1 = lz1;
00231
00232         x2 = lx2;
00233         y2 = ly2;
00234         z2 = lz2;
00235     }
00236
00238     double x1;
00240     double y1;
00242     double z1;
00243
00245     double x2;
00247     double y2;
00249     double z2;
00250 };
00251
00255 struct DXFLIB_EXPORT DL_XLineData {
00260     DL_XLineData(double bx, double by, double bz,
00261                  double dx, double dy, double dz) :
00262         bx(bx), by(by), bz(bz),
00263         dx(dx), dy(dy), dz(dz) {
00264     }
00265
00267     double bx;
00269     double by;
00271     double bz;
00272
00274     double dx;
00276     double dy;
00278     double dz;
00279 };
00280
00284 struct DXFLIB_EXPORT DL_RayData {
00289     DL_RayData(double bx, double by, double bz,
00290                double dx, double dy, double dz) :
00291         bx(bx), by(by), bz(bz),
00292         dx(dx), dy(dy), dz(dz) {
00293     }
00294
00296     double bx;
00298     double by;
00300     double bz;
00301
00303     double dx;
00305     double dy;
00307     double dz;
00308 };
00309
00310
00311
00315 struct DXFLIB_EXPORT DL_ArcData {
00320     DL_ArcData(double acx, double acy, double acz,
00321                double aRadius,
00322                double aAngle1, double aAngle2) {
00323
00324         cx = acx;
00325         cy = acy;
00326         cz = acz;
00327         radius = aRadius;
00328         angle1 = aAngle1;
00329         angle2 = aAngle2;
00330     }
00331
00333     double cx;
00335     double cy;
00337     double cz;
00338
00340     double radius;
```

```
00342     double angle1;
00344     double angle2;
00345 };
00346
00347
00348
00352 struct DXFLIB_EXPORT DL_CircleData {
00357     DL_CircleData(double acx, double acy, double acz,
00358                   double aRadius) {
00359
00360         cx = acx;
00361         cy = acy;
00362         cz = acz;
00363         radius = aRadius;
00364     }
00365
00367     double cx;
00369     double cy;
00371     double cz;
00372
00374     double radius;
00375 };
00376
00377
00378
00382 struct DXFLIB_EXPORT DL_PolylineData {
00387     DL_PolylineData(int pNumber, int pMVerteces, int pNVerteces, int pFlags, double pElevation = 0.0)
    {
00388         number = pNumber;
00389         m = pMVerteces;
00390         n = pNVerteces;
00391         elevation = pElevation;
00392         flags = pFlags;
00393     }
00394
00396     unsigned int number;
00397
00399     unsigned int m;
00400
00402     unsigned int n;
00403
00405     double elevation;
00406
00408     int flags;
00409 };
00410
00411
00412
00416 struct DXFLIB_EXPORT DL_VertexData {
00421     DL_VertexData(double px=0.0, double py=0.0, double pz=0.0,
00422                   double pBulge=0.0) {
00423         x = px;
00424         y = py;
00425         z = pz;
00426         bulge = pBulge;
00427     }
00428
00430     double x;
00432     double y;
00434     double z;
00437     double bulge;
00438 };
00439
00440
00444 struct DXFLIB_EXPORT DL_TraceData {
00445     DL_TraceData() {
00446         thickness = 0.0;
00447         for (int i=0; i<4; i++) {
00448             x[i] = 0.0;
00449             y[i] = 0.0;
00450             z[i] = 0.0;
00451         }
00452     }
00453
00458     DL_TraceData(double sx1, double sy1, double sz1,
00459                  double sx2, double sy2, double sz2,
00460                  double sx3, double sy3, double sz3,
00461                  double sx4, double sy4, double sz4,
00462                  double sthickness=0.0) {
00463
00464         thickness = sthickness;
00465
00466         x[0] = sx1;
00467         y[0] = sy1;
00468         z[0] = sz1;
00469
00470         x[1] = sx2;
```

```
00471         y[1] = sy2;
00472         z[1] = sz2;
00473
00474         x[2] = sx3;
00475         y[2] = sy3;
00476         z[2] = sz3;
00477
00478         x[3] = sx4;
00479         y[3] = sy4;
00480         z[3] = sz4;
00481     }
00482
00484     double thickness;
00485
00487     double x[4];
00488     double y[4];
00489     double z[4];
00490 };
00491
00492
00493
00494
00495
00499 typedef DL_TraceData DL_SolidData;
00500
00501
00505 typedef DL_TraceData DL_3dFaceData;
00506
00507
00511 struct DXFLIB_EXPORT DL_SplineData {
00516     DL_SplineData(int degree,
00517                   int nKnots,
00518                   int nControl,
00519                   int nFit,
00520                   int flags) :
00521         degree(degree),
00522         nKnots(nKnots),
00523         nControl(nControl),
00524         nFit(nFit),
00525         flags(flags) {
00526     }
00527
00529     unsigned int degree;
00530
00532     unsigned int nKnots;
00533
00535     unsigned int nControl;
00536
00538     unsigned int nFit;
00539
00541     int flags;
00542
00543     double tangentStartX;
00544     double tangentStartY;
00545     double tangentStartZ;
00546     double tangentEndX;
00547     double tangentEndY;
00548     double tangentEndZ;
00549 };
00550
00551
00552
00556 struct DXFLIB_EXPORT DL_KnotData {
00557     DL_KnotData() {}
00562     DL_KnotData(double pk) {
00563         k = pk;
00564     }
00565
00567     double k;
00568 };
00569
00570
00571
00575 struct DXFLIB_EXPORT DL_ControlPointData {
00580     DL_ControlPointData(double px, double py, double pz, double weight) {
00581         x = px;
00582         y = py;
00583         z = pz;
00584         w = weight;
00585     }
00586
00588     double x;
00590     double y;
00592     double z;
00594     double w;
00595 };
00596
```

```
00597
00598
00602 struct DXFLIB_EXPORT DL_FitPointData {
00607     DL_FitPointData(double x, double y, double z) : x(x), y(y), z(z) {}
00608
00610     double x;
00612     double y;
00614     double z;
00615 };
00616
00617
00618
00622 struct DXFLIB_EXPORT DL_EllipseData {
00627     DL_EllipseData(double cx, double cy, double cz,
00628                    double mx, double my, double mz,
00629                    double ratio,
00630                    double angle1, double angle2)
00631         : cx(cx),
00632           cy(cy),
00633           cz(cz),
00634           mx(mx),
00635           my(my),
00636           mz(mz),
00637           ratio(ratio),
00638           angle1(angle1),
00639           angle2(angle2) {
00640     }
00641
00643     double cx;
00645     double cy;
00647     double cz;
00648
00650     double mx;
00652     double my;
00654     double mz;
00655
00657     double ratio;
00659     double angle1;
00661     double angle2;
00662 };
00663
00664
00665
00669 struct DXFLIB_EXPORT DL_InsertData {
00674     DL_InsertData(const std::string& name,
00675                   double ipx, double ipy, double ipz,
00676                   double sx, double sy, double sz,
00677                   double angle,
00678                   int cols, int rows,
00679                   double colSp, double rowSp) :
00680           name(name),
00681           ipx(ipx), ipy(ipy), ipz(ipz),
00682           sx(sx), sy(sy), sz(sz),
00683           angle(angle),
00684           cols(cols), rows(rows),
00685           colSp(colSp), rowSp(rowSp) {
00686     }
00687
00689     std::string name;
00691     double ipx;
00693     double ipy;
00695     double ipz;
00697     double sx;
00699     double sy;
00701     double sz;
00703     double angle;
00705     int cols;
00707     int rows;
00709     double colSp;
00711     double rowSp;
00712 };
00713
00714
00715
00719 struct DXFLIB_EXPORT DL_MTextData {
00724     DL_MTextData(double ipx, double ipy, double ipz,
00725                  double dirx, double diry, double dirz,
00726                  double height, double width,
00727                  int attachmentPoint,
00728                  int drawingDirection,
00729                  int lineSpacingStyle,
00730                  double lineSpacingFactor,
00731                  const std::string& text,
00732                  const std::string& style,
00733                  double angle) :
00734           ipx(ipx), ipy(ipy), ipz(ipz),
00735           dirx(dirx), diry(diry), dirz(dirz),
```

```
00736           height(height), width(width),
00737           attachmentPoint(attachmentPoint),
00738           drawingDirection(drawingDirection),
00739           lineSpacingStyle(lineSpacingStyle),
00740           lineSpacingFactor(lineSpacingFactor),
00741           text(text),
00742           style(style),
00743           angle(angle) {
00744
00745     }
00746
00748     double ipx;
00750     double ipy;
00752     double ipz;
00754     double dirx;
00756     double diry;
00758     double dirz;
00760     double height;
00762     double width;
00770     int attachmentPoint;
00776     int drawingDirection;
00782     int lineSpacingStyle;
00786     double lineSpacingFactor;
00788     std::string text;
00790     std::string style;
00792     double angle;
00793 };
00794
00795
00796
00800 struct DXFLIB_EXPORT DL_TextData {
00805     DL_TextData(double ipx, double ipy, double ipz,
00806                 double apx, double apy, double apz,
00807                 double height, double xScaleFactor,
00808                 int textGenerationFlags,
00809                 int hJustification,
00810                 int vJustification,
00811                 const std::string& text,
00812                 const std::string& style,
00813                 double angle)
00814         : ipx(ipx), ipy(ipy), ipz(ipz),
00815           apx(apx), apy(apy), apz(apz),
00816           height(height), xScaleFactor(xScaleFactor),
00817           textGenerationFlags(textGenerationFlags),
00818           hJustification(hJustification),
00819           vJustification(vJustification),
00820           text(text),
00821           style(style),
00822           angle(angle) {
00823     }
00824
00826     double ipx;
00828     double ipy;
00830     double ipz;
00831
00833     double apx;
00835     double apy;
00837     double apz;
00838
00840     double height;
00842     double xScaleFactor;
00844     int textGenerationFlags;
00852     int hJustification;
00858     int vJustification;
00860     std::string text;
00862     std::string style;
00864     double angle;
00865 };
00866
00870 struct DXFLIB_EXPORT DL_ArcAlignedTextData {
00871
00873     std::string text;
00875     std::string font;
00877     std::string style;
00878
00880     double cx;
00882     double cy;
00884     double cz;
00886     double radius;
00887
00889     double xScaleFactor;
00891     double height;
00893     double spacing;
00895     double offset;
00897     double rightOffset;
00899     double leftOffset;
00901     double startAngle;
```

```
00903     double endAngle;
00908     bool reversedCharacterOrder;
00913     int direction;
00920     int alignment;
00925     int side;
00927     bool bold;
00929     bool italic;
00931     bool underline;
00933     int characerSet;
00935     int pitch;
00940     bool shxFont;
00942     bool wizard;
00944     int arcHandle;
00945 };
00946
00950 struct DXFLIB_EXPORT DL_AttributeData : public DL_TextData {
00951     DL_AttributeData(const DL_TextData& tData, const std::string& tag)
00952         : DL_TextData(tData), tag(tag) {
00953
00954     }
00955
00960     DL_AttributeData(double ipx, double ipy, double ipz,
00961                 double apx, double apy, double apz,
00962                 double height, double xScaleFactor,
00963                 int textGenerationFlags,
00964                 int hJustification,
00965                 int vJustification,
00966                 const std::string& tag,
00967                 const std::string& text,
00968                 const std::string& style,
00969                 double angle)
00970         : DL_TextData(ipx, ipy, ipz,
00971                 apx, apy, apz,
00972                 height, xScaleFactor,
00973                 textGenerationFlags,
00974                 hJustification,
00975                 vJustification,
00976                 text,
00977                 style,
00978                 angle),
00979           tag(tag) {
00980     }
00981
00983     std::string tag;
00984 };
00985
00986
00990 struct DXFLIB_EXPORT DL_DimensionData {
00995     DL_DimensionData(double dpx, double dpy, double dpz,
00996                      double mpx, double mpy, double mpz,
00997                      int type,
00998                      int attachmentPoint,
00999                      int lineSpacingStyle,
01000                      double lineSpacingFactor,
01001                      const std::string& text,
01002                      const std::string& style,
01003                      double angle,
01004                      double linearFactor = 1.0,
01005                      double dimScale = 1.0) :
01006         dpx(dpx), dpy(dpy), dpz(dpz),
01007         mpx(mpx), mpy(mpy), mpz(mpz),
01008         type(type),
01009         attachmentPoint(attachmentPoint),
01010         lineSpacingStyle(lineSpacingStyle),
01011         lineSpacingFactor(lineSpacingFactor),
01012         text(text),
01013         style(style),
01014         angle(angle),
01015         linearFactor(linearFactor),
01016         dimScale(dimScale) {
01017
01018     }
01019
01021     double dpx;
01023     double dpy;
01025     double dpz;
01027     double mpx;
01029     double mpy;
01031     double mpz;
01051     int type;
01059     int attachmentPoint;
01065     int lineSpacingStyle;
01069     double lineSpacingFactor;
01077     std::string text;
01079     std::string style;
01084     double angle;
01088     double linearFactor;
```

```
01092     double dimScale;
01093     bool arrow1Flipped;
01094     bool arrow2Flipped;
01095 };
01096
01097
01098
01102 struct DXFLIB_EXPORT DL_DimAlignedData {
01107     DL_DimAlignedData(double depx1, double depy1, double depz1,
01108                       double depx2, double depy2, double depz2) {
01109
01110         epx1 = depx1;
01111         epy1 = depy1;
01112         epz1 = depz1;
01113
01114         epx2 = depx2;
01115         epy2 = depy2;
01116         epz2 = depz2;
01117     }
01118
01120     double epx1;
01122     double epy1;
01124     double epz1;
01125
01127     double epx2;
01129     double epy2;
01131     double epz2;
01132 };
01133
01134
01135
01139 struct DXFLIB_EXPORT DL_DimLinearData {
01144     DL_DimLinearData(double ddpx1, double ddpy1, double ddpz1,
01145                      double ddpx2, double ddpy2, double ddpz2,
01146                      double dAngle, double dOblique) {
01147
01148         dpx1 = ddpx1;
01149         dpy1 = ddpy1;
01150         dpz1 = ddpz1;
01151
01152         dpx2 = ddpx2;
01153         dpy2 = ddpy2;
01154         dpz2 = ddpz2;
01155
01156         angle = dAngle;
01157         oblique = dOblique;
01158     }
01159
01161     double dpx1;
01163     double dpy1;
01165     double dpz1;
01166
01168     double dpx2;
01170     double dpy2;
01172     double dpz2;
01173
01175     double angle;
01177     double oblique;
01178 };
01179
01180
01181
01185 struct DXFLIB_EXPORT DL_DimRadialData {
01190     DL_DimRadialData(double ddpx, double ddpy, double ddpz, double dleader) {
01191         dpx = ddpx;
01192         dpy = ddpy;
01193         dpz = ddpz;
01194
01195         leader = dleader;
01196     }
01197
01199     double dpx;
01201     double dpy;
01203     double dpz;
01204
01206     double leader;
01207 };
01208
01209
01210
01214 struct DXFLIB_EXPORT DL_DimDiametricData {
01219     DL_DimDiametricData(double ddpx, double ddpy, double ddpz, double dleader) {
01220         dpx = ddpx;
01221         dpy = ddpy;
01222         dpz = ddpz;
01223
01224         leader = dleader;
```

```
01225     }
01226
01228     double dpx;
01230     double dpy;
01232     double dpz;
01233
01235     double leader;
01236 };
01237
01238
01239
01243 struct DXFLIB_EXPORT DL_DimAngular2LData {
01248     DL_DimAngular2LData(double ddpx1, double ddpy1, double ddpz1,
01249                         double ddpx2, double ddpy2, double ddpz2,
01250                         double ddpx3, double ddpy3, double ddpz3,
01251                         double ddpx4, double ddpy4, double ddpz4) {
01252
01253         dpx1 = ddpx1;
01254         dpy1 = ddpy1;
01255         dpz1 = ddpz1;
01256
01257         dpx2 = ddpx2;
01258         dpy2 = ddpy2;
01259         dpz2 = ddpz2;
01260
01261         dpx3 = ddpx3;
01262         dpy3 = ddpy3;
01263         dpz3 = ddpz3;
01264
01265         dpx4 = ddpx4;
01266         dpy4 = ddpy4;
01267         dpz4 = ddpz4;
01268     }
01269
01271     double dpx1;
01273     double dpy1;
01275     double dpz1;
01276
01278     double dpx2;
01280     double dpy2;
01282     double dpz2;
01283
01285     double dpx3;
01287     double dpy3;
01289     double dpz3;
01290
01292     double dpx4;
01294     double dpy4;
01296     double dpz4;
01297 };
01298
01299
01303 struct DXFLIB_EXPORT DL_DimAngular3PData {
01308     DL_DimAngular3PData(double ddpx1, double ddpy1, double ddpz1,
01309                         double ddpx2, double ddpy2, double ddpz2,
01310                         double ddpx3, double ddpy3, double ddpz3) {
01311
01312         dpx1 = ddpx1;
01313         dpy1 = ddpy1;
01314         dpz1 = ddpz1;
01315
01316         dpx2 = ddpx2;
01317         dpy2 = ddpy2;
01318         dpz2 = ddpz2;
01319
01320         dpx3 = ddpx3;
01321         dpy3 = ddpy3;
01322         dpz3 = ddpz3;
01323     }
01324
01326     double dpx1;
01328     double dpy1;
01330     double dpz1;
01331
01333     double dpx2;
01335     double dpy2;
01337     double dpz2;
01338
01340     double dpx3;
01342     double dpy3;
01344     double dpz3;
01345 };
01346
01347
01348
01352 struct DXFLIB_EXPORT DL_DimOrdinateData {
01357     DL_DimOrdinateData(double ddpx1, double ddpy1, double ddpz1,
```

```
01358                       double ddpx2, double ddpy2, double ddpz2,
01359                       bool dxtype) {
01360
01361         dpx1 = ddpx1;
01362         dpy1 = ddpy1;
01363         dpz1 = ddpz1;
01364
01365         dpx2 = ddpx2;
01366         dpy2 = ddpy2;
01367         dpz2 = ddpz2;
01368
01369         xtype = dxtype;
01370     }
01371
01373     double dpx1;
01375     double dpy1;
01377     double dpz1;
01378
01380     double dpx2;
01382     double dpy2;
01384     double dpz2;
01385
01387     bool xtype;
01388 };
01389
01390
01391
01395 struct DXFLIB_EXPORT DL_LeaderData {
01400     DL_LeaderData(int arrowHeadFlag,
01401                   int leaderPathType,
01402                   int leaderCreationFlag,
01403                   int hooklineDirectionFlag,
01404                   int hooklineFlag,
01405                   double textAnnotationHeight,
01406                   double textAnnotationWidth,
01407                   int number,
01408                   double dimScale = 1.0) :
01409         arrowHeadFlag(arrowHeadFlag),
01410         leaderPathType(leaderPathType),
01411         leaderCreationFlag(leaderCreationFlag),
01412         hooklineDirectionFlag(hooklineDirectionFlag),
01413         hooklineFlag(hooklineFlag),
01414         textAnnotationHeight(textAnnotationHeight),
01415         textAnnotationWidth(textAnnotationWidth),
01416         number(number),
01417         dimScale(dimScale) {
01418
01419     }
01420
01422     int arrowHeadFlag;
01424     int leaderPathType;
01426     int leaderCreationFlag;
01428     int hooklineDirectionFlag;
01430     int hooklineFlag;
01432     double textAnnotationHeight;
01434     double textAnnotationWidth;
01436     int number;
01438     double dimScale;
01439 };
01440
01441
01442
01446 struct DXFLIB_EXPORT DL_LeaderVertexData {
01451     DL_LeaderVertexData(double px=0.0, double py=0.0, double pz=0.0) {
01452         x = px;
01453         y = py;
01454         z = pz;
01455     }
01456
01458     double x;
01460     double y;
01462     double z;
01463 };
01464
01465
01466
01470 struct DXFLIB_EXPORT DL_HatchData {
01474     DL_HatchData() {}
01475
01480     DL_HatchData(int numLoops,
01481                  bool solid,
01482                  double scale,
01483                  double angle,
01484                  const std::string& pattern,
01485                  double originX = 0.0,
01486                  double originY = 0.0) :
01487         numLoops(numLoops),
```

```
01488         solid(solid),
01489         scale(scale),
01490         angle(angle),
01491         pattern(pattern),
01492         originX(originX),
01493         originY(originY) {
01494
01495     }
01496
01498     int numLoops;
01500     bool solid;
01502     double scale;
01504     double angle;
01506     std::string pattern;
01508     double originX;
01509     double originY;
01510 };
01511
01512
01513
01517 struct DXFLIB_EXPORT DL_HatchLoopData {
01521     DL_HatchLoopData() {}
01526     DL_HatchLoopData(int hNumEdges) {
01527         numEdges = hNumEdges;
01528     }
01529
01531     int numEdges;
01532 };
01533
01534
01535
01539 struct DXFLIB_EXPORT DL_HatchEdgeData {
01543     DL_HatchEdgeData() : defined(false), x1(0.0), y1(0.0), x2(0.0), y2(0.0) {
01544     }
01545
01550     DL_HatchEdgeData(double x1, double y1,
01551                      double x2, double y2) :
01552         defined(true),
01553         type(1),
01554         x1(x1),
01555         y1(y1),
01556         x2(x2),
01557         y2(y2) {
01558     }
01559
01564     DL_HatchEdgeData(double cx, double cy,
01565                      double radius,
01566                      double angle1, double angle2,
01567                      bool ccw) :
01568         defined(true),
01569         type(2),
01570         cx(cx),
01571         cy(cy),
01572         radius(radius),
01573         angle1(angle1),
01574         angle2(angle2),
01575         ccw(ccw) {
01576     }
01577
01582     DL_HatchEdgeData(double cx, double cy,
01583                      double mx, double my,
01584                      double ratio,
01585                      double angle1, double angle2,
01586                      bool ccw) :
01587         defined(true),
01588         type(3),
01589         cx(cx),
01590         cy(cy),
01591         angle1(angle1),
01592         angle2(angle2),
01593         ccw(ccw),
01594         mx(mx),
01595         my(my),
01596         ratio(ratio) {
01597     }
01598
01603     DL_HatchEdgeData(unsigned int degree,
01604                      bool rational,
01605                      bool periodic,
01606                      unsigned int nKnots,
01607                      unsigned int nControl,
01608                      unsigned int nFit,
01609                      const std::vector<double>& knots,
01610                      const std::vector<std::vector<double> >& controlPoints,
01611                      const std::vector<std::vector<double> >& fitPoints,
01612                      const std::vector<double>& weights,
01613                      double startTangentX,
```

```
01614                     double startTangentY,
01615                     double endTangentX,
01616                     double endTangentY) :
01617         defined(true),
01618         type(4),
01619         degree(degree),
01620         rational(rational),
01621         periodic(periodic),
01622         nKnots(nKnots),
01623         nControl(nControl),
01624         nFit(nFit),
01625         controlPoints(controlPoints),
01626         knots(knots),
01627         weights(weights),
01628         fitPoints(fitPoints),
01629         startTangentX(startTangentX),
01630         startTangentY(startTangentY),
01631         endTangentX(endTangentX),
01632         endTangentY(endTangentY) {
01633     }
01634
01638     bool defined;
01639
01643     int type;
01644
01645     // line edges:
01646
01648     double x1;
01650     double y1;
01652     double x2;
01654     double y2;
01655
01657     double cx;
01659     double cy;
01661     double radius;
01663     double angle1;
01665     double angle2;
01667     bool ccw;
01668
01670     double mx;
01672     double my;
01674     double ratio;
01675
01676
01678     unsigned int degree;
01679     bool rational;
01680     bool periodic;
01682     unsigned int nKnots;
01684     unsigned int nControl;
01686     unsigned int nFit;
01687
01688     std::vector<std::vector<double> > controlPoints;
01689     std::vector<double> knots;
01690     std::vector<double> weights;
01691     std::vector<std::vector<double> > fitPoints;
01692
01693     double startTangentX;
01694     double startTangentY;
01695
01696     double endTangentX;
01697     double endTangentY;
01698
01700     std::vector<std::vector<double> > vertices;
01701     //bool closed;
01702 };
01703
01704
01705
01709 struct DXFLIB_EXPORT DL_ImageData {
01714     DL_ImageData(const std::string& iref,
01715                  double iipx, double iipy, double iipz,
01716                  double iux, double iuy, double iuz,
01717                  double ivx, double ivy, double ivz,
01718                  int iwidth, int iheight,
01719                  int ibrightness, int icontrast, int ifade) {
01720         ref = iref;
01721         ipx = iipx;
01722         ipy = iipy;
01723         ipz = iipz;
01724         ux = iux;
01725         uy = iuy;
01726         uz = iuz;
01727         vx = ivx;
01728         vy = ivy;
01729         vz = ivz;
01730         width = iwidth;
01731         height = iheight;
```

```
01732         brightness = ibrightness;
01733         contrast = icontrast;
01734         fade = ifade;
01735     }
01736
01739     std::string ref;
01741     double ipx;
01743     double ipy;
01745     double ipz;
01747     double ux;
01749     double uy;
01751     double uz;
01753     double vx;
01755     double vy;
01757     double vz;
01759     int width;
01761     int height;
01763     int brightness;
01765     int contrast;
01767     int fade;
01768 };
01769
01770
01771
01775 struct DXFLIB_EXPORT DL_ImageDefData {
01780     DL_ImageDefData(const std::string& iref,
01781                     const std::string& ifile) {
01782         ref = iref;
01783         file = ifile;
01784     }
01785
01788     std::string ref;
01789
01791     std::string file;
01792 };
01793
01794
01795
01799 struct DXFLIB_EXPORT DL_DictionaryData {
01800     DL_DictionaryData(const std::string& handle) : handle(handle) {}
01801     std::string handle;
01802 };
01803
01804
01805
01809 struct DXFLIB_EXPORT DL_DictionaryEntryData {
01810     DL_DictionaryEntryData(const std::string& name, const std::string& handle) :
01811         name(name), handle(handle) {}
01812
01813     std::string name;
01814     std::string handle;
01815 };
01816
01817 #endif
01818
01819 // EOF
```

6.7 dl_exception.h

```
00001 /**********************************************************************
00002 ** Copyright (C) 2001-2013 RibbonSoft, GmbH. All rights reserved.
00003 ** Copyright (C) 2001 Robert J. Campbell Jr.
00004 **
00005 ** This file is part of the dxflib project.
00006 **
00007 ** This file is free software; you can redistribute it and/or modify
00008 ** it under the terms of the GNU General Public License as published by
00009 ** the Free Software Foundation; either version 2 of the License, or
00010 ** (at your option) any later version.
00011 **
00012 ** Licensees holding valid dxflib Professional Edition licenses may use
00013 ** this file in accordance with the dxflib Commercial License
00014 ** Agreement provided with the Software.
00015 **
00016 ** This file is provided AS IS with NO WARRANTY OF ANY KIND, INCLUDING THE
00017 ** WARRANTY OF DESIGN, MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE.
00018 **
00019 ** See http://www.ribbonsoft.com for further details.
00020 **
00021 ** Contact info@ribbonsoft.com if any conditions of this licensing are
00022 ** not clear to you.
00023 **
00024 **********************************************************************/
```

```
00025
00026 #ifndef DL_EXCEPTION_H
00027 #define DL_EXCEPTION_H
00028
00029 #include "dl_global.h"
00030
00031 #if _MSC_VER > 1000
00032 #pragma once
00033 #endif // _MSC_VER > 1000
00034
00038 class DXFLIB_EXPORT DL_Exception {}
00039 ;
00040
00044 class DXFLIB_EXPORT DL_NullStrExc : public DL_Exception {}
00045 ;
00046
00050 class DXFLIB_EXPORT DL_GroupCodeExc : public DL_Exception {
00051     DL_GroupCodeExc(int gc=0) : groupCode(gc) {}
00052     int groupCode;
00053 };
00054 #endif
00055
```

6.8 dl_extrusion.h

```
00001 /**********************************************************************
00002 ** Copyright (C) 2001-2013 RibbonSoft, GmbH. All rights reserved.
00003 **
00004 ** This file is part of the dxflib project.
00005 **
00006 ** This file is free software; you can redistribute it and/or modify
00007 ** it under the terms of the GNU General Public License as published by
00008 ** the Free Software Foundation; either version 2 of the License, or
00009 ** (at your option) any later version.
00010 **
00011 ** Licensees holding valid dxflib Professional Edition licenses may use
00012 ** this file in accordance with the dxflib Commercial License
00013 ** Agreement provided with the Software.
00014 **
00015 ** This file is provided AS IS with NO WARRANTY OF ANY KIND, INCLUDING THE
00016 ** WARRANTY OF DESIGN, MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE.
00017 **
00018 ** See http://www.ribbonsoft.com for further details.
00019 **
00020 ** Contact info@ribbonsoft.com if any conditions of this licensing are
00021 ** not clear to you.
00022 **
00023 **********************************************************************/
00024
00025 #ifndef DL_EXTRUSION_H
00026 #define DL_EXTRUSION_H
00027
00028 #include "dl_global.h"
00029
00030 #include <math.h>
00031
00032
00038 class DXFLIB_EXPORT DL_Extrusion {
00039
00040 public:
00041
00045     DL_Extrusion() {
00046         direction = new double[3];
00047         setDirection(0.0, 0.0, 1.0);
00048         setElevation(0.0);
00049     }
00050
00051
00055     ~DL_Extrusion() {
00056         delete[] direction ;
00057     }
00058
00059
00068     DL_Extrusion(double dx, double dy, double dz, double elevation) {
00069         direction = new double[3];
00070         setDirection(dx, dy, dz);
00071         setElevation(elevation);
00072     }
00073
00074
00075
00079     void setDirection(double dx, double dy, double dz) {
00080         direction[0]=dx;
```

```
00081         direction[1]=dy;
00082         direction[2]=dz;
00083     }
00084
00085
00086
00090     double* getDirection() const {
00091         return direction;
00092     }
00093
00094
00095
00099     void getDirection(double dir[]) const {
00100         dir[0]=direction[0];
00101         dir[1]=direction[1];
00102         dir[2]=direction[2];
00103     }
00104
00105
00106
00110     void setElevation(double elevation) {
00111         this->elevation = elevation;
00112     }
00113
00114
00115
00119     double getElevation() const {
00120         return elevation;
00121     }
00122
00123
00124
00128     DL_Extrusion operator = (const DL_Extrusion& extru) {
00129         setDirection(extru.direction[0], extru.direction[1], extru.direction[2]);
00130         setElevation(extru.elevation);
00131
00132         return *this;
00133     }
00134
00135
00136
00137 private:
00138     double *direction;
00139     double elevation;
00140 };
00141
00142 #endif
00143
```

6.9 dl_global.h

```
00001 #if defined(DXFLIB_DLL)
00002 #   ifdef _WIN32
00003 #       if defined(DXFLIB_LIBRARY)
00004 #           define DXFLIB_EXPORT __declspec(dllexport)
00005 #       else
00006 #           define DXFLIB_EXPORT __declspec(dllimport)
00007 #       endif
00008 #   else
00009 #       define DXFLIB_EXPORT
00010 #   endif
00011 #else
00012 #   define DXFLIB_EXPORT
00013 #endif
```

6.10 dl_writer.h

```
00001 /**********************************************************************
00002 ** Copyright (C) 2001-2013 RibbonSoft, GmbH. All rights reserved.
00003 ** Copyright (C) 2001 Robert J. Campbell Jr.
00004 **
00005 ** This file is part of the dxflib project.
00006 **
00007 ** This file is free software; you can redistribute it and/or modify
00008 ** it under the terms of the GNU General Public License as published by
00009 ** the Free Software Foundation; either version 2 of the License, or
00010 ** (at your option) any later version.
00011 **
00012 ** Licensees holding valid dxflib Professional Edition licenses may use
```

```
00013 ** this file in accordance with the dxflib Commercial License
00014 ** Agreement provided with the Software.
00015 **
00016 ** This file is provided AS IS with NO WARRANTY OF ANY KIND, INCLUDING THE
00017 ** WARRANTY OF DESIGN, MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE.
00018 **
00019 ** See http://www.ribbonsoft.com for further details.
00020 **
00021 ** Contact info@ribbonsoft.com if any conditions of this licensing are
00022 ** not clear to you.
00023 **
00024 **********************************************************************/
00025
00026 #ifndef DL_WRITER_H
00027 #define DL_WRITER_H
00028
00029 #include "dl_global.h"
00030
00031 #ifndef _WIN32
00032 #include <strings.h>
00033 #endif
00034
00035 #if _MSC_VER > 1000
00036 #pragma once
00037 #endif // _MSC_VER > 1000
00038
00039 #include <cstring>
00040 #include <iostream>
00041 #include <algorithm>
00042
00043 #include "dl_attributes.h"
00044 #include "dl_codes.h"
00045
00046
00047
00058 class DXFLIB_EXPORT DL_Writer {
00059 public:
00063     DL_Writer(DL_Codes::version version) : m_handle(0x30) {
00064         this->version = version;
00065         modelSpaceHandle = 0;
00066         paperSpaceHandle = 0;
00067         paperSpace0Handle = 0;
00068     }
00069
00070     virtual ~DL_Writer() {}
00071     ;
00072
00082     void section(const char* name) const {
00083         dxfString(0, "SECTION");
00084         dxfString(2, name);
00085     }
00086
00097     void sectionHeader() const {
00098         section("HEADER");
00099     }
00100
00111     void sectionTables() const {
00112         section("TABLES");
00113     }
00114
00125     void sectionBlocks() const {
00126         section("BLOCKS");
00127     }
00128
00139     void sectionEntities() const {
00140         section("ENTITIES");
00141     }
00142
00153     void sectionClasses() const {
00154         section("CLASSES");
00155     }
00156
00167     void sectionObjects() const {
00168         section("OBJECTS");
00169     }
00170
00179     void sectionEnd() const {
00180         dxfString(0, "ENDSEC");
00181     }
00182
00195     void table(const char* name, int num, int h=0) const {
00196         dxfString(0, "TABLE");
00197         dxfString(2, name);
00198         if (version>=DL_VERSION_2000) {
00199             if (h==0) {
00200                 handle();
00201             }
```

```
00202             else {
00203                 dxfHex(5, h);
00204             }
00205             dxfString(100, "AcDbSymbolTable");
00206         }
00207         dxfInt(70, num);
00208     }
00209
00223     void tableLayers(int num) const {
00224         table("LAYER", num, 2);
00225     }
00226
00240     void tableLinetypes(int num) const {
00241         //linetypeHandle = 5;
00242         table("LTYPE", num, 5);
00243     }
00244
00258     void tableAppid(int num) const {
00259         table("APPID", num, 9);
00260     }
00261
00275     void tableStyle(int num) const {
00276         table("STYLE", num, 3);
00277     }
00278
00287     void tableEnd() const {
00288         dxfString(0, "ENDTAB");
00289     }
00290
00299     void dxfEOF() const {
00300         dxfString(0, "EOF");
00301     }
00302
00311     void comment(const char* text) const {
00312         dxfString(999, text);
00313     }
00314
00325     void entity(const char* entTypeName) const {
00326         dxfString(0, entTypeName);
00327         if (version>=DL_VERSION_2000) {
00328             handle();
00329         }
00330     }
00331
00346     void entityAttributes(const DL_Attributes& attrib) const {
00347
00348         // layer name:
00349         dxfString(8, attrib.getLayer());
00350
00351         // R12 doesn't accept BYLAYER values. The value has to be missing
00352         //   in that case.
00353         if (version>=DL_VERSION_2000 || attrib.getColor()!=256) {
00354             dxfInt(62, attrib.getColor());
00355         }
00356         if (version>=DL_VERSION_2000 && attrib.getColor24()!=-1) {
00357             dxfInt(420, attrib.getColor24());
00358         }
00359         if (version>=DL_VERSION_2000) {
00360             dxfInt(370, attrib.getWidth());
00361         }
00362         if (version>=DL_VERSION_2000) {
00363             dxfReal(48, attrib.getLinetypeScale());
00364         }
00365         std::string linetype = attrib.getLinetype();
00366         std::transform(linetype.begin(), linetype.end(), linetype.begin(), ::toupper);
00367         if (version>=DL_VERSION_2000 || linetype=="BYLAYER") {
00368             dxfString(6, attrib.getLinetype());
00369         }
00370     }
00371
00375     void subClass(const char* sub) const {
00376         dxfString(100, sub);
00377     }
00378
00387     void tableLayerEntry(unsigned long int h=0)  const {
00388         dxfString(0, "LAYER");
00389         if (version>=DL_VERSION_2000) {
00390             if (h==0) {
00391                 handle();
00392             } else {
00393                 dxfHex(5, h);
00394             }
00395             dxfString(100, "AcDbSymbolTableRecord");
00396             dxfString(100, "AcDbLayerTableRecord");
00397         }
00398     }
00399
```

```
00408     void tableLinetypeEntry(unsigned long int h=0)  const {
00409         dxfString(0, "LTYPE");
00410         if (version>=DL_VERSION_2000) {
00411             if (h==0) {
00412                 handle();
00413             } else {
00414                 dxfHex(5, h);
00415             }
00416             //dxfHex(330, 0x5);
00417             dxfString(100, "AcDbSymbolTableRecord");
00418             dxfString(100, "AcDbLinetypeTableRecord");
00419         }
00420     }
00421
00430     void tableAppidEntry(unsigned long int h=0)  const {
00431         dxfString(0, "APPID");
00432         if (version>=DL_VERSION_2000) {
00433             if (h==0) {
00434                 handle();
00435             } else {
00436                 dxfHex(5, h);
00437             }
00438             //dxfHex(330, 0x9);
00439             dxfString(100, "AcDbSymbolTableRecord");
00440             dxfString(100, "AcDbRegAppTableRecord");
00441         }
00442     }
00443
00452     void sectionBlockEntry(unsigned long int h=0)  const {
00453         dxfString(0, "BLOCK");
00454         if (version>=DL_VERSION_2000) {
00455             if (h==0) {
00456                 handle();
00457             } else {
00458                 dxfHex(5, h);
00459             }
00460             //dxfHex(330, blockHandle);
00461             dxfString(100, "AcDbEntity");
00462             if (h==0x1C) {
00463                 dxfInt(67, 1);
00464             }
00465             dxfString(8, "0");                  // TODO: Layer for block
00466             dxfString(100, "AcDbBlockBegin");
00467         }
00468     }
00469
00478     void sectionBlockEntryEnd(unsigned long int h=0)  const {
00479         dxfString(0, "ENDBLK");
00480         if (version>=DL_VERSION_2000) {
00481             if (h==0) {
00482                 handle();
00483             } else {
00484                 dxfHex(5, h);
00485             }
00486             //dxfHex(330, blockHandle);
00487             dxfString(100, "AcDbEntity");
00488             if (h==0x1D) {
00489                 dxfInt(67, 1);
00490             }
00491             dxfString(8, "0");                  // TODO: Layer for block
00492             dxfString(100, "AcDbBlockEnd");
00493         }
00494     }
00495
00496     void color(int col=256) const {
00497         dxfInt(62, col);
00498     }
00499     void linetype(const char *lt) const {
00500         dxfString(6, lt);
00501     }
00502     void linetypeScale(double scale) const {
00503         dxfReal(48, scale);
00504     }
00505     void lineWeight(int lw) const {
00506         dxfInt(370, lw);
00507     }
00508
00509     void coord(int gc, double x, double y, double z=0) const {
00510         dxfReal(gc, x);
00511         dxfReal(gc+10, y);
00512         dxfReal(gc+20, z);
00513     }
00514
00515     void coordTriplet(int gc, const double* value) const {
00516         if (value) {
00517             dxfReal(gc, *value++);
00518             dxfReal(gc+10, *value++);
```

```
00519             dxfReal(gc+20, *value++);
00520         }
00521     }
00522
00523     void resetHandle() const {
00524         m_handle = 1;
00525     }
00526
00530     unsigned long handle(int gc=5) const {
00531         // handle has to be hex
00532         dxfHex(gc, m_handle);
00533         return m_handle++;
00534     }
00535
00539     unsigned long getNextHandle() const {
00540         return m_handle;
00541     }
00542
00550     virtual void dxfReal(int gc, double value) const = 0;
00551
00559     virtual void dxfInt(int gc, int value) const = 0;
00560
00568     virtual void dxfBool(int gc, bool value) const {
00569         dxfInt(gc, (int)value);
00570     }
00571
00579     virtual void dxfHex(int gc, int value) const = 0;
00580
00588     virtual void dxfString(int gc, const char* value) const = 0;
00589
00597     virtual void dxfString(int gc, const std::string& value) const = 0;
00598
00599 protected:
00600     mutable unsigned long m_handle;
00601     mutable unsigned long modelSpaceHandle;
00602     mutable unsigned long paperSpaceHandle;
00603     mutable unsigned long paperSpace0Handle;
00604
00608     DL_Codes::version version;
00609 private:
00610 };
00611
00612 #endif
```

6.11 dl_writer_ascii.h

```
00001 /****************************************************************************
00002 ** Copyright (C) 2001-2013 RibbonSoft, GmbH. All rights reserved.
00003 ** Copyright (C) 2001 Robert J. Campbell Jr.
00004 **
00005 ** This file is part of the dxflib project.
00006 **
00007 ** This file is free software; you can redistribute it and/or modify
00008 ** it under the terms of the GNU General Public License as published by
00009 ** the Free Software Foundation; either version 2 of the License, or
00010 ** (at your option) any later version.
00011 **
00012 ** Licensees holding valid dxflib Professional Edition licenses may use
00013 ** this file in accordance with the dxflib Commercial License
00014 ** Agreement provided with the Software.
00015 **
00016 ** This file is provided AS IS with NO WARRANTY OF ANY KIND, INCLUDING THE
00017 ** WARRANTY OF DESIGN, MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE.
00018 **
00019 ** See http://www.ribbonsoft.com for further details.
00020 **
00021 ** Contact info@ribbonsoft.com if any conditions of this licensing are
00022 ** not clear to you.
00023 **
00024 **********************************************************************/
00025
00026 #ifndef DL_WRITER_ASCII_H
00027 #define DL_WRITER_ASCII_H
00028
00029 #include "dl_global.h"
00030
00031 #if _MSC_VER > 1000
00032 #pragma once
00033 #endif // _MSC_VER > 1000
00034
00035 #include "dl_writer.h"
00036 #include <fstream>
00037 #include <string>
```

```
00038
00049 class DXFLIB_EXPORT DL_WriterA : public DL_Writer {
00050 public:
00051     DL_WriterA(const char* fname, DL_Codes::version version=DL_VERSION_2000)
00052             : DL_Writer(version), m_ofile(fname) {}
00053     virtual ~DL_WriterA() {}
00054
00055     bool openFailed() const;
00056     void close() const;
00057     void dxfReal(int gc, double value) const;
00058     void dxfInt(int gc, int value) const;
00059     void dxfHex(int gc, int value) const;
00060     void dxfString(int gc, const char* value) const;
00061     void dxfString(int gc, const std::string& value) const;
00062
00063     static void strReplace(char* str, char src, char dest);
00064
00065 private:
00069     mutable std::ofstream m_ofile;
00070
00071 };
00072
00073 #endif
00074
```

Index