OpenDWG™ Toolkit
Reference
VERSION 2.0

The OpenDWG Alliance
http://www.opendwg.org
TABLE OF CONTENTS

1. GENERAL .......................................................................................................................... 13
   PREFACE ............................................................................................................................ 13

2. OPENDWG TOOLKIT LIBRARIES .................................................................................... 15
   OPENDWG TOOLKIT FOR WINDOWS .............................................................................. 15
   OPENDWG TOOLKIT FOR UNIX ...................................................................................... 16
   OPENDWG TOOLKIT FOR DEC ALPHA UNDER WINDOWS NT ....................................... 16
   OPENDWG TOOLKIT FOR MACINTOSH ......................................................................... 16

3. PROGRAMMING WITH OPENDWG TOOLKIT ................................................................. 19
   AUTOCAD DWG AND DXF FILES ...................................................................................... 19
   SUPPORTED ENVIRONMENTS ......................................................................................... 19
   INCLUDE FILES ................................................................................................................ 20
   FUNCTION PROTOTYPES ................................................................................................. 20
   EXAMPLE PROGRAMS ....................................................................................................... 20
   STRUCTURE ALIGNMENT ISSUES .................................................................................... 21
   STACK USAGE .................................................................................................................. 21
   OPENDWG TOOLKIT initialization DATA LOADING ......................................................... 21
   SOME TROUBLESHOOTING TIPS ...................................................................................... 23
   MIGRATION FROM VERSION 1.0 TO VERSION 2.0 ............................................................. 23

4. OPENDWG TOOLKIT DATA ORGANIZATION ............................................................... 25
   GENERAL ............................................................................................................................ 25
   VIRTUAL MEMORY ............................................................................................................. 25
   HANDLES .......................................................................................................................... 26
   LINKED LISTS .................................................................................................................... 26
   BLObs ................................................................................................................................ 26
   AD_OBJHANDLES ........................................................................................................... 26
   HEADER .............................................................................................................................. 27
   ENTITIES AND BLOCKS ..................................................................................................... 27
   TABLES ............................................................................................................................... 27
   CLASSES ............................................................................................................................ 28
   LAYERS .............................................................................................................................. 28
   LINETYPES ....................................................................................................................... 28
   SHAPE FILES ................................................................................................................... 28
   VIEWS .............................................................................................................................. 28
   VPORTS ............................................................................................................................ 28
   DIMSTYLES ....................................................................................................................... 28
   UCS (user coordinate systems) ......................................................................................... 28
   REGISTERED APPLICATIONS ......................................................................................... 29
   OBJECTS ............................................................................................................................. 29
   EXTENDED DATA ............................................................................................................ 29

5. CREATING PROGRAMS USING OPENDWG TOOLKIT .................................................... 31

6. ADDING ADVANCED OPENDWG TOOLKIT FEATURES .................................................. 35
   FILE I/O AND MEMORY MANAGEMENT .......................................................................... 35
   ERROR HANDLING .......................................................................................................... 36
   OPENDWG TOOLKIT WARNING ERRORS ........................................................................ 37
7. FUNCTIONS BY CATEGORY ............................................................................. 45

BLOB MANAGEMENT ...................................................................................... 45
BLOCK HEADERS .............................................................................................. 45
CLASSES ............................................................................................................ 45
DATE/TIME FUNCTIONS .................................................................................. 45
DICTIONARY OBJECTS ...................................................................................... 45
DIMSTYLES ......................................................................................................... 45
ENTITIES ............................................................................................................. 45
ENTITY LONGDATA ........................................................................................... 46
ENTITY TYPES ................................................................................................... 46
ERROR HANDLING ............................................................................................. 46
EXTENDED DATA .............................................................................................. 46
EXTERNAL REFERENCES ..................................................................................... 46
GROUPS ................................................................................................................ 46
HEADER ............................................................................................................. 46
INTERFACE FUNCTIONS .................................................................................... 46
LAYERS ................................................................................................................ 47
LINETYPES .......................................................................................................... 47
MISCELLANEOUS .............................................................................................. 47
MLSTYLES .......................................................................................................... 47
OBJECT TYPES .................................................................................................. 47
OBJECTS ............................................................................................................. 47
OBJHANDLE MANIPULATION .......................................................................... 47
PROXY DATA ....................................................................................................... 48
PROXY GRAPHICS ............................................................................................ 48
PROXY OBJECTS ............................................................................................... 48
RASER IMAGES .................................................................................................. 48
REGAPPS ............................................................................................................ 49
SHAPEFILES ..................................................................................................... 49
STARTUP/SHUTDOWN ..................................................................................... 49
TEXT POSITIONING ............................................................................................ 49
UCS ....................................................................................................................... 49
UNICODE HANDLING ......................................................................................... 49
VIEWS ................................................................................................................ 49
VPORTS .............................................................................................................. 49

8. OTHER OPENDWG TOOLKIT CONCEPTS....................................................... 51

VERSION DEPENDENCIES .............................................................................. 51

9. FUNCTION REFERENCE .............................................................................. 53

ADD_ACCEPTR14BETAFILES ............................................................................ 53
ADD_ADDBLOCKHEADER .................................................................................. 53
ADD_ADDBMP .................................................................................................... 54
ADD_ADDBMPFROMBUFFER .......................................................................... 54
ADD_ADDCLASS ................................................................................................. 55
ADD_ADDDIMSTYLE ......................................................................................... 55
ADD_ADDENTITYToList .................................................................................... 57
<table>
<thead>
<tr>
<th>Function</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ADReadMTextBlock</td>
<td>147</td>
</tr>
<tr>
<td>ADReadPrimColors</td>
<td>148</td>
</tr>
<tr>
<td>ADReadPrimFlags</td>
<td>149</td>
</tr>
<tr>
<td>ADReadPrimLayerIDs</td>
<td>150</td>
</tr>
<tr>
<td>ADReadPrimLineTypeIDs</td>
<td>150</td>
</tr>
<tr>
<td>ADReadPrimMarkers</td>
<td>151</td>
</tr>
<tr>
<td>ADReadPrimNormals</td>
<td>151</td>
</tr>
<tr>
<td>ADReadPrimOrientation</td>
<td>152</td>
</tr>
<tr>
<td>ADReadPrimVisibilities</td>
<td>153</td>
</tr>
<tr>
<td>ADReadXRecordItem</td>
<td>153</td>
</tr>
<tr>
<td>ADReadProxyDataBinary</td>
<td>154</td>
</tr>
<tr>
<td>ADReadProxyDataBool</td>
<td>156</td>
</tr>
<tr>
<td>ADReadProxyDataChar</td>
<td>156</td>
</tr>
<tr>
<td>ADReadProxyDataDouble</td>
<td>156</td>
</tr>
<tr>
<td>ADReadProxyDataInt16</td>
<td>157</td>
</tr>
<tr>
<td>ADReadProxyDataInt32</td>
<td>157</td>
</tr>
<tr>
<td>ADReadProxyDataObjHandle</td>
<td>158</td>
</tr>
<tr>
<td>ADReadProxyDataPoint2D</td>
<td>158</td>
</tr>
<tr>
<td>ADReadProxyDataPoint3D</td>
<td>159</td>
</tr>
<tr>
<td>ADReadProxyDataScale3D</td>
<td>159</td>
</tr>
<tr>
<td>ADReadProxyDataString</td>
<td>160</td>
</tr>
<tr>
<td>ADReadProxyDataUInt16</td>
<td>160</td>
</tr>
<tr>
<td>ADReadProxyDataUInt32</td>
<td>161</td>
</tr>
<tr>
<td>ADReadProxyDataUInt8</td>
<td>161</td>
</tr>
<tr>
<td>ADReadProxyDataVector2D</td>
<td>161</td>
</tr>
<tr>
<td>ADReadProxyDataVector3D</td>
<td>163</td>
</tr>
<tr>
<td>ADReadProxyDataVoidStData</td>
<td>163</td>
</tr>
<tr>
<td>ADReadProxyEntityObjID</td>
<td>164</td>
</tr>
<tr>
<td>ADReadProxyObjectObjID</td>
<td>164</td>
</tr>
<tr>
<td>ADReplaceBlockHeader</td>
<td>165</td>
</tr>
<tr>
<td>ADReplaceDimstyle</td>
<td>165</td>
</tr>
<tr>
<td>ADReplaceEntity</td>
<td>166</td>
</tr>
<tr>
<td>ADReplaceLayer</td>
<td>167</td>
</tr>
<tr>
<td>ADReplaceLineType</td>
<td>167</td>
</tr>
<tr>
<td>ADReplaceObject</td>
<td>168</td>
</tr>
<tr>
<td>ADReplaceRegapp</td>
<td>169</td>
</tr>
<tr>
<td>ADReplaceShapefile</td>
<td>169</td>
</tr>
<tr>
<td>ADReplaceUCS</td>
<td>170</td>
</tr>
<tr>
<td>ADReplaceView</td>
<td>170</td>
</tr>
<tr>
<td>ADReplaceVPor</td>
<td>171</td>
</tr>
<tr>
<td>ADRewindBlob</td>
<td>171</td>
</tr>
<tr>
<td>ADTextEntity</td>
<td>172</td>
</tr>
<tr>
<td>ADSaveFile</td>
<td>172</td>
</tr>
<tr>
<td>ADSeekBlockHeader</td>
<td>173</td>
</tr>
<tr>
<td>ADSeekDimstyle</td>
<td>173</td>
</tr>
<tr>
<td>ADSeekEntity</td>
<td>174</td>
</tr>
<tr>
<td>ADSeekLayer</td>
<td>174</td>
</tr>
<tr>
<td>ADSeekLineType</td>
<td>175</td>
</tr>
<tr>
<td>ADSeekObject</td>
<td>175</td>
</tr>
<tr>
<td>ADSeekRegapp</td>
<td>176</td>
</tr>
<tr>
<td>ADSeekShapefile</td>
<td>177</td>
</tr>
<tr>
<td>ADSeekUCS</td>
<td>177</td>
</tr>
<tr>
<td>ADSeekView</td>
<td>178</td>
</tr>
<tr>
<td>ADSeekVPor</td>
<td>178</td>
</tr>
<tr>
<td>ADSetAd2CloseInitFileFN</td>
<td>179</td>
</tr>
</tbody>
</table>
10. DATA REFERENCE

HEADER DATA: ............................................................................. 235
ENTITY DATA: ........................................................................... 252
ENTITY HEADER: ......................................................................... 252
ENTITY BODY: ............................................................................ 253
3D FACE: ................................................................................... 253
3D LINE: .................................................................................... 254
ACIS DATA: ................................................................................ 255
1. GENERAL

PREFACE

OpenDWG Toolkit (ODT) enables you to write C or C++ language programs that read and write AutoCAD DWG and DXF files outside of Autodesk AutoCAD, thus greatly improving the speed and convenience of processes utilizing information contained within AutoCAD drawings.

This product is intended for experienced C programmers only; therefore we have not attempted to describe operation of the compilers, where to store include files on your hard disk, and C language information. Please consult your compiler reference manuals for information on their operation.
2. OpenDWG Toolkit Libraries

The following sections describe OpenDWG Toolkit for the various platforms on which it is supported.

Libraries for Windows are compressed with PKZIP; the UNIX files are compressed with GZIP. BE SURE TO DOWNLOAD NONLIBS.ZIP AS WELL AS ANY NEEDED LIBRARIES. NONLIBS.ZIP contains the header files for OpenDWG Toolkit, and if you do not download these files and use them along with newly downloaded libraries OpenDWG Toolkit may not work.

OpenDWG Toolkit for Windows

Following is a table of the names for the libraries contained in Windows version of OpenDWG Toolkit. Please note that we are constantly adding new libraries to the ODT distribution; this may not be a complete list. You may consult the file LIBINFO.TXT which will be installed with your software to see which libraries have been provided with ODT.

<table>
<thead>
<tr>
<th>Library Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ADB45W.LIB</td>
<td>Borland C++ 4.5, large model</td>
</tr>
<tr>
<td>ADB45N.LIB</td>
<td>Borland C++ 4.5, Win32 (NT/95)</td>
</tr>
<tr>
<td>ADV2N.LIB</td>
<td>Visual C++ 2.0 for Windows NT</td>
</tr>
<tr>
<td>ADV2ND.LIB</td>
<td>Visual C++ 2.0 for Windows NT, -MD option (for DLL creation)</td>
</tr>
<tr>
<td>ADV2NT.LIB</td>
<td>Visual C++ 2.0 for Windows NT, -MT option (multi-threaded)</td>
</tr>
<tr>
<td>ADV22N.LIB</td>
<td>Visual C++ 2.2 for Windows NT</td>
</tr>
<tr>
<td>ADV22ND.LIB</td>
<td>Visual C++ 2.2 for Windows NT, -MD option (for DLL creation)</td>
</tr>
<tr>
<td>ADV22NT.LIB</td>
<td>Visual C++ 2.2 for Windows NT, -MT option (multi-threaded)</td>
</tr>
<tr>
<td>ADV4N.LIB</td>
<td>Visual C++ 4.0 for Windows NT</td>
</tr>
<tr>
<td>ADV4ND.LIB</td>
<td>Visual C++ 4.0 for Windows NT, -MD option (for DLL creation)</td>
</tr>
<tr>
<td>ADV4NT.LIB</td>
<td>Visual C++ 4.0 for Windows NT, -MT option (multi-threaded)</td>
</tr>
<tr>
<td>ADV5N.LIB</td>
<td>Visual C++ 5.0 for Windows NT</td>
</tr>
<tr>
<td>ADV5ND.LIB</td>
<td>Visual C++ 5.0 for Windows NT, -MD option (for DLL creation)</td>
</tr>
<tr>
<td>ADV5NT.LIB</td>
<td>Visual C++ 5.0 for Windows NT, -MT option (multi-threaded)</td>
</tr>
<tr>
<td>ADV6N.LIB</td>
<td>Visual C++ 6.0 for Windows NT</td>
</tr>
<tr>
<td>ADV6ND.LIB</td>
<td>Visual C++ 6.0 for Windows NT, -MD option (for DLL creation)</td>
</tr>
<tr>
<td>ADV6NT.LIB</td>
<td>Visual C++ 6.0 for Windows NT, -MT option (multi-threaded)</td>
</tr>
</tbody>
</table>
OpenDWG Toolkit for UNIX

This section describes the OpenDWG Toolkit libraries for the various supported UNIX platforms (Sun Sparcstation, IBM RS/6000, Dec Alpha, HP 9000/700, and SGI).

The ODT libraries have the following names: ad2.a, and ad2pic.a for the library compiled with the appropriate compiler switches for position-independent code for the platform.

To compile and link the adex.r.c example program for ODT, you could use the command:

```bash
cc adex.r.c -o adexr ad2.a -lm
```

Other commands will also successfully compile and link; this is simply an example.

OpenDWG Toolkit for DEC Alpha under Windows NT

The OpenDWG Toolkit libraries have the following names: adv4na.lib, adv4nta.lib (compiled with -MT for multi-threaded applications) and adv4nda.lib (compiled with -MD for DLL applications).

We have also supplied example programs adexrw, adexcpw, and adexww in their own directories as examples of how to read, copy, and write a DWG or DXF respectively.

OpenDWG Toolkit for Macintosh

The Macintosh, particularly in 68K mode, has numerous options for compiled programs. We have listed below the options used for the libraries provided with OpenDWG Toolkit. Please choose the appropriate library for your particular situation. Please note that certain options, in particular 8 byte doubles, are required by ODT. Please also note that we are not sure whether some of these options (Source model, for instance) actually make any difference in the compiled ODT library.

The OpenDWG Toolkit libraries all have names starting with ad2cw. The next character indicates the version of the CodeWarrior compiler (9 for 9, 0 for 10, 1 for CodeWarrior Professional version 1). The next character indicates the option set used to compile the library. If the library is for PowerPC the final character "p" is appended. Thus for instance, "ad2cw9bp.o" is the library for CodeWarrior 9 with option set b, for Power PC.
<table>
<thead>
<tr>
<th>Feature</th>
<th>&quot;a&quot;</th>
<th>&quot;b&quot;</th>
<th>&quot;c&quot;</th>
</tr>
</thead>
<tbody>
<tr>
<td>A5 Relative data</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Target Mac 68K</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Source model</td>
<td>Apple C</td>
<td>Custom</td>
<td>Apple C</td>
</tr>
<tr>
<td>Code model</td>
<td>Smart</td>
<td>Large</td>
<td>Smart</td>
</tr>
<tr>
<td>Struct alignment</td>
<td>68K</td>
<td>68K</td>
<td>68K</td>
</tr>
<tr>
<td>68020 code</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>68881 code</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>4 byte ints</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>8 byte doubles</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>far data</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>far methods</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>far strings</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>A6 stack</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Feature</th>
<th>&quot;a&quot;</th>
<th>&quot;b&quot;</th>
<th>&quot;c&quot;</th>
</tr>
</thead>
<tbody>
<tr>
<td>Target PPC</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Source model</td>
<td>Apple C</td>
<td>Custom</td>
<td>Apple C</td>
</tr>
<tr>
<td>Struct alignment</td>
<td>PPC</td>
<td>68K</td>
<td>68K</td>
</tr>
<tr>
<td>Static data in TOC</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>FMADD &amp; FMSUB</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
</tbody>
</table>
3. PROGRAMMING WITH OpenDWG Toolkit

AUTOCAD DWG and DXF FILES

DWG is the native AutoCAD file format. It is an undocumented binary format in which AutoCAD stores drawing information. OpenDWG Toolkit provides a method for loading, modifying and saving these files without need for the programmer to understand the format of the underlying file.

DXF (Drawing Interchange Format) format is a standard promulgated by Autodesk for transfer of data to other programs. There are two flavors of DXF: ASCII and binary. Both are supported by OpenDWG Toolkit, and both can be loaded, modified and saved just like DWG files.

Both the DWG and DXF formats have changed over time. OpenDWG Toolkit supports DWG and DXF files from versions 2.5 through AutoCAD 2000.

When a file is loaded by ODT, it is resolved into a format consistent with the AutoCAD 2000 format. Thus, all files may be used without (much) regard to the originating format or version.

SUPPORTED ENVIRONMENTS

Environments supported by OpenDWG Toolkit include:

Windows: ODT typically requires about 3 to 3.5 times the size of the DWG files to be loaded for virtual memory, so if you want to load 20 Mb files, you probably should have 70 Mb (or more) of swap space set aside. Windows example programs are provided. For real-mode compilers, large model is the only supported model; however we do provide (for Microsoft C) a library version compiled with the -gx switch to allow multiple invocation. The Windows distribution includes Win32-compatible libraries, which are usable under Windows 95 and Windows NT.

UNIX: UNIX example programs are provided.

Macintosh: Macintosh example programs are provided.

The ODT interface is virtually identical across the supported platforms. However, certain platforms may have certain features "tweaked" for performance.
INCLUDE FILES

You should include the ODT include files in your programs. The include files for OpenDWG Toolkit are ad2.h and odio.h.

If you are using the default I/O and memory management routines provided with ODT, you should include one of the following statements in your program before the statement that includes the odio.h file:

```c
#define OD_GENERIC_READ
/* For Windows NT only, you can use the following statement. */
/* For more information, see File I/O and Memory Management. */
#define OD_FAST_READ
```

FUNCTION PROTOTYPES

Function prototypes are provided for all OpenDWG Toolkit functions. The prototype may be used either with or without a complete set of parameters. To activate the parameters (which should be done for almost all supported platforms), you should include the statement

```c
#define AD_PROTOTYPES
```

before the `#include` statements that reference the ODT headers.

EXAMPLE PROGRAMS

OpenDWG Toolkit includes a number of example programs as a guide to its use. You should use these example programs as a guide to developing your own. The examples are:

- adexr.c -- console-mode read example program
- adexcp.c -- console-mode copy example program
- adexw.c -- console-mode write example program
- adexzo.c -- console-mode example illustrating write of proxy entities and objects

Windows only:

- adexrw.c -- Windows read example program
- adexcpw.c -- Windows copy example program
- adexww.c -- Windows write example program
STRUCTURE ALIGNMENT ISSUES

OpenDWG Toolkit is compiled with the default structure alignment for each supported compiler. ODT will not work if you use a non-default alignment without taking remedial action. The reason for this is obvious – if the compiler thinks ODT structures have a different alignment than that with which they were compiled, you will be unable to use the data stored in those structures.

If you wish to use a non-default alignment, and your compiler supports mixed-alignment programming, you should use your compiler's method for handling this. For instance, Visual C provides a "pack" pragma which can be used to indicate the packing of the OpenDWG Toolkit structures. An example of usage of this pragma (for Visual C 4) is:

```c
#pragma pack(push)        /* saves current packing */
#pragma pack(8)           /* set to default, which is 8 for VC4 */
#include <ad2.h>
#pragma pack(pop)         /* pop previous packing */
```

STACK USAGE

Programs utilizing OpenDWG Toolkit should be compiled with a large stack, preferably 40K or more.

OpenDWG Toolkit INITIALIZATION DATA LOADING

OpenDWG Toolkit is supplied with an initialization file, adinit.dat, which contains the initialization data for OpenDWG Toolkit. This file is normally loaded by passing its path as one of the parameters to adInitAd2().

For those of you who wish to embed this data in your application in another way, rather than as an external file, we have provided an alternative method for loading the data. You may use this method as follows:

You must define four interface functions for OpenDWG Toolkit to call: open, read a character, read characters, and close. You may perform these operations however you wish, as long as they work properly. Thus you may hide the data inside a larger init file, in a Macintosh resource, or wherever you wish.
Define four functions as follows (the names are your choice):

```c
short myopeninitfile(void)
{
    /* your code here -- perform open */
    /* for instance (assuming myinitfile is global):
       myinitfile=fopen("adinit.dat","rb");
       if (myinitfile!=NULL) return(1);
       return(0);
    */
}

char myreadinitfilechar(void)
{
    /* your code here -- return the next character of init data */
    /* for instance --
       return(fgetc(myinitfile));
    */
}

void myreadinitfilechars(char *str,unsigned short num)
{
    /* your code here -- put the next num bytes into str */
    /* for instance --
       fread(str,num,1,myinitfile);
    */
}

void mycloseinitfile(void)
{
    /* your code here -- close the init file */
    /* for instance --
       fclose(myinitfile);
    */
}
```

Tell OpenDWG Toolkit where to find your functions:

```c
adSetAd2OpenInitFileFn(myopeninitfile);
adSetAd2ReadInitFileCharFn(myreadinitfilechar);
adSetAd2ReadInitFileCharsFn(myreadinitfilechars);
adSetAd2CloseInitFileFn(mycloseinitfile);
```

NOTE: These assignments must be performed BEFORE adInitAd2() is called.

At the call to adInitAd2(), pass in a 1 for the userfns parameter, indicating that you have defined functions for OpenDWG Toolkit to use to load the init data. You should pass NULL to adInitAd2() for the init file path if you redefine these functions.
SOME TROUBLESHOOTING TIPS:

1. If you are having trouble loading a file, please verify that it will load in AutoCAD, and run the function `adDwgSmellsBad()` on the file. If `adDwgSmellsBad()` returns a value greater than 0, there may be an error in the file. If the file loads successfully in AutoCAD, please try an AutoCAD "RECOVER" on the file to see if any errors are uncovered. It is also occasionally illuminating to do an AutoCAD DXF out and DXF back in on the file. A very large percentage of file load problems are caused by corrupt data within AutoCAD DWG files.

2. If the file loads successfully, please try copying it using the supplied OpenDWG Toolkit `adexcp` or `adexcpw` sample programs. If this process is not successful, or the file produced cannot be loaded into AutoCAD, there is a greater likelihood that the problem lies within ODT and not your program.

3. Please note that OpenDWG Toolkit may be stricter than AutoCAD when it comes to loading DXF files. We have done a great deal of work on ODT to improve its flexibility in loading poorly written DXF files; however there may still be DXF files that AutoCAD will load but ODT will not. Should you find such a DXF file, we would appreciate seeing a copy.

MIGRATION FROM VERSION 1.0 TO VERSION 2.0

The following changes will need to be considered when migrating from version 1.0 to version 2.0 of the Toolkit. Please refer to the appropriate section in the remainder of this document for a more detail description of each change.

1. `adLoadFile` takes an extra argument.
2. `adSaveFile` takes 3 extra arguments.
3. `adGetBlockHandle` should be used to retrieve the paper space block, since the "Paper_Space name is no longer unique.
4. `adSetDefaultBlockheader` should be called for each block header created. New fields were added to the structure, and this will ensure that the new fields are properly initialized.
5. The values that were stored in viewport entity MVIEW extended data have been moved into the AD_VPENT structure, and the MVIEW extended data section is no longer present. This change reflects the corresponding change that was made to the file format.
6. Support for r14 beta files is now disabled by default, and can be enabled by calling `adAcceptR14BetaFiles`.
7. Block names have been replaced by handles in several structures. This was done to reflect to corresponding changes in the format.
4. OpenDWG Toolkit DATA ORGANIZATION

GENERAL

Files loaded or created by OpenDWG Toolkit are stored in a virtual memory subsystem. This system, which may either be OpenDWG Toolkit's own virtual memory system or a system provided by the operating system upon which the program is running, is used to provide access to the data stored within the files. The structure of the data as stored by OpenDWG Toolkit, is shown in figure entitled "OpenDWG Toolkit DATABASE ORGANIZATION". Note that the File Control Block is a structure held internally by OpenDWG Toolkit; the handle passed to an OpenDWG Toolkit function selects the FCB used internally by OpenDWG Toolkit.

VIRTUAL MEMORY

OpenDWG Toolkit requires an external virtual memory implementation to operate. Drawing file data is loaded into virtual memory and operated on there. The interface functions for virtual memory are discussed in section 6.

Since most modern operating systems provide some sort of virtual memory, it is not typically necessary for the ODT user to write complicated functions to handle it; often simply calling malloc and free is sufficient. However, a complete interface is provided in case a more complete level of interface to an existing virtual memory system is required.
HANDLES

Upon opening a file, OpenDWG Toolkit returns a handle to the file. This is simply a way for OpenDWG Toolkit to keep track of which file you wish to act on. You will use this handle as a parameter when you make subsequent calls to OpenDWG Toolkit to act on the file.

LINKED LISTS

OpenDWG Toolkit utilizes a linked list manager, providing a method for creating, adding to and deleting from linked lists. Most information loaded by OpenDWG Toolkit is stored in linked lists, including all entities, table entries (layers, linetypes, etc.), block headers and blocks. Generally, access to a given linked list is initiated by a call to `adStartXXXXXGet()`, followed by subsequent calls to `adGetXXXXX`, where `XXXXX` is the name of the table or list from which to read. Additions to a list are made by `adAddXXXXX()`, deletions by `adDeleteXXXXX()`. Items in a list can be sought using `adSeekXXXXX()`. A linked list is created with `adCreateList()`, and deleted with `adDeleteList()`.

BLOBs

BLOBs (binary large objects) are used by OpenDWG Toolkit to hold binary objects of arbitrary size. Accessing data contained in a BLOB is very similar to accessing data in a FILE. You simply open the blob with `adStartBlobRead()`, read data from the blob with `adReadBlobBytes()`, and close the blob with `adEndBlobRead()`. Similarly, a blob can be created with `adCreateBlob()`, initialized for write with `adStartBlobWrite()`, written to with `adWriteBlobBytes()`, and closed with `adEndBlobWrite()`. You can append to a blob with `adStartBlobAppend()`, followed by write commands. The control structure for a blob is typedef’d `AD_BLOB_CTRL`.

AD_OBJHANDLES

AD_OBJHANDLEs are unique 8 byte identifiers which are used to specify items within a file. For instance, the layer upon which a drawing entity resides is identified by an AD_OBJHANDLE. All entities have unique AD_OBJHANDLEs, as do all table entries, etc. The data in an AD_OBJHANDLE is stored right justified in an 8 character field, with zeroes filled in on the left.

Note that AD_OBJHANDLEs as stored by OpenDWG Toolkit are binary data. When these are written in ASCII (as, for instance, DXF files, or when viewed through the LIST command in AutoCAD) they are shown with two ASCII hex characters per byte, or a maximum of 16 characters. The binary representation is therefore twice as compact and should be used rather than the ASCII representation.

We provide functions `adAsciiObjHandleToBinary` and `adBinaryObjHandleToAscii` to facilitate conversion between these formats.

A typed AD_OBJHANDLE is an object handle with an additional field that indicates its type. They are stored in structure `AD_TYPEDOBJHANDLE`:

```c
typedef struct typedobjhandlestru {
    AD_OBJHANDLE typedhandle;
    char handletype;
};
```
AD_TYPEDOBJHANDLE,*PAD_TYPEDOBJHANDLE;

The possible types of typed objhandles are:

AD_ARBITRARY_HANDLE
AD_SOFT_POINTER_HANDLE
AD_HARD_POINTER_HANDLE
AD_SOFT_OWNER_HANDLE
AD_HARD_OWNER_HANDLE

Most AD_OBJHANDLEs within OpenDWG Toolkit are not typed. This is not because they do not have types; rather it is because, for the functions they serve, they can only be of one particular type. It is generally only for user-defined proxy entities and objects that there is a need to designate the type of an AD_OBJHANDLE.

HEADER

OpenDWG Toolkit stores an open file's header in regular memory. In order to access the header, you should use the adHeaderPointer() function, which returns a pointer to an AD_DWGHDR.

The values for all header variables are defined in the section entitled "DATA REFERENCE".

ENTITIES and BLOCKS

Entities and blocks are stored in virtual memory. All entities are accessed by first locating the block header to which the entities are attached, then reading the list of entities. There are two special blocks. These are *MSPACE_BLOCK and *PSPACE_BLOCK, which hold the modelspace and paperspace entities respectively. All blocks have as their first entity a BLOCK entity, and as the last an ENDBLK entity, except for the two special blocks, which lack these entities.

There are various functions for handling entities within OpenDWG Toolkit. Among these are:

adStartEntityGet initiate read of entities
adGetEntity get the next entity in a list
adSeekEntity seek to an entity by its handle
adAddEntity add an entity to the list
adReplaceEntity replace an entity in the list
adDeleteEntity delete an entity
adNumEntities returns the number of entities in a list

A number of entities have as part of them BLOBs which hold lengthy data. For instance, a spline contains an indeterminate number of data points; these points are held in a BLOB.

The values for all entity variables are defined in the section entitled "DATA REFERENCE".

TABLES

There are numerous tables in a DWG or DXF file; they are:
CLASSES
Contains the classes defined in a file. These are generally defined by external programs which create their own proxy entities or objects. The corresponding OpenDWG Toolkit structure is AD_CLASS.

LAYERs
Contains the drawing's layers. Each layer has such attributes as ON/OFF, FROZEN/THAWED, color and linetype. The corresponding OpenDWG Toolkit structure definition is AD_LAY.

LINETYPES
Contains the drawing's linetypes. Each linetype has a name, a string describing its appearance, and an array containing the control parameters which determine the appearance of the linetype. The corresponding OpenDWG Toolkit structure definition is AD_LTYPE.

SHAPE FILES
Contains the drawing's shape file (style) definitions. Each shape file has a name, an indication of the font in use, a height, an oblique angle, and other parameters. The corresponding OpenDWG Toolkit structure definition is AD_SHPTB.

VIEWS
Contains the drawing's named view definitions. Each view definition specifies a 3-D view. The corresponding OpenDWG Toolkit structure definition is AD_VIEW.

VPORTS
Contains the drawing's viewport definitions. Note that these are distinct from paperspace viewport entities; rather they are modelspace viewports which display modelspace data. Named viewport groups consist of a set of viewports, all with the same name. The current viewport (or viewports) has the name "*ACTIVE". The corresponding OpenDWG Toolkit structure definition is AD_VPORT.

DIMSTYLES
Contains the definitions for the drawing's dimension styles. Dimension styles are collections of variables which specify how dimensions are to be created. The corresponding OpenDWG Toolkit structure definition is AD_DIMSTYLE.

UCS (user coordinate systems)
Contains the drawing's named user coordinate system definitions. A UCS is a specification of a local coordinate system to be used instead of the world coordinate system. The corresponding OpenDWG Toolkit structure definition is AD_UCS.
REGISTERED APPLICATIONS
Contains the drawing's registered applications. These are applications which have registered themselves with the CAD system. The corresponding OpenDWG Toolkit structure definition is AD_APP.

OBJECTS
The OBJECT table stores a number of types of AutoCAD objects, including dictionary entries, multi-line styles, groups, and application defined proxy objects. Proxy objects can be either "generic" proxies or special proxies such as OLE2FRAME, XRECORD, SPATIAL_FILTER, IDBUFFER, DICTIONARYVAR, RASTERVARIABLES, IMAGEDEFFREACTOR, and IMAGEDEF.

Each of these tables is stored by OpenDWG Toolkit as a linked list. Several OpenDWG Toolkit functions are defined for each table type. They are:

adStartXXXXXGet initiate read of the list
adGetXXXXX get the next item in the list
adAddXXXXX add an item to the end of the list
adReplaceXXXXX replace an item in the list
adSeekXXXXX seek an item in the list

The values for all table variables are defined in the section entitled "DATA REFERENCE".

EXTENDED DATA
Extended data is data specifically added to an object by an application program. OpenDWG Toolkit defines a number of types of extended data; the AD_XD structure contains this definition. Extended data is limited to 16K in size. Extended data is stored in a BLOB, and is typically pointed to by a field in an structure called xdblob.

The values for all table variables are defined in the section entitled "DATA REFERENCE".
5. CREATING PROGRAMS USING OpenDWG Toolkit

In this section we will demonstrate the creation of a simple OpenDWG Toolkit program.

Here is a fragment of a simple OpenDWG Toolkit program:

```c
#define AD_PROTOTYPES
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include <string.h>
#include "ad2.h"
#define OD_GENERIC_READ
#include " odio.h" 

/* OpenDWG Toolkit structure pointers */
/* allocate these at startup, release when done */
PAD_DWGHDR adhd;
PAD_ENT_HDR adenhd;
PAD_ENT aden;
PAD_TB adtb;
PAD_XD adxd;

void main(int argc,char **argv)
{
     AD_DB_HANDLE dwghandle;
     short initerr;

     if (!allocateadptrs()) { 
         printf("couldn't allocate OpenDWG Toolkit data space\n");
         exit(1);
    }

     if (!adInitAd2("adinit.dat",0,&initerr)){
         printf("Unable to initialize OpenDWG Toolkit\n");
         printf("error is %d: (%s)\n",initerr,adErrorStr(initerr));
         exit(1);
    }

     adSetupDwgRead();
     adSetupDxfRead();

     if (!((dwghandle=adLoadFile(infile,AD_PRELOAD_ALL,1))) { /* init input*/
         printf("Error opening file %s\n",infile); /* and read header */
         printf("error code is %d: %s\n",adError(),adErrorStr(adError()));
         return;
    }

     processfile(dwghandle);
     adCloseFile(dwghandle);
     adCloseAd2();
     freeadptrs();
}
```

This fragment illustrates several aspects of OpenDWG Toolkit programming. First note the use of the #define AD_PROTOTYPES statement, and the inclusion of "ad2.h".

Next, we define various interface structures that our program will use to pass data back and forth to OpenDWG Toolkit. These are:

- adhd -- header structure
- adenhd -- entity header structure
aden -- entity body structure
adtb -- union containing the various table structures
adx< -- extended data structure

The first statement in main() calls allocateadptrs() to allocate the memory pointed to by the various pointers described above. Note that allocateadptrs() is not an OpenDWG Toolkit function; it is defined elsewhere in our program.

Next we call adInitAd2() to initialize the OpenDWG Toolkit (read and write) portion of OpenDWG Toolkit. We pass a number of parameters to adInitAd2(). The first parameter is a string indicating the path to the adinit.dat initialization file. The next parameter is 0 indicating that we are not using our own function calls to load the init file; we let OpenDWG Toolkit do it. See our example programs for a more sophisticated method for finding this file. The final parameter returns an error code if adInitAd2() fails.

After this we call adSetupDwgRead() and adSetupDxfRead() to prepare for read and write of these file types.

Next we call adLoadFile() to load the file into virtual memory.

We then call a function called processfile(), which we have not yet defined, which acts on the file.

When we are through processing the file, we call adCloseFile() to close the file.

Finally, we call adCloseAd2() to terminate OpenDWG Toolkit, and call freeadptrs() (which, like allocateadptrs(), is defined in this program, not OpenDWG Toolkit) to free the pointers allocated by allocateadptrs().
Let's continue to add to our program:

```c
void processfile(AD_DB_HANDLE dwghandle)
{
    short i;
    PAD_DWHHDR adhd;

    adhd=adHeaderPointer(dwghandle);
    writeheaderinfo(adhd);

    printf("\nLAYERs: \n");                          /* read layer list */
    adStartLayerGet(dwghandle);
    for (i=0; i<(short)adNumLayers(dwghandle); i++) {
        adGetLayer(dwghandle,&adtb->lay);
        if (!adtb->lay.purgedflag) {
            printf(" %s\n",adtb->lay.name);
            if (adtb->lay.xdblob!=AD_VMNULL) printxdblob(adtb->lay.xdblob);
        }
    }

    printf("\nLinetypes: \n");                        /* read linetype list */
    adStartLinetypeGet(dwghandle);
    for (i=0; i<(short)adNumLinetypes(dwghandle); i++) {
        adGetLinetype(dwghandle,&adtb->ltype);
        if (!adtb->ltype.purgedflag) {
            printf(" %s\n",adtb->ltype.name);
            if (adtb->ltype.xdblob!=AD_VMNULL)
                printxdblob(adtb->ltype.xdblob);
        }
    }
}
```

The `processfile()` function reads the header from virtual memory and prints it, then reads the layer and linetype tables. Note the use of `dwghandle` as a parameter to the OpenDWG Toolkit functions.
Next, here's the code for allocateadptrs() and freeadptrs():

```c
short allocateadptrs(void)
{
    if ((adhd=(PAD_DWGHDR)malloc(sizeof(AD_DWGHDR)))!=NULL) {
        if ((adenhd=(PAD_ENT_HDR)malloc(sizeof(AD_ENT_HDR)))!=NULL) {
            if ((aden=(PAD_ENT)malloc(sizeof(AD_ENT)))!=NULL) {
                if ((adtb=(PAD_TB)malloc(sizeof(AD_TB)))!=NULL) {
                    if ((adxd=(PAD_XD)malloc(sizeof(AD_XD)))!=NULL) {
                        return(1);
                    }
                    free(adtb);
                }
                free(aden);
            }
            free(adenhd);
        }
        free(adhd);
    }
    return(0);
}

void freeadptrs(void)
{
    free(adxd);
    free(adtb);
    free(aden);
    free(adenhd);
    free(adhd);
}
```

These functions simply allocate storage for the OpenDWG Toolkit data, and free it.

In this example we used `adLoadFile()` to load an existing DWG file; we could also have created a new file from scratch using `adNewFile()`.

Most of the code shown here was taken from the example `adexr.c`. Two other examples, `adexw.c` and `adexcp.c`, show how to write a drawing file from scratch, and how to copy one drawing file to another.
6. ADDING ADVANCED OpenDWG Toolkit FEATURES

FILE I/O AND MEMORY MANAGEMENT

OpenDWG Toolkit has a defined set of required functions to handle file input/output and memory management. These functions must be included in your ODT program. We provide a header file, odio.h, which contains a reference implementation of these functions. You are REQUIRED to implement each function listed below to creating a working ODT program.

Below is a summary of the required functions.

File I/O:

All file I/O functions should have the same functionality as their C library counterparts. The "void *f" entries can be, for instance, a normal FILE * pointer, or they can represent any other data the user wants to use to indicate a source for data.

The one odd function here is odior_fgetdxfline(), which reads the next line of data from a DXF file. Because of the various possible line termination conditions for DXF, we found it necessary to do a very particular sort of line read function; similar to, but not the same as, fgets. Odio.h contains an example of how this function should work.

Read functions and write functions are separately defined to allow for high-performance cached reads to be implemented if desired. Odio.h contains one such implementation, which can be turned on using #define OD_FAST_READ before the #include " odio.h" statement.

File read functions:

int odior_fclose(void *f)
short odior_feof(void *f) unsigned char odior_fgetc(void *f) char
*odior_fgetdxfline(char *s, unsigned int n, void **f2)
long odior_flength(void *f) void *odior_fopen (void *path, char *accessmodes) int
odior_fread (void *buf, unsigned int size, unsigned int num, void *f) int odior_fseek (void
*f, long offset, int whence) long odior_ftell(void *f)

File write functions:

int odiow_fclose(void *f) void *odiow_fopen (void *path, char *accessmodes) int
odiow_fputs(const char *str, void *f)
int odiow_fseek (void *f, long offset, int whence) long odiow_ftell(void *f) int
odiow_fwrite(void *b, int size, int count, void *f);
Temporary Memory Management:

All temporary memory management routines should utilize pointers through which data can be accessed directly, as the traditional malloc and free allow.

```c
void *odmem_malloc(int size)
void odmem_free(void *loc)
```

Virtual Memory Management:

OpenDWG Toolkit provides a set of generic routines for access to any virtual memory subsystem you may wish to implement or run under. All access to data stored in virtual memory is done through the `odvm_readbytes` and `odvm_writebytes` functions. This allows for mediated access, indirecled access (a la Mac), etc. It is also possible to simply make the init and term functions no-ops, use malloc and free for the vm malloc and free, and use memcpy for readbytes and writebytes, which is the sensible way to operate on systems which have virtual memory built into their operating systems. The reference implementation in “odio.h” does the following:

```c
short odvm_init(void)    /* initialize VM subsystem */
short odvm_term (void)   /* terminate VM subsystem */
short odvm_readbytes(char *ptr,AD_VMADDR vmloc,unsigned short bytes)
    /* read from VM */
short odvm_writebytes(AD_VMADDR vmloc,char *ptr,unsigned short bytes)
short odvm_free(AD_VMADDR ptr)  /* free VM memory */
AD_VMADDR odvm_malloc(unsigned num_bytes)  /* allocate VM memory */
```

ERROR HANDLING

General errors:

OpenDWG Toolkit provides a function `adError()` which can be used to gain more information about errors which occur while using ODT. Generally ODT functions return 1 on success and 0 on failure; when a failure occurs you can use `adError()` to retrieve a value that indicates the type of error. These values correspond to the constants defined in ad2.h (for instance, AD_CANT_OPEN_FILE). While you can look in ad2.h to find the corresponding defined constant, a much easier method is to use `adErrorStr()` to return a string which describes the error. For instance, you might say:

```c
if (myerrno=adError()) printf("error: %s\n",adErrorStr(myerrno));
```

Note that `adErrorStr()` does drag a large amount of static string data into your program; if you find you are running low on space you might try omitting it.

If you receive the specific message "Duplicate handle in file", you can also call the function `adErrorObjhandle()` to see which handle was duplicated.
OpenDWG Toolkit WARNING ERRORS

OpenDWG Toolkit generates very few warning errors. In fact, the only warnings as of this writing are errors on SHAPE entities and ACIS-based entities (BODY, SOLID3D, and REGION).

A warning is generated when attempting to write a DXF file with SHAPE entities in it. If ODT cannot find the appropriate SHX file (which it needs to perform this operation successfully), it will call a warning error function if one is defined (and omit the SHAPE entity from the output file).

A warning is also generated if an attempt is made to write a DWG file with ACIS-based data that did not originate from a DWG file. ODT is not capable of creating such entities.

To define a warning error function, simply write a function accepting a short as its argument, and assign the function using adSetAd2WarningFn. Here is an example:

```c
void mywarning(short errorno)
{
    printf("warning error %d: s\n", adError(), adErrorStr(adError()));
}
```

then somewhere else in your program (but after adInitAd2()), as an initialization step,

```c
adSetAd2WarningFn(mywarning);
```

OpenDWG Toolkit CRITICAL ERRORS

OpenDWG Toolkit generates a critical error when it is unable to perform a buffer fill operation from a file, when there is a write error, or if a memory allocation fails. To trap these errors, which very rarely occur, you should define a critical error function. Here is an example:

```c
short criterrhandler(short num)
{
    char ch;

    /* file read errors can be retried */
    if (num==AD_CRITERR_FILEREADERROR) {
        printf("File read error: Abort/Retry?\n");
        ch=fgetc(stdin);
        if (ch=='R' || ch=='r') return(0); /* return value ignored */
        exit(1);
    }

    /* disk full can be retried */
    if (num==AD_CRITERR_DISKFULL) {
        printf("Critical error: disk full\n");
        /* Returning 1 on a disk error means retry. Returning 0 means
        close the file and generate critical error AD_WRITE_ABORT (to allow
        a graceful exit). */
        return(0);
    }

    /* write abort means disk full was reached and the program requested */
    /* termination. This is a fatal error. */
    if (num==AD_CRITERR_WRITEABORT) {
        printf("Critical error: write abort\n");
        /* You should delete the partial file here. AD DOES NOT do so. */
        exit(1);
    }
```
/* malloc errors are fatal */
if (num==AD_CRITERR_MALLOCERROR) {
    printf("critical error %d, aderror %d (%s), terminating\n", num, adError(), adErrorStr(adError()));
    exit(1);
}
/* as are buffer overruns */
if (num==AD_CRITERR_BUFFEROVERRUN) {
    printf("critical error -- buffer overrun, terminating\n");
    exit(1);
}
if (num==AD_CRITERR_BADDATA) {
    printf("critical error -- bad data detected, terminating\n");
    exit(1);
}
return(0);

Your critical error function should do one of three things; either exit, return some value to ODT, or longjmp (presumably from a previous setjmp) to a safe location for a restart or soft shutdown.

Below we discuss the various types of errors:

For file read errors, the value you return to ODT is ignored. If your critical error function returns to ODT, it will retry the read.

File write errors are a bit more complicated. You may simply abort if you wish. If your function returns 0 to OpenDWG Toolkit, it will close the file being written and generate an AD_CRITERR_WRITEABORT critical error. If your function returns 1, it will simply retry the write operation.

As described above, writeabort is a fatal error generated when your program has requested ODT simply to close the file and generate a second error.

Mallos cannot be retried, and should be treated as fatal errors.

To tell OpenDWG Toolkit where to find your critical error function, use the **adSetAd2CriticalErrorFn()** as shown below:

```
adSetAd2CriticalErrorFn(criticalerrorhandler);
```

You should set this function before calling adInitAd2().
EXAMINE SHAPE FILE CALLBACK

At times, particularly when dealing with DXF files, OpenDWG Toolkit needs to be able to open external .SHX files and read information from them. By default, ODT calls its own function adDefaultExamineFile(char *shapefile, char *newshapefile) to search for new shape files. The source code for adDefaultExamineFile is contained in the file default.c which is included with ODT. adDefaultExamineFile uses the following (Release 12 compatible) method for searching for shape files:

If the file has a fully specified path, search there. If not found, return failure.
If the file does not have a fully specified path, search in the current directory, then in all directories pointed to by the ACAD environment variable, if any.

The file’s path, if found, is returned in newshapefile.

You may wish to modify this search method. For instance, to emulate Release 13 and later, rather than returning failure after a fully specified path search fails, you would extract the filename part of the specification and search down the ACAD environment variable paths.

To modify the search method, simply define your own search function which takes two char * parameters; one file OpenDWG Toolkit tells you it wants, and one for you to place the appropriate path in. Then use the function adSetAd2ExamineshapefileFn to point to your function. Here is a simple example:

```c
void myshapefilesearch(char *findthis, char *returnstr)
{
    /* always return SIMPLEX */
    strcpy(returnstr,"SIMPLEX.SHX");
}
```

then somewhere else in your program, as an initialization step,

```c
adSetAd2ExamineshapefileFn(myshapefilesearch);
```

Using this hook allows you, for instance, to let the user specify a directory in which SHX files are to be found.

MONITORING FILE LOADS/SAVES

OpenDWG Toolkit provides several functions for monitoring loads and saves of files. Three of these functions are odometers which indicate the progress through the file during certain operations. For each of these functions, you define a function of your own which accepts a short integer as its argument, and register your function with OpenDWG Toolkit using the appropriate adSetAd2... function.

The three types of monitoring functions are Loadometer, Saveometer, and Dxfscanometer. Loadometer monitors the loading of files. Saveometer monitors the saving of files. Dxfscanometer monitors the scan of DXF files, which is an initial scan that is performed only on DXF files and which takes place before the load starts.

Here is an example of such a function:

```c
void printpercent(short percent)
{
    printf("so far, %d percent\n",percent);
}
```
and for initialization,

```c
adSetAd2DxfscanometerFn(printpercent);
```

and similarly for the other two functions.

The remaining OpenDWG Toolkit monitoring function allows you to examine entities as they are being loaded. `adSetAd2ExamineEntityLoad()` is the function used to register your function with OpenDWG Toolkit. Here is an example of such a function:

```c
void myexamineentityduringload(PAD_ENT_HDR adenhd, PAD_ENT aden)
{
    printf("entity type is %d\n",adenhd->enttype);
}
```

and for initialization,

```c
adSetAd2ExamineEntityLoad(myexamineentityduringload);
```

### ALIGNING TEXT WITH `adComputeNewTextParms()`

One difficulty encountered when creating text (or text-esque entities, such as attributes and attdefs) within OpenDWG Toolkit is computing the appropriate control points for text that is not drawn with left justification. AutoCAD draws all text from the stored lower left point, regardless of whether this text was designated as being centered, fit, middle, aligned, etc. By drawing all text from the stored lower left point, it becomes unnecessary for AutoCAD to compute this lower left point “on-the-fly” during operation of AutoCAD. AutoCAD simply computes and stores the lower left point when the text is created or modified.

Therefore, the OpenDWG Toolkit programmer must also compute a lower left point when creating text that is not drawn from the lower left. This is accomplished by using `adComputeNewTextParms()`. This ODT function computes the appropriate lower left point for a text entity, given its base point (stored in the `secondtextloc` field in the `tdata` substructure), and its justification. An example of how to do this is given below:

```c
. . .
```
adenhd->enttype==AD_ENT_TEXT;
adSetEntityDefaults(handle, adenhd, aden);
aden->text.tdata.justification=AD_TEXT_JUST_CENTER
aden->text.tdata.secondtextloc[0]=5.0; /* center around 5,6 */
aden->text.tdata.secondtextloc[1]=6.0; /* center around 5,6 */
strcpy(aden->text.textstr, "this is a test");
adComputeNewTextParms(handle, aden->text.textstr, aden->text.pt0, &aden->text.tdata, "SIMPLEX", NULL);

`adComputeNewTextParms` will compute the appropriate lower left point for this text string, justification, and font file. Although this example does not show it, `adComputeNewTextParms` will also take into account the width factor of the text.

**ABORTING LOAD OF A FILE**

You can abort a file load by calling `adStopLoading()`. Normally this action is performed during a call out by OpenDWG Toolkit to a function set using either `adSetAd2ExamineEntityLoadFn()` or `adSetAd2LoadometerFn()`. Note that ODT does not terminate loading right away. Your function calling `adStopLoading()` should return to ODT, which will in turn terminate at its next checkpoint.

**UNICODE SUPPORT**

OpenDWG Toolkit contains a number of functions to support the Unicode standard. Please read the following discussion of these issues:

**Background -- what are code pages?**

A code page is essentially an index to a translation table indicating the mapping of hex values to displayed characters. Some common code pages are dos850, which is the default code page for AutoCAD R13 DOS for U.S. English, and ansi_1250, which is the default code page for AutoCAD R13 Windows for U.S. English. The combination of the code page and the character value defines the character to be displayed. Thus, for instance, the character ° (degree symbol) is represented by the hex value 0xF8 in code page dos850, but by hex value 0xB0 in code page ansi_1250. Thus, it is important to ensure that the characters you store in the AutoCAD database are the correct ones for the code page of the drawing.
How do I determine the code page for a drawing?

The drawing's code page is given by the value of adhd->dwgcodepage. You can get the name of the code page using adShortCodePageToAscii(), but there is usually no reason to do this; all of the OpenDWG Toolkit functions related to code pages want the integer value.

What is Unicode?

Unicode is a character representation scheme promulgated by The Unicode Consortium. In Unicode, characters are represented as 16 bit unsigned values. This allows most characters devised by mankind to be mapped into unique 16 bit values. AutoCAD allows direct entry of Unicode values under certain circumstances, particularly MTEXT, through use of the \U+xxxx syntax, where xxxx represents the hex value of the character. Release 13 of AutoCAD introduced the use of font files based on Unicode.

How do I know whether or not Unicode fonts are being used in the drawing?

Unfortunately, it is not easy to make this determination. The best heuristic, in our opinion, is to use adFileVersionLoaded() to determine the version of the file which was loaded, and, if this version is >=AD_ACAD13, to assume that Unicode fonts are in use. If it is possible for you to search for the font files, you can examine the first 30 characters of the font and see if the word "unifont" appears, indicating a Unicode font.

How do I make sure I'm using the right characters in my drawing if Unicode fonts are in use?

Most code pages map the most common characters (A..Z, a..z, and 0..9) onto the same hex values, so for English uses there may be no need for special heroics. For other languages or special symbols, however, we believe that the following set of steps is the most prudent to translate a character to the drawing's code page:

a) Determine the code page you are using; that is, from which code page does the character set in which you are programming derive?

b) Determine the code page of the drawing (adhd->dwgcodepage).

c) Translate the character to Unicode using adCodePageCharToUnicode().

d) Translate the unicode-translated character to the drawing's code page using adUnicodeToCodePageChar(). If the translation fails, use the \U+xxxx representation if valid.

We use this double-translation method (source --> Unicode --> target) because it eliminates the need for the n-squared translation tables which would be required to translate from every possible source to every possible target. Instead, we need simply to be able to translate to and from Unicode.

Here's a working example. Assume that we are using characters originating in the dos850 code page. The representation of the degree symbol, as we discussed above, is 0xF8 for this code page. We wish to write this character into a text string in a drawing which we have loaded in OpenDWG Toolkit.
short ourcodepage,newchar;
unsigned short unicodechar;
PAD_DWG_HDR adhd;

/* we know our code page is dos850 */
success=adAsciiCodePageToShort("dos850",&ourcodepage);

/* pull in the header to get the dwgcodepage. */
/* Normally you would want to do this just once, rather than once per character */
adhd=adHeaderPointer(handle);

/* translate to Unicode */
adCodePageCharToUnicode(ourcodepage,0xF8,&unicodechar);

/* translate to the target code page */
adUnicodeToCodePageChar(unicodechar,adhd->dwgcodepage,&newchar);

newchar now contains the character translated to the drawing's code page.

Why can't I just change adhd->dwgcodepage to my code page? When can I avoid this translation?

This strategy would not work. Since all of the strings already present in the file are in the old code page, simply changing this value in the header will not magically cause all of the strings in the file to change also. You can change adhd->dwgcodepage, however, if you are creating a new drawing from scratch. In this case you could bypass completely any need for translations as described here. It is also possible that you are programming using the same code page with which the drawing was created; in this case, no translations would be required.

What changes have been made to OpenDWG Toolkit to accommodate Unicode?

The most obvious change is that OpenDWG Toolkit now has an initialization file, adinit.dat, which contains the Unicode translation tables. We have tried for years to avoid having to load an initialization file, but there is really no sensible way to load 450Kb of static data otherwise. We have added a final parameter to adInitAd2() which is the path to the init file. If you pass in NULL, ODT will still run, but you will not be able to use the new functions adUnicodeToCodePageChar() and adCodePageCharToUnicode().

There are five OpenDWG Toolkit functions which assist with Unicode:

- adAsciiCodePageToShort -- gives the integer index for an ASCII code page name.
- adCodePageCharToUnicode -- converts a character on a given code page to its Unicode equivalent.
- adNumValidCodePages -- returns the number of code pages supported by ODT
- adShortCodePageToAscii -- gives the name of the code page indicated by a given integer index.
- adUnicodeToCodePageChar -- converts a given Unicode character to its code page equivalent.
# 7. FUNCTIONS BY CATEGORY

Below is a taxonomy of OpenDWG Toolkit functions by category.

<table>
<thead>
<tr>
<th>BLOB MANAGEMENT</th>
<th>CLASSES</th>
</tr>
</thead>
<tbody>
<tr>
<td>adBlobSeek</td>
<td>adAddClass</td>
</tr>
<tr>
<td>adBlobSize</td>
<td>adDeleteClass</td>
</tr>
<tr>
<td>adBlobSizeFromVmaddr</td>
<td>adGetClass</td>
</tr>
<tr>
<td>adBlobTell</td>
<td>adNextClassnum</td>
</tr>
<tr>
<td>adCreateBlob</td>
<td>adNumClasses</td>
</tr>
<tr>
<td>adDeleteBlob</td>
<td>adStartClassGet</td>
</tr>
<tr>
<td>adEndBlobAppend</td>
<td></td>
</tr>
<tr>
<td>adEndBlobRead</td>
<td></td>
</tr>
<tr>
<td>adEndBlobWrite</td>
<td></td>
</tr>
<tr>
<td>adReadBlob2Double</td>
<td></td>
</tr>
<tr>
<td>adReadBlob3Double</td>
<td></td>
</tr>
<tr>
<td>adReadBlobByte</td>
<td></td>
</tr>
<tr>
<td>adReadBlobBytes</td>
<td></td>
</tr>
<tr>
<td>adReadBlobDouble</td>
<td></td>
</tr>
<tr>
<td>adReadBlobLong</td>
<td></td>
</tr>
<tr>
<td>adReadBlobObjhandle</td>
<td></td>
</tr>
<tr>
<td>adReadBlobShort</td>
<td></td>
</tr>
<tr>
<td>adReadBlobString</td>
<td></td>
</tr>
<tr>
<td>adRewindBlob</td>
<td></td>
</tr>
<tr>
<td>adStartBlobAppend</td>
<td></td>
</tr>
<tr>
<td>adStartBlobRead</td>
<td></td>
</tr>
<tr>
<td>adStartBlobWrite</td>
<td></td>
</tr>
<tr>
<td>adWriteBlob2Double</td>
<td></td>
</tr>
<tr>
<td>adWriteBlob3Double</td>
<td></td>
</tr>
<tr>
<td>adWriteBlobByte</td>
<td></td>
</tr>
<tr>
<td>adWriteBlobBytes</td>
<td></td>
</tr>
<tr>
<td>adWriteBlobDouble</td>
<td></td>
</tr>
<tr>
<td>adWriteBlobLong</td>
<td></td>
</tr>
<tr>
<td>adWriteBlobObjhandle</td>
<td></td>
</tr>
<tr>
<td>adWriteBlobShort</td>
<td></td>
</tr>
<tr>
<td>adWriteBlobString</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>BLOCK HEADERS</th>
<th>DATE/TIME FUNCTIONS</th>
</tr>
</thead>
<tbody>
<tr>
<td>adAddBlockheader</td>
<td>adDecodeAcadDate</td>
</tr>
<tr>
<td>adDeleteBlockheader</td>
<td>adDecodeAcadTime</td>
</tr>
<tr>
<td>adFindBlockheaderByName</td>
<td>adEncodeAcadDate</td>
</tr>
<tr>
<td>adGetBlockHandle</td>
<td>adEncodeAcadTime</td>
</tr>
<tr>
<td>adGetBlockheader</td>
<td></td>
</tr>
<tr>
<td>adNumBlockheaders</td>
<td></td>
</tr>
<tr>
<td>adReplaceBlockheader</td>
<td></td>
</tr>
<tr>
<td>adSeekBlockheader</td>
<td></td>
</tr>
<tr>
<td>adStartBlockheaderGet</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>DIMSTYLES</th>
<th>DICTIONARY OBJECTS</th>
</tr>
</thead>
<tbody>
<tr>
<td>adAddDimstyle</td>
<td>adReadDicItem</td>
</tr>
<tr>
<td>adDeleteDimstyle</td>
<td>adWriteDicItem</td>
</tr>
<tr>
<td>adFindDimstyleByName</td>
<td></td>
</tr>
<tr>
<td>adGetDimstyle</td>
<td></td>
</tr>
<tr>
<td>adNumDimstyles</td>
<td></td>
</tr>
<tr>
<td>adReplaceDimstyle</td>
<td></td>
</tr>
<tr>
<td>adSeekDimstyle</td>
<td></td>
</tr>
<tr>
<td>adsetDefaultDimstyle</td>
<td></td>
</tr>
<tr>
<td>adsetDefaultDimstyleFromHeader</td>
<td></td>
</tr>
<tr>
<td>adStartDimstyleGet</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>ENTITIES</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>adAddEntityToList</td>
<td></td>
</tr>
<tr>
<td>adAddEntityToListAfter</td>
<td></td>
</tr>
<tr>
<td>adAddReactorToEntity</td>
<td></td>
</tr>
<tr>
<td>adDeleteEntity</td>
<td></td>
</tr>
<tr>
<td>adEntityList</td>
<td></td>
</tr>
<tr>
<td>adGetEntity</td>
<td></td>
</tr>
<tr>
<td>adNumEntities</td>
<td></td>
</tr>
<tr>
<td>adReplaceEntity</td>
<td></td>
</tr>
<tr>
<td>adSeekEntity</td>
<td></td>
</tr>
<tr>
<td>adSetEntityDefaults</td>
<td></td>
</tr>
<tr>
<td>adSetEntityDefaultsFromHeader</td>
<td></td>
</tr>
<tr>
<td>adStartEntityGet</td>
<td></td>
</tr>
</tbody>
</table>
ENTITY LONGDATA
adReadAcisString
adReadMtextBlock
adWriteAcisString
adWriteMtextBlock

ENTITY TYPES
adArcAlignedTextEnttype
adHatchEnttype
adImageEnttype
adLwplineEnttype
adOle2frameEnttype
adRtextEnttype
adWipeoutEnttype

ERROR HANDLING
adError
adErrorClear
adErrorObjhandle
adErrorStr

EXTENDED DATA
adReadExtendeddata
adWriteExtendeddata

EXTERNAL REFERENCES
adCreateXrefBlock

GROUPS
adCreateGroup

HEADER
adHeaderPointer
adReadHeaderBlock
adSetHeaderDefaults
adWriteHeaderBlock

INTERFACE FUNCTIONS
adSetAd2CloseInitFileFn
adSetAd2CriticalErrorFn
adSetAd2DxfscanometerFn
adSetAd2ExamineEntityLoadFn
adSetAd2ExamineshapefileFn
adSetAd2LoadometerFn
adSetAd2OpenInitFileFn
adSetAd2ReadInitFileCharFn
adSetAd2SaveometerFn
adSetAd2VmmallocFns
adSetAd2WarningFn

LAYERS
adAddLayer
adDeleteLayer
adFindLayerByName
adGetLayer
adGetLayerState
adNumLayers
adReplaceLayer
adSeekLayer
adSetDefaultLayer
adSetLayerState
adStartLayerGet

LINETYPES
adAddLinetype
adDeleteLinetype
adFindLinetypeByName
adGetLinetype
adNumLinetypes
adReplaceLinetype
adSeekLinetype
adSetDefaultLinetype
adStartLinetypeGet

MLSTYLES
adReadMlstyleSeginfo
adSetDefaultMlstyle
adWriteMlstyleSeginfo

OBJECT TYPES
adDictionaryvarObjtype
adDictionarywdfltObjtype
adIdbufferObjtype
adImagedefObjtype
adImagedefreactorObjtype
adLayerindexObjtype
adLayoutObjtype
adObjectptrObjtype
adPlaceholderObjtype
adRastervariablesObjtype
adSortentstableObjtype
adSpatialfilterObjtype
adSpatialindexObjtype
adVbaProjectObjtype
adWipeoutVariablesObjtype
adXrecordObjtype

OBJECTS
adAddObject
adDeleteObject
adGetObject
adNumObjects
adReplaceObject
adSeekObject
adSetObjectDefaults
adStartObjectGet
adWriteXrecordItem

OBJHANDLE MANIPULATION
adAsciiObjhandleToBinary
adBinaryObjhandleToAscii
adGenerateObjhandle
adHanclear
adHancmp
adHancpy
adIncrementObjhandle
adObjectptrObjtype
adSetNewFileStartHandle
adSetCallbackUserData
adSetDxfCompleteness
adShadowObjhandlePtr
adValHdrStr
PROXY DATA
adEndProxyDataRead
adEndProxyDataWrite
adReadProxyDataBinary
adReadProxyDataBool
adReadProxyDataChar
adReadProxyDataDouble
adReadProxyDataInt16
adReadProxyDataInt32
adReadProxyDataObjhandle
adReadProxyDataPoint2d
adReadProxyDataPoint3d
adReadProxyDataScale3d
adReadProxyDataString
adReadProxyDataTypedObjhandle
adReadProxyDataVector2d
adReadProxyDataVector3d
adReadProxyDataVoidstardata
adReadProxyEntityObjid
adReadProxyObjectObjid
adStartProxyDataRead
adStartProxyDataWrite
adWriteProxyDataBinary
adWriteProxyDataBool
adWriteProxyDataChar
adWriteProxyDataDouble
adWriteProxyDataInt16
adWriteProxyDataInt32
adWriteProxyDataObjhandle
adWriteProxyDataPoint2d
adWriteProxyDataPoint3d
adWriteProxyDataScale3d
adWriteProxyDataString
adWriteProxyDataTypedObjhandle
adWriteProxyDataVector2d
adWriteProxyDataVector3d
adWriteProxyDataVoidstardata
adWriteProxyEntityObjid
adWriteProxyObjectObjid

PROXY GRAPHICS
adHasPrimTraits
adNumsFromFaceEntryList
adPrimsHaveColors
adPrimsHaveLayers
adPrimsHaveLinetypes
adPrimsHaveMarkers
adPrimsHaveNormals
adPrimsHaveOrientation
adPrimsHaveVisibilities
adReadGrblobData
adReadGrblobVertexPt
adReadGrshellFaceEntries
adReadPrimColors
adReadPrimFlags
adReadPrimLayerids
adReadPrimLinetypeids
adReadPrimMarkers
adReadPrimNormals
adReadPrimOrientation
adReadPrimVisibilities
adStartGrblobDataRead
adStartGrblobDataWrite
adWriteGrblobData
adWriteGrblobVertexPt
adWriteGrshellFaceEntries

PROXY OBJECTS
adReadXrecordItem
adReadProxyDataObjectObjid
adWriteXrecordItem
adWriteProxyDataObjectObjid

RASTER IMAGES
adAddBmp
adBmpSize
adBmpSizeFromFile
adDeleteBmp
adDeleteWmf
adExtractBmpFromFileToFile
adExtractBmpToFile
adFileHasBmp
adFileHasWmf
REGAPPS
adAddRegapp
adDeleteRegapp
adFindRegappByName
adGetRegapp
adNumRegapps
adReplaceRegapp
adSeekRegapp
adSetDefaultRegapp
adStartRegappGet

SHAPEFILES
adAddShapefile
adDeleteShapefile
adFindShapefileByName
adGetShapefile
adNumShapefiles
adReplaceShapefile
adSeekShapefile
adSetDefaultShapefile
adStartShapefileGet

STARTUP/SHUTDOWN
adCloseAd2
adCloseFile
adInitAd2
adLoadFile
adNewFile
adSaveFile
adSetupDwgRead
adSetupDxfRead
adStopLoading

TEXT POSITIONING
adComputeNewTextParms

UCS
adAddUcs
adDeleteUcs
adFindUcsByName
adGetUcs
adNumUcs
adReplaceUcs
adSeekUcs
adSetDefaultUcs
adStartUcsGet

UNICODE HANDLING
adAsciiCodePageToShort
adCodePageCharToUnicode
adNumValidCodePages
adShortCodePageToAscii
adUnicodeToCodePageChar

VIEWS
adAddView
adDeleteView
adFindViewByName
adGetView
adNumViews
adReplaceView
adSeekView
adSetDefaultView
adStartViewGet

VPORTS
adAddVport
adDeleteVport
adGetVport
adNumVports
adReplaceVport
adSeekVport
adSetDefaultVport
adStartVportGet
8. OTHER OpenDWG Toolkit CONCEPTS

VERSION DEPENDENCIES

We have done our best to minimize/eliminate any version dependencies for the OpenDWG Toolkit programmer. It should be virtually unnecessary for most ODT operations to have any dependence on either the version of the AutoCAD file that was loaded, or the format (DWG or DXF). Unfortunately there are still some circumstances where the version is important.

Throughout this manual we have attempted to tag data fields with indications such as (9), (12+), etc. These tags, for instance, would indicate respectively a field that is significant only for Release 9 of AutoCAD, and a field that was significant for Release 12 and greater.

The same is true of entities and certain tables. Autodesk introduced new entities into AutoCAD during its history, as well as new tables. We have attempted to indicate these as well. When writing a file, OpenDWG Toolkit will simply omit entities and tables that are not appropriate for a given version.

Entity (and object) handles have also been phased in by Autodesk over time. In Release 13 and later, all objects (not only entities, but all table entries) have handles, and they are mandatory. In Releases 11 and 12, main entities (i.e., entities not in blocks) and entities in blocks both could have handles; however handles were optional. In Release 10 only main entities could have handles, and they were optional. Versions prior to Release 10 did not have handles.

In order to provide a unified interface to objects in DWG and DXF files, ODT automatically generates and adds object handles to items being read. These handles are not written out when a file is saved to a version of AutoCAD that does not support/require them. This is significant to the OpenDWG Toolkit programmer because the object handles identifying items in the file may not be the same when the file is reloaded into ODT, since they are not stored in the file. Therefore it may be necessary to be careful about the versions of AutoCAD you wish to support, if you intend to depend on object handles being the same from session to session.

Below, we describe the versions of AutoCAD and the features that were added to each.

Version 2.5:

Entity types: ATTDEF, ATTRIBUTE, BLOCK, CIRCLE, ENDBLOCK, INSERT, LINE, POLYLINE, POINT, SEQEND, SHAPE, SOLID, TEXT, TRACE, VERTEX
Table types: BLOCKHEADER, LAYER, LINETYPE, SHAPEFILE, VIEW

Version 2.6:

Entity types added: DIMENSION

Release 10:

Table types added: UCS, VPORT

Release 11:
Entity types added: VIEWPORT
Table types added: DIMENSION STYLE, REGISTERED APPLICATION

Release 13:

Entity types added: CONSTRUCTION LINE, ELLIPSE, LEADER, MULTI-LINE, MULTI-LINE TEXT, RAY, REGION, SOLID3D, SPLINE, TOLERANCE, PROXY
Table types added: CLASS, OBJECT(DICTIONARY, GROUP, MULTI-LINE STYLE, PROXY, XRECORD)

Release 14:

Entity types added: LIGHTWEIGHT POLYLINE, HATCH, IMAGE
Object types added: IDBUFFER, DICTIONARYVAR, IMAGEDEFREACTOR, IMAGEDEF, RASTERVARIABLES, SPATIALFILTER, SPATIALINDEX, LAYERINDEX, OBJECTPTR, SORTENTTABLE

Release 2000:

Entity types added: ARCALIGNEDTEXT, RTEXT, WIPEOUT
Object types added: DICTIONARYWDFLT, PLACEHOLDER, LAYOUT, WIPEOUTVARIABLES, VBAPROJECT
9. FUNCTION REFERENCE

This section contains definitions for all OpenDWG Toolkit functions.

adAcceptR14BetaFiles

Summary

void adAcceptR14BetaFiles();

Description

Instructs ODT to allow loading of R14 Beta files (by default ODT will return an error when attempting to load a file of this type).

Return Value

void.

See Also

adLoadedR14BetaFile

Example

adAcceptR14BetaFiles();

adAddBlockheader

Summary

short adAddBlockheader(AD_DB_HANDLE handle, PAD_BLKH blkh);

Description

Adds the block header pointed to by blkh to the block header list.

Return Value

Returns 1 on success, 0 on failure. On failure, adError() should be checked for an indication of the cause of the failure.

See Also

adDeleteBlockheader, adNumBlockheaders, adSeekBlockheader

Example

adtb->blkh.flag=0;
adtb->blkh.xrefindex = -1;
adtb->blkh.purgedflag=0;
adtb->blkh.xdblob=AD_VMNULL;
strcpy(adtb->blkh.name,"BLKNAME");
adGenerateObjhandle(wrhandle,adtb->blkh.objhandle);
if (!adAddBlockheader(wrhandle,&adtb->blkh)) writeerror(wrhandle);

adAddBmp

Summary

short adAddBmp(AD_DB_HANDLE handle, AD_FILE_DESIGNATOR fs);

Description

Adds a BMP raster image file to the current drawing. AD_FILE_DESIGNATOR is char * for most platforms. Note that BMP raster images are only actually written to a file for DWG files R13 and above.

Return Value

Returns 1 on success, 0 on failure. On failure, adError() should be checked for an indication of the cause of the failure.

See Also

adDeleteBmp, adDeleteWmf, adExtractBmpToFile, adFileHasBmp, adFileHasWmf

Example

adAddBmp(handle,"c:\bmps\test.bmp");

adAddBmpFromBuffer

Summary

short adAddBmpFromBuffer(AD_DB_HANDLE handle, void *buffer, unsigned short size);

Description

Adds a BMP raster image file to the current drawing. buffer points to the BMP data to be added, and size is the size of the data in bytes. Note that BMP raster images are only actually written to a file for DWG files R13 and above.

Return Value

Returns 1 on success, 0 on failure. On failure, adError() should be checked for an indication of the cause of the failure.

See Also

adDeleteBmp, adDeleteWmf, adExtractBmpToFile, adFileHasBmp, adFileHasWmf

Example
adAddBmpFromBuffer(handle,bmpbuf,5314);

adAddClass

Summary

short adAddClass (AD_DB_HANDLE handle, PAD_CLASS cls);

Description

Adds the class pointed to by cls to the class list.

Return Value

Returns 1 on success, 0 on failure. On failure, adError() should be checked for an indication of the cause of the failure.

See Also

adNumClasses, adGetClass

Example

/* add the class for our proxies */
class.classnum=adNextClassnum(wrhandle);
class.version=0;
strcpy(class.classdxfname,"MARCOMPSAMP");
strcpy(class.cplusplusclassname,"MarCompSamp");
strcpy(class.appname,"MarComp Sample");
class.wasaproxy=0;
class.itemclassid=AD_ITEM_CLASS_PROXY_ENTITY;
adAddClass(wrhandle,&class);  

adAddDimstyle

Summary

short adAddDimstyle (AD_DB_HANDLE handle, PAD_DIMSTYLE dimst);

Description

Adds the dimension style pointed to by dimst to the dimension style list.

Return Value

Returns 1 on success, 0 on failure. On failure, adError() should be checked for an indication of the cause of the failure.

See Also

adDeleteDimstyle, adNumDimstyles

Example

adAddBmpFromBuffer(handle,bmpbuf,5314);
AD_DIMSTYLE dimst;

adSetDefaultDimstyle(&adtb->dimst);
strncpy(adtb->dimst.name, name);
adGenerateObjectHandle(wrhandle, adtb->dimst.objhandle);
if (!adAddDimstyle(wrhandle, &adtb->dimst)) writeerror(wrhandle);
**adAddEntityToList**

**Summary**

short adAddEntityToList (AD_DB_HANDLE handle, AD_VMADDR addlist, PAD_ENT_HDR adenhd, PAD_ENT aden);

**Description**

Adds an entity pointed to by adenhd and aden to the entity list passed in addlist.

**Return Value**

Returns 1 on success, 0 on failure. On failure, adError() should be checked for an indication of the cause of the failure.

**See Also**

adDeleteEntity, adNumEntities, adAddEntityToListAfter, adSeekEntity, adReplaceEntity

**Example**

/* draw a circle */
adenhd->enttype=AD_ENT_CIRCLE;
adSetEntityDefaults(wrHandle, adenhd, aden);
aden->circle.radius=3.0;
aden->circle.pt0[0]=3.0;
aden->circle.pt0[1]=3.0;
adGenerateObjHandle(wrHandle, adenhd->enthandle);
adAddEntityToList(wrHandle, mspaceentlist, adenhd, aden);

**adAddEntityToListAfter**

**Summary**

short adAddEntityToListAfter (AD_DB_HANDLE handle, AD_VMADDR addlist, PAD_ENT_HDR adenhd, PAD_ENT aden, AD_OBJHANDLE afterthisobj);

**Description**

Adds the entity pointed to by adenhd and aden to the list passed in addlist. The entity is added after the entity whose handle is afterthisobj. If afterthisobj is all zeroes, the entity is inserted at the head of the list.

**Return Value**

Returns 1 on success, 0 on failure. On failure, adError() should be checked for an indication of the cause of the failure.
See Also

adDeleteEntity, adNumEntities, adAddEntityToList, adSeekEntity, adReplaceEntity

Example

/* draw a circle */
adenhd->enttype=AD_ENT_CIRCLE;
adSetEntityDefaults(wrhandle, adenhd, aden);
aden->circle.radius=3.0;
aden->circle.pt0[0]=3.0;
aden->circle.pt0[1]=3.0;
adGenerateObjhandle(wrhandle, adenhd->enthandle);
adHanclear(temphn);
temphn[7]=0xC0; /* add after ent with handle C0 */
adAddEntityToListAfter(wrhandle, mspaceentlist, adenhd, aden, temphn);

adAddLayer

Summary

short adAddLayer (AD_DB_HANDLE handle, PAD_LAY lay);

Description

Adds the layer pointed to by lay to the layer list.

Return Value

Returns 1 on success, 0 on failure. On failure, adError() should be checked for an indication of the cause of the failure.

See Also

adDeleteLayer, adNumLayers, adReplaceLayer, adSeekLayer, adGetLayer

Example

adSetDefaultLayer(wrhandle, &adtb->lay);
strcpy(adtb->lay.name, name);
adtb->lay.color=color;
adHancpy(adtb->lay.linetypeobjhandle, firstltypeobjhandle);
adGenerateObjhandle(wrhandle, adtb->lay.objhandle);
if (!adAddLayer(wrhandle, &adtb->lay)) writeerror(wrhandle);
adAddLinetype

Summary

short adAddLinetype (AD_DB_HANDLE handle, PAD_LTYPE ltp);

Description

Adds the linetype pointed to by ltp to the linetype list.

Return Value

Returns 1 on success, 0 on failure. On failure, adError() should be checked for an indication of the cause of the failure.

See Also

adDeleteLinetype, adNumLinetypes, adReplaceLinetype, adSeekLayer, adGetLinetype

Example

/* and another */
adSetDefaultLinetype(&adtb->ltype);
strcpy(adtb->ltype.name, "DASHED");
strcpy(adtb->ltype.text,"__ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ 

10/25/99 OpenDWG Toolkit Version 2.0 59
adAddObject

Summary

    short adAddObject (AD_DB_HANDLE handle, PAD_OBJ_HDR adobhd, PAD_OBJ adob);

Description

    Adds the object pointed to by adobhd and adob to the object list. This function should be used to add all
    object types to the object list. This includes multi-line styles, dictionary entries, groups, proxy objects, xrecords,
    idbuffers, dictionaryvars, imagedefreactors, imagedefs, rastervariables, and spatialfilters.

Return Value

    Returns 1 on success, 0 on failure. On failure, adError() should be checked for an indication of the cause
    of the failure.

See Also

    adNumObjects, adDeleteObject, adReplaceObject, adSeekObject, adGetObject

Example

    AD_OBJ_HDR adobhd;
    AD_OBJ adob;
    PAD_BLOB_CTRL bcptr;
    AD_DB_HANDLE handle;
    AD_DIC_ITEM dicitem;
    adobhd.objtype=AD_OBJ_DICTIONARY;
    adSetObjectDefaults(handle,&adobhd,&adob);
    adGenerateObjhandle(handle,adobhd.objhandle);
    adob.dic.numdicitems=2;
    adobhd.numreactors=0;
    if ((adob.dic.itemblob=adCreateBlob())==AD_VMNULL) { return(0); }
    bcptr=adStartBlobWrite(adob.dic.itemblob);
    strcpy(dicitem.str,"ACAD_GROUP");
    adGenerateObjhandle(handle,dicitem.itemhandle);
    adHancpy(dicgrouphandle,dicitem.itemhandle);
    if (adWriteDicItem(bcptr,&dicitem)) {
        strcpy(dicitem.str,"ACAD_MLINESTYLE");
        adGenerateObjhandle(handle,dicitem.itemhandle);
        if (adWriteDicItem(bcptr,&dicitem)) {
            adEndBlobWrite(bcptr);
            if (adAddObject(handle,&adobhd,&adob)) {
                goto SUCCESS1;
            }
        }
    }
    adDeleteBlob(adob.dic.itemblob);
    return(0);

SUCCESS1:
    . . .
adAddReactorToEntity

Summary

short adAddReactorToEntity (PAD_ENT_HDR adenhd, AD_OBJHANDLE reactor);

Description

Adds a persistent reactor to the entity pointed to by adenhd.

Return Value

Returns 1 on success, 0 on failure. On failure, adError() should be checked for an indication of the cause of the failure.

See Also

adReplaceEntity

Example

if (!adSeekEntity(handle,entlist,temphandle,&adenhd,&aden)) {
    adin->aderror=AD_CANT_SEEK_ENTITY;
    return(0);
}
adAddReactorToEntity(&adenhd,grphdr.objhandle);
if (!adReplaceEntity(handle,entlist,adenhd.enthandle,&adenhd,&aden)) {
    adin->aderror=AD_CANT_REPLACE_ENTITY;
    return(0);
}

adAddRegapp

Summary

short adAddRegapp (AD_DB_HANDLE handle, PAD_APP app);

Description

Adds the registered application pointed to by app to the regapp list.

Return Value

Returns 1 on success, 0 on failure. On failure, adError() should be checked for an indication of the cause of the failure.

See Also

adNumRegapps, adDeleteRegapp, adReplaceRegapp, adSeekRegapp, adGetRegapp

Example

AD_APP app;
adSetDefaultRegapp(&app);
adGenerateObjhandle(handle,app.objhandle);
strcpy(app.name,"ACAD");
if (adAddRegapp(handle,&app)) return(0);

adAddShapefile

Summary

     short adAddShapefile (AD_DB_HANDLE handle, PAD_SHPTB shptb);

Description

     Adds the shape file (style) pointed to by shptb to the shape file list.

Return Value

     Returns 1 on success, 0 on failure. On failure, adError() should be checked for an indication of the cause of the failure.

See Also

     adNumShapefiles, adDeleteShapefile, adReplaceShapefile, adSeekShapefile, adGetShapefile

Example

     /* write a shapefile entry */
adSetDefaultShapefile(&adtb->shptb);
strcpy(adtb->shptb.name,"ANOTHERSTYLE");
adtb->shptb.height=0.2;
adtb->shptb.width=1.0;
adtb->shptb.angle=0.0;
adtb->shptb.flag2=0;
adtb->shptb.lastshapeheight=0.2;
strcpy(adtb->shptb.file,"txt");
adGenerateObjhandle(wrhandle,adtb->shptb.objhandle);
if (!adAddShapefile(wrhandle,&adtb->shptb)) writeerror(wrhandle);

adAddUcs

Summary

     short adAddUcs (AD_DB_HANDLE handle, PAD_UCS ucs);

Description

     Adds the user coordinate system entry pointed to by ucs to the UCS list.

Return Value

     Returns 1 on success, 0 on failure. On failure, adError() should be checked for an indication of the cause of the failure. See Also adNumUcs, adDeleteUcs, adReplaceUcs, adSeekUcs, adGetUcs
Example

```
AD_UCS theucs;theucs.purgedflag=0;
theucs.xrefindex=-1;
theucs.flag=0;
strcpy(theucs.name,"MYUCS");
theucs.xdir[0]=0.0; theucs.xdir[1]=1.0; theucs.xdir[2]=0.0;
theucs.ydir[0]=0.0; theucs.ydir[1]=0.0; theucs.ydir[2]=1.0;
theucs.xdblob=AD_VMNULL;
adGenerateObjhandle(handle,theucs.objhandle);
adAddUcs(handle,&theucs);
```

adAddView

**Summary**

```
short adAddView (AD_DB_HANDLE handle, PAD_VIEW view);
```

**Description**

Adds the view description pointed to by `view` to the view list.

**Return Value**

Returns 1 on success, 0 on failure. On failure, `adError()` should be checked for an indication of the cause of the failure.

**See Also**

`adNumViews, adDeleteView, adReplaceView, adSeekView, adGetView`
Example

AD_VIEW theview;

adSetDefaultView(&theview);
adGenerateObjhandle(handle, theview.objhandle);
adAddView(handle, &theview);

adAddVport

Summary

short adAddVport (AD_DB_HANDLE handle, PAD_VPORT vport);

Description

Adds the viewport description pointed to by vport to the viewport list.

Return Value

Returns 1 on success, 0 on failure. On failure, adError() should be checked for an indication of the cause of the failure.

See Also

adNumVports, adDeleteVport, adReplaceVport, adSeekVport, adGetVport

Example

AD_VPORT vport;

adSetDefaultVport(&vport);
strcpy(vport.name, "MYVPORT");
adGenerateObjhandle(handle, vport.objhandle);
adAddVport(handle, &vport);

adAdVersion

Summary

char *adAdVersion (void);

Description

Returns a pointer to the OpenDWG Toolkit version string.

Return Value

Pointer to string.
See Also

None

Example

printf("OpenDWG Toolkit version: %s\n",adAdVersion());

adAllowInvalidDxfVersions

Summary

void adAllowInvalidDxfVersions (char allow);

Description

Calling this function with a value of 1 causes OpenDWG Toolkit to suppress the check for the version of
the DXF file being read. This may allow OpenDWG Toolkit to be able to read certain poorly written DXF files
which omit this information. However, we cannot guarantee OpenDWG Toolkit's successful operation when this
feature is used.

Return Value

There is no return value.

See also

None

Example

adAllowInvalidDxfVersions(1);

adArcAlignedTextEnttype

Summary

short adArcAlignedTextEnttype(AD_DB_HANDLE handle);

Description

Returns the entity type value for AD_ARCALIGNEDTEXT entities.

Return Value

Returns the entity type value for AD_ARCALIGNEDTEXT entities.

See Also

None
Example

arcalignedtextenttype=adArcAlignedTextEnttype(handle);

adAsciiCodePageToShort

Summary

short adAsciiCodePageToShort(char *codepagestr, short *index);

Description

Sets the value pointed to by \texttt{index} to the integer value representing the code page whose ASCII name is pointed to by \texttt{codepagestr}. The code page name must be in lower case.

Return Value

Returns 1 on success, 0 on failure.

See Also

adShortCodePageToAscii

Example

success=adAsciiCodePageToShort("dos850", &adhd->dwgcodepage);

adAsciiHexToBinary

Summary

void adAsciiHexToBinary (char *readbuf, char *storebuf, short *validchars);

Description

Converts the ASCII hex data in \texttt{readbuf} to binary.

Return Value

There is no return value. The converted hexadecimal is stored in \texttt{storebuf}, and the number of characters converted is placed in \texttt{validchars}.

See Also

adBinaryHexToAscii

Example

adAsciiHexToBinary("ABCDEF01", storebuf, &numchars);
adAsciiObjhandleToBinary

Summary

void adAsciiObjhandleToBinary (char *readbuf, unsigned char *handlebuf);

Description

Converts the AD_OBJHANDLE in readbuf from ASCII representation to binary.

Return Value

There is no return value. The converted handle is stored in handlebuf.

See Also

adBinaryObjhandleToAscii

Example

adAsciiObjhandleToBinary("ABCDEF01", theobjhandle);

adAudit

Summary

short adAudit(AD_DB_HANDLE handle, short auditmode, short verbosemode, short (*auditmsg)(char *msg));

Description

Audits the file indicated by handle. This process scans the entire database and attempts to find and report on errors within the database. Generally, adAudit tests for invalid AD_OBJHANDLEs, invalid blob pointers, and other anomalies.

adAudit takes the handle of the file to be audited as its first parameter. The second parameter is auditmode, which should be set to one of the defined constants AD_SAFEMODE or AD_UNSAFEMODE. Generally AD_UNSAFEMODE performs a more thorough test, but may also perform tests which could crash OpenDWG Toolkit or cause protection faults.

The third parameter is verbosemode, which should be set to one of the defined constants AD_VERBOSITY_NONE, AD_VERBOSITY_QUIET, or AD_VERBOSITY_VERBOSE. NONE causes no output. QUIT causes output only when there is an error found. VERBOSE gives a full report on the test as it proceeds.

The final parameter, auditmsg, is a pointer to a function which accepts a character pointer as its argument. adAudit uses this function to pass progress messages to your program. Passing a NULL value causes no messages to be emitted. Also, returning 0 from the function called through the auditmsg pointer causes the audit to terminate.

Return Value

adAudit returns 1 if no errors are found, 0 if there are errors.
See Also

None

Example

/* declare the function to receive audit info */
short myauditmessage(char *str)
{
    printf("%s\n", str);
}

/* somewhere else in the program */
adAudit(handle, AD_UNSAFE_MODE, AD_VERBOSITY_QUIET, myauditmessage);

adBinaryHexToAscii

Summary

void adBinaryHexToAscii (unsigned char *bindata, short bindatalength, char *tempstr);

Description

Converts the binary hex data in bindata of length bindatalength to ASCII.

Return Value

There is no return value. The converted hexadecimal is stored in tempstr.

See Also

adAsciiHexToBinary

Example

char bindata[128];
short len;
char storebuf[256];
adBinaryHexToAscii(bindata, len, storebuf);

adBinaryObjhandleToAscii

Summary

void adBinaryObjhandleToAscii (AD_OBJHANDLE enthandle, char *tempstr);

Description

Converts the AD_OBJHANDLE in enthandle from binary to ASCII. The maximum size for the returned string is 16, so you must provide an array of at least size 17 to store the ASCII string, including the NULL terminator.

Return Value
There is no return value. The converted handle is stored in `tempstr`.

See Also

`adAsciiObjhandleToBinary`

Example

```c
AD_OBJHANDLE theobjhandle;
char storebuf[17];
adBinaryObjhandleToAscii(theobjhandle,storebuf);
```

### adBlobSeek

**Summary**

```c
short adBlobSeek (PAD_BLOB_CTRL bcptr, long offset);
```

**Description**

Seeks to location `offset` in the BLOB indicated by `bcptr`. This function should only be performed on blobs open for read.

**Return Value**

Returns 1 on success, 0 on failure. On failure, `adError()` should be checked for an indication of the cause of the failure.

See Also

`adReadBlobBytes, adReadBlobByte`

**Example**

```c
success=adBlobSeek(bcptr,125L);
```

### adBlobSize

**Summary**

```c
long adBlobSize (PAD_BLOB_CTRL bcptr);
```

**Description**

Returns the size of the BLOB indicated by `bcptr`.

**Return Value**

Returns 1 on success, 0 on failure. On failure, `adError()` should be checked for an indication of the cause of the failure.
See Also

    adBlobSizeFromVmaddr

Example

    size=adBlobSize(bcptr);
adBlobSizeFromVmaddr

Summary

long adBlobSizeFromVmaddr (AD_VMADDR blob);

Description

Returns the size of the blob with address blob. This function can be used to get the size of a blob which is currently closed, without need to open it for read.

Return Value

Returns 1 on success, 0 on failure. On failure, adError() should be checked for an indication of the cause of the failure.

See Also

adBlobSize

Example

size=adBlobSizeFromVmaddr (adenhd->xdblob);

adBlobTell

Summary

long adBlobTell (PAD_BLOB_CTRL bcptr);

Description

Returns the current location in the BLOB indicated by bcptr.

Return Value

Returns 1 on success, 0 on failure. On failure, adError() should be checked for an indication of the cause of the failure.

See Also

adBlobSizeFromVmaddr, adBlobSize

Example

loc=adBlobTell (bcptr);
**adBmpSize**

**Summary**

unsigned short adBmpSize(AD_DB_HANDLE handle);

**Description**

Returns the size of the BMP image attached to the file indicated by handle. To be used, generally, with adExtractBmpToBuffer and adExtractBmpToFile.

**Return Value**

As above.

**See Also**

adExtractBmpToBuffer, adExtractBmpToFile, etc.

**Example**

```c
if (adFileHasBmp(handle)) {
    size=adBmpSize(handle);
    bmpptr=malloc(size);
    adExtractBmpToBuffer(handle,bmpptr);
}
```

**adBmpSizeFromFile**

**Summary**

unsigned short adBmpSizeFromFile(PATHSPEC filepath);

**Description**

Returns the size of the BMP image stored in the file pointed to by filepath. To be used, generally, with adExtractBmpFromFileToBuffer, adExtractBmpFromFileToFile.

**Return Value**

Returns the size of the BMP image, or 0 if the file does not contain one.

**See Also**

adExtractBmpFromFileToBuffer, adExtractBmpFromFileToFile, etc.

**Example**

```c
size=adBmpSizeFromFile("c:\\dwgs\\13.dwg");
```
adCalcPlineArcCenter

Summary

void adCalcPlineArcCenter (double *pt1, double *pt2, double inclang, double *cenx, double *ceny, double *rad);

Description

Given start and endpoints (pt1 and pt2) of a polyline arc, sets the center and radius of the arc. A polyline vertex bulge is the tangent of ¼ the included angle of the arc, so to get the included angle you should take 4.0*atan(bulge). If the included angle is 0.0, there is no arc.

Return Value

There is no return value.

See Also

adBlobSizeFromVmaddr, adBlobSize

Example

incl=4.0*atan(aden->vertex.bulge);
if (incl==0.0) printf("no arc\n");
else adCalcPlineArcCenter(aden->vertex.pt0[0],
 aden->vertex.pt0[1], incl, &cenx,& ceny, &radius);

adCloseAd2

Summary

void adCloseAd2 (void);

Description

Closes OpenDWG Toolkit 2.

Return Value

There is no return value.

See Also

adInitAd2

Example

adCloseAd2();
adCloseFile

Summary

short adCloseFile (AD_DB_HANDLE handle);

Description

Closes a file previously opened by either adNewFile or adLoadFile.

Return Value

Returns 1 on success, 0 on failure. On failure, adError() should be checked for an indication of the cause of the failure.

See Also

adLoadFile, adNewFile

Example

adCloseFile(handle);

adCodePageCharToUnicode

Summary

short adCodePageCharToUnicode(short origcodepage, short codepagechar, unsigned short *unicodechar);

Description

Translates a character from a given codepage to its Unicode representation.

Return Value

Returns 1 on success, 0 on failure. On success, the value pointed to by unicodechar is set to the translated value.

See Also

adUnicodeToCodePageChar

Example

success=adCodePageCharToUnicode(12,0xF8,&unicodeval);
adComputeBulge

Summary

    short adComputeBulge(double *stpt, double *endpt, double *center, double *bulge);

Description

    Assists in creating a polyline with an arc segment. Given an arc start point, end point, and center, this
    function computes and returns the proper bulge value for a polyline vertex.

Return Value

    Returns 1 on success, 0 on failure.

See Also

    None

Example

    /* set up next end point and center of arc for bulge computation */
    /* arc is drawn FROM the current vertex TO the next one */
    endpt[0]=28.0;
    endpt[1]=7.0;
    endpt[2]=0.0;
    center[0]=27.0;
    center[1]=6.0;
    center[2]=0.0;
    adComputeBulge(aden->vertex.pt0,endpt,center,&aden->vertex.bulge);

adComputeNewTextParms

Summary

    short adComputeNewTextParms (char *str, double *pt0, PAD_TDATA tdata, char *fontfile);

Description

    Computes new text parameters from the string passed in str, based on the points in tdata, and stores the
    result in pt0. The new parameters are based on the font name passed in fontfile.

Return Value

    Returns 1 on success, 0 on failure. On failure, adError() should be checked for an indication of the cause
    of the failure.

See Also

    Section entitled "Aligning text with adComputeNewTextParms..."

Example

    PAD_ENT_HDR adenhd;
PAD_ENT aden;

adGetEntity(handle, aden, adenhd);
/* assuming it's a text entity */
aden->text.tdata.justification=AD_TEXT_JUST_RIGHT;
aden->text.tdata.secondtextloc[0]=5.0;
aden->text.tdata.secondtextloc[1]=5.0;
adComputeNewTextParms(handle, aden->text.textstr, aden->text.pt0,
 &aden->text.tdata,"SIMPLEX",NULL);

adConvertPlinesOnLoad

Summary

void adConvertPlinesOnLoad (short convert);

Description

Tells OpenDWG Toolkit whether or not to convert polylines to lightweight polylines at load time. By
default, OpenDWG Toolkit will make this conversion. Passing a 1 tells OpenDWG Toolkit to do the conversion,
passing 0 tells it not to do the conversion. This function should be executed after adInitAd2().

Return Value

There is no return value.

See Also

None

Example

adConvertPlinesOnLoad(0);

adCreateBlob

Summary

AD_VMADDR adCreateBlob(void);

Description

Creates an OpenDWG Toolkit BLOB (Binary Large Object). This is an arbitrary size chunk of binary data.

Return Value

Returns the AD_VMADDR of the created blob; AD_VMNULL if failure. On failure, adError() should be checked for an indication of the cause of the failure.

See Also
**adWriteBlobBytes, adEndBlobWrite**

**Example**

```c
AD_VMADDR theblob;
PAD_BLOB_CTRL bcptr;

theblob=adCreateBlob();
bcptr=adStartBlobWrite(theblob);
...  
adEndBlobWrite(bcptr);
```

**adCreateGroup**

**Summary**

short adCreateGroup(AD_DB_HANDLE handle, char *groupname, char *groupdesc, AD_OBJHANDLE groupobjhandle, long numgroupitems, AD_VMADDR groupentblob, AD_VMADDR entlist, short unnamed, short selectable);

**Description**

Creates an AutoCAD group. This function encapsulates most of the tedious process of creating a group. The passed-in parameters are:

- **handle**: the file’s handle
- **groupname**: the name of the group
- **groupdesc**: description of the group
- **groupobjhandle**: the desired object handle for this group
- **numgroupitems**: the number of items in the group
- **groupentblob**: a BLOB containing the object handles of the entities to be grouped
- **entlist**: the entity list in which the entities reside
- **unnamed**: 1 if the group is to have no name (typically 0)
- **selectable**: 1 if the group is to be selectable (typically 1)

To create the group, create a BLOB containing the object handles for the entities to be placed in the group. Use `adGenerateObjhandle()` to generate an object handle for the group. Then call `adCreateGroup()` with the appropriate parameters. Note that the BLOB you give this function is not copied, but used directly.

**Return Value**

Returns 1 on success, 0 on failure. On failure, `adError()` should be checked for an indication of the cause of the failure.
See Also

None

Example

AD_VMADDR groupentblob;
PAD_BLOB_CTRL bcptr;
short i;

groupentblob=adCreateBlob();
bcptr=adStartBlobWrite(groupentblob);
for (i=0; i<12; i++)
adWriteBlobObjhandle(bcptr,groupentobjhandle[i]);
adEndBlobWrite(bcptr);
adGenerateObjhandle(wrhandle,groupobjhandle);

if (!adCreateGroup(wrhandle,"MY_GROUP",groupobjhandle,12L,
groupentblob,mspaceentlist,0,1)) {
    printf("creategroup failed, error %d: s\n",adError(),
    adErrorStr(adError()));
}

adCreateXrefBlock

Summary

short adCreateXrefBlock(AD_DB_HANDLE handle, char *blkpath, char *blkname, char overlaid,
AD_OBJHANDLE blkobjhandle);

Description

Creates an AutoCAD xref block. This function encapsulates most of the tedious process of creating an
xref block. The passed in parameters are:

handle the file's handle
blkpath the path to the xref
blkname the name of the block
overlaid 0 if normal attached xref, 1 if an overlaid xref

Return Value

Returns 1 on success, 0 on failure. On failure, adError() should be checked for an indication of the cause
of the failure. On return, the blkobjhandle parameter will contain the object handle of the blockheader created by
adCreateXref().

See Also

None

Example

/* add an external reference */
adCreateXrefBlock(wrhandle,"j:\13.dwg","13",0,xrefblkobjhandle);
/* now add an insertion of that block */
writeaninsert(xrefblkobjhandle,5.0,5.0,0);
**adCtrlObjhandlePtr**

**Summary**

```
AD_OBJHANDLE *adCtrlObjhandlePtr(AD_DB_HANDLE handle);
```

**Description**

Returns a pointer to the control handles for an OpenDWG Toolkit database. These handles can be accessed using the following defined constants as indices to the returned pointer:

- BLKCTRL
- LAYERCTRL
- SHPFILECTRL
- LTYPECTRL
- VIEWCTRL
- UCSCTRL
- VPORTCTRL
- REGAPPCTRL
- DIMSTYLECTRL
- VPENTCTRL

**Return Value**

As described above.

**See Also**

- `adShadowObjhandlePtr`

**Example**

```c
AD_OBJHANDLE *ohptr;

ohptr=adCtrlObjhandlePtr(handle);
/* now the blockcontrolobjhandle, for instance, is at:
   ohptr[BLKCTRL]; */
```
adDecodeAcadDate

Summary

    void   adDecodeAcadDate (long *longs, short *day, short *month, short year,
                           short *hour, short *mins, short *sec);

Description

    Decodes an AutoCAD encoded date into day, month, year, hour, minute, and second. Use this function to
decode the header variables tdcreate (date and time of file creation) and tdupdate (date and time of last update).

Return Value

    There is no return value; however the passed-in variables day, month, year, hour, mins, and sec are set to the
appropriate values.

See Also

    adDecodeAcadTime, adEncodeAcadDate, adEncodeAcadTime

Example

    adhd=adHeaderPointer(dwghandle);
    adDecodeAcadDate(adhd->tdcreate,
                     &day,&month,&year,&hour,&mins,&sec);
    printf("Created: %02d/%.02d/%d, %.02d:%.02d:%.02d\n",
           month,day,year,hour,mins,sec);

adDecodeAcadTime

Summary

    void   adDecodeAcadTime (long *longs, short *days, short *hour,
                           short *mins, short *sec);

Description

    Decodes an AutoCAD encoded time into days, hours, minutes, and seconds. Use this function to decode
the header variables tdindwg (time in file) and tusrtimer (user timer).

Return Value

    There is no return value; however the passed in variables days, hour, mins, and sec are set to the
appropriate values.

See Also

    adDecodeAcadDate, adEncodeAcadDate, adEncodeAcadTime

Example

    adhd=adHeaderPointer(dwghandle);
adDecodeAcadTime(adhd->tdindwg,&day,&hour,&mins,&sec);
printf("Total time: %d days, %d hours, %d minutes, %d seconds\n",
day,hour,mins,sec);

adDefaultExamineFile

Summary

void adDefaultExamineFile (char *oldfile, char *newfile);

Description

This is the default OpenDWG Toolkit examine file function. This function takes the file passed in oldfile and searches for it, both in the current directory and (if no path is specified in oldfile) in all directories pointed to by the ACAD environment variable, if any.

Return Value

There is no return value. On return, newfile contains the located file, if any.

See Also

None

Example

char findloc[128];
adDefaultExamineFile("simplex.shx",findloc);

adDeleteBlob

Summary

short adDeleteBlob (AD_VMADDR blob);

Description

Deletes a blob from virtual memory.

Return Value

Returns 1 on success, 0 on failure. On failure, adError() should be checked for an indication of the cause of the failure.
See Also

adCreateBlob

Example

AD_VMADDR theblob;
PAD_BLOB_CTRL bcptr;

theblob=adCreateBlob();
bcptr=adStartBlobWrite();
...
adEndBlobWrite();
...
adDeleteBlob(theblob);

adDeleteBlockheader

Summary

short adDeleteBlockheader (AD_DB_HANDLE handle, AD_OBJHANDLE blhhandle);

Description

Deletes block header the blockheader with AD_OBJHANDLE blhhandle from the block header list. Note that adDeleteBlockheader does NOT delete the entities associated with a block; you should delete them separately if you wish to delete a block.

Return Value

Returns 1 on success, 0 on failure. On failure, adError() should be checked for an indication of the cause of the failure.

See Also

adAddBlockheader, adNumBlockheaders, adGetBlockheader, adSeekBlockheader

Example

adDeleteBlockheader(blkh.objhandle);

adDeleteBmp

Summary

short adDeleteBmp (AD_DB_HANDLE handle);

Description

Deletes a BMP image from the drawing file indicated by handle.

Return Value
Returns 1 on success, 0 on failure. On failure, `adError()` should be checked for an indication of the cause of the failure.

See Also

`adAddBmp, adExtractBmpToFile`

Example

```c
adDeleteBmp(handle);
```

### adDeleteClass

**Summary**

```c
short adDeleteClass (AD_DB_HANDLE handle, short classnum);
```

**Description**

Deletes the class whose classnum is `classnum` from the class list.

**Return Value**

Returns 1 on success, 0 on failure. On failure, `adError()` should be checked for an indication of the cause of the failure.

See Also

`adAddClass, adNumClasses, adGetClass`

Example

```c
adDeleteClass(clss.classnum);
```

### adDeleteDimstyle

**Summary**

```c
short adDeleteDimstyle (AD_DB_HANDLE handle, AD_OBJHANDLE dimhandle);
```

**Description**

Deletes the dimension style whose AD_OBJHANDLE is `dimhandle` from the dimstyle list.

**Return Value**

Returns 1 on success, 0 on failure. On failure, `adError()` should be checked for an indication of the cause of the failure.

See Also
adAddDimstyle, adReplaceDimstyle, adNumDimstyles, adGetDimstyle, adSeekDimstyle

Example

adDeleteDimstyle(dimst.objhandle);

adDeleteEntity

Summary

short adDeleteEntity (AD_DB_HANDLE handle, AD_VMADDR list, AD_OBJHANDLE enthandle, short deleteblobs);

Description

Deletes the entity whose AD_OBJHANDLE is enthandle from the list indicated by list. The deleteblobs parameter indicates whether any blobs associated with this entity should be deleted, and should be set to one of the defined constants AD_DELETE_BLOBS or AD_LEAVE_BLOBS. For normal deletion blobs should be deleted. Blobs should be kept if you have retrieved the entity with adGetEntity() and are planning on using it further; in this case, presumably, you would like the blobs to continue to be valid.

Return Value

Returns 1 on success, 0 on failure. On failure, adError() should be checked for an indication of the cause of the failure.

See Also

adAddEntity, adSeekEntity, adNumEntities, adGetEntity, adReplaceEntity

Example

adDeleteEntity(handle,thelist,adenhd->enthandle);

adDeleteLayer

Summary

short adDeleteLayer (AD_DB_HANDLE handle, AD_OBJHANDLE layhandle);

Description

Deletes the layer whose AD_OBJHANDLE is layhandle from the layer list.

Return Value

Returns 1 on success, 0 on failure. On failure, adError() should be checked for an indication of the cause of the failure.

See Also
adAddLayer, adReplaceLayer, adSeekLayer, adNumLayers, adGetLayer

Example

adDeleteLayer(lay.objhandle);

adDeleteLinetype

Summary

short adDeleteLinetype (AD_DB_HANDLE handle, AD_OBJHANDLE ltphandle);

Description

Deletes the linetype whose AD_OBJHANDLE is ltphandle from the linetype list.

Return Value

Returns 1 on success, 0 on failure. On failure, adError() should be checked for an indication of the cause of the failure.

See Also

adAddLinetype, adReplaceLinetype, adSeekLinetype, adNumLinetypes, adGetLinetype

Example

adDeleteLinetype(ltype.objhandle);

adDeleteObject

Summary

short adDeleteObject (AD_DB_HANDLE handle, AD_OBJHANDLE objhandle);

Description

Deletes the object whose AD_OBJHANDLE is objhandle from the object list.

Return Value

Returns 1 on success, 0 on failure. On failure, adError() should be checked for an indication of the cause of the failure.

See Also

adAddDicObject, adAddMlstyle, adAddProxyObject

Example

adDeleteObject(dic.objhandle);
adDeleteRegapp

Summary

short adDeleteRegapp (AD_DB_HANDLE handle, AD_OBJHANDLE apphandle);

Description

Deletes the registered application whose AD_OBJHANDLE is apphandle from the regapp list.

Return Value

Returns 1 on success, 0 on failure. On failure, adError() should be checked for an indication of the cause of the failure.

See Also

adAddRegapp, adReplaceRegapp, adSeekRegapp, adNumRegapps, adGetRegapp

Example

adDeleteRegapp(app.objhandle);

adDeleteShapefile

Summary

short adDeleteShapefile (AD_DB_HANDLE handle, AD_OBJHANDLE shchandle);

Description

Deletes the shapefile (style) entry whose AD_OBJHANDLE is shchandle from the shapefile list.

Return Value

Returns 1 on success, 0 on failure. On failure, adError() should be checked for an indication of the cause of the failure.

See Also

adAddShapefile, adReplaceShapefile, adSeekShapefile, adNumShapefiles, adGetShapefile

Example

adDeleteShapefile(shptb.objhandle);
adDeleteUcs

Summary

short adDeleteUcs (AD_DB_HANDLE handle, AD_OBJHANDLE ucshandle);

Description

Deletes the UCS whose AD_OBJHANDLE is ucshandle from the UCS list.

Return Value

Returns 1 on success, 0 on failure. On failure, adError() should be checked for an indication of the cause of the failure.

See Also

adAddUcs, adReplaceUcs, adSeekUcs, adNumUcs, adGetUcs

Example

adDeleteUcs(ucs.objhandle);

adDeleteView

Summary

short adDeleteView (AD_DB_HANDLE handle, AD_OBJHANDLE viewhandle);

Description

Deletes the view whose AD_OBJHANDLE is viewhandle from the view list.

Return Value

Returns 1 on success, 0 on failure. On failure, adError() should be checked for an indication of the cause of the failure.

See Also

adAddView, adReplaceView, adSeekView, adNumViews, adGetView

Example

adDeleteView(view.objhandle);

adDeleteVport

Summary

short adDeleteVport (AD_DB_HANDLE handle, AD_OBJHANDLE vpohandle);
Description

Deletes the vport whose AD_OBJHANDLE is `vpohandle` from the vport list.

Return Value

Returns 1 on success, 0 on failure. On failure, `adError()` should be checked for an indication of the cause of the failure.

See Also

`adAddVport`, `adReplaceVport`, `adSeekVport`, `adNumVports`, `adGetVport`

Example

```c
adDeleteVport(vport.objhandle);
```

### adDeleteWmf

Summary

```c
short adDeleteWmf (AD_DB_HANDLE handle);
```

Description

Deletes a WMF image from the current drawing.

Return Value

Returns 1 on success, 0 on failure. On failure, `adError()` should be checked for an indication of the cause of the failure.

See Also

`adAddWmf`, `adFileHasWmf`

Example

```c
adDeleteWmf(handle);
```

### adDictionaryvarObjtype

Summary

```c
short adDictionaryvarObjtype(AD_DB_HANDLE handle);
```

Description

Returns the object type for a DICTIONARYVAR object.
Return Value

Returns the object type for a DICTIONARYVAR object.

See Also

None

Example

    adobhd.objtype=adDictionaryvarObjtype(handle);

adDictionarywdfltObjtype

Summary

short adDictionarywdfltObjtype (AD_DB_HANDLE handle);

Description

Returns the object type value for AD_DICTIONARYWDFLT objects.

Return Value

Returns the object type value for AD_DICTIONARYWDFLT objects.

See Also

None

Example

    adobhd.objtype=adDictionarywdfltObjtype(handle);

adDwgSmellsBad

Summary

short adDwgSmellsBad(void *path);

Description

Attempts to determine whether a DWG file is corrupt. Note that this function will only actually do the checking for files from AutoCAD R11 and higher.

Return Value

Returns 0 if no errors are found. Returns -1 if the file to be smelled is prior to R11. Returns 1 if errors are found in the file. If errors are found, adError() may be checked for an indication of the type of error.

See Also
Example

```c
retval=adDwgSmellsBad("c:\test.dwg");
if (retval==-1) printf("file is pre-R11\n");
else if (retval==0) printf("file is OK\n");
else printf("errors found: %s\n", adErrorStr(adError()));
```

adDxfQualityFlags

**Summary**

```c
long adDxfQualityFlags(AD_DB_HANDLE handle);
```

**Description**

This function gives information about which sections of a DXF file were found by OpenDWG Toolkit while reading it.

You can take the return value from this function and AND it with any of the following defined constants to determine whether or not that section of the DXF file was found by ODT.

- AD_DXF_HEADER_PRESENT
- AD_DXF_LTYPE_TABLE_PRESENT
- AD_DXF_LAYER_TABLE_PRESENT
- AD_DXF_SHAPEFILE_TABLE_PRESENT
- AD_DXF_VIEW_TABLE_PRESENT
- AD_DXF_VPORT_TABLE_PRESENT
- AD_DXF_REGAPP_TABLE_PRESENT
- AD_DXF_DIMSTYLE_TABLE_PRESENT
- AD_DXF_UCS_TABLE_PRESENT
- AD_DXF_BLKREC_TABLE_PRESENT

Note that while it is not necessarily the case, by any means, that files missing one or more of these section are bad, and indeed ODT tries to load DXF files which are very incomplete, it is certainly possible that odd results may occur when the header, layer, linetype, or shapefile sections have been omitted from the DXF file. Also note that certain sections (registered applications, for instance), did not exist in older versions of AutoCAD, so their absence should not be considered an error.

**Return Value**

Returns set of flags as a long integer.

**See Also**

None

**Example**

```c
if (adDxfQualityFlags(handle) & AD_DXF_LAYER_TABLE_PRESENT)
    printf("layer table present\n");
else printf("layer table not present\n");
```
adEncodeAcadDate

Summary

void adEncodeAcadDate (short day, short month, short year, short hour, short mins, short sec, long *longs);

Description

Encodes a day, month, year, hour, minute, and second into an AutoCAD date. Use this function to encode the header variables \texttt{tdcreate} (date and time of file creation) and \texttt{tdupdate} (date and time of last update).

Return Value

There is no return value; however \texttt{longs} is set to the appropriate values.

See Also

adDecodeAcadTime, adDecodeAcadDate, adEncodeAcadTime

Example

adEncodeAcadDate(day,month,year,hour,mins,sec,adhd.tdcreate);

adEncodeAcadTime

Summary

void adEncodeAcadTime (short days, short hour, short mins, short sec, long *longs);

Description

Encodes days, hours, minutes, and seconds into an encoded AutoCAD time. Use this function to encode the header variables \texttt{tdindwg} (time in file) and \texttt{tdusrtimer} (user timer).

Return Value

There is no return value; however \texttt{longs} is set to the appropriate values.

See Also

adDecodeAcadDate, adEncodeAcadDate, adDecodeAcadTime

Example

adEncodeAcadTime(&day,&hour,&mins,&sec,adhd.tdindwg);
adEndBlobAppend

Summary

short adEndBlobAppend (PAD_BLOB_CTRL bcptr);

Description

Terminates append to a BLOB, and frees the temporary memory buffer.

Return Value

Returns 1 on success, 0 on failure. On failure, \texttt{adError()} should be checked for an indication of the cause of the failure.

See Also

\texttt{adCreateBlob}, \texttt{adStartBlobAppend}

Example

AD_VMADDR theblob;
PAD_BLOB_CTRL bcptr;

\begin{verbatim}
bcptr=adStartBlobAppend(theblob);
\end{verbatim}

adEndBlobAppend(bcptr);

adEndBlobRead

Summary

short adEndBlobRead (PAD_BLOB_CTRL bcptr);

Description

Terminates read of a BLOB, and frees the temporary memory buffer.

Return Value

Returns 1 on success, 0 on failure. On failure, \texttt{adError()} should be checked for an indication of the cause of the failure.

See Also

\texttt{adCreateBlob}, \texttt{adStartBlobRead}
Example

AD_VMADDR theblob;
PAD_BLOB_CTRL bcptr;

. . .
bcptr=adStartBlobRead(theblob);
. . .
adEndBlobRead(bcptr);

adEndBlobWrite

Summary

short adEndBlobWrite (PAD_BLOB_CTRL bcptr);

Description

Terminates write of a BLOB, and frees the temporary memory buffer.

Return Value

Returns 1 on success, 0 on failure. On failure, adError() should be checked for an indication of the cause of the failure.

See Also

adStartBlobWrite, adWriteBlobBytes

Example

AD_VMADDR theblob;
PAD_BLOB_CTRL bcptr;

theblob=adCreateBlob();
bcptr=adStartBlobWrite(theblob);
. . .
adEndBlobWrite(bcptr);

adEndProxyDataRead

Summary

short adEndProxyDataRead (AD_DB_HANDLE handle);

Description

Ends read of proxy (object or entity) data.
Return Value

Returns 1 on success, 0 on failure. On failure, adError() should be checked for an indication of the cause of the failure.

See Also

adStartProxyDataRead, adReadProxyDataXXXX

Example

adEndProxyDataRead(handle);

adEndProxyDataWrite

Summary

short adEndProxyDataWrite (AD_DB_HANDLE handle);

Description

Ends write of proxy (entity or object) data.

Return Value

Returns 1 on success, 0 on failure. On failure, adError() should be checked for an indication of the cause of the failure.

See Also

adStartProxyDataWrite, adWriteProxyDataXXXX

Example

adEndProxyDataWrite(handle);

adEntityList

Summary

AD_VMADDR adEntityList (AD_DB_HANDLE handle, AD_OBJHANDLE whichblk);

Description

Returns the AD_VMADDR of the entity list associated with the block header whose handle is whichblk.

Return Value

As described above. AD_VMNULL is returned if there is no entity list or there is an error. On error, adError() should be checked for an indication of the cause of the failure.
See Also

    adGetEntity

Example

    entlist=adEntityList(handle,blkh.objhandle);

adError

Summary

    short adError(void);

Description

    Returns a value indicating the type of error last encountered by AD.

Return Value

    Returns the value of the last error. This value is an index into the ader enum defined in ad2.h. While it is possible to simply look through those values, it is easier to use adErrorStr() (below) to get an English-language sentence indicating the problem.

See Also

    None

Example

    printf("error was %s\n",adErrorStr(adError()));

adErrorClear

Summary

    void adErrorClear(void);

Description

    Sets the OpenDWG Toolkit internal "last error" value to 0 (AD_NO_ERROR).

Return Value

    There is no return value.
See Also

adError(), adErrorStr()

Example

printf("error was %s\n",adErrorStr(adError()));
adErrorClear();

adErrorObjhandle

Summary

void adErrorObjhandle (AD_OBJHANDLE errhandle);

Description

Returns the AD_OBJHANDLE of the object for which an error was found. Note that this is implemented for only a small number of errors, chiefly attempts to add an object with a duplicate handle (AD_ATTEMPT_TO_ADD_DUPLICATE_HANDLE).

Return Value

There is no return value.

See Also

adError(), adErrorStr()

Example

char *buf;
AD_OBJHANDLE temphandle;

printf("error was %s\n",adErrorStr(adError()));
adErrorObjhandle(temphandle);
adBinaryObjhandleToAscii(temphandle,buf);
printf("handle: %s\n",buf);

adErrorStr

Summary

char *adErrorStr (short aderror);

Description

Returns the English-language description of the error whose number is aderror.
Return Value

Returns a pointer to the description.

See Also

None

Example

    printf("error was %s\n", adErrorStr(adError()));

adExtractBmpFromFileToBuffer

Summary

    short adExtractBmpFromFileToBuffer (void *source, void *buffer, unsigned short *bytes);

Description

    Extracts the BMP image, if any, from a disk file to a memory buffer. buffer points to the in-memory buffer. bytes receives the number of bytes placed in the buffer. You can get the size of the required buffer by using adBmpSizeFromFile().

Return Value

    Returns 1 on success, 0 on failure. On failure, adError() should be checked for an indication of the cause of the failure.

See Also

    adAddBmp, adFileHasBmp, adDeleteBmp

Example

    char *mybuf;
    unsigned short numbytes;
    numbytes = adBmpSizeFromFile("c:\test.dwg");
    if (numbytes) {
        mybuf = malloc(numbytes);
        adExtractBmpFromFileToBuffer("c:\test.dwg", mybuf, &numbytes);
    }

adExtractBmpFromFileToFile

Summary

    short adExtractBmpFromFileToFile (void *source, void *fn);
Description

Extracts the BMP image, if any, from the source to a file.

Return Value

Returns 1 on success, 0 on failure. On failure, adError() should be checked for an indication of the cause of the failure.

See Also

adAddBmp, adFileHasBmp, adDeleteBmp

Example

adExtractBmpFromFileToFile("c:\\test.dwg","c:\\test.bmp");

adExtractBmpToBuffer

Summary

short adExtractBmpToBuffer (AD_DB_HANDLE handle, void *buffer, unsigned short *bytes);

Description

Extracts the BMP image, if any, from a loaded file to a memory buffer. handle is the file handle. buffer points to the in-memory buffer. bytes receives the number of bytes placed in the buffer. You can use adBmpSize() to determine the size of the buffer necessary.

Return Value

Returns 1 on success, 0 on failure. On failure, adError() should be checked for an indication of the cause of the failure.

See Also

adBmpSize, adAddBmp, adFileHasBmp, adDeleteBmp

Example

char mybuf[65520];
unsigned short numbytes;
adExtractBmpToBuffer("c:\\test.dwg",mybuf,&numbytes);

adExtractBmpToFile

Summary

short adExtractBmpToFile (AD_DB_HANDLE handle,AD_FILE_DESIGNATOR fn);
Description

Extracts the BMP image from the file indicated by handle to a file. AD_FILE_DESIGNATOR is char * for most platforms, FSSpec * for Macintosh.

Return Value

Returns 1 on success, 0 on failure. On failure, adError() should be checked for an indication of the cause of the failure.

See Also

adAddBmp, adFileHasBmp, adDeleteBmp

Example

adExtractBmpToFile(handle,"c:\test.bmp");

adFileDisplayableInPaperspace

Summary

short adFileDisplayableInPaperspace (AD_DB_HANDLE handle);

Description

Returns 1 if this file can be displayed in a paperspace view.

Return Value

Returns 1 on success, 0 on failure. On failure, adError() should be checked for an indication of the cause of the failure.

See Also

adFileSavedInModelSpace

Example

yesitcan=adFileDisplayableInPaperspace(handle);

adFileHasBmp

Summary

short adFileHasBmp (AD_DB_HANDLE handle);

Description

Returns 1 if this drawing file has a BMP raster image attached to it, otherwise 0.
Return Value

As described above.

See Also

adFileHasWmf, adAddBmp

Example

hasbmp=adFileHasBmp(handle);

adFileHasWmf

Summary

short adFileHasWmf (AD_DB_HANDLE handle);

Description

Returns 1 if this drawing file has a WMF image attached to it, otherwise 0.

Return Value

As described above.

See Also

adFileHasBmp, adAddWmf

Example

haswmf=adFileHasWmf(handle);

adFileSavedInModelSpace

Summary

short adFileSavedInModelSpace (AD_DB_HANDLE handle, short tilemode);

Description

Returns 1 if the file was saved with a modelspace view, 0 if a paperspace view.

Return Value

As described above.
See Also

adFileDisplayableInPaperspace

Example

PAD_DWGHDR adhd;
adh=adHeaderPointer(handle);
inms=adFileSavedInModelSpace(handle, adhd->tilemode);

adFileTypeLoaded

Summary

short adFileTypeLoaded (AD_DB_HANDLE handle);

Description

Indicates the type of file which was loaded when handle handle was returned.

Return Value

Returns one of the defined constants AD_DWG, AD_DXF or AD_BDXF.

See Also

adFileVersionLoaded

Example

filetype=adFileTypeLoaded(handle);

adFileVersionLoaded

Summary

short adFileVersionLoaded (AD_DB_HANDLE handle);

Description

Indicates the version of the loaded file whose identifier is handle.

Return Value

Returns one of the defined constants AD_ACAD25, AD_ACAD26, AD_ACAD9, AD_ACAD10, AD_ACAD11, AD_ACAD13, or AD_ACAD14.

See Also

adFileTypeLoaded
Example

```
    fileversion=adFileVersionLoaded(handle);
```

### adFindBlockheaderByName

**Summary**

```
short adFindBlockheaderByName(AD_DB_HANDLE handle, char *name, AD_OBJHANDLE thehandle);
```

**Description**

Returns the object handle of the blockheader whose name is *name* in *thehandle*.

**Return Value**

Returns 1 on success, 0 on failure.

**See Also**

- adSeekBlockheader

**Example**

```
    success=adFindBlockheaderByName(handle,"MYBLOCK", thehandle);
```

### adFindDimstyleByName

**Summary**

```
short adFindDimstyleByName(AD_DB_HANDLE handle, char *name, AD_OBJHANDLE thehandle);
```

**Description**

Returns the object handle of the dimension style whose name is *name* in *thehandle*.

**Return Value**

Returns 1 on success, 0 on failure.

**See Also**

- adSeekDimstyle

**Example**

```
    success=adFindDimstyleByName(handle,"STANDARD", thehandle);
```
**adFindLayerByName**

Summary

```c
short adFindLayerByName(AD_DB_HANDLE handle, char *name, AD_OBJHANDLE thehandle);
```

Description

Returns the object handle of the layer whose name is `name` in `thehandle`.

Return Value

Returns 1 on success, 0 on failure.

See Also

- adSeekLayer

Example

```c
success=adFindLayerByName(handle,"0",thehandle);
```

**adFindLinetypeByName**

Summary

```c
short adFindLinetypeByName(AD_DB_HANDLE handle, char *name, AD_OBJHANDLE thehandle);
```

Description

Returns the object handle of the linetype whose name is `name` in `thehandle`.

Return Value

Returns 1 on success, 0 on failure.

See Also

- adSeekLinetype

Example

```c
success=adFindLinetypeByName(handle,"CONTINUOUS",thehandle);
```
adFindRegappByName

Summary

short adFindRegappByName(AD_DB_HANDLE handle, char *name, AD_OBJHANDLE thehandle);

Description

Returns the object handle of the registered application whose name is name in thehandle.

Return Value

Returns 1 on success, 0 on failure.

See Also

adSeekRegapp

Example

success=adFindRegappByName(handle,"ACAD", thehandle);

adFindShapefileByName

Summary

short adFindShapefileByName(AD_DB_HANDLE handle, char *name, AD_OBJHANDLE thehandle);

Description

Returns the object handle of the shapefile whose name is name in thehandle.

Return Value

Returns 1 on success, 0 on failure.

See Also

adSeekShapefile

Example

success=adFindShapefileByName(handle,"STANDARD", thehandle);
adFindUcsByName

Summary

    short adFindUcsByName(AD_DB_HANDLE handle, char *name, AD_OBJHANDLE thehandle);

Description

    Returns the object handle of the UCS whose name is **name** in **thehandle**.

Return Value

    Returns 1 on success, 0 on failure.

See Also

    adSeekUcs

Example

    success=adFindUcsByName(handle,"MYUCS",thehandle);

adFindViewByName

Summary

    short adFindViewByName(AD_DB_HANDLE handle, char *name, AD_OBJHANDLE thehandle);

Description

    Returns the object handle of the view whose name is **name** in **thehandle**.

Return Value

    Returns 1 on success, 0 on failure.

See Also

    adSeekView

Example

    success=adFindViewByName(handle,"STANDARD",thehandle);
adGenerateObjHandle

Summary

void adGenerateObjHandle (AD_DB_HANDLE handle, unsigned char *newhandle);

Description

Generates a new AD_OBJHANDLE to be used to identify an entry in the drawing file. This function should be used whenever a new object which has an AD_OBJHANDLE (which is almost everything in the file) is added to the database. The new AD_OBJHANDLE is the identifier for that item, and must be unique. adGenerateObjHandle generates the "next" handle by incrementing from the "last" one, guaranteeing uniqueness.

Return Value

There is no return value.

See Also

None

Example

AD_OBJHANDLE thehandle;
adGenerateObjHandle(handle, thehandle);

adGetBlockHandle

Summary

short adGetBlockheader (AD_DB_HANDLE handle, AD_OBJHANDLE handle, short type);

Description

Retrieves the block handle for either the *Model_Space block (pass AD_MODELSPACE_HANDLE for type), or the *Paper_Space block (pass AD_PAPERSPACE_HANDLE for type).

Return Value

Returns 1 on success, 0 on failure.

See Also

None.
Example

```c
if (adGetBlockHandle(dwghandle, handle, AD_MODELSPACE_HANDLE))
{
  ...
}
```

---

**adGetBlockheader**

**Summary**

```c
short adGetBlockheader (AD_DB_HANDLE handle, PAD_BLK adblkh);
```

**Description**

Retrieves the next blockheader in the blockheader list, storing it in the buffer pointed to by `adblkh`.

**Return Value**

Returns 1 on success, 0 on failure. On failure, `adError()` should be checked for an indication of the cause of the failure.

**See Also**

`adStartBlockheaderGet, adDeleteBlockheader, adSeekBlockheader, adNumBlockheaders`

**Example**

```c
for (i=0; i<(short)adNumBlockheaders(dwghandle); i++) {
  adGetBlockheader (dwghandle,&adtb->blkh);
  if (!adtb->blkh.purgedflag) {
    printf(" %s\n",adtb->blkh.name);
    if (adtb->blkh.xdblob!=AD_VMNULL)
      printfdblob(adtb->blkh.xdblob);
  }
}
```

---

**adGetClass**

**Summary**

```c
short adGetClass (AD_DB_HANDLE handle, PAD_CLASS clss);
```

**Description**

Retrieves the next class entry in the class list, storing it in the buffer pointed to by `clss`.

**Return Value**

Returns 1 on success, 0 on failure. On failure, `adError()` should be checked for an indication of the cause of the failure.

**See Also**
adStartClassGet, adDeleteClass, adNumClasses

Example

AD_CLASS cls;

adStartClassGet(dwghandle);
printf("there are %ld classes\n",adNumClasses(dwghandle));
for (i=0; i<adNumClasses(dwghandle); i++) {
adGetClass(dwghandle,&cls);
printf("class: Classdxfname %s\n",cls.classdxfname);
printf(" C++clsname %s\n",cls.cplusplusclassname);
printf(" Appname %s\n",cls.appname);
}

adGetDimstyle

Summary

short adGetDimstyle (AD_DB_HANDLE handle, PAD_DIMSTYLE addimst);

Description

Retrieves the next dimension style from the dimstyle list, storing it in the buffer pointed to by addimst.

Return Value

Returns 1 on success, 0 on failure. On failure, adError() should be checked for an indication of the cause of the failure.

See Also

adStartDimstyleGet, adDeleteDimstyle, adReplaceDimstyle, adNumDimstyles, adSeekDimstyle

Example

printf("\nDIMSTYLES:\n");
adStartDimstyleGet(dwghandle);
for (i=0; i<(short)adNumDimstyles(dwghandle); i++) {
adGetDimstyle(dwghandle,&adtdimst);
if (!adtdimst.purgedflag) {
    printf(" %s\n",adtdimst.name);
    if (adtdimst.xdblob!=AD_VMNULL)
        printxdblob(adtdimst.xdblob);
}
}

adGetEntity

Summary

short adGetEntity (AD_VMADDR list,PAD_ENT_HDR adenhd,PAD_ENT aden);

Description
Retrieves the next entity from the list at AD_VMADDR list, storing the header in the buffer pointed to by adenhd and the body in the buffer pointed to by aden.

Return Value

Returns 1 on success, 0 on failure. On failure, adError() should be checked for an indication of the cause of the failure.

See Also

adStartEntityGet, adDeleteEntity, adReplaceEntity, adNumEntities, adSeekEntity

Example

```c
printf("\nBLOCKS:\n");
adStartBlockheaderGet(dwghandle);
/* note -- -2 so we don't get the model and paperspace blocks */
for (i=0; i<(short)adNumBlockheaders(dwghandle)-2; i++) {
adGetBlockheader (dwghandle,&adtb->blkh);
printf("BLOCK: %s\n",adtb->blkh.name);
if (!adtb->blkh.purgedflag) {
adStartEntityGet(adtb->blkh.entitylist);
do {
    returnval=adGetEntity(adtb->blkh.entitylist,adenhd,aden);
    if (returnval) {
        if (adenhd->xdblob!=AD_VMNULL)
            printxdblob(adenhd->xdblob);
        if (!adtb->blkh.purgedflag)
            printentity(adenhd,aden);
        readremainderofentity(adenhd,aden);
    }
} while (returnval && adenhd->enttype!=AD_ENT_ENDBLK);
}
```

adGetLayer

Summary

short adGetLayer (AD_DB_HANDLE handle, PAD_LAY adlay);

Description

Retrieves the next layer entry from the layer list, storing it in the buffer pointed to by adlay.

Return Value

Returns 1 on success, 0 on failure. On failure, adError() should be checked for an indication of the cause of the failure.

See Also

adStartLayerGet, adDeleteLayer, adReplaceLayer, adNumLayers, adSeekLayer

Example
printf("\nLAYERS:\n");
adStartLayerGet(dwghandle);
for (i=0; i<(short)adNumLayers(dwghandle); i++) {
adGetLayer(dwghandle,&adtb->layer);
    if (!adtb->layer.purgedflag) {
        printf(" %s\n",adtb->layer.name);
        if (adtb->layer.xdblob!=AD_VMNULL)
            printxDBlob(adtb->layer.xdblob);
    }
}

adGetLayerState

Summary

    short adGetLayerState (AD_DB_HANDLE handle, AD_OBJHANDLE layhandle, char *on, char *frozen, char *vpfrozen, char *locked);

Description

    Gets the current state of the layer indicated by layhandle.

Return Value

    Returns 1 on success, 0 on failure. On failure, adError() should be checked for an indication of the cause of the failure. On success, on is set to 1 if the layer is on, otherwise 0. Frozen is set to 1 if the layer is frozen, otherwise 0. Vpfrozen is set to 1 if the layer is to be frozen in new vports, otherwise 0. Locked is set to 1 if the layer is locked, otherwise 0.

See Also

    adSetLayerState

Example

    adGetLayerState(handle,layhandle,&on,&frozen,&vpfrozen,&locked);
    printf("layer is ");
    if (on) printf("on and ");
    else printf("off and ");
    if (frozen) printf("frozen\n");
    else printf("thawed\n");
adGetLinetype

Summary

short adGetLinetype (AD_DB_HANDLE handle, PAD_LTYPE adltype);

Description

Retrieves the next linetype from the linetype list, and stores it in the buffer pointed to by adltype.

Return Value

Returns 1 on success, 0 on failure. On failure, adError() should be checked for an indication of the cause of the failure.

See Also

adStartLinetypeGet, adDeleteLinetype, adReplaceLinetype, adNumLinetypes, adSeekLinetype

Example

printf("LINETYPES:\n");
adStartLinetypeGet(dwghandle);
for (i=0; i<(short)adNumLinetypes(dwghandle); i++) {
adGetLinetype(dwghandle,&adltypedebug);
if (!adltypedebug.purgedflag) {
printf(" %s\n",adltypedebug.name);
if (adltypedebug.xdblob!=AD_VMNULL)
printxdblob(adltypedebug.xdblob);
}
}

adGetObject

Summary

short adGetObject (AD_DB_HANDLE handle, PAD_OBJ_HDR adobhd, PAD_OBJ adob);

Description

Retrieves the next object from the object list, and stores it in the buffers pointed to by adobhd and adob.

Return Value

Returns 1 on success, 0 on failure. On failure, adError() should be checked for an indication of the cause of the failure.
See Also

adStartObject Get, adDeleteObject, adReplaceObject, adSeekObject

Example

```c
adStartObjectGet(dwghandle);
while (adGetObject(dwghandle,&adobhd,&adob)) {
  if (adobhd.objtype==AD_OBJ_DICTIONARY)
    exampleprintf("DICTIONARY OBJECT\n");
  else if (adobhd.objtype==AD_OBJ_MLINESTYLE)
    exampleprintf("MLSTYLE OBJECT\n");
  else if (adobhd.objtype==AD_OBJ_GROUP)
    exampleprintf("GROUP OBJECT\n");
  else { /* others are proxies */
    if (adobhd.objtype==adXrecordObjtype(dwghandle))
      exampleprintf("XRECORD OBJECT\n");
    else if (adobhd.objtype==adIdbufferObjtype(dwghandle))
      exampleprintf("IDBUFFER OBJECT\n");
    else if (adobhd.objtype==adDictionaryvarObjtype(dwghandle))
      exampleprintf("DICTIONARYVAR OBJECT\n");
    else if (adobhd.objtype==adImagedefreactorObjtype(dwghandle))
      exampleprintf("IMAGEDEF_REACTOR\n");
    else if (adobhd.objtype==adImagedefObjtype(dwghandle))
      exampleprintf("IMAGEDEF_REACTOR\n");
    else if (adobhd.objtype==adRastervariablesObjtype(dwghandle))
      exampleprintf("RASTERVARIABLES\n");
    else if (adobhd.objtype==adSpatialfilterObjtype(dwghandle))
      exampleprintf("SPATIAL_FILTER OBJECT\n");
    else if (AD_IS_A_GENERIC_PROXYOBJ(adobhd.objtype))
      exampleprintf("GENERIC PROXY OBJECT\n");
  }
}
```

adGetRegapp

Summary

short adGetRegapp (AD_DB_HANDLE handle, PAD_APP adapp);

Description

Retrieves the next registered application entry from the regapp list, storing it in the buffer pointed to by adapp.

Return Value

Returns 1 on success, 0 on failure. On failure, adError() should be checked for an indication of the cause of the failure.

See Also

adStartRegappGet, adDeleteRegapp, adReplaceRegapp, adNumRegapps, adSeekRegapp

Example

```c
printf("\nREGAPPS:\n");
adStartRegappGet(dwghandle);
for (i=0; i<(short)adNumRegapps(dwghandle); i++) {
  adGetRegapp(dwghandle,&adtb->app);
  if (!adtb->app.purgedflag) {
```
printf(" %s\n", adtb->app.name);
if (adtb->app.xdblob!=AD_VMNULL)
    printxdblob(adtb->app.xdblob);
}
}

adGetShapefile

Summary

short adGetShapefile (AD_DB_HANDLE handle, PAD_SHPTB adshptb);

Description

Retrieves the next shapefile (style) entry from the shapefile list, storing it in the buffer pointed to by adshptb.

Return Value

Returns 1 on success, 0 on failure. On failure, adError() should be checked for an indication of the cause of the failure.

See Also

adStartShapefileGet, adDeleteShapefile, adReplaceShapefile, adNumShapefiles, adSeekShapefile

Example

printf("\nSHAPE FILES:\n");
adStartShapefileGet(dwghandle);
for (i=0; i<(short)adNumShapefiles(dwghandle); i++) {
adGetShapefile(dwghandle,&adtb->shptb);
    if (!adtb->shptb.purgedflag) {
        printf(" %15s ", adtb->shptb.name);
        printf(" %s\n", adtb->shptb.file);
        if (adtb->shptb.xdblob!=AD_VMNULL)
            printxdblob(adtb->shptb.xdblob);
    }
}
**adGetUcs**

**Summary**

short adGetUcs (AD_DB_HANDLE handle, PAD_UCS aducs);

**Description**

Retrieves the next UCS entry from the UCS list, storing the result in the buffer pointed to by `aducs`.

**Return Value**

Returns 1 on success, 0 on failure. On failure, `adError()` should be checked for an indication of the cause of the failure.

**See Also**

`adStartUcsGet, adDeleteUcs, adReplaceUcs, adNumUcs, adSeekUcs`

**Example**

```c
printf("\nUCS:\n");
adStartUcsGet(dwghandle);
for (i=0; i<(short)adNumUcs(dwghandle); i++) {
adGetUcs(dwghandle,&adtb->ucs);
  if (!adtb->ucs.purgedflag) {
    printf(" %s\n",adtb->ucs.name);
    if (adtb->ucs.xdblob!=AD_VMNULL) {
      printxdblob(adtb->ucs.xdblob);
    }
  }
}
```

**adGetView**

**Summary**

short adGetView (AD_DB_HANDLE handle, PAD_VIEW adview);

**Description**

Retrieves the next view from the view list, storing the result in the buffer pointed to by `adview`.

**Return Value**

Returns 1 on success, 0 on failure. On failure, `adError()` should be checked for an indication of the cause of the failure.

**See Also**

`adStartViewGet, adDeleteView, adReplaceView, adNumViews, adSeekView`
Example

printf("\nVIEWS:\n");
adStartViewGet(dwghandle);
for (i=0; i<(short)adNumViews(dwghandle); i++) {
adGetView(dwghandle,&adtb->view);
if (!adtb->view.purgedflag) {
    printf("%s\n",adtb->view.name);
    if (adtb->view.xdblob!=AD_VMNULL)
        printxdblob(adtb->view.Xdblob);
}
}

adGetVport

Summary

short adGetVport (AD_DB_HANDLE handle, PAD_VPORT advport);

Description

Retrieves the next vport from the vport list, storing the result in the buffer pointed to by advport.

Return Value

Returns 1 on success, 0 on failure. On failure, adError() should be checked for an indication of the cause of the failure.

See Also

adStartVportGet, adDeleteVport, adReplaceVport, adNumVports, adSeekVport

Example

printf("\nVPORTS:\n");
adStartVportGet(dwghandle);
for (i=0; i<(short)adNumVports(dwghandle); i++) {
adGetVport(dwghandle,&adtb->vport);
if (!adtb->vport.purgedflag) {
    printf("%s\n",adtb->vport.name);
    if (adtb->vport.xdblob!=AD_VMNULL)
        printxdblob(adtb->vport.Xdblob);
}
}

adHanclear

Summary

adHanclear(AD_OBJHANDLE a);
Description

A macro that sets the AD_OBJHANDLE to all zeroes (the null objhandle).

Return Value

This macro calls memset, and therefore returns a pointer to the objhandle.

See Also

adHancpy, adHancmp

Example

adHanclear(blkh->objhandle);

adHancmp

Summary

adHancmp(AD_OBJHANDLE a, AD_OBJHANDLE b);

Description

A macro that uses memcmp to compare two AD_OBJHANDLES.

Return Value

-1 if a<b, 0 if equal, 1 if a>b.

See Also

adHancpy, adHanclear

Example

if (!adHancmp(blkh->objhandle,blkh2->objhandle)) printf("found\n");

adHancpy

Summary

adHancpy(AD_OBJHANDLE a, AD_OBJHANDLE b);

Description

A macro that copies handle b onto a.
Return Value

This macro calls memcpy, which returns a pointer to b.

See Also

adHanclear, adHancmp

Example

adHancpy(blkh->objhandle,myhandle);

adHasPrimTraits

Summary

adHasPrimTraits(flags);

Description

A macro that determines whether the primitives contained in a proxy entity graphic have traits (color, layer, linetype, marker, visibility, orientation, fillon) assigned to them.

Return Value

This macro returns nonzero if traits are present.

See Also

adPrimsHave..., adReadPrim...

Example

adReadPrimFlags(bcptr,&flags);
if (adHasPrimTraits(flags)) {
   . . .

adHatchEnttype

Summary

short adHatchEnttype(AD_DB_HANDLE handle);

Description

Returns the entity type value for AD_HATCH entities.

Return Value

Returns the entity type value for AD_HATCH entities.

See Also
Example

    hatchenttype=adHatchEnttype(handle);

**adHeaderPointer**

**Summary**

    PAD_DWGHDR adHeaderPointer(AD_DB_HANDLE handle);

**Description**

    Returns a pointer to the file header for a loaded file.

**Return Value**

    Returns a pointer to the file header for a loaded file.

**See Also**

    None

Example

    PAD_DWGHDR adhd;
    adhd=adHeaderPointer(handle);
    printf("min extents are %f,%f\n",adhd->extmin[0],adhd->extmin[1]);

**adIdbufferObjtype**

**Summary**

    short adIdbufferObjtype(AD_DB_HANDLE handle);

**Description**

    Returns the object type value for AD_IDBUFFER objects.

**Return Value**

    Returns the object type value for AD_IDBUFFER objects.

**See Also**

    None
Example

adobhd.objtype=adIdbufferObjtype(handle);

adImageEnttype

Summary

short adImageEnttype(AD_DB_HANDLE handle);

Description

Returns the entity type value for AD_IMAGE entities.

Return Value

Returns the entity type value for AD_IMAGE entities.

See Also

None

Example

adenhd.enttype=adImageEnttype(handle);

adImagedefObjtype

Summary

short adImagedefObjtype(AD_DB_HANDLE handle);

Description

Returns the object type value for AD_IMAGEDEF objects.

Return Value

Returns the object type value for AD_IMAGEDEF objects.

See Also

None

Example

adobhd.objtype=adImagedefObjtype(handle);
adImagedefreactorObjtype

Summary

   short adImagedefreactorObjtype(AD_DB_HANDLE handle);

Description

   Returns the object type value for AD_IMAGEDEFREACTOR objects.

Return Value

   Returns the object type value for AD_IMAGEDEFREACTOR objects.

See Also

   None

Example

   adobhd.objtype=adImagedefreactorObjtype(handle);

adIncrementObjhandle

Summary

   void adIncrementObjhandle(AD_OBJHANDLE objhandle);

Description

   Increases the passed in objhandle by 1. THIS FUNCTION IS NOT TO BE USED TO STEP TO THE NEXT
   HANDLE TO ADD OBJECTS TO DATABASE. Use adGenerateObjhandle() for that purpose.
   adIncrementObjhandle() is to be used in those situations where it is known that several handles were created
   consecutively; it allows you to save the first handle only, and step through the rest using
   adIncrementObjhandle().

Return Value

   There is no return value.

See Also

   None
Example

AD_OBJHANDLE firstobjhandle, temphandle;
short i, num;

adHancpy(temphandle, firstobjhandle);
for (i=0; i<num; i++) {
    /* do something to temphandle here */
    adIncrementObjhandle(temphandle);
}

adInitAd2

Summary

short adInitAd2 (void *initfilepath, short userfns, short *initerror)

Description

Initializes OpenDWG Toolkit. The parameters are:

initfilepath: Path to the adinit.dat file.
userfns: If 1, use user-defined functions to load initialization data.
initerror: Returns the error code if an error is encountered.

The userfns variable is used in conjunction with adSetAd2OpenInitFileFn() and related functions to create an alternate method for loading the ADINIT.DAT file, if desired.

Return Value

Returns 1 on success, 0 on failure. On failure, the value returned in initerror should be checked for an indication of the cause of the failure.

See Also

adCloseAd2

Example

if (!adInitAd2("adinit.dat", 0, &initerr)) {
    printf("Unable to initialize OpenDWG Toolkit\n");
    printf("error is %d: %s\n", initerr, adErrorStr(initerr));
    exit(1);
}
adLayerindexObjtype

Summary
short adLayerindexObjtype(AD_DB_HANDLE handle);

Description
Returns the object type value for AD_LAYERINDEX objects.

Return Value
Returns the object type value for AD_LAYERINDEX objects.

See Also
None

Example
adobhd.objtype=adLayerindexObjtype(handle);

adLayoutObjtype

Summary
short adLayoutObjtype (AD_DB_HANDLE handle);

Description
Returns the object type value for AD_LAYOUT objects.

Return Value
Returns the object type value for AD_LAYOUT objects.

See Also
None

Example
adobhd.objtype=adLayoutObjtype(handle);

adLoadedR14Beta

Summary
short adLoadedR14Beta();
Description

Returns true if an R14 Beta file has been loaded, false otherwise.

Return Value

Returns true if an R14 Beta file has been loaded, false otherwise.

See Also

adAcceptR14BetaFiles

Example

if (adLoadedR14BetaFile()) {...}

adLoadFile

Summary

AD_DB_HANDLE adLoadFile (void *fn, char preloadstrat, short addA2kObjs);

Description

Loads the file indicated by fn into OpenDWG Toolkit. AD_FILE_DESIGNATOR is void * for most systems. preloadstrat indicates the amount of the file to be loaded; at this time the only valid value is AD_PRELOAD_ALL, which loads the entire file. addA2kObjs determines whether AutoCAD 2000 default objects are added to the database. This will normally be set to 1, except in the case where a pre-AutoCAD 2000 file is being loaded and then saved back to R14 or earlier. This argument is provided as a means to eliminate the proxy warning box that appears when R14 and earlier files that contain AutoCAD 2000 objects are loaded into older versions of AutoCAD.

Return Value

Returns a handle to the drawing, or NULL on failure. On failure, adError() should be checked for an indication of the cause of the failure. The handle returned is used for subsequent calls to other OpenDWG Toolkit functions.

See Also

adSaveFile, adCloseFile

Example

if ((dwghandle=adLoadFile(infile,AD_PRELOAD_ALL,1))==NULL) {
    printf("Error opening file %s\n",infile);
    printf("error code is %d:%s\n",adError(),
        adErrorStr(adError()));
    return;
}

adLwplineEnttype
Summary

short adLwplineEnttype(AD_DB_HANDLE handle);

Description

Returns the entity type value for AD_LWPLINE entities.

Return Value

Returns the entity type value for AD_LWPLINE entities.

See Also

None

Example

lwplineenttype=adLwplineEnttype(handle);

adNewFile

Summary

AD_DB_HANDLE adNewFile (AD_OBJHANDLE *stockobjhandle, PAD_CLASS classes, short numClasses, short version);

Description

Creates a new file in virtual memory. OpenDWG Toolkit adds the normal entries for a default "empty file" in AutoCAD, including layer 0, linetypes CONTINUOUS, BYLAYER and BYBLOCK, dimstyle STANDARD, shapefile STANDARD, regapp ACAD, and the standard objects. The stockobjhandle parameter allows control over the handles assigned to these stock objects. To let OpenDWG Toolkit generate these object handles, simply pass NULL for stockobjhandle. If not NULL, OpenDWG Toolkit will use the handles pointed to by stockobjhandle as handles for the default objects, as follows:

- stockobjhandle[0]: Block control object
- stockobjhandle[1]: Layer control object
- stockobjhandle[2]: Shapefile (style) control object
- stockobjhandle[3]: Linetype control object
- stockobjhandle[4]: View control object
- stockobjhandle[5]: UCS control object
- stockobjhandle[6]: Viewport control object
- stockobjhandle[7]: Registered application control object
- stockobjhandle[8]: Dimension style control object
- stockobjhandle[9]: Viewport entity control object
- stockobjhandle[10]: Modelspace blockheader
- stockobjhandle[11]: Paperspace blockheader
- stockobjhandle[12]: Modelspace BLOCK entity
- stockobjhandle[13]: Modelspace ENDBLK entity
- stockobjhandle[14]: Paperspace BLOCK entity
- stockobjhandle[15]: Paperspace ENDBLK entity
- stockobjhandle[16]: BYBLOCK linetype
stockobjhandle[17]: BYLAYER linetype
stockobjhandle[18]: CONTINUOUS linetype
stockobjhandle[19]: Layer 0
stockobjhandle[20]: Viewport *ACTIVE*
stockobjhandle[21]: Registered application ACAD
stockobjhandle[22]: Shapefile (style) STANDARD
stockobjhandle[23]: Dimension style STANDARD
stockobjhandle[24]: Main dictionary object
stockobjhandle[25]: Group dictionary object
stockobjhandle[26]: Multi-line style dictionary object
stockobjhandle[27]: Multi-line style STANDARD
stockobjhandle[28]: Placeholder object
stockobjhandle[29]: Layout dictionary object
stockobjhandle[30]: Dictionary with default object
stockobjhandle[31]: Model Space layout object
stockobjhandle[32]: Paper Space layout object
stockobjhandle[33]: Plotsettings dictionary object
stockobjhandle[34]: Wipeoutvariables object

Note that these handles must be unique (obviously); however they are not required to be in ascending order or to start from any particular value. The handseed for the new file will be 1 greater than the maximum handle in the stockobjhandle array. classes is a list of default classes to add to the file. If this argument is NULL, ODT will create the default class list. numClasses is the number classes in the classes list. If version is AD_ACAD2000, then AutoCAD 2000 default objects will be added to the new file, otherwise these objects will not be added.

Return Value

Returns the handle to the newly opened file on success, NULL on failure. On failure, adError() should be checked for an indication of the cause of the failure.

See Also

adLoadFile, adSaveFile

Example

handle=adNewFile(NULL,NULL,0,AD_ACAD2000);

adNextClassNum

Summary

short adNextClassNum (AD_DB_HANDLE handle);

Description

Returns the next available class id; use this when creating new classes to add to the file.

Return Value

As described above.
See Also

adAddClass

Example

see adAddClass for an example

adNumBlockheaders

Summary

long adNumBlockheaders (AD_DB_HANDLE handle);

Description

Returns the number of blockheaders in the file indicated by handle.

Return Value

As described above.

See Also

adStartBlockheaderGet, adGetBlockheader

Example

num=adNumBlockheaders(handle);

adNumClasses

Summary

long adNumClasses(AD_DB_HANDLE handle);

Description

Returns the number of classes in the file indicated by handle.

Return Value

As described above.

See Also

adStartClassGet, adGetClass

Example

num=adNumClasses(handle);
adNumDimstyles

Summary
long adNumDimstyles(AD_DB_HANDLE handle);

Description
Returns the number of dimension styles in the drawing indicated by handle.

Return Value
As described above.

See Also
adStartDimstyleGet, adGetDimstyle

Example
num=adNumDimstyles(handle);

adNumEntities

Summary
long adNumEntities (AD_DB_HANDLE handle, AD_OBJHANDLE theblk);

Description
Returns the number of entities attached to the block indicated by theblk.

Return Value
As described above.

See Also
adStartEntityGet, adGetEntity

Example
num=adNumEntities(handle,blkh.objhandle);

adNumLayers
long adNumLayers (AD_DB_HANDLE handle);

Description

Returns the number of layers in the drawing indicated by handle.

Return Value

As described above.

See Also

adStartLayerGet, adGetLayer

Example

num=adNumLayers(handle);

adNumLinetypes

Summary

long adNumLinetypes (AD_DB_HANDLE handle);

Description

Returns the number of linetypes in the drawing indicated by handle.

Return Value

As described above.

See Also

adStartLinetypeGet, adGetLinetype

Example

num=adNumLinetypes(handle);

adNumObjects

Summary

long adNumObjects (AD_DB_HANDLE handle);

Description

Returns the number of objects in the drawing indicated by handle.

Return Value
As described above.

See Also

adStartObjectGet, adSeekObject

Example

num=adNumObjects(handle);

adNumRegapps

Summary

long adNumRegapps (AD_DB_HANDLE handle);

Description

Returns the number of registered applications in the drawing indicated by handle.

Return Value

As described above.

See Also

adStartRegappGet, adGetRegapp

Example

num=adNumRegapps(handle);

adNumsFromFaceEntryList

Summary

void adNumsFromFaceEntryList (long *faceentry,long numentries,long *numfaces,
long *numedges);

Description

Determines the number of faces and edges for the faces defined by a face entry list for a proxy graphic data item.

Return Value

The function has no return value. The number of faces and edges are returned in numfaces and numedges respectively.

See Also
None

Example

```
adNumsFromFaceEntryList(faceentry, numentries, &numfaces, &numedges);
```

See also adexzo.c for an example

---

### adNumShapefiles

**Summary**

```
long adNumShapefiles (AD_DB_HANDLE handle);
```

**Description**

Returns the number of shapefiles (styles) in the drawing indicated by `handle`.

**Return Value**

As described above.

**See Also**

`adStartShapefileGet, adGetShapefile`

**Example**

```
um=adNumShapefiles(handle);
```

---

### adNumUcs

**Summary**

```
long adNumUcs (AD_DB_HANDLE handle);
```

**Description**

Returns the number of user coordinate system definitions in the drawing indicated by `handle`.

**Return Value**

As described above.

**See Also**

`adStartUcsGet, adGetUcs`

**Example**

```
num=adNumUcs(handle);
```
**adNumValidCodePages**

**Summary**

short adNumValidCodePages(void);

**Description**

Returns the number of code pages supported by OpenDWG Toolkit.

**Return Value**

As above.

**See Also**

none

**Example**

```c
short i;
char codepagestr[30];

printf("Supported code pages:\n");
for (i=0; i<adNumValidCodePages(); i++) {
    adShortCodePageToAscii(i,codepagestr);
    printf("%d) %s\n",i,codepagestr);
}
```

**adNumViews**

**Summary**

long adNumViews (AD_DB_HANDLE handle);

**Description**

Returns the number of view definitions in the drawing file indicated by handle.

**Return Value**

As described above.

**See Also**

adStartViewGet, adGetView

**Example**

```c
num=adNumViews(handle);
```

**adNumVports**
Summary

long adNumVports (AD_DB_HANDLE handle);

Description

Returns the number of vports in the drawing file indicated by handle.

Return Value

As described above.

See Also

adStartVportGet, adGetVport

Example

num=adNumVports(handle);

adObjectptrObjtype

Summary

short adObjectptrObjtype(AD_DB_HANDLE handle);

Description

Returns the object type value for AD_OBJECTPTR objects.

Return Value

Returns the object type value for AD_OBJECTPTR objects.

See Also

None

Example

adobhd.objtype=adObjectptrObjtype(handle);

adObjhandleValidChars

Summary

char adObjhandleValidChars (AD_OBJHANDLE han);

Description
Returns the number of valid characters in an AD_OBJHANDLE; that is, 8 minus the number of leading zeroes. AD_OBJHANDLEs are right justified in an 8 character field, and padded on the left with zeroes.

Return Value

Returns the number of valid characters in the object handle.

See Also

None

Example

validchars=adObjhandleValidChars(thehandle);

adOle2frameEnttype

Summary

short adOle2frameEnttype(AD_DB_HANDLE handle);

Description

Returns the entity type value for AD_ENT_OLE2FRAME entities.

Return Value

Returns the entity type value for AD_ENT_OLE2FRAME entities.

See Also

None

Example

ole2frameenttype=adOle2frameEnttype(handle);

adPlaceholderObjtype

Summary

short adPlaceholderObjtype (AD_DB_HANDLE handle);

Description

Returns the object type value for AD_PLACEHOLDER objects.

Return Value

Returns the object type value for AD_PLACEHOLDER objects.
adPrimsHaveColors

Summary

adPrimsHaveColors (flags);

Description

A macro which determines whether the primitives associated with a proxy graphic have colors assigned.

Return Value

Returns nonzero if primitive colors are present.

See Also

adHasPrimTraits, adPrimsHave..., adReadPrim...

Example

if (adPrimsHaveColors (primflag)) {
    exampleprintf("mesh edge colors:\n");
    adReadPrimColors (bcptr, numedges, shortarray);
    for (i=0; i<(unsigned short)numedges; i++)
        exampleprintf("%d\n", shortarray[i]);
}

adPrimsHaveLayers

Summary

adPrimsHaveLayers (flags);

Description

A macro which determines whether the primitives associated with a proxy graphic have layers assigned.

Return Value

Returns nonzero if primitive layers are present.

See Also

adHasPrimTraits, adPrimsHave..., adReadPrim...
Example

```c
if (adPrimsHaveLayers(primflag)) {
    exampleprintf("mesh edge layer ids:\n");
adReadPrimLayerids(bcptr,numedges,shortarray);
    for (i=0; i<(unsigned short)numedges; i++)
        exampleprintf("%d\n",shortarray[i]);
}
```

**adPrimsHaveLinetypes**

**Summary**

adPrimsHaveLinetypes (flags);

**Description**

A macro which determines whether the primitives associated with a proxy graphic have linetypes assigned.

**Return Value**

Returns nonzero if primitive linetypes are present.

**See Also**

adHasPrimTraits, adPrimsHave..., adReadPrim...

**Example**

```c
if (adPrimsHaveLinetypes(primflag)) {
    exampleprintf("mesh edge linetype ids:\n");
adReadPrimLinetypeids(bcptr,numedges,shortarray);
    for (i=0; i<(unsigned short)numedges; i++)
        exampleprintf("%d\n",shortarray[i]);
}
```

**adPrimsHaveMarkers**

**Summary**

adPrimsHaveMarkers (flags);

**Description**

A macro which determines whether the primitives associated with a proxy graphic have graphic markers assigned.

**Return Value**

Returns nonzero if primitive markers are present.
See Also

adHasPrimTraits, adPrimsHave..., adReadPrim...

Example

```c
if (adPrimsHaveMarkers(primflag)) {
    exampleprintf("mesh edge markers:
    ");
adReadPrimMarkers(bcptr,numedges,&markers);
    for (i=0; i<(unsigned short)numedges; i++) {
        exampleprintf("%ld
",markers[i]);
    }
}
```

adPrimsHaveNormals

Summary

adPrimsHaveNormals (flags);

Description

A macro which determines whether the primitives associated with a proxy graphic have normals assigned.

Return Value

Returns nonzero if primitive normals are present.

See Also

adHasPrimTraits, adPrimsHave..., adReadPrim...

Example

```c
if (adPrimsHaveNormals(primflag)) {
    exampleprintf("mesh face normals:
    ");
adReadPrimNormals(bcptr,numfaces,normal);
}
```

adPrimsHaveOrientation

Summary

adPrimsHaveOrientation (flags);

Description

A macro which determines whether the primitives associated with a proxy graphic have orientations assigned.

Return Value
Returns nonzero if primitive orientations are present.

See Also

adHasPrimTraits, adPrimsHave..., adReadPrim...

Example

```c
if (adPrimsHaveOrientation(primflag)) {
    exampleprintf("shell face orientation:\n");
    adReadPrimOrientation(bcptr, &orientation);
    exampleprintf("%ld\n", orientation);
}
```

---

adPrimsHaveVisibilities

Summary

adPrimsHaveVisibilities (flags);

Description

A macro which determines whether the primitives associated with a proxy graphic have visibilities assigned.

Return Value

Returns nonzero if primitive visibilities are present.

See Also

adHasPrimTraits, adPrimsHave..., adReadPrim...

Example

```c
if (adPrimsHaveVisibilities(primflag)) {
    exampleprintf("shell face visibilities:\n");
    adReadPrimVisibilities(bcptr, numfaces, &visibility);
}
```

---

adRastervariablesObjtype

Summary

short adRastervariablesObjtype(AD_DB_HANDLE handle);

Description

Returns the object type value for AD_RASTERVARIABLES objects.

Return Value

Returns the object type value for AD_RASTERVARIABLES objects.
Example

```c
rvobjtype=adRastervariablesObjtype(handle);
```

### adReadAcisString

**Summary**

```c
short adReadAcisString (PAD_BLOB_CTRL bcptr, char *str);
```

**Description**

Reads the next string from the blob indicated by `bcptr` and stores it in `str`. The maximum length of this string is 512 bytes.

**Return Value**

Returns 1 on success, 0 on failure. On failure, `adError()` should be checked for an indication of the cause of the failure.

**See Also**

`adStartBlobRead, adEndBlobRead`

### adReadBlob2Double

**Summary**

```c
short adReadBlob2Double (PAD_BLOB_CTRL bcptr,double *doub);
```

**Description**

Reads 2 double precision reals from the blob indicated by `bcptr` into `*doub`.

**Return Value**

Returns 1 on success, 0 on failure. On failure, `adError()` should be checked for an indication of the cause of the failure.

**See Also**
adReadBlobDouble, adWriteBlobDouble, adStartBlobRead

Example

PAD_BLOB_CTRL bcptr;
double doub[2];
bcptr=adStartBlobRead(theblob);
adReadBlob2Double(theblob,doub);

adReadBlob3Double

Summary

short adReadBlob3Double (PAD_BLOB_CTRL bcptr, double *doub);

Description

Reads 3 double precision reals from the blob indicated by bcptr into *doub.

Return Value

Returns 1 on success, 0 on failure. On failure, adError() should be checked for an indication of the cause of the failure.

See Also

adReadBlobDouble, adWriteBlobDouble, adStartBlobRead

Example

PAD_BLOB_CTRL bcptr;
double doub[3];
bcptr=adStartBlobRead(theblob);
adReadBlob3Double(theblob,doub);
adReadBlobByte

Summary
short adReadBlobByte (PAD_BLOB_CTRL bcptr, char *buf);

Description
Reads a single byte from the blob indicated by bcptr into buf.

Return Value
Returns 1 on success, 0 on failure. On failure, adError() should be checked for an indication of the cause of the failure.

See Also
adStartBlobRead, adEndBlobRead

Example
PAD_BLOB_CTRL bcptr;
char ch;
bcptr=adStartBlobRead(theblob);
adReadBlobByte(theblob,&ch);

adReadBlobBytes

Summary
short adReadBlobBytes (PAD_BLOB_CTRL bcptr, char *buf, short num);

Description
Reads num bytes from the blob indicated by bcptr into buf.

Return Value
Returns 1 on success, 0 on failure. On failure, adError() should be checked for an indication of the cause of the failure.

See Also
adStartBlobRead, adEndBlobRead

Example
PAD_BLOB_CTRL bcptr;
char buf[128];
bcptr=adStartBlobRead(theblob);
adReadBlobBytes(theblob,buf,128);
### adReadBlobDouble

**Summary**

```
short adReadBlobDouble (PAD_BLOB_CTRL bcptr,double *doub);
```

**Description**

Reads a double precision real from the blob indicated by `bcptr` into `*doub`.

**Return Value**

Returns 1 on success, 0 on failure. On failure, `adError()` should be checked for an indication of the cause of the failure.

**See Also**

`adWriteBlobDouble, adStartBlobRead`

**Example**

```c
PAD_BLOB_CTRL bcptr;
double *doub;

bcptr=adStartBlobRead(theblob);
adReadBlobDouble(theblob,&doub);
```

### adReadBlobLong

**Summary**

```
short adReadBlobLong (PAD_BLOB_CTRL bcptr,long *l);
```

**Description**

Reads a long from the blob indicated by `bcptr` into `*l`.

**Return Value**

Returns 1 on success, 0 on failure. On failure, `adError()` should be checked for an indication of the cause of the failure.

**See Also**

`adWriteBlobLong, adStartBlobRead`

**Example**

```c
PAD_BLOB_CTRL bcptr;
long *l;

bcptr=adStartBlobRead(theblob);
adReadBlobLong(theblob,&l);
```
adReadBlobObjhandle

Summary

    short adReadBlobObjhandle (PAD_BLOB_CTRL bcptr,AD_OBJHANDLE thehandle);

Description

    Reads an AD_OBJHANDLE from the blob indicated by bcptr into thehandle.

Return Value

    Returns 1 on success, 0 on failure. On failure, adError() should be checked for an indication of the cause of the failure.

See Also

    adWriteBlobObjhandle

Example

    PAD_BLOB_CTRL bcptr;
    AD_OBJHANDLE thehandle;
    bcptr=adStartBlobRead(theblob);
    adReadBlobObjhandle(theblob,thehandle);

adReadBlobShort

Summary

    short adReadBlobShort (PAD_BLOB_CTRL bcptr,short *s);

Description

    Reads a short from the blob indicated by bcptr into *s.
Return Value

Returns 1 on success, 0 on failure. On failure, adError() should be checked for an indication of the cause of the failure.

See Also

adWriteBlobShort, adStartBlobRead

Example

PAD_BLOB_CTRL bcptr;
short *s;
bcptr=adStartBlobRead(theblob);
adReadBlobShort(theblob,&s);

adReadBlobString

Summary

short adReadBlobString (PAD_BLOB_CTRL bcptr,char *str, unsigned short maxsize);

Description

Reads a short from the blob indicated by bcptr into str. maxsize is the size of the buffer to contain the string.

Return Value

Returns 1 on success, 0 on failure. On failure, adError() should be checked for an indication of the cause of the failure.

See Also

adWriteBlobString, adStartBlobWrite

Example

PAD_BLOB_CTRL bcptr;
char str[40];
bcptr=adStartBlobRead(theblob);
adReadBlobString(theblob,str,40);

adReadDicItem

Summary

short adReadDicItem (PAD_BLOB_CTRL bcptr, PAD_DICITEM dicitem);
Description

Reads an item from the dictionary object item blob indicated by bcptr, storing the result in dicitem.

Return Value

Returns 1 on success, 0 on failure. On failure, adError() should be checked for an indication of the cause of the failure.

See Also

adStartObjectGet, adGetObject, adStartBlobRead

Example

```c
if (adobhd.objtype==AD_OBJ_DICTIONARY) {
  if (adob.dic.numdicitems>0L) {
    bcptr=adStartBlobRead(adob.dic.itemblob);
    printf("contains %d items\n",adob.dic.numdicitems);
    for (i=0; i<adob.dic.numdicitems; i++) {
      adReadDicItem(bcptr,&dicitem);
      printf("item: %s\n",dicitem.str);
    }
    adEndBlobRead(bcptr);
  }
}
```

---

**adReadExtendeddata**

**Summary**

short adReadExtendeddata (PAD_BLOB_CTRL bcptr, PAD_XD adxd);

**Description**

Reads the next item of extended data from the blob indicated by bcptr, into the buffer pointed to by adxd.

**Return Value**

Returns 1 on success, 0 on failure. On failure, adError() should be checked for an indication of the cause of the failure.

**See Also**

adStartBlobRead, adEndBlobRead
Example

PAD_BLOB_CTRL bcptr;
AD_XD adXd;
...

bcptr=adStartBlobRead(xdblob);
while (adReadExtendedData(bcptr,&adXd)) {
    printExtendedData(&adXd);
} adEndBlobRead(bcptr);

**adReadGrblobData**

**Summary**

short adReadGrblobData (PAD_BLOB_CTRL bcptr, PAD_GR_DATA data);

**Description**

Reads the next item of data from the proxy entity graphical representation blob indicated by bcptr, and stores it in the buffer indicated by data. data->grtype contains the type of the graphical data. See the Data Reference section of this manual for more information.

**Return Value**

Returns 1 on success, 0 on failure. On failure, adError() should be checked for an indication of the cause of the failure.

**See Also**

adStartBlobRead, adEndBlobRead, prnstuff.h file for a complete example.

Example

if (AD IS A PROXYENT(adenhd->enttype)) {
    if (aden->proxyent.grblob!=AD_VMNULL) {
        bcptr=adStartBlobRead(aden->proxyent.grblob);
        adStartGrBlobDataRead(bcptr);
        while (adReadGrblobData(bcptr,&grdata)) {
            printf("grdata type: %d\n",grdata.grtype);
            if (grdata.grtype==AD_GRENT_PLINE) {
                for (i=0; i<(short)grdata.grbody.pline.numpts; i++) {
                    adReadGrblobVertexPt(bcptr,pt);
                    printf("vertex: %lf %lf %lf\n",pt[0],pt[1],pt[2]);
                }
            }
        }
    }
    adEndBlobRead(bcptr);
}

**adReadGrblobVertexPt**

**Summary**
short adReadGrblobVertexPt (PAD_BLOB_CTRL bcptr, double *pt);

**Description**

Reads a vertex point for a data item of type AD_GRENT_PLINE or AD_GRENT_POLYGON from the proxy entity graphical representation blob indicated by `bcptr`, storing the result in `pt`.

**Return Value**

Returns 1 on success, 0 on failure. On failure, `adError()` should be checked for an indication of the cause of the failure.

**See Also**

`adStartBlobRead`, `adReadGrblobData`, `adEndBlobRead`

**Example**

See `adReadGrblobData` for an example.

---

**adReadGrshellFaceEntries**

**Summary**

short adReadGrshellFaceEntries(PAD_BLOB_CTRL bcptr, long numentries, long *faceentry);

**Description**

Reads the face entries for a data item of type AD_GRENT_SHELL from the proxy entity graphical representation blob indicated by `bcptr`, storing the result in `faceentry`.

**Return Value**

Returns 1 on success, 0 on failure. On failure, `adError()` should be checked for an indication of the cause of the failure.

**See Also**

`adHasPrimTraits`, `adPrimsHave...`, `adReadPrim...`

**Example**

```c
adReadGrshellNumFaceEntries(bcptr,&numfaceentries);
numfaces=facecount=numedges=0L;
adReadGrshellFaceEntries(bcptr,numfaceentries,&entry);
```

---

**adReadHeaderBlock**

**Summary**
short adReadHeaderBlock (AD_DB_HANDLE handle, PAD_DWGHDR adhd);

Description

Reads the current header for the drawing indicated by handle and stores it in the buffer pointed to by adhd. NOTE – THIS IS AN OBSOLETE FUNCTION. YOU SHOULD USE adHeaderPointer() INSTEAD.

Return Value

Returns 1 on success, 0 on failure. On failure, adError() should be checked for an indication of the cause of the failure.

See Also

adHeaderPointer, adWriteHeaderBlock

Example

AD_DWGHDR adhd;
adReadHeaderBlock(handle, &adhd);

adReadMlstyleSeginfo

Summary

short adReadMlstyleSeginfo (PAD_BLOB_CTRL bcptr, PAD_MLSTYLESEGINFO seginfo);

Description

Reads the next group of multi-line style segment information from the blob indicated by bcptr, and stores it in the buffer pointed to by seginfo.

Return Value

Returns 1 on success, 0 on failure. On failure, adError() should be checked for an indication of the cause of the failure.

See Also

adStartBlobRead, adEndBlobRead

Example

See adGetMlstyle for an example.

adReadMtextBlock

Summary

short adReadMtextBlock (PAD_BLOB_CTRL bcptr, char *str);
Description

Returns the next piece of mtext from an mtext blob indicated by bcptr, and stores it in str. Note that an mtext entity which is shorter than 256 bytes will not have a blob, and will have its string data stored in the textstr field of the entity.

Return Value

Returns 1 on success, 0 on failure. On failure, adError() should be checked for an indication of the cause of the failure.

See Also

adStartBlobRead, adEndBlobRead

Example

```c
else if (adenhd->enttype==AD_ENT_MTEXT) {
    if (aden->mtext.ldblob==AD_VMNULL)
        printf("mtext str %s\n",aden->mtext.textstr);
    else {
        bcptr=adStartBlobRead(aden->mtext.ldblob);
        while (adReadMtextBlock(bcptr,aden->mtext.textstr)) {
            printf("next string is %s\n",aden->mtext.textstr);
        }
        adEndBlobRead(bcptr);
    }
}
```

adReadPrimColors

Summary

short adReadPrimColors (PAD_BLOB_CTRL bcptr, short numcolors, short *shortarray);

Description

Reads the array of primitive colors assigned to the primitives in a proxy entity graphic. Reads numcolors entries from the blob indicated by bcptr, and stores the entries in the buffer pointed to by shortarray.

Return Value

Returns 1 on success, 0 on failure. On failure, adError() should be checked for an indication of the cause of the failure.

See Also

adHasPrimTraits, adPrimsHave..., adReadPrim...

Example

```c
if (adPrimsHaveColors(primflag)) {
    exampleprintf("mesh face colors: \n");
    adReadPrimColors(bcptr,numfaces,shortarray);
    for (i=0; i<(unsigned short)numfaces; i++)
        exampleprintf("%d\n",shortarray[i]);
}
```
adReadPrimFlags

Summary

short adReadPrimFlags (PAD_BLOB_CTRL bcptr, long *flags);

Description

Reads a set of flags from a proxy entity graphic that indicate the presence of primitive traits within the graphic. Reads from the blob indicated by bcptr, and stores the flags in flags.

Return Value

Returns 1 on success, 0 on failure. On failure, adError() should be checked for an indication of the cause of the failure.

See Also

adHasPrimTraits, adPrimsHave..., adReadPrim...

Example

    adReadPrimFlags(bcptr,&primflag);
    if (adHasPrimTraits(primflag)) {
        . . .
adReadPrimLayerids

**Summary**

short adReadPrimLayerids (PAD_BLOB_CTRL bcptr, short numlayerids, short *shortarray);

**Description**

Reads the array of primitive layers assigned to the primitives in a proxy entity graphic. Reads numlayerids entries from the blob indicated by bcptr, and stores the entries in the buffer pointed to by shortarray.

**Return Value**

Returns 1 on success, 0 on failure. On failure, adError() should be checked for an indication of the cause of the failure.

**See Also**

adHasPrimTraits, adPrimsHave..., adReadPrim...

**Example**

```c
if (adPrimsHaveLayers(primflag)) {
    exampleprintf("mesh face layer ids:\n");
    adReadPrimLayerids(bcptr,numfaces,shortarray);
    for (i=0; i<(unsigned short)numfaces; i++)
        exampleprintf("%d\n",shortarray[i]);
}
```

adReadPrimLinetypeids

**Summary**

short adReadPrimLinetypeids (PAD_BLOB_CTRL bcptr, short numlinetypeids, short *shortarray);

**Description**

Reads the array of primitive linetypes assigned to the primitives in a proxy entity graphic. Reads numlinetypeids entries from the blob indicated by bcptr, and stores the entries in the buffer pointed to by shortarray.

**Return Value**

Returns 1 on success, 0 on failure. On failure, adError() should be checked for an indication of the cause of the failure.

**See Also**

adHasPrimTraits, adPrimsHave..., adReadPrim...

**Example**

```c
if (adPrimsHaveLinetypes(primflag)) {
    exampleprintf("mesh edge linetype ids:\n");
```
adReadPrimLinetypeids(bcptr,numedges,shortarray);
for (i=0; i<(unsigned short)numedges; i++)
  exampleprintf("%d\n",shortarray[i]);
}

adReadPrimMarkers

Summary

short adReadPrimMarkers (PAD_BLOB_CTRL bcptr, long numitems, long *marker);

Description

Reads the markers for the primitives in a proxy entity graphic. Reads from the blob indicated by bcptr, and stores the markers in the buffer pointed to by marker.

Return Value

Returns 1 on success, 0 on failure. On failure, adError() should be checked for an indication of the cause of the failure.

See Also

adHasPrimTraits, adPrimsHave..., adReadPrim...

Example

if (adPrimsHaveMarkers(primflag)) {
  exampleprintf("mesh edge markers:\n");
  adReadPrimMarkers(bcptr,numedges,&marker);
}

adReadPrimNormals

Summary

short adReadPrimNormals (PAD_BLOB_CTRL bcptr, long numitems, double *normal);

Description

Reads the normal vectors assigned to the primitives in a proxy entity graphic. Reads from the blob indicated by bcptr, and stores the normals in the buffer pointed to by normal. Note that normals are X, Y and Z magnitudes; these are stored in the normal array as follows:

normal[0]: first X magnitude
normal[1]: first Y magnitude
normal[2]: first Z magnitude
normal[3]: second X magnitude
normal[4]: second Y magnitude
normal[5]: second Z magnitude
etc.
Return Value

Returns 1 on success, 0 on failure. On failure, \texttt{adError()} should be checked for an indication of the cause of the failure.

See Also

\texttt{adHasPrimTraits, adPrimsHave..., adReadPrim...}

Example

\begin{verbatim}
if (adPrimsHaveNormals(primflag)) {
    exampleprintf("mesh face normals:\n");
    adReadPrimNormal(bcptr,numfaces,normal);
}
\end{verbatim}

\texttt{adReadPrimOrientation}

Summary

\begin{verbatim}
short adReadPrimOrientation (PAD_BLOB_CTRL bcptr, long *orientation);
\end{verbatim}

Description

Reads the next orientation assigned to the primitives in a proxy entity graphic. Reads from the blob indicated by \texttt{bcptr}, and stores the marker in the buffer pointed to by \texttt{orientation}.

Return Value

Returns 1 on success, 0 on failure. On failure, \texttt{adError()} should be checked for an indication of the cause of the failure.

See Also

\texttt{adHasPrimTraits, adPrimsHave..., adReadPrim...}

Example

\begin{verbatim}
if (adPrimsHaveOrientation(primflag)) {
    exampleprintf("mesh face orientation:\n");
    adReadPrimOrientation(bcptr, &orientation);
    exampleprintf("%ld\n",orientation);
}
\end{verbatim}
adReadPrimVisibilities

Summary

    short adReadPrimVisibilities (PAD_BLOB_CTRL bcptr, long numitems, long *visibility);

Description

    Reads the visibilities assigned to the primitives in a proxy entity graphic. Reads from the blob indicated by bcptr, and stores the visibilities in the buffer pointed to by visibility.

Return Value

    Returns 1 on success, 0 on failure. On failure, adError() should be checked for an indication of the cause of the failure.

See Also

    adHasPrimTraits, adPrimsHave..., adReadPrim...

Example

    if (adPrimsHaveVisibilities(primflag)) {
        exampleprintf("shell edge visibilities:\n");
        adReadPrimVisibilities(bcptr,numfaces,&visibility);
    }

adReadXrecordItem

Summary

    short adReadXrecordItem (PAD_BLOB_CTRL bcptr, PAD_XD adxd);

Description

    Reads the next item from the xrecord data pointed to by bcptr into the buffer pointed to by adxd. Note that the structure passed to this function is the same as that passed to read extended data.

Return Value

    Returns 1 on success, 0 on failure. On failure, adError() should be checked for an indication of the cause of the failure.

See Also

    adStartBlobRead, adEndBlobRead
Example

```c
if (adtb->proxyobj.type==AD_ZT_XRECORD) {
    printf("xrecord data:\n\n");
    bcptr=adStartBlobRead(adtb->proxyobj.datablob);
    while (adReadXrecordItem(bcptr,&adxd)) {
        printf("type %d, value ",adxd.xddxfnumber);
        if ((adxd.xddxfnumber>=0 && adxd.xddxfnumber<=9) ||
            adxd.xddxfnumber==10) {
            printf("%s\n",adxd.xddata.xdstring);
        } else if (adxd.xddxfnumber>=10 && adxd.xddxfnumber<=19) {
            printf("%lf, %lf, %lf\n",adxd.xddata.xd3dpt[0],
                adxd.xddata.xd3dpt[1],adxd.xddata.xd3dpt[2]);
        } else if ((adxd.xddxfnumber>=60 && adxd.xddxfnumber<=79) ||
                (adxd.xddxfnumber>=170 && adxd.xddxfnumber<=175)) {
            printf("%d\n",adxd.xddata.xdint);
        } else if (adxd.xddxfnumber>=310 && adxd.xddxfnumber<=319) {
            for (i=0; i<adxd.xdbindatalength; I++)
                printf("%.02X",((unsigned)adxd.xddata.xdbindata[i]) &
                        255);
            printf("\n");
        } else if (adxd.xddxfnumber>=330 && adxd.xddxfnumber<=369) {
            locprinthandle(adxd.xddata.xdhandle);
        }
    }
}
```

adReadProxyDataBinary

**Summary**

short adReadProxyDataBinary (char *a, short *len);

**Description**

Reads a proxy (object or entity) binary data into the buffer pointed to by a and returns the length in len.

**Return Value**

Returns 1 on success, 0 on failure. On failure, adError() should be checked for an indication of the cause of the failure.

**See Also**

adStartProxyDataRead, adReadProxyData...

**Example**

```c
printf("now reading entity data\n");
/* YOU MUST KNOW THE DATA DEFINITION FOR THE ENTITY TO DO THIS! */
```
if (aden->proxyent.entdatablob!=AD_VMNULL) {
    adStartProxyDataRead(dwghandle,
        aden->proxyent.entdatablob);

    adReadProxyDataChar(&ch);
    printf("character is %c\n",ch);

    adReadProxyDataString(str);
    printf("string is %s\n",str);

    adReadProxyDataBinary(str,&len);
    printf("binary data is:\n");
    for (i=0; i<len; i++) {
        printf("%.02X ",str[i]);
        if ((i % 8)==7) printf("\n");
    }
    printf("\n");

    adReadProxyDataObjhandle(oh);
    printf("objhandle is ");
    locprinthandle(oh);
    printf("\n");

    adReadProxyDataInt32(&l);
    printf("Int32 was %ld\n",l);

    adReadProxyDataInt16(&i);
    printf("Int16 was %d\n",i);

    adReadProxyDataUInt32(&ul);
    printf("UInt32 was %ld\n",ul);

    adReadProxyDataUInt16(&ui);
    printf("UInt16 was %d\n",ui);

    adReadProxyDataUInt8(&uc);
    printf("UInt8 was %d\n",uc);

    adReadProxyDataBool(&uc);
    printf("Bool was %d\n",uc);

    adReadProxyDataDouble(&d);
    printf("Double was %lf\n",d);

    adReadProxyDataPoint2d(d2);
    printf("Point2d was %lf %lf\n",d2[0],d2[1]);

    adReadProxyDataPoint3d(d2);
    printf("Point3d was %lf %lf %lf\n",d2[0],d2[1],d2[2]);

    adReadProxyDataVector2d(d2);
    printf("Vector2d was %lf %lf\n",d2[0],d2[1]);

    adReadProxyDataVector3d(d2);
    printf("Vector3d was %lf %lf %lf\n",d2[0],d2[1],d2[2]);

    adReadProxyDataScale3d(d2);
    printf("Scale3d was %lf %lf %lf\n",d2[0],d2[1],d2[2]);

    adReadProxyDataVoidstarData(bufdata,50);
    printf("voidstar data is:\n");
    for (i=0; i<50; i++) {
        printf("%.02X ",bufdata[i]);
        if ((i % 8)==7) printf("\n");
    }
    printf("\n");
} else printf("entdatablob was NULL\n");
adReadProxyDataBool

Summary

short adReadProxyDataBool (unsigned char *a);

Description

Reads a proxy (object or entity) Boolean into the buffer pointed to by a.

Return Value

Returns 1 on success, 0 on failure. On failure, adError() should be checked for an indication of the cause of the failure.

See Also

adStartProxyDataRead, adReadProxyData...

Example

See adReadProxyDataBinary for an example.

adReadProxyDataChar

Summary

short adReadProxyDataChar (char *a);

Description

Reads a proxy (object or entity) character into the buffer pointed to by a.

Return Value

Returns 1 on success, 0 on failure. On failure, adError() should be checked for an indication of the cause of the failure.

See Also

adStartProxyDataRead, adReadProxyData...

Example

See adReadProxyDataBinary for an example.

adReadProxyDataDouble
Summary

short adReadProxyDataDouble (double *a);

Description

Reads a proxy (object or entity) double into the buffer pointed to by a.

Return Value

Returns 1 on success, 0 on failure. On failure, adError() should be checked for an indication of the cause of the failure.

See Also

adStartProxyDataRead, adReadProxyData...

Example

See adReadProxyDataBinary for an example.

adReadProxyDataInt16

Summary

short adReadProxyDataInt16 (short *a);

Description

Reads a proxy (object or entity) binary 16 bit short integer into the buffer pointed to by a.

Return Value

Returns 1 on success, 0 on failure. On failure, adError() should be checked for an indication of the cause of the failure.

See Also

adStartProxyDataRead, adReadProxyData...

Example

See adReadProxyDataBinary for an example.

adReadProxyDataInt32

Summary

short adReadProxyDataInt32 (long *a);

Description
Reads a proxy (object or entity) 32 bit integer into the buffer pointed to by a.

Return Value

Returns 1 on success, 0 on failure. On failure, adError() should be checked for an indication of the cause of the failure.

See Also

adStartProxyDataRead, adReadProxyData...

Example

See adReadProxyDataBinary for an example.

adReadProxyDataObjhandle

Summary

short adReadProxyDataObjhandle (AD_OBJHANDLE a);

Description

Reads a proxy (object or entity) AD_OBJHANDLE into the buffer pointed to by a.

Return Value

Returns 1 on success, 0 on failure. On failure, adError() should be checked for an indication of the cause of the failure.

See Also

adStartProxyDataRead, adReadProxyData...

Example

See adReadProxyDataBinary for an example.

adReadProxyDataPoint2d

Summary

short adReadProxyDataPoint2d (double *a);

Description

Reads a proxy (object or entity) 2D point (X and Y coordinates) into the buffer pointed to by a.

Return Value
Returns 1 on success, 0 on failure. On failure, \texttt{adError()} should be checked for an indication of the cause of the failure.

See Also

\texttt{adStartProxyDataRead, adReadProxyData...}

Example

See \texttt{adReadProxyDataBinary} for an example.

\noalign{\hline}

\textbf{adReadProxyDataPoint3d}

\noalign{\hline}

Summary

\begin{verbatim}
short adReadProxyDataPoint3d (double *a);
\end{verbatim}

Description

Reads a proxy (object or entity) 3D point \((X, Y \text{ and } Z\) coordinates) into the buffer pointed to by \(a\).

Return Value

Returns 1 on success, 0 on failure. On failure, \texttt{adError()} should be checked for an indication of the cause of the failure.

See Also

\texttt{adStartProxyDataRead, adReadProxyData...}

Example

See \texttt{adReadProxyDataBinary} for an example.

\noalign{\hline}

\textbf{adReadProxyDataScale3d}

\noalign{\hline}

Summary

\begin{verbatim}
short adReadProxyDataScale3d (double *a);
\end{verbatim}

Description

Reads a proxy (object or entity) scale value into the buffer pointed to by \(a\).

Return Value

Returns 1 on success, 0 on failure. On failure, \texttt{adError()} should be checked for an indication of the cause of the failure.

See Also
adStartProxyDataRead, adReadProxyData...

Example

See adReadProxyDataBinary for an example.

adReadProxyDataString

Summary

short adReadProxyDataString (char *a);

Description

Reads a proxy (object or entity) string into the buffer pointed to by a.

Return Value

Returns 1 on success, 0 on failure. On failure, adError() should be checked for an indication of the cause of the failure.

See Also

adStartProxyDataRead, adReadProxyData...

Example

See adReadProxyDataBinary for an example.

adReadProxyDataUInt16

Summary

short adReadProxyDataUInt16 (unsigned short *a);

Description

Reads a proxy (object or entity) unsigned short 16 bit integer into the buffer pointed to by a.

Return Value

Returns 1 on success, 0 on failure. On failure, adError() should be checked for an indication of the cause of the failure.

See Also

adStartProxyDataRead, adReadProxyData...

Example

See adReadProxyDataBinary for an example.
**adReadProxyDataUInt32**

Summary

short adReadProxyDataUInt32 (unsigned long *a);

Description

Reads a proxy (object or entity) unsigned short 32 bit integer into the buffer pointed to by a.

Return Value

Returns 1 on success, 0 on failure. On failure, adError() should be checked for an indication of the cause of the failure.

See Also

adStartProxyDataRead, adReadProxyData...

Example

See adReadProxyDataBinary for an example.

**adReadProxyDataUInt8**

Summary

short adReadProxyDataUInt8 (unsigned char *a);

Description

Reads a proxy (object or entity) unsigned character into the buffer pointed to by a.

Return Value

Returns 1 on success, 0 on failure. On failure, adError() should be checked for an indication of the cause of the failure.

See Also

adStartProxyDataRead, adReadProxyData...

Example

See adReadProxyDataBinary for an example.

**adReadProxyDataVector2d**
Summary

    short adReadProxyDataVector2d (double *a);

Description

    Reads a proxy (object or entity) 2D vector (X and Y values) into the buffer pointed to by a.

Return Value

    Returns 1 on success, 0 on failure. On failure, adError() should be checked for an indication of the cause of the failure.

See Also

    adStartProxyDataRead, adReadProxyData...

Example

    See adReadProxyDataBinary for an example.
adReadProxyDataVector3d

Summary

short adReadProxyDataVector3d (double *a);

Description

Reads a proxy (object or entity) 3D vector (X, Y and Z values) into the buffer pointed to by a.

Return Value

Returns 1 on success, 0 on failure. On failure, adError() should be checked for an indication of the cause of the failure.

See Also

adStartProxyDataRead, adReadProxyData...

Example

See adReadProxyDataBinary for an example.

adReadProxyDataVoidstardata

Summary

short adReadProxyDataVoidstardata (char *a,unsigned short len);

Description

Reads proxy entity binary data into the buffer pointed to by a. The data's length should be given in the len parameter.

Return Value

Returns 1 on success, 0 on failure. On failure, adError() should be checked for an indication of the cause of the failure.

See Also

adStartProxyDataRead, adReadProxyData...

Example

See adReadProxyDataBinary for an example.
adReadProxyEntityObjid

Summary

short adReadProxyEntityObjid (PAD_BLOB_CTRL bcptr, PAD_TYPEDOBJHANDLE oh);

Description

Reads a proxy (object or entity) object ID into the buffer pointed to by oh.

Return Value

Returns 1 on success, 0 on failure. On failure, adError() should be checked for an indication of the cause of the failure.

See Also

adStartProxyDataRead, adReadProxyData...

Example

if (aden->proxyent.numobjids!=0L) {
    bcptr=adStartBlobRead(aden->proxyent.objidblob);
    for (i=0; i<(short)adtb->proxyent.numobjids; i++) {
        adReadProxyEntityObjid(bcptr,&objidhandle);
        printf("obj ID handle: ");
        locprinthandle(objidhandle.typedhandle);
    }
    adEndBlobRead(bcptr);
}

adReadProxyObjectObjid

Summary

short adReadProxyObjectObjid (PAD_BLOB_CTRL bcptr, PAD_TYPEDOBJHANDLE oh);

Description

Reads a proxy object object ID into the buffer pointed to by oh.

Return Value

Returns 1 on success, 0 on failure. On failure, adError() should be checked for an indication of the cause of the failure.

See Also

adStartBlobRead, adEndBlobRead
if (adtb->proxyobj.numobjids!=0L) {
    bcptr=adStartBlobRead(adtb->proxyobj.objidblob);
    for (i=0; i<(short)adtb->proxyobj.numobjids; i++) {
        adReadProxyObjectObjid(bcptr,&objidhandle);
        printf("obj ID handle: ");
        locprinthandle(objidhandle.typedhandle);
    }
    adEndBlobRead(bcptr);
}

**adReplaceBlockheader**

**Summary**

short adReplaceBlockheader (AD_DB_HANDLE handle, PAD_BLKH theblkh);

**Description**

Replaces the blockheader in the list whose AD_OBJHANDLE is theblkh->objhandle with the blockheader stored in the buffer pointed to by theblkh.

**Return Value**

Returns 1 on success, 0 on failure. On failure, adError() should be checked for an indication of the cause of the failure.

**See Also**

adStartBlockheaderGet, adGetBlockheader, adDeleteBlockheader

**Example**

AD_BLKH blkh;

adGetBlockheader(handle,&blkh);
strcpy(blkh.name,"NEWNAME");
adReplaceBlockheader(handle,&blkh);

**adReplaceDimstyle**

**Summary**

short adReplaceDimstyle (AD_DB_HANDLE handle, PAD_DIMSTYLE thedimstyle);

**Description**

Replaces the dimension style in the list whose AD_OBJHANDLE is thedimstyle->objhandle with the dimstyle stored in the buffer pointed to by thedimstyle.

**Example**

AD_BLKH blkh;

adGetBlockheader (handle,&blkh);
strcpy(blkh.name,"NEWNAME");
adReplaceDimstyle (handle,&blkh);
Return Value

Returns 1 on success, 0 on failure. On failure, `adError()` should be checked for an indication of the cause of the failure.

See Also

`adStartDimstyleGet, adGetDimstyle, adDeleteDimstyle`

Example

```c
AD_DIMSTYLE dimst;
adGetDimstyle(handle, &dimst);
strcpy(dimst.name, "NEWDIMSTYLE");
adReplaceDimstyle(handle, &dimst);
```

adReplaceEntity

Summary

```c
short adReplaceEntity(AD_DB_HANDLE handle, AD_VMADDR list, AD_OBJHANDLE oldhandle, PAD_ENT_HDR adenhd, PAD_ENT aden);
```

Description

Replaces the entity in list `list` whose `AD_OBJHANDLE` is `oldhandle` with the entity pointed to by `adenhd` and `aden`. Note that replacing entities can be a bit tricky, due to the potential presence of blobs of long data, so please understand the following rules regarding this function:

- `adReplaceEntity` will first compare the entity types of the current entity in the list and the entity to replace it. If they are not the same, the entity currently in the list is deleted, along with any associated blobs (including extended entity data), and the new entity is added to the list in place of the old one.

- If the entity types are the same, the entity currently in the list is deleted but its blobs are retained. Then the address of each blob in the old entity is compared to the address of the corresponding blob in the new entity. If they are the same, it is assumed that the old blob was meant to be preserved; otherwise, the old blob is deleted.

Return Value

Returns 1 on success, 0 on failure. On failure, `adError()` should be checked for an indication of the cause of the failure.

See Also

`adStartEntityGet, adGetEntity, adDeleteEntity`
Example

/* now demonstrate replace of same-type entity */
/* first get handle of just-added ent, and change it */
adHancpy(holdhandle,adenhd->enthandle);
/* just change the handle */
adGenerateObjhandle(wrhandle,adenhd->enthandle);
adGenerateObjhandle(wrhandle,adenhd->enthandle);
adGenerateObjhandle(wrhandle,adenhd->enthandle);
/* now replace */
adReplaceEntity(wrhandle,mspaceentlist,holdhandle,adenhd,aden);

adReplaceLayer

Summary

short adReplaceLayer (AD_DB_HANDLE handle, PAD_LAY thelay);

Description

Replaces the layer in the list whose AD_OBJHANDLE is thelay->objhandle with the layer stored in the buffer pointed to by thelay.

Return Value

Returns 1 on success, 0 on failure. On failure, adError() should be checked for an indication of the cause of the failure.

See Also

adStartLayerGet, adGetLayer, adDeleteLayer

Example

AD_LAYER lay;
adGetLayer(handle,&lay);
strcpy(lay.name,"NEWLAYER");
adReplaceLayer(handle,&lay);

adReplaceLinetype

Summary

short adReplaceLinetype (AD_DB_HANDLE handle, PAD_LTYPE theltp);

Description

Replaces the linetype in the list whose AD_OBJHANDLE is theltp->objhandle with the linetype stored in the buffer pointed to by theltp.
Return Value

Returns 1 on success, 0 on failure. On failure, adError() should be checked for an indication of the cause of the failure.

See Also

adStartLinetypeGet, adGetLinetype, adDeleteLinetype

Example

AD_LTYPE ltp;
adGetLinetype(handle,&ltp);
strcpy(ltp.name,"NEWLTYPE");
adReplaceLinetype(handle,&ltp);

adReplaceObject

Summary

short adReplaceObject (AD_DB_HANDLE handle, PAD_OBJ_HDR adobhd, PAD_OBJ adob);

Description

Replaces the object in the list whose AD_OBJHANDLE is adobhd->objhandle with the object stored in the buffer pointed to by adobhd and adob.

Return Value

Returns 1 on success, 0 on failure. On failure, adError() should be checked for an indication of the cause of the failure.

See Also

adStartObjectGet, adGetObject, adDeleteObject

Example

AD_OBJ_HDR adobhd;
AD_OBJ adob;

adGetObject(handle,&adobhd,&adob);
if (adobhd.objtype==AD_OBJ_MLINESTYLE) {
    strcpy(adob.mlstyle.name,"NEWMLSTYLENAME");
adReplaceObject(handle,&adobhd,&adob);
}
adReplaceRegapp

Summary

    short adReplaceRegapp (AD_DB_HANDLE handle, PAD_APP theapp);

Description

    Replaces the registered application in the list whose AD_OBJHANDLE is theapp->objhandle with the regapp stored in the buffer pointed to by theapp.

Return Value

    Returns 1 on success, 0 on failure. On failure, adError() should be checked for an indication of the cause of the failure.

See Also

    adStartRegappGet, adGetRegapp, adDeleteRegapp

Example

    AD_APP app;
    adGetRegapp(handle,&app);
    strcpy(app.name,"NEWAPP");
    adReplaceRegapp(handle,&app);

adReplaceShapefile

Summary

    short adReplaceShapefile (AD_DB_HANDLE handle, PAD_SHPTB theshptb);

Description

    Replaces the shapefile (style) in the list whose AD_OBJHANDLE is theshptb->objhandle with the shapefile stored in the buffer pointed to by theshptb.

Return Value

    Returns 1 on success, 0 on failure. On failure, adError() should be checked for an indication of the cause of the failure.

See Also

    adStartShapefileGet, adGetShapefile, adDeleteShapefile
Example

    AD_SHPTB shptb;
    adGetShapefile(handle,&shptb);
    strcpy(shptb.name,"NEWSHPTB");
    adReplaceShapefile(handle,&shptb);

**adReplaceUcs**

Summary

    short adReplaceUcs (AD_DB_HANDLE handle, PAD_UCS theucs);

Description

    Replaces the user coordinate system entry in the list whose AD_OBJHANDLE is \texttt{theucs->objhandle} with the ucs stored in the buffer pointed to by \texttt{theucs}.

Return Value

    Returns 1 on success, 0 on failure. On failure, \texttt{adError()} should be checked for an indication of the cause of the failure.

See Also

    adStartUcsGet, adGetUcs, adDeleteUcs

Example

    AD_UCS ucs;
    adGetUcs(handle,&ucs);
    strcpy(ucs.name,"NEWUCS");
    adReplaceUcs(handle,&ucs);

**adReplaceView**

Summary

    short adReplaceView (AD_DB_HANDLE handle, PAD_VIEW theview);

Description

    Replaces the view entry in the list whose AD_OBJHANDLE is \texttt{theview->objhandle} with the view stored in the buffer pointed to by \texttt{theview}.

Return Value

    Returns 1 on success, 0 on failure. On failure, \texttt{adError()} should be checked for an indication of the cause of the failure.

See Also
adStartViewGet, adGetView, adDeleteView

Example

AD_VIEW view;
adGetView(handle,&view);
strcpy(view.name,"NEWVIEW");
adReplaceView(handle,&view);

adReplaceVport

Summary

short adReplaceVport (AD_DB_HANDLE handle, PAD_VPORT thevport);

Description

Replaces the vport definition in the list whose AD_OBJHANDLE is \texttt{thevport->objhandle} with the vport stored in the buffer pointed to by \texttt{thevport}.

Return Value

Returns 1 on success, 0 on failure. On failure, \texttt{adError()} should be checked for an indication of the cause of the failure.

See Also

adStartVportGet, adGetVport, adDeleteVport

Example

AD_VPORT vport;
adGetVport(handle,&vport);
strcpy(vport.name,"NEWVPORT");
adReplaceVport(handle,&vport);

adRewindBlob

Summary

void adRewindBlob (PAD_BLOB_CTRL bcptr);

Description

Rewinds the blob indicated by \texttt{bcptr} to the start. Should be used only on blobs open for read.

Return Value

There is no return value.
See Also

adStartBlobRead, adReadBlobBytes

Example

PAD_BLOB_CTRL bcptr;
adRewindBlob(bcptr);

adRtextEnttype

Summary

short adRtextEnttype(AD_DB_HANDLE handle);

Description

Returns the entity type value for AD_RTEXT entities.

Return Value

Returns the entity type value for AD_RTEXT entities.

See Also

None

Example

rtextenttype = adRtextEnttype(handle);

adSaveFile

Summary

short adSaveFile (AD_DB_HANDLE handle, void *name, char filetype, short version, short dxfnegz, short dxfdecprec, short dxfwritezeroes, char r12dxfvbls);

Description

Saves the file indicated by handle. name, , indicates the filename to which the file is to be saved. filetype should be one of the defined constants AD_DWG, AD_DXF, or AD_BDXF. version is the AutoCAD version of the file to be created, and should be one of the defined constants AD_ACAD25, AD_ACAD26, AD_ACAD9, AD_ACAD10, AD_ACAD11, AD_ACAD13, or AD_ACAD14.

The final four parameters are significant only if the file being written is a DXF file; you may simply send 0 for each of these values if you are not writing a DXF file.

dxfnegz, if set to 1, causes OpenDWG Toolkit to emit -0.0 (rather than 0.0) for small negative numbers, as AutoCAD does. dxfdecprec is the decimal precision of the DXF file; AutoCAD's default is 6. dxfwritezeroes, if set to 1, causes OpenDWG Toolkit to write Z coordinates with 0 values in cases where the comparable AutoCAD
version emits them. Finally, r12dxvbls causes OpenDWG Toolkit to emit a few extra header variables which are in AutoCAD Release 12 but not in Release 11; this variable is only significant if you are writing a Release 11 version DXF file.

**Return Value**

Returns 1 on success, 0 on failure. On failure, adError() should be checked for an indication of the cause of the failure.

**See Also**

adLoadFile

**Example**

```c
#ifdef DODXF
    adSaveFile(rdhandle, argv[2], AD_DXF, adFileVersionLoaded(rdhandle), 0, 6, 1, 1);
#endif
#ifdef DODWG
    adSaveFile(rdhandle, argv[2], AD_DWG, adFileVersionLoaded(rdhandle), 0, 6, 1, 1);
#endif
```

**adSeekBlockheader**

**Summary**

short adSeekBlockheader (AD_DB_HANDLE handle, AD_OBJHANDLE thehandle, PAD_BLKH adblkh);

**Description**

Searches for the blockheader whose handle is thehandle, and returns it, if found, in the buffer pointed to by adblkh. The current position in the list will be just past the object sought.

**Return Value**

Returns 1 on success, 0 on failure. On failure, adError() should be checked for an indication of the cause of the failure.

**See Also**

adStartBlockheaderGet, adGetBlockheader, adReplaceBlockheader, adDeleteBlockheader

**Example**

```c
adSeekBlockheader(handle, thehandle, &blkh);
```

**adSeekDimstyle**

**Summary**
short adSeekDimstyle (AD_DB_HANDLE handle, AD_OBJHANDLE thehandle, PAD_DIMSTYLE addimst);

**Description**

Searches for the dimension style whose handle is `thehandle`, and returns it, if found, in the buffer pointed to by `addimst`. The current position in the list will be just past the object sought.

**Return Value**

Returns 1 on success, 0 on failure. On failure, `adError()` should be checked for an indication of the cause of the failure.

**See Also**

`adStartDimstyleGet`, `adGetDimstyle`, `adReplaceDimstyle`, `adDeleteDimstyle`

**Example**

```c
adSeekDimstyle (handle, thehandle, &dimst);
```

---

**adSeekEntity**

**Summary**

`adSeekEntity (AD_DB_HANDLE handle, AD_VMADDR list, AD_OBJHANDLE thehandle, PAD_ENT_HDR adenhd, PAD_ENT aden);`

**Description**

Searches for the entity whose handle is `thehandle`, and returns it, if found, in the entity header pointed to by `adenhd`, and the entity body pointed to by `aden`. The current position in the list will be just past the object sought.

**Return Value**

Returns 1 on success, 0 on failure. On failure, `adError()` should be checked for an indication of the cause of the failure.

**See Also**

`adStartEntityGet`, `adGetEntity`, `adDeleteEntity`

**Example**

```c
adSeekEntity (handle, entlist, thehandle, &adenhd, &aden);
```

---

**adSeekLayer**

**Summary**

---
short adSeekLayer (AD_DB_HANDLE handle, AD_OBJHANDLE thehandle, PAD_LAY adlay);

Description

Searches for the layer whose handle is thehandle, and returns it, if found, in the buffer pointed to by adlay. The current position in the list will be just past the object sought.

Return Value

Returns 1 on success, 0 on failure. On failure, adError() should be checked for an indication of the cause of the failure.

See Also

adStartLayerGet, adGetLayer, adReplaceLayer, adDeleteLayer

Example

adSeekLayer (handle, thehandle, &lay);

adSeekLinetype

Summary

short adSeekLinetype (AD_DB_HANDLE handle, AD_OBJHANDLE thehandle, PAD_LTYPE adltp);

Description

Searches for the linetype whose handle is thehandle, and returns it, if found, in the buffer pointed to by adltp. The current position in the list will be just past the object sought.

Return Value

Returns 1 on success, 0 on failure. On failure, adError() should be checked for an indication of the cause of the failure.

See Also

adStartLinetypeGet, adGetLinetype, adReplaceLinetype, adDeleteLinetype

Example

adSeekLinetype (handle, thehandle, &ltp);

adSeekObject

Summary
short adSeekObject (AD_DB_HANDLE handle, AD_OBJHANDLE thehandle, PAD_OBJ_HDR adobhd, PAD_OBJ adob);

Description

Searches for the object whose handle is thehandle in the object list, and returns it in the structures pointed to by adobhd and adob. The current position in the list will be just past the object sought.

Return Value

Returns 1 on success, 0 on failure. On failure, adError() should be checked for an indication of the cause of the failure.

See Also

adStartObjectGet, adGetObject

Example

adSeekObject (handle, thehandle, &adobhd, &adob);

adSeekRegapp

Summary

short adSeekRegapp (AD_DB_HANDLE handle, AD_OBJHANDLE thehandle, PAD_APP theapp);

Description

Searches for the registered application whose handle is thehandle, and returns it, if found, in the buffer pointed to by adapp. The current position in the list will be just past the object sought.

Return Value

Returns 1 on success, 0 on failure. On failure, adError() should be checked for an indication of the cause of the failure.

See Also

adStartRegappGet, adGetRegapp, adReplaceRegapp, adDeleteRegapp

Example

adSeekRegapp (handle, thehandle, &app);
adSeekShapefile

Summary

short adSeekShapefile (AD_DB_HANDLE handle, AD_OBJHANDLE thehandle, PAD_SHPTB theshptb);

Description

Searches for the shapefile (style) entry whose handle is thehandle, and returns it, if found, in the buffer pointed to by theshptb. The current position in the list will be just past the object sought.

Return Value

Returns 1 on success, 0 on failure. On failure, adError() should be checked for an indication of the cause of the failure.

See Also

adStartShapefileGet, adGetShapefile, adReplaceShapefile, adDeleteShapefile

Example

adSeekShapefile(handle, thehandle, &shptb);

adSeekUcs

Summary

short adSeekUcs (AD_DB_HANDLE handle, AD_OBJHANDLE thehandle, PAD_UCS theucs);

Description

Searches for the user coordinate system whose handle is thehandle, and returns it, if found, in the buffer pointed to by theucs. The current position in the list will be just past the object sought.

Return Value

Returns 1 on success, 0 on failure. On failure, adError() should be checked for an indication of the cause of the failure.

See Also

adStartUcsGet, adGetUcs, adReplaceUcs, adDeleteUcs
Example

adSeekUcs(handle, thehandle, &ucs);

adSeekView

Summary

short adSeekView (AD_DB_HANDLE handle, AD_OBJHANDLE thehandle, PAD_VIEW theview);

Description

Searches for the view whose handle is thehandle, and returns it, if found, in the buffer pointed to by theview. The current position in the list will be just past the object sought.

Return Value

Returns 1 on success, 0 on failure. On failure, adError() should be checked for an indication of the cause of the failure.

See Also

adStartViewGet, adGetView, adReplaceView, adDeleteView

Example

adSeekView(handle, thehandle, &view);

adSeekVport

Summary

short adSeekVport (AD_DB_HANDLE handle, AD_OBJHANDLE thehandle, PAD_VPORT thevport);

Description

Searches for the vport whose handle is thehandle, and returns it, if found, in the buffer pointed to by thevport. The current position in the list will be just past the object sought.

Return Value

Returns 1 on success, 0 on failure. On failure, adError() should be checked for an indication of the cause of the failure.

See Also

adStartVportGet, adGetVport, adReplaceVport, adDeleteVport

Example
adSeekVport(handle,thehandle,&vport);

adSetAd2CloseInitFileFn

Summary

void adSetAd2CloseInitFileFn (void (*closeinitfile)(void));

Description

Sets the function to be called by OpenDWG Toolkit when the initialization file data load is completed. See the section "Loading Initialization File" for more information.

Return Value

There is no return value.

See Also

adSetAd2OpenInitFileFn, adSetAd2ReadInitFileCharFn, adSetAd2ReadInitFileCharsFn

Example

adSetAd2CloseInitFileFn(myclosefunction);

adSetAd2CriticalErrorFn

Summary

void adSetAd2CriticalErrorFn (short (*criticalerror)(short errorno));

Description

Sets the function to be called by OpenDWG Toolkit when a critical error is encountered. **errorno** indicates the type of error. Currently the defined error types are AD_CRITERR_FILEREADERROR, AD_CRITERR_MALLOCERROR, AD_CRITERR_DISKFULL, and AD_CRITERR_WRITEABORT. You should call this function BEFORE calling **adInitAd2()**.

It is VERY important that you install a critical error handler to handle these situations. See, for instance, adexr.c for an example of a critical error handler.

Return Value

There is no return value.
See Also

The example programs illustrate critical error handlers.

Example

```c
adSetAd2CriticalErrorFn (mycriticalerrorfunction);

short criterrhandler (short num)
{
    char ch;

    /* file read errors can be retried */
    if (num==AD_CRITERR_FILEREADERROR) {
        printf("File read error: Abort/Retry?\n");
        ch=fgetc(stdin);
        if (ch=='R' || ch=='r') return(0); /* return value ignored */
        exit(1);
    }

    /* disk full can be retried */
    if (num==AD_CRITERR_DISKFULL) {
        printf("Critical error: disk full\n");
        /* Returning 1 on a disk error means retry. Returning 0 means
        close the file and generate critical error AD_WRITE_ABORT (to allow
        a graceful exit). */
        return(0);
    }

    /* write abort means disk full was reached and the program requested */
    /* termination. This is a fatal error. */
    if (num==AD_CRITERR_WRITEABORT) {
        printf("Critical error: write abort\n");
        /* You should delete the partial file here. AD DOES NOT do so. */
        exit(1);
    }

    /* malloc errors are fatal */
    if (num==AD_CRITERR_MALLOCERROR) {
        printf("Critical error %d, aderror %d (%s), terminating\n", num, adError(), adErrorStr(adError()));
        exit(1);
    }

    /* as are buffer overruns */
    if (num==AD_CRITERR_BUFFEROVERRUN) {
        printf("Critical error -- buffer overrun, terminating\n");
        exit(1);
    }

    if (num==AD_CRITERR_BADDATA) {
        printf("Critical error -- bad data detected, terminating\n");
        exit(1);
    }
    return(0);
}
```

---

**adSetAd2DxfscanometerFn**

**Summary**

void adSetAd2DxfscanometerFn (void (*scanometer)(short percent));
Description

Sets the function to be called by OpenDWG Toolkit when a DXF file is being scanned. This function is passed a percentage-complete indicator that can be used to display the process of AD's initial scan of a DXF file.

Return Value

There is no return value.

See Also

adSetAd2LoadometerFn, adSetAd2SaveometerFn

Example

adSetAd2DxfscanometerFn(myscanometerfunction);

adSetAd2ExamineEntityLoadFn

Summary

void adSetAd2ExamineEntityLoadFn ( 
    void (*examineentity)(PAD_ENT_HDR adenhd, PAD_ENT aden));

Description

Sets the function to be called by OpenDWG Toolkit when an entity is being loaded. This function is passed pointers to the entity head and body.

Return Value

There is no return value.

See Also

None

Example

adSetAd2ExamineEntityLoadFn(myexaminefunction);
adSetAd2ExamineshapefileFn

Summary

void adSetAd2ExamineshapefileFn (void (*examineshapefile)(char *shapefile,
char *newshapefile, void *userdata));

Description

Sets the function to be called by OpenDWG Toolkit when a shape file is to be loaded to *examineshapefile*. *examineshapefile* should accept a char * argument which is the shapefile OpenDWG Toolkit needs to load; the full path should be returned in *newshapefile*. The returned path has a maximum of 256 characters. The *userdata* pointer is simply whatever pointer, if any, was set using adSetCallbackUserdata().

Return Value

There is no return value.

See Also

adSetCallbackUserdata

Example

adSetAd2ExamineshapefileFn(myexaminefunction);

adSetAd2LoadometerFn

Summary

void adSetAd2LoadometerFn (void (*loadometer)(short errorno));

Description

Sets the function to be called by OpenDWG Toolkit when a file is being loaded. This function is passed a percentage indicating how much of the file has been loaded, for use as a progress indicator.

Return Value

There is no return value.

See Also

adSetAd2SaveometerFn, adSetAd2DxfscanometerFn

Example

adSetAd2LoadometerFn(myloadometerfunction);
adSetAd2OpenInitFileFn

Summary

void adSetAd2OpenInitFileFn (void (*openinitfile)(void));

Description

Sets the function to be called by OpenDWG Toolkit when AD's init file data is to be opened. See the section "Loading Initialization File" and the function definition for adInitAd2() for further information.

Return Value

There is no return value.

See Also

adSetAd2CloseInitFileFn, adSetAd2ReadInitFileCharFn, adSetAd2ReadInitFileCharsFn

Example

adSetAd2OpenInitFileFn(myopeninitfilefunction);

adSetAd2ReadInitFileCharFn

Summary

void adSetAd2ReadInitFileCharFn (char (*readinitfilechar)(void));

Description

Sets the function to be called by OpenDWG Toolkit when it needs to read a single character of init file data. See the section "Loading Initialization File" and the function definition for adInitAd2() for further information.

Return Value

There is no return value.

See Also

adSetAd2CloseInitFileFn, adSetAd2ReadInitFileCharsFn, adSetAd2OpenInitFileFn

Example

adSetAd2ReadInitFileCharFn(myreadinitfilecharfunction);

adSetAd2ReadInitFileCharsFn

Summary
void adSetAd2ReadInitFileCharsFn (void (*readinitchars)(char *string, short num));

Description

Sets the function to be called by OpenDWG Toolkit when it needs to read several characters of init file data. See the section "Loading Initialization File" and the function definition for \texttt{adInitAd2()} for further information.

Return Value

There is no return value.

See Also

\texttt{adSetAd2CloseInitFileFn, adSetAd2ReadInitFileCharFn, adSetAd2OpenInitFileFn}

Example

\texttt{adSetAd2ReadInitFileCharsFn(myreadinitfilecharsfunction);} 

\texttt{adSetAd2SaveometerFn}

Summary

void adSetAd2SaveometerFn (void (*loadometer)(short errorno));

Description

Sets the function to be called by OpenDWG Toolkit when a file is being saved. This function is passed a percentage indicating how much of the file has been saved, for use as a progress indicator.

Return Value

There is no return value.

See Also

None

Example

\texttt{adSetAd2SaveometerFn(mysaveometerfunction);} 

\texttt{adSetAd2WarningFn}

Summary

void adSetAd2WarningFn (void (*warning)(short errorno));

Description


Sets the function to be called by OpenDWG Toolkit when a warning is generated. **Errorno** indicates the type of warning; right now the only defined value is 1 for a shape file which could not resolved for DXF output.

**Return Value**

There is no return value.

**See Also**

None

**Example**

`adSetAd2WarningFn(mywarning);`

---

### adSetCallbackUserdata

**Summary**

`void adSetCallbackUserdata (AD_DB_HANDLE handle, void *userdata);`

**Description**

Sets a pointer which is passed along with calls from OpenDWG Toolkit back to user defined functions. This pointer is simply supplied to user functions called from OpenDWG Toolkit during operation. The data is specific to the handle with which this function is called; that is, calling this function with different handles will set different userdata pointers for the various files which may be open.

The only function to which this applies is `adSetAd2ExamineShapefileFn`.

**Return Value**

There is no return value.

**See Also**

`adSetAd2ExamineShapefileFn`

**Example**

`adSetCallbackUserdata(mydata);`

---

### adSetDefaultDimstyle

**Summary**

`void adSetDefaultDimstyle (AD_DB_HANDLE handle, PAD_DIMSTYLE dimst);`

**Description**
Initializes the dimension style pointed to by `dimst` to the OpenDWG Toolkit defaults.

**Return Value**

None.

**See Also**

`adAddDimstyle`

**Example**

```c
adSetDefaultDimstyle(handle, &dimst);
```

---

**adSetDefaultDimstyleFromHeader**

**Summary**

```c
void adSetDefaultDimstyleFromHeader (AD_DB_HANDLE handle, PAD_DIMSTYLE dimst);
```

**Description**

Initializes the dimension style pointed to by `dimst` to the values stored in the header. This function is used internally by OpenDWG Toolkit to create a STANDARD dimstyle for files prior to Release 13, which do not have one.

**Return Value**

None.

**See Also**

`adAddDimstyle`

**Example**

```c
adSetDefaultDimstyleFromHeader(handle, &dimst);
```

---

**adSetDefaultLayer**

**Summary**

```c
void adSetDefaultLayer (AD_DB_HANDLE handle, PAD_LAY lay);
```

**Description**

Initializes the layer pointed to by `lay` to the OpenDWG Toolkit defaults.

**Return Value**

None.
See Also

adAddLayer

Example

adSetDefaultLayer(AD_DB_HANDLE handle, &lay);

---

adSetDefaultLinetype

Summary

void adSetDefaultLinetype (PAD_LTYPE ltp);

Description

Initializes the linetype pointed to by ltp to the OpenDWG Toolkit defaults.

Return Value

None.

See Also

adAddLinetype

Example

adSetDefaultLinetype(&ltp);

---

adSetDefaultMlstyle

Summary

void adSetDefaultMlstyle(PAD_MLSTYLE mlstyle);

Description

Initializes the multi-line style pointed to by mlstyle to the OpenDWG Toolkit defaults.

Return Value

None.

See Also

adAddMlstyle

Example

adSetDefaultMlstyle(&mlstyle);
**adSetDefaultRegapp**

**Summary**

    void adSetDefaultRegapp (PAD_APP app);

**Description**

Initializes the registered application pointed to by `app` to the OpenDWG Toolkit defaults.

**Return Value**

None.

**See Also**

`adAddRegapp`

**Example**

    adSetDefaultRegapp(&app);

---

**adSetDefaultShapefile**

**Summary**

    void adSetDefaultShapefile (PAD_SHPTB shptb);

**Description**

Initializes the shapefile(style) pointed to by `shptb` to the OpenDWG Toolkit defaults.
Summary

void adSetDefaultUcs (PAD_UCS ucs);

Description

Initializes the UCS pointed to by ucs to the OpenDWG Toolkit defaults.

Return Value

None.

See Also

adAddShapefile

Example

adSetDefaultShapefile(&shptb);

adSetDefaultView

Summary

void adSetDefaultView (PAD_VIEW vie);

Description

Initializes the view pointed to by vie to the OpenDWG Toolkit defaults.

Return Value

None.
See Also

adAddView

Example

adSetDefaultView(&view);

\textbf{adSetDefaultVport}

\textbf{Summary}

void adSetDefaultVport (PAD_VPORT vpo);

\textbf{Description}

Initializes the vport pointed to by vpo to the OpenDWG Toolkit defaults.

\textbf{Return Value}

None.

\textbf{See Also}

adAddVport

Example

adSetDefaultVport(&vport);

\textbf{adSetDxfWriteCompleteness}

\textbf{Summary}

void adSetDxfWriteCompleteness(short completeness);

\textbf{Description}

Controls the completeness of DXF files written by OpenDWG Toolkit. Note that we recommend always writing complete DXF files. However, in those circumstances where it is necessary to write a less-than-complete file, you can call this function to tell OpenDWG Toolkit to write a less-than-complete DXF file. Call this function with one of the following defined constants:

- **AD_DXF_COMPLETE**: complete dxf file
- **AD_DXF_MINIMALHEADER1**: header has extents and version only
- **AD_DXF_MINIMALHEADER2**: header has extents only
- **AD_DXF_ENTITIESONLY**: entities only

\textbf{Return Value}
None.

See Also

None

Example

adSetDxfWriteCompleteness(AD_DXF_ENTITIESONLY);

---

**adSetEntityDefaults**

**Summary**

void adSetEntityDefaults (AD_DB_HANDLE handle, PAD_ENT_HDR adenhd, PAD_ENT aden);

**Description**

Initializes the entity header pointed to by `adenhd` and the entity body pointed to by `aden` to the OpenDWG Toolkit defaults. Note that `adenhd->enttype` must be set to the type of the entity to be initialized. You should also be sure to have valid layers and linetypes in the drawing, as OpenDWG Toolkit will use them to set the defaults.

**Return Value**

None.

See Also

`adAddEntity`

**Example**

```
adhd.enttype=AD_ENT_LINE;
adSetEntityDefaults(handle, &adenhd, &aden);
```

---

**adSetHandseed**

**Summary**

void adSetHandseed (AD_DB_HANDLE handle, AD_OBJHANDLE newhandseed);

---
**Description**

Sets the handseed for the file indicated by `handle` to `newhandseed`. The handseed value is the value of the next handle to be used when new objects are added to the database. Use this function with caution; it would probably be a VERY BAD IDEA to set this value to a handle lower than that of objects already in the database, unless you are certain that you will be removing those objects.

**Return Value**

None.

**See Also**

`adSetNewFileStartHandle`

**Example**

```c
/* note the 35 is 29 in octal */
    adSetHandseed(handle,"\0\0\0\0\0\0\0\035");
```

---

**adSetHeaderDefaults**

---

**Summary**

`void adSetHeaderDefaults (PAD_DWGHDR adhd);`

**Description**

Initializes the header structure pointed to by `adhd` to the OpenDWG Toolkit defaults.

**Return Value**

None.

**See Also**

`adHeaderPointer, adReadHeaderBlock, adWriteHeaderBlock`

**Example**

```c
    adSetHeaderDefaults(&adhd);
```

---

**adSetLayerState**

---

**Summary**

`short adSetLayerState (AD_DB_HANDLE handle, AD_OBJHANDLE layhandle, char on, char frozen, char vpfrozen, char locked);`

**Description**
Sets the state of the layer indicated by layhandle.

**Return Value**

Returns 1 on success, 0 on failure. On failure, adError() should be checked for an indication of the cause of the failure. On success, the layer is set to the states defined by on, frozen, vpfrozen, and locked. Each of those values should be passed in as 1 for true, 0 for false, or AD_LAYER_NOCHANGE if no change is desired.

**See Also**

adGetLayerState

**Example**

```c
printf("LAYERS:\n")
adStartLayerGet(dwghandle);
for (i=0; i<(short)adNumLayers(dwghandle); i++) {
adGetLayer(dwghandle,&adtb->lay);
/* set layers on and thawed */
adSetLayerState(dwghandle,adtb->lay.objhandle,
1,0,0,0);
}
```

### adSetNewFileStartHandle

**Summary**

void adSetNewFileStartHandle (AD_OBJHANDLE nfsh);

**Description**

Sets the starting handle to be used for files created with AD. This allows you to set these handles above other handles which you may wish to preserve as you bring data into a new file.

**Return Value**

There is no return value.

**See Also**

adNewFile

**Example**

```c
AD_OBJHANDLE newhandle;
adHanclear(newhandle);
newhandle[7]=254; /* reserve the bottom 253 handles */
adSetNewFileStartHandle(newhandle);
```

### adSetObjectDefaults

**Summary**
void adSetObjectDefaults (AD_DB_HANDLE handle, PAD_OBJ_HDR adobhd, PAD_OBJ adob);

Description

Initializes the object header pointed to by adobhd and the object body pointed to by adob to the OpenDWG Toolkit defaults. Note that adobhd->objtype must be set to the type of the object to be initialized.

Return Value

None.

See Also

adAddEntity

Example

AD_OBJ_HDR adobhd;
AD_OBJ adob;

adobhd.enttype=AD_OBJ_DICTIONARY;
adSetObjectDefaults(handle,&adobhd,&adob);

adSetupDwgRead

Summary

void adSetupDwgRead (void);

Description

Sets up OpenDWG Toolkit to read DWG files. This function should be executed after adInitAd2 but before any DWG files are operated on.

Return Value

There is no return value.

See Also

adSetupDxfRead

Example

adSetupDwgRead();

adSetupDxfRead

Summary

void adSetupDxfRead (void);
Description

Sets up OpenDWG Toolkit to read DXF files. This function should be executed after \texttt{adInitAd2} but before any DXF files are operated on.

Return Value

There is no return value.

See Also

\texttt{adSetupDwgRead}

Example

\begin{verbatim}
adSetupDxfRead();
\end{verbatim}

\textbf{adShadowObjhandlePtr}

\begin{verbatim}
Summary

AD_OBJHANDLE *adShadowObjhandlePtr(AD_DB_HANDLE handle);

Description

Returns a pointer to several AD_OBJHANDLEs which are shadowed by OpenDWG Toolkit. OpenDWG Toolkit keeps several AD_OBJHANDLEs which are commonly needed in this array. Note that OpenDWG Toolkit does not automatically update these shadowed object handles. Thus, for instance, if you decide to modify the BYBLOCK linetype's object handle, you should also change the object handle shadowed in this array.

You can use the following defined constants as indices against the pointer returned by this function:

\begin{verbatim}
MLINEDICSHADOW                   multi line style dictionary object
GROUPDICSHADOW                   group dictionary object
DICTIONARYSHADOW                 main dictionary shadow
MSPACEBLKSHADOW                  modelspace blockheader
PSPACEBLKSHADOW                  paperspace blockheader
BYBLOCKLTPSHADOW                 byblock linetype
BYLAYERLTPSHADOW                 bylayer linetype
MSPACEBLKENTSHPADSHADOW          modelspace block entity
MSPACEENDBLKENTSHPADSHADOW       modelspace end block entity
PSPACEBLKENTSHPADSHADOW          paperspace block entity
PSPACEENDBLKENTSHPADSHADOW       paperspace end block entity
\end{verbatim}

NOTE – This function will probably go away. We strongly suggest that rather than using these shadows, you actually search the lists for the object handles as needed, except that there is no way to search for the final 4 items (modelspace and paperspace start and end block entities), so it is legitimate to use this function to access those, although we may provide a function specifically to access them in the future.

Return Value

\end{verbatim}
Returns a pointer to an array of AD_OBJHANDLEs.

See Also

adCtrlObjhandlePtr

Example

```c
AD_OBJHANDLE *ohptr;

ohptr=adShadowObjhandlePtr(handle);
/* now, for instance, the byblock linetype objhandle
   is available as ohptr[BYBLOCKLTPSHADOW] */
```

adShortCodePageToAscii

Summary

```c
short  adShortCodePageToAscii(short index, char *codepagestr);
```

Description

Sets the value pointed to by `codepagestr` to the name of the code page indicated by `index`.

Return Value

Returns 1 on success, 0 on failure. On success, `codepagestr` points to the code page name.

See Also

adAsciiCodePageToShort

Example

```c
char codepagestr[30];

success=adShortCodePageToAscii(adhd->dwgcodepage,codepagestr);
```

adSortentstableObjtype

Summary

```c
short adSortentstableObjtype(AD_DB_HANDLE handle);
```

Description

Returns the object type value for AD_SORTENTSTABLE objects.

Return Value

Returns the object type value for AD_SORTENTSTABLE objects.
adSpatialfilterObjtype

Summary

short adSpatialfilterObjtype(AD_DB_HANDLE handle);

Description

Returns the object type value for AD_SPATIALFILTER objects.

Return Value

Returns the object type value for AD_SPATIALFILTER objects.

See Also

None

Example

adobhd.objtype=adSpatialfilterObjtype(handle);
**adSpatialindexObjtype**

**Summary**

```
short adSpatialindexObjtype(AD_DB_HANDLE handle);
```

**Description**

Returns the object type value for AD_SPATIALINDEX objects.

**Return Value**

Returns the object type value for AD_SPATIALINDEX objects.

**See Also**

None

**Example**

```
adobhd.objtype=adSpatialindexObjtype(handle);
```

**adStartBlobAppend**

**Summary**

```
PAD_BLOB_CTRL adStartBlobAppend (AD_VMADDR blob);
```

**Description**

Initiates append to an OpenDWG Toolkit blob. Also allocates a memory buffer for appending to this blob. You should run `adEndBlobAppend` when you are through appending to the blob, to deallocate the memory.

**Return Value**

Returns a pointer to an AD_BLOB_CTRL, an OpenDWG Toolkit blob control object; or NULL on failure.

**See Also**

`adWriteBlobBytes`, `adEndBlobAppend`

**Example**

```
AD_VMADDR theblob;
PAD_BLOB_CTRL bcptr;
bcptr=adStartBlobAppend(theblob);
```

**adStartBlobRead**
Summary

    PAD_BLOB_CTRL adStartBlobRead (AD_VMADDR blob);

Description

    Initiates read of an OpenDWG Toolkit blob. Also allocates a memory buffer for reads from this blob. You should run adEndBlobRead when you are through reading the blob, to deallocate the memory.

Return Value

    Returns a pointer to an AD_BLOB_CTRL, an OpenDWG Toolkit blob control object; or NULL on failure.

See Also

    adReadBlobBytes, adEndBlobRead

Example

    AD_VMADDR theblob;
    PAD_BLOB_CTRL bcptr;
    bcptr=adStartBlobRead(theblob);

adStartBlobWrite

Summary

    PAD_BLOB_CTRL adStartBlobWrite (AD_VMADDR blob);

Description

    Initiates write of an OpenDWG Toolkit blob. Also allocates a memory buffer for writes to this blob. You should run adEndBlobWrite when you are through writing the blob, to deallocate the memory.

Return Value

    Returns a pointer to an AD_BLOB_CTRL, an OpenDWG Toolkit blob control object, or NULL on failure.

See Also

    adWriteBlobBytes, adEndBlobWrite
AD_VMADDR theblob;
PAD_BLOB_CTRL bcptr;

theblob=adCreateBlob();
bcptr=adStartBlobWrite(theblob);
adWriteBlobBytes(bcptr, buf, num);

adStartBlockheaderGet

Summary

short adStartBlockheaderGet (AD_DB_HANDLE handle);

Description

Rewinds the blockheader list for subsequent calls to adGetBlockheader.

Return Value

Returns 1 on success, 0 on failure. On failure, adError() should be checked for an indication of the cause of the failure.

See Also

adGetBlockheader

Example

See adGetBlockheader for an example.

adStartClassGet

Summary

short adStartClassGet (AD_DB_HANDLE handle);

Description

Rewinds the class list for subsequent calls to adGetClass.

Return Value

Returns 1 on success, 0 on failure. On failure, adError() should be checked for an indication of the cause of the failure.

See Also

adGetClass

Example
See `adGetClass` for an example.

**adStartDimstyleGet**

**Summary**

short adStartDimstyleGet (AD_DB_HANDLE handle);

**Description**

Rewinds the dimension style list for subsequent calls to `adGetDimstyle`.

**Return Value**

Returns 1 on success, 0 on failure. On failure, `adError()` should be checked for an indication of the cause of the failure.

**See Also**

`adGetDimstyle`

**Example**

See `adGetDimstyle` for an example.

**adStartEntityGet**

**Summary**

short adStartEntityGet (AD_VMADDR list);

**Description**

Rewinds the entity list indicated by list for subsequent calls to `adGetEntity`.

**Return Value**

Returns 1 on success, 0 on failure. On failure, `adError()` should be checked for an indication of the cause of the failure.

**See Also**

`adGetEntity`

**Example**

See `adGetEntity` for an example.
adStartGrblobDataRead

Summary

short adStartGrblobDataRead (PAD_BLOB_CTRL bcptr);

Description

Initiates read of a proxy entity graphics data blob.

Return Value

Returns 1 on success, 0 on failure. On failure, adError() should be checked for an indication of the cause of the failure.

See Also

adReadGrblobData

Example

See adReadGrblobData for an example.

adStartGrblobDataWrite

Summary

short adStartGrblobDataWrite (PAD_BLOB_CTRL bcptr);

Description

Initiates write of a proxy entity graphics data blob.

Return Value

Returns 1 on success, 0 on failure. On failure, adError() should be checked for an indication of the cause of the failure.

See Also

adWriteGrblobData

Example

See adWriteGrblobData for an example, and adexzo.c
adStartLayerGet

Summary

    short adStartLayerGet (AD_DB_HANDLE handle); Description  Rewinds the layer list for subsequent calls to adGetLayer.

Return Value  Returns 1 on success, 0 on failure. On failure, adError() should be checked for an indication of the cause of the failure.

See Also

    adGetLayer

Example

    See adGetLayer for an example.

adStartLinetypeGet

Summary

    short adStartLinetypeGet (AD_DB_HANDLE handle);

Description

    Rewinds the linetype list for subsequent calls to adGetLinetype.

Return Value

    Returns 1 on success, 0 on failure. On failure, adError() should be checked for an indication of the cause of the failure.

See Also

    adGetLinetype

Example

    See adGetLinetype for an example.
adStartObjectGet

Summary
short adStartObjectGet (AD_DB_HANDLE handle);

Description
Rewinds the object list for subsequent calls to adGetObject.

Return Value
Returns 1 on success, 0 on failure. On failure, adError() should be checked for an indication of the cause of the failure.

See Also
adGetObject

Example
See adGetObject for an example.

adStartRegappGet

Summary
short adStartRegappGet (AD_DB_HANDLE handle);

Description
Rewinds the registered application list for subsequent calls to adGetRegapp.

Return Value
Returns 1 on success, 0 on failure. On failure, adError() should be checked for an indication of the cause of the failure.

See Also
adGetRegapp

Example
See adGetRegapp for an example.
adStartShapefileGet

Summary

short adStartShapefileGet (AD_DB_HANDLE handle);

Description

Rewinds the shapefile (style) list for subsequent calls to \texttt{adGetShapefile}.

Return Value

Returns 1 on success, 0 on failure. On failure, \texttt{adError()} should be checked for an indication of the cause of the failure.

See Also

\texttt{adGetShapefile}

Example

See \texttt{adGetShapefile} for an example.

adStartUcsGet

Summary

short adStartUcsGet (AD_DB_HANDLE handle);

Description

Rewinds the user coordinate system list for subsequent calls to \texttt{adGetUcs}.

Return Value

Returns 1 on success, 0 on failure. On failure, \texttt{adError()} should be checked for an indication of the cause of the failure.

See Also

\texttt{adGetUcs}

Example

See \texttt{adGetUcs} for an example.
adStartViewGet

Summary

short adStartViewGet (AD_DB_HANDLE handle);

Description

Rewinds the view list for subsequent calls to adGetView.

Return Value

Returns 1 on success, 0 on failure. On failure, adError() should be checked for an indication of the cause of the failure.

See Also

adGetView

Example

See adGetView for an example.

adStartVportGet

Summary

short adStartVportGet (AD_DB_HANDLE handle);

Description

Rewinds the vport list for subsequent calls to adGetVport.

Return Value

Returns 1 on success, 0 on failure. On failure, adError() should be checked for an indication of the cause of the failure.

See Also

adGetVport

Example

See adGetVport for an example.
adStartProxyDataRead

Summary
short adStartProxyDataRead (AD_DB_HANDLE handle, AD_VMADDR proxydata);

Description
Initiates read of proxy (object or entity) data.

Return Value
Returns 1 on success, 0 on failure. On failure, adError() should be checked for an indication of the cause of the failure.

See Also
adReadProxyData...

Example
See adReadProxyDataBinary for an example.

adStartProxyDataWrite

Summary
short adStartProxyDataWrite (AD_DB_HANDLE handle);

Description
Initiates write of proxy (object or entity) data.

Return Value
Returns 1 on success, 0 on failure. On failure, adError() should be checked for an indication of the cause of the failure.

See Also
adWriteProxyData..., adEndProxyDataWrite

Example
adSetEntityDefaults(wrhandle,adenhd,aden);
adStartProxyDataWrite(wrhandle);
/* we write some random proxy data here. You would substitute the appropriate data for your application */
adWriteProxyDataString("a test string");
pt[0]=1.0; pt[1]=2.0; pt[2]=3.0;
adWriteProxyDataPoint3d(pt);
aden->proxyent.entdatablob=
adEndProxyDataWrite(wrhandle,&aden->proxyent.entdatabits);
**adStopLoading**

**Summary**

```c
void adStopLoading (void);
```

**Description**

Terminates a file load. Normally you would call this function during a callout from OpenDWG Toolkit such as the ones established by `adSetAd2ExamineEntityLoadFn()` or `adSetAd2LoadometerFn()`. Note that the load does not terminate immediately; rather `adStopLoading()` will return to your calling function, which should in turn return to AD. Load will stop at AD's next internal checkpoint.

**Return Value**

There is no return value.

**See Also**

`adSetAd2ExamineEntityLoadFn`, `adSetAd2LoadometerFn`

**Example**

```c
adSetAd2LoadometerFn(myloadometer);
```

```c
...  
void myloadometer(short percent)
{
    printf("percent complete: %d\n",percent);
    if (kbhit()) adStopLoading();
}
```

---

**adTextBoundingBox**

**Summary**

```c
short  adTextBoundingBox(AD_DB_HANDLE,unsigned char *textstr,PAD_TDATA tdata, short vertflag,
                        char *shfilename, char *ffilename, short usesimplex, PDOUBRECTFULL textbb, double *bottomy);
```

**Description**

Returns the bounding box for a text string, and the low descender y value. Parameters are:

- **handle** the drawing handle
- **textstr** the text string
- **tdata** pointer to the text draw data
- **vertflag** text is vertical (if 1), horiz if 0.
- **shfilename** name of the font file
- **bffilename** name of the bigfont, if any (NULL if none)
- **usesimplex** 1==use AD emulator font if font cannot be found, 0==fail if font cannot be found.
- **textbb** receives the bounding box for the text (baseline to top, full width)
- **bottomy** the maximum extent of the text downward, below
the baseline.

Return Value

Returns 1 on success, 0 on failure.

See Also

None

Example

DoubRectFull thebb;
double bottomy;

/* set up desired tdata values first, then */
success=adTextBoundingBox(handle,"test string",
&tdata,0,"SIMPLEX",NULL,&thebb,&bottomy);

adUnicodeToCodePageChar

Summary

short adUnicodeToCodePageChar(unsigned short unicodechar, short codepage, short *codepagechar);

Description

Translates a Unicode value to the corresponding code page character.

Return Value

Returns 1 on success, 0 on failure. On success the value pointed to by codepagechar contains the translated character.

See Also

adCodePageCharToUnicode

dexample

Example

short codepage;
short codepagechar;

success=adAsciiCodePageToShort("dos850", &codepage);
success=adUnicodeToCodePageChar(0x2205, codepage, &codepagechar);

adValHdrStr

Summary

short adValHdrStr (char *acadverr);

Description
Indicates the version of the header version string passed in.

**Return Value**

Returns -1 if the string is not from a version of AutoCAD which is supported by OpenDWG Toolkit, otherwise returns one of the defined constants AD_ACAD25, AD_ACAD26, AD_ACAD9, AD_ACAD10, AD_ACAD11, AD_ACAD13, or AD_ACAD14.

**See Also**

None

**Example**

```c
printf("acad version index is %d\n",adValHdrStr("AC1002"));
```

---

### `adVbaProjectObjtype`

**Summary**

```c
short adVbaProjectObjtype (AD_DB_HANDLE handle);
```

**Description**

Returns the object type value for AD_VBAPROJECT objects.

**Return Value**

Returns the object type value for AD_VBAPROJECT objects.

**See Also**

None

**Example**

```c
adobhd.objtype=adVbaProjectObjtype(handle);
```

---

### `adWipeoutEnttype`

**Summary**

```c
short adWipeoutEnttype(AD_DB_HANDLE handle);
```

**Description**

Returns the entity type value for AD_WIPEOUT entities.

**Return Value**
Returns the entity type value for AD_WIPEOUT entities.

See Also

None

Example

```c
wipeoutenttype=adWipeoutEnttype(handle);
```

adWipeoutVariablesObjtype

Summary

```c
short adWipeoutVariablesObjtype (AD_DB_HANDLE handle);
```

Description

Returns the object type value for AD_WIPEOUTVARIABLES objects.

Return Value

Returns the object type value for AD_WIPEOUTVARIABLES objects.

See Also

None

Example

```c
adobhd.objtype=adWipeoutVariablesObjtype(handle);
```

adWriteAcisString

Summary

```c
short adWriteAcisString (PAD_BLOB_CTRL bcptr, char *str);
```

Description

Writes an ACIS string to the blob indicated by `bcptr`.

Return Value

Returns 1 on success, 0 on failure. On failure, `adError()` should be checked for an indication of the cause of the failure.

See Also

`adStartBlobWrite`, `adEndBlobWrite`, `adCreateBlob`
Example

    adWriteAcisString(bcptr,"105 661 1 0");
adWriteBlob2Double

**Summary**

short adWriteBlob2Double (PAD_BLOB_CTRL bcptr, double *doub);

**Description**

Writes two doubles pointed to by **doub** to the blob indicated by **bcptr**.

**Return Value**

Returns 1 on success, 0 on failure. On failure, **adError()** should be checked for an indication of the cause of the failure.

**See Also**

adStartBlobWrite, adEndBlobWrite, adCreateBlob

**Example**

```c
AD_VMADDR theblob;
PAD_BLOB_CTRL bcptr;
double doub[2];
doub[0]=8.0; doub[1]=3.0;
theblob=adCreateBlob();
bcptr=adStartBlobWrite(theblob);
adWriteBlob2Double(bcptr,doub);
```

adWriteBlob3Double

**Summary**

short adWriteBlob3Double (PAD_BLOB_CTRL bcptr, double *doub);

**Description**

Writes three doubles pointed to by **doub** to the blob indicated by **bcptr**.

**Return Value**

Returns 1 on success, 0 on failure. On failure, **adError()** should be checked for an indication of the cause of the failure.

**See Also**

adStartBlobWrite, adEndBlobWrite, adCreateBlob
Example

```c
AD_VMADDR theblob;
PAD_BLOB_CTRL bcptr;
double doub[3];

doub[0]=8.0; doub[1]=3.0; doub[2]=2.0;

theblob=adCreateBlob();
bcptr=adStartBlobWrite(theblob);
adWriteBlob3Double(bcptr,doub);
```

### adWriteBlobByte

**Summary**

short adWriteBlobByte (PAD_BLOB_CTRL bcptr, char ch);

**Description**

Writes a single byte ch to the blob indicated by bcptr.

**Return Value**

Returns 1 on success, 0 on failure. On failure, adError() should be checked for an indication of the cause of the failure.

**See Also**

adStartBlobWrite, adEndBlobWrite, adCreateBlob

**Example**

```c
AD_VMADDR theblob;
PAD_BLOB_CTRL bcptr;

theblob=adCreateBlob();
bcptr=adStartBlobWrite(theblob);
adWriteBlobByte(bcptr,'A');
```

### adWriteBlobBytes

**Summary**

short adWriteBlobBytes (PAD_BLOB_CTRL bcptr, char *buf, short num);

**Description**

Writes num bytes from buffer buf to the blob indicated by bcptr.
Return Value

Returns 1 on success, 0 on failure. On failure, adError() should be checked for an indication of the cause of the failure.

See Also

adStartBlobWrite, adEndBlobWrite, adCreateBlob

Example

AD_VMADDR theblob;
PAD_BLOB_CTRL bcptr;
theblob=adCreateBlob();
bcptr=adStartBlobWrite(theblob);
adWriteBlobBytes(bcptr,"AAA",3);

adWriteBlobDouble

Summary

short adWriteBlobDouble (PAD_BLOB_CTRL bcptr, double *doub);

Description

 Writes the double pointed to by doub to the blob indicated by bcptr.

Return Value

Returns 1 on success, 0 on failure. On failure, adError() should be checked for an indication of the cause of the failure.

See Also

adStartBlobWrite, adEndBlobWrite, adCreateBlob

Example

AD_VMADDR theblob;
PAD_BLOB_CTRL bcptr;
double dSub;

doub=8.0;
theblob=adCreateBlob();
bcptr=adStartBlobWrite(theblob);
adWriteBlobDouble(bcptr,&doub);
adWriteBlobLong

Summary

short adWriteBlobLong (PAD_BLOB_CTRL bcptr, long *l);

Description

Writes the long pointed to by l to the blob indicated by bcptr.

Return Value

Returns 1 on success, 0 on failure. On failure, adError() should be checked for an indication of the cause of the failure.

See Also

adStartBlobWrite, adEndBlobWrite, adCreateBlob

Example

AD_VMADDR theblob;
PAD_BLOB_CTRL bcptr;
long l;

l=8L;
theblob=adCreateBlob();
bcptr=adStartBlobWrite(theblob);
adWriteBlobLong(bcptr,&l);

adWriteBlobObjhandle

Summary

short adWriteBlobObjhandle (PAD_BLOB_CTRL bcptr,AD_OBJHANDLE thehandle);

Description

Writes an AD_OBJHANDLE to the blob indicated by bcptr.

Return Value

Returns 1 on success, 0 on failure. On failure, adError() should be checked for an indication of the cause of the failure.

See Also

adReadBlobObjhandle
Example

PAD_BLOB_CTRL bcptr;
AD_OBJHANDLE thehandle;
AD_VMADDR theblob;

theblob=adCreateBlob();
bcptr=adStartBlobWrite(theblob);
adWriteBlobObjhandle(theblob, thehandle);

adWriteBlobShort

Summary

short adWriteBlobShort (PAD_BLOB_CTRL bcptr, short *s);

Description

Writes the short pointed to by s to the blob indicated by bcptr.

Return Value

Returns 1 on success, 0 on failure. On failure, adError() should be checked for an indication of the cause of the failure.

See Also

adStartBlobWrite, adEndBlobWrite, adCreateBlob

Example

AD_VMADDR theblob;
PAD_BLOB_CTRL bcptr;
short s;

s=8L;
theblob=adCreateBlob();
bcptr=adStartBlobWrite(theblob);
adWriteBlobShort(bcptr, &s);

adWriteBlobString

Summary

short adWriteBlobString (PAD_BLOB_CTRL bcptr, char *str);

Description

Writes a string to the blob indicated by bcptr into str.
Return Value

Returns 1 on success, 0 on failure. On failure, adError() should be checked for an indication of the cause of the failure.

See Also

adReadBlobString, adStartBlobWrite

Example

PAD_BLOB_CTRL bcptr;
bcptr=adStartBlobWrite(theblob);
adWriteBlobString(theblob,"THIS IS A TEST");

adWriteDicItem

Summary

short adWriteDicItem (PAD_BLOB_CTRL bcptr, PAD_DICITEM dicitem);

Description

Writes a dictionary item to the blob indicated by bcptr.

Return Value

Returns 1 on success, 0 on failure. On failure, adError() should be checked for an indication of the cause of the failure.

See Also

adStartBlobWrite, adEndBlobWrite, adCreateBlob

Example

adWriteDicItem(bcptr,&dicitem);

adWriteExtendedData

Summary

short adWriteExtendedData (PAD_BLOB_CTRL bcptr, PAD_XD adxd);

Description

Writes an extended data item to the blob indicated by bcptr.
Return Value

Returns 1 on success, 0 on failure. On failure, adError() should be checked for an indication of the cause of the failure.

See Also

adStartBlobWrite, adEndBlobWrite, adCreateBlob

Example

adenhd->xdblob=adCreateBlob();
bcptr=adStartBlobWrite(adenhd->xdblob);

adx->xdxfnumber=1001;
adHancpy(adx->xddata.xdappobjhandle,acadappobjhandle);
adWriteExtendeddata(bcptr,adx);

adx->xdxfnumber=1000;
strcpy(adx->xddata.xdstring,"MVIEW");
adWriteExtendeddata(bcptr,adx);

adWriteGrblobData

Summary

short adWriteGrblobData (PAD_BLOB_CTRL bcptr, PAD_GR_DATA data);

Description

Writes an data to the proxy entity graphics blob indicated by bcptr.

Return Value

Returns 1 on success, 0 on failure. On failure, adError() should be checked for an indication of the cause of the failure.

See Also

adStartBlobWrite, adEndBlobWrite, adCreateBlob

Example

/* here are some sample graphic image writes */
aden->proxyent.grblob=adCreateBlob();
bcptr=adStartBlobWrite(aden->proxyent.grblob);
adStartGrblobDataWrite(bcptr);
/* circle has no "additional" data */
data.grtype=AD_GRENT_CIRCLE;
data.grbody.circle.pt0[0]=1.0;
data.grbody.circle.pt0[1]=2.0;
data.grbody.circle.pt0[2]=
data.grbody.circle.normal[0]=
data.grbody.circle.normal[1]=0.0;
data.grbody.circle.normal[2]=1.0;
data.grbody.circle.radius=3.0;
adWriteGrblobData(bcptr,&data);

adWriteGrblobVertexPt

Summary

short adWriteGrblobVertexPt (PAD_BLOB_CTRL bcptr, double *pt);

Description

Writes an 3D point to the proxy entity graphics blob indicated by bcptr. Generally this is used to add the vertices to a polyline or polygon.

Return Value

Returns 1 on success, 0 on failure. On failure, adError() should be checked for an indication of the cause of the failure.

See Also

adStartBlobWrite, adEndBlobWrite, adCreateBlob

Example

/* here are some sample graphic image writes */
aden->proxyent.grblob=adCreateBlob();
bcptr=adStartBlobWrite(aden->proxyent.grblob);
adStartGrblobDataWrite(bcptr);

/* polyline has vertices following */
data.grtype=AD_GRENT_POLYLINE;
data.grbody.pline.numpts=5L;
adWriteGrblobData(bcptr,&data);

for (i=0; i<5; i++) {
    adWriteGrblobVertexPt(bcptr,pt);
    pt[0]+=1.0;
    pt[1]=1.0-pt[1];
}

adWriteGrshellFaceEntries

Summary
short adWriteGrshellFaceEntries(PAD_BLOB_CTRL bcptr, long numentries, long *faceentry);

**Description**

Writes the face entries for a data item of type AD_GRENT_SHELL to the proxy entity graphical representation blob indicated by `bcptr`.

**Return Value**

Returns 1 on success, 0 on failure. On failure, `adError()` should be checked for an indication of the cause of the failure.

**See Also**

`adHasPrimTraits, adPrimsHave..., adReadPrim...`

**Example**

```c
adWriteGrshellNumFaceEntries(bcptr,&numfaceentries);
adWriteGrshellFaceEntries(bcptr,numfaceentries,&entry);
```

---

**adWriteHeaderBlock**

**Summary**

```c
short adWriteHeaderBlock (AD_DB_HANDLE handle, PAD_DWGHDR adhd);
```

**Description**

Writes the header data pointed to by `adhd` into the drawing. **NOTE – THIS IS AN OBSOLETE FUNCTION. YOU SHOULD USE adHeaderPointer() INSTEAD.**

**Return Value**

Returns 1 on success, 0 on failure. On failure, `adError()` should be checked for an indication of the cause of the failure.

**See Also**

`adReadHeaderBlock`

**Example**

```c
adReadHeaderBlock(handle,&adhd);
adhd.tilemode=0;
adWriteHeaderBlock(handle,&adhd);
```

---

**adWriteMlstyleSeginfo**

**Summary**

```c
short adWriteMlstyleSeginfo(PAD_BLOB_CTRL bcptr, PAD_MLSTYLESEGINFO seginfo);
```
Description

Writes a set of multi-line style segment info to the blob indicated by bcptr.

Return Value

Returns 1 on success, 0 on failure. On failure, adError() should be checked for an indication of the cause of the failure.

See Also

adStartBlobWrite, adEndBlobWrite, adCreateBlob

Example

adSetDefaultMlstyle(&mlstyle);
strcpy(mlstyle.name,"STANDARD");
if ((mlstyle.seginfoblob=adCreateBlob())==AD_VMNULL) {
    deleteobjectlist(handle);
    return(0);
}

bcptr=adStartBlobWrite(mlstyle.seginfoblob);
adHancpy(mlstyle.objhandle,mlstylehandle);
adHancpy(adhd->curmlstyleobjhandle,mlstylehandle);
mlstyle.fillcolor=0;
mlstyle.numreactors=1;
seginfo.offset=0.5;
seginfo.ltindex=AD_LTP_BYLAYER;
seginfo.segcolor=AD_COLOR_BYLAYER;
if (adWriteMlstyleSeginfo(bcptr,&seginfo)) {
    seginfo.offset=-0.5;
    if (adWriteMlstyleSeginfo(bcptr,&seginfo)) {
        dic.numdicitems=2;
        adEndBlobWrite(bcptr);
        if (adAddMlstyle(handle,&mlstyle)) {
            goto SUCCESS3;
        }
    }
}
}
adWriteMtextBlock

Summary

short adWriteMtextBlock (PAD_BLOB_CTRL bcptr, char *str);

Description

Writes a block of mtext to the blob indicated by bcptr.

Return Value

Returns 1 on success, 0 on failure. On failure, adError() should be checked for an indication of the cause of the failure.

See Also

adStartBlobWrite, adEndBlobWrite, adCreateBlob

Example

adenhd->enttype=AD_ENT_MTEXT;
adSetEntityDefaults(wrHandle, adenhd, aden);
aden->mtext.pt0[0]=10.0;
aden->mtext.pt0[1]=9.0;
aden->mtext.ht=0.2;
aden->mtext.reflectwid=8.0;
aden->mtext.ldblob=adCreateBlob();
bcptr=adStartBlobWrite(aden->mtext.ldblob);
for (i=0; i<30; i++)
  adWriteMtextBlock(bcptr,"This is even more mtext. ");
adEndBlobWrite(bcptr);
adGenerateObjhandle(wrhandle, adenhd->enthandle);
adAddEntityToList(wrhandle, mspaceentlist, adenhd, aden);

adWriteXrecordItem

Summary

short adWriteXrecordItem (PAD_BLOB_CTRL bcptr, PAD_XD adxd);

Description

Writes an xrecord item to the blob indicated by bcptr.

Return Value

Returns 1 on success, 0 on failure. On failure, adError() should be checked for an indication of the cause of the failure.
See Also

adStartBlobWrite, adEndBlobWrite, adCreateBlob

Example

```c
if ((adxd.xddxfnumber>=1 && adxd.xddxfnumber<=9) ||
    adxd.xddxfnumber==102) {
    strcpy(adxd.xddata.xdstring,adin->globalreadbuf);
adWriteXrecordItem(bcptr,&adxd);
}
```

adWriteProxyDataBinary

**Summary**

short adWriteProxyDataBinary (char *a, short len);

**Description**

Writes binary data to proxy data (object or entity) of length len.

**Return Value**

Returns 1 on success, 0 on failure. On failure, adError() should be checked for an indication of the cause of the failure.

See Also

adStartProxyDataWrite, adEndProxyDataWrite

Example

```c
adWriteProxyDataBinary("a test string",13);
```

adWriteProxyDataBool

**Summary**

short adWriteProxyDataBool (unsigned char a);

**Description**

Writes a Boolean to proxy data (object or entity).

**Return Value**

Returns 1 on success, 0 on failure. On failure, adError() should be checked for an indication of the cause of the failure.
See Also

adStartProxyDataWrite, adEndProxyDataWrite

Example

```
adWriteProxyDataBool(0);
```

---

### adWriteProxyDataChar

**Summary**

short adWriteProxyDataChar (char a);

**Description**

Writes a proxy data (object or entity) character.

**Return Value**

Returns 1 on success, 0 on failure. On failure, adError() should be checked for an indication of the cause of the failure.

See Also

adStartProxyDataWrite, adEndProxyDataWrite

Example

```
adWriteProxyDataChar('a');
```

---

### adWriteProxyDataDouble

**Summary**

short adWriteProxyDataChar (double a);

**Description**

Writes a proxy data (object or entity) double.

**Return Value**

Returns 1 on success, 0 on failure. On failure, adError() should be checked for an indication of the cause of the failure.

See Also

adStartProxyDataWrite, adEndProxyDataWrite
Example

    adWriteProxyDataDouble(3.0);

adWriteProxyDataInt16

Summary

    short adWriteProxyDataInt16 (short a);

Description

    Writes a proxy data (object or entity) short (16 bit integer).

Return Value

    Returns 1 on success, 0 on failure. On failure, adError() should be checked for an indication of the cause of the failure.

See Also

    adStartProxyDataWrite, adEndProxyDataWrite

Example

    adWriteProxyDataInt16(128);

adWriteProxyDataInt32

Summary

    short adWriteProxyDataInt32 (short a);

Description

    Writes a proxy data (object or entity) long (32 bit integer).

Return Value

    Returns 1 on success, 0 on failure. On failure, adError() should be checked for an indication of the cause of the failure.

See Also

    adStartProxyDataWrite, adEndProxyDataWrite

Example

    adWriteProxyDataInt32(128L);
adWriteProxyDataObjhandle

**Summary**

short adWriteProxyDataObjhandle (AD_OBJHANDLE a);

**Description**

Writes a proxy data (object or entity) AD_OBJHANDLE.

**Return Value**

Returns 1 on success, 0 on failure. On failure, `adError()` should be checked for an indication of the cause of the failure.

**See Also**

adStartProxyDataWrite, adEndProxyDataWrite

**Example**

```c
adWriteProxyDataObjhandle(adenhd->curmlstyleobjhandle);
```

adWriteProxyDataPoint2d

**Summary**

short adWriteProxyDataPoint2d (double *a);

**Description**

Writes a proxy data (object or entity) 2D point (X and Y coordinates).

**Return Value**

Returns 1 on success, 0 on failure. On failure, `adError()` should be checked for an indication of the cause of the failure.

**See Also**

adStartProxyDataWrite, adEndProxyDataWrite

**Example**

```c
double pt[2];
pt[0]=1.0; pt[1]=2.0;
adWriteProxyDataPoint2d(pt);
```
adWriteProxyDataPoint3d

Summary

short adWriteProxyDataPoint3d (double *a);

Description

Writes a proxy data (object or entity) 3D point (X, Y and Z coordinates).

Return Value

Returns 1 on success, 0 on failure. On failure, adError() should be checked for an indication of the cause of the failure.

See Also

adStartProxyDataWrite, adEndProxyDataWrite

Example

double pt[3];
pt[0]=1.0; pt[1]=2.0; pt[2]=3.0
adWriteProxyDataPoint3d(pt);

adWriteProxyDataScale3d

Summary

short adWriteProxyDataScale3d (double *a);

Description

Writes a proxy data (object or entity) 3D scale (X, Y and Z scales).

Return Value

Returns 1 on success, 0 on failure. On failure, adError() should be checked for an indication of the cause of the failure.

See Also

adStartProxyDataWrite, adEndProxyDataWrite

Example

double pt[3];
pt[0]=1.0; pt[1]=2.0; pt[2]=3.0
adWriteProxyDataScale3d(pt);
**adWriteProxyDataString**

**Summary**

short adWriteProxyDataString (char *a);

**Description**

Writes a proxy data (object or entity) string.

**Return Value**

Returns 1 on success, 0 on failure. On failure, **adError()** should be checked for an indication of the cause of the failure.

**See Also**

adStartProxyDataWrite, adEndProxyDataWrite

**Example**

adWriteProxyDataString("this is a test");

---

**adWriteProxyDataTypedObjhandle**

**Summary**

short adWriteProxyDataTypedObjhandle (PAD_TYPEDOBJHANDLE a);

**Description**

Writes a proxy data (object or entity) typed AD_OBJHANDLE. Types of handles are the defined constants AD_SOFT_POINTER_HANDLE, AD_HARD_POINTER_HANDLE, AD_SOFTOWNERHANDLE, and AD_HARDOWNERHANDLE.

**Return Value**

Returns 1 on success, 0 on failure. On failure, **adError()** should be checked for an indication of the cause of the failure.

**See Also**

adStartProxyDataWrite, adEndProxyDataWrite

**Example**

AD_TYPEDOBJHANDLE oh;

oh.handletype=AD_SOFT_POINTER_HANDLE;
adHancpy(oh.oh.typehandle, adhd->curmlstyleobjhandle);
adWriteProxyDataTypedObjhandle(&oh);
adWriteProxyDataUInt16

Summary

short adWriteProxyDataUInt16 (short a);

Description

Writes a proxy data (object or entity) unsigned short (16 bit integer).

Return Value

Returns 1 on success, 0 on failure. On failure, adError() should be checked for an indication of the cause of the failure.

See Also

adStartProxyDataWrite, adEndProxyDataWrite

Example

unsigned short myuint=24;
adWriteProxyDataUInt16(myuint);

adWriteProxyDataUInt32

Summary

short adWriteProxyDataUInt32 (short a);

Description

Writes a proxy data (object or entity) unsigned long (32 bit integer).

Return Value

Returns 1 on success, 0 on failure. On failure, adError() should be checked for an indication of the cause of the failure.

See Also

adStartProxyDataWrite, adEndProxyDataWrite

Example

unsigned long myulong=24;
adWriteProxyDataUInt32(myulong);
adWriteProxyDataUInt8

Summary

short adWriteProxyDataUInt8 (short a);

Description

Writes a proxy data (object or entity) unsigned char (8 bit integer).

Return Value

Returns 1 on success, 0 on failure. On failure, adError() should be checked for an indication of the cause of the failure.

See Also

adStartProxyDataWrite, adEndProxyDataWrite

Example

unsigned char myuchar=24;
adWriteProxyDataUInt8(myuchar);

adWriteProxyDataVector2d

Summary

short adWriteProxyDataVector2d (double *a);

Description

Writes a proxy data (object or entity) 2D vector (X and Y magnitudes).

Return Value

Returns 1 on success, 0 on failure. On failure, adError() should be checked for an indication of the cause of the failure.

See Also

adStartProxyDataWrite, adEndProxyDataWrite

Example

double *vec[2];
vec[0]=0.0;
vec[1]=1.0;
adWriteProxyDataVector2d(vec);
**adWriteProxyDataVector3d**

**Summary**

```c
short adWriteProxyDataVector3d (double *a);
```

**Description**

Writes a proxy data (object or entity) 3D vector (X, Y and Z magnitudes).

**Return Value**

Returns 1 on success, 0 on failure. On failure, `adError()` should be checked for an indication of the cause of the failure.

**See Also**

`adStartProxyDataWrite, adEndProxyDataWrite`

**Example**

```c
double *vec[3];
vec[0]=0.0;
vec[1]=1.0;
vec[2]=0.0;
adWriteProxyDataVector3d(vec);
```

**adWriteProxyDataVoidstardata**

**Summary**

```c
short adWriteProxyDataVoidstardata(char *a, unsigned short size);
```

**Description**

Writes proxy data (object or entity) unformatted data of length size.

**Return Value**

Returns 1 on success, 0 on failure. On failure, `adError()` should be checked for an indication of the cause of the failure.

**See Also**

`adStartProxyDataWrite, adEndProxyDataWrite`
Example

```c
char mybuf[256];
adWriteProxyDataVoidstardata(mybuf,223);
```

**adWriteProxyEntityObjid**

**Summary**

short adWriteProxyEntityObjid (PAD_BLOB_CTRL bcptr, PAD_TYPEDOBJHANDLE oh);

**Description**

Writes a proxy entity object ID to the blob indicated by `bcptr`.

**Return Value**

Returns 1 on success, 0 on failure. On failure, `adError()` should be checked for an indication of the cause of the failure.

**See Also**

`adStartBlobWrite, adEndBlobWrite, adCreateBlob`

**Example**

```c
PAD_TYPEDOBJHANDLE mytypedobjhandle;
adWriteProxyEntityObjid(bcptr,mytypedobjhandle);
```

**adWriteProxyObjectObjid**

**Summary**

short adWriteProxyObjectObjid (PAD_BLOB_CTRL bcptr, PAD_TYPEDOBJHANDLE oh);

**Description**

Writes a proxy object object ID to the blob indicated by `bcptr`.

**Return Value**

Returns 1 on success, 0 on failure. On failure, `adError()` should be checked for an indication of the cause of the failure.

**See Also**

`adStartBlobWrite, adEndBlobWrite, adCreateBlob`

**Example**

```c
PAD_TYPEDOBJHANDLE mytypedobjhandle;
adWriteProxyObjectObjid(bcptr,mytypedobjhandle);
```
adXrecordObjtype

Summary

short adXrecordObjtype(AD_DB_HANDLE handle);

Description

Returns the object type value for AD_XRECORD objects.

Return Value

Returns the object type value for AD_XRECORD objects.

See Also

None

Example

adobhd.objtype=adXrecordObjtype(handle);
10. DATA REFERENCE

This section contains prose descriptions of the data structures and defined constants in ad2.h.

HEADER DATA:

Here we describe the AD_DWGHDR structure, a pointer to which is returned by adHeaderPointer().

<table>
<thead>
<tr>
<th>Variable</th>
<th>Version</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>short acadmaintver;</td>
<td>14+</td>
<td>Maintenance version, set to 0.</td>
</tr>
<tr>
<td>char acadver [7];</td>
<td>2.5+</td>
<td>AutoCAD version string:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>&quot;AC1002&quot;: Version 2.5</td>
</tr>
<tr>
<td></td>
<td></td>
<td>&quot;AC1003&quot;: Version 2.6</td>
</tr>
<tr>
<td></td>
<td></td>
<td>&quot;AC1004&quot;: Release 9</td>
</tr>
<tr>
<td></td>
<td></td>
<td>&quot;AC1006&quot;: Release 10</td>
</tr>
<tr>
<td></td>
<td></td>
<td>&quot;AC1009&quot;: Release 11 &amp; 12</td>
</tr>
<tr>
<td></td>
<td></td>
<td>&quot;AC1012&quot;: Release 13</td>
</tr>
<tr>
<td></td>
<td></td>
<td>&quot;AC1014&quot;: Release 14</td>
</tr>
<tr>
<td>double angbase</td>
<td>(2.5+)</td>
<td>Angle considered to be 0, in radians. Default is 0.0.</td>
</tr>
<tr>
<td>short angdir</td>
<td>(2.5+)</td>
<td>Direction of angle measure (0==counterclockwise, 1==clockwise). Default is 0.</td>
</tr>
<tr>
<td>short attdia;</td>
<td>(9+)</td>
<td>If 0, prompts are issued on command line for attribute value entry, if 1, a dialog box is used. Default is 0.</td>
</tr>
<tr>
<td>short attmode</td>
<td>(2.5+)</td>
<td>Controls display of attributes. 0==off, 1==normal, 2==on. Default is 1.</td>
</tr>
<tr>
<td>short attreq;</td>
<td>(9+)</td>
<td>If 0, defaults are assumed for all attributes, otherwise user is prompted according to the setting of attdia. Default is 1.</td>
</tr>
<tr>
<td>short aunits;</td>
<td>(2.5+)</td>
<td>Angular units mode: 0==decimal degrees, 1==degrees/minutes/seconds, 2==gradians, 3==radians, 4==surveyor's units. Default is 0.</td>
</tr>
<tr>
<td>short auprec;</td>
<td>(2.5+)</td>
<td>Decimal places of precision for angular units. Default is 0.</td>
</tr>
<tr>
<td>short axismode;</td>
<td>(2.5-11)</td>
<td>1 turns axis on, 0 off. Pre R13 only. Default is 0.</td>
</tr>
<tr>
<td>double axisunit[2];</td>
<td>(2.5-11)</td>
<td>Units for axis on X and Y. Default is 0.0, 0.0.</td>
</tr>
<tr>
<td>double backz;</td>
<td>(10+)</td>
<td>Offset of back clipping plane from the target plane in the current viewport. Default is 0.0.</td>
</tr>
<tr>
<td>short blipmode;</td>
<td>(2.5+)</td>
<td>0==marker blips off, 1==marker blips on. Default is 1.</td>
</tr>
<tr>
<td>Variable</td>
<td>Type</td>
<td>Description</td>
</tr>
<tr>
<td>---------------</td>
<td>--------</td>
<td>------------------------------------------------------------------------------</td>
</tr>
<tr>
<td><code>cecolor</code></td>
<td><code>short</code></td>
<td>Current entity color. Sets color for new entities. Some of the standard AutoCAD colors are set to defined constants by OpenDWG Toolkit: AD_COLOR_RED, AD_COLOR_YELLOW, AD_COLOR_GREEN, AD_COLOR_CYAN, AD_COLOR_BLUE, AD_COLOR_MAGENTA, AD_COLOR_WHITE, AD_COLOR_BYLAYER, AD_COLOR_BYBLOCK. Default is AD_COLOR_BYLAYER.</td>
</tr>
<tr>
<td><code>celtscale</code></td>
<td><code>double</code></td>
<td>Current entity linetype scale. Default is 1.0.</td>
</tr>
<tr>
<td><code>celweight</code></td>
<td><code>short</code></td>
<td>Current entity lineweight. Default is -1.</td>
</tr>
<tr>
<td><code>cepsntype</code></td>
<td><code>short</code></td>
<td>Plotstyle type of new objects: 0 = PlotStyle by layer, 1 = PlotStyle by block, 2 = PlotStyle by dictionary default, 3 = PlotStyle by object ID/handle. Default is 0.</td>
</tr>
<tr>
<td><code>chamfera</code></td>
<td><code>double</code></td>
<td>First chamfer distance. Default is 0.0.</td>
</tr>
<tr>
<td><code>chamferb</code></td>
<td><code>double</code></td>
<td>Second chamfer distance. Default is 0.0.</td>
</tr>
<tr>
<td><code>chamferc</code></td>
<td><code>double</code></td>
<td>Length of chamfers. Default is 0.0.</td>
</tr>
<tr>
<td><code>chamferd</code></td>
<td><code>double</code></td>
<td>Angle of chamfers. Default is 0.0.</td>
</tr>
<tr>
<td><code>cmljust</code></td>
<td><code>short</code></td>
<td>Current multiline justification. 0 = top, 1 = middle, 2 = bottom</td>
</tr>
<tr>
<td><code>cmlscale</code></td>
<td><code>double</code></td>
<td>Current multiline scale. Default is 1.0.</td>
</tr>
<tr>
<td><code>coords</code></td>
<td><code>short</code></td>
<td>Controls update of coordinate display. 0 = updated when picks are performed, 1 = continuous update, 2 = displays distance and angle from last point when either distances or angles are requested. Default is 0.</td>
</tr>
<tr>
<td><code>cyclecurr</code></td>
<td><code>short</code></td>
<td>Unused by AutoCAD. Default is 1.</td>
</tr>
<tr>
<td><code>cyclekeep</code></td>
<td><code>short</code></td>
<td>Unused by AutoCAD. Default is 5.</td>
</tr>
<tr>
<td><code>delobj</code></td>
<td><code>char</code></td>
<td>If 0, objects used to create other objects are deleted. If 1, objects used to create other objects are retained. Default is 1.</td>
</tr>
<tr>
<td><code>dimadec</code></td>
<td><code>short</code></td>
<td>Number of precision places displayed in angular dimensions. Default is 0.</td>
</tr>
<tr>
<td><code>dimalt</code></td>
<td><code>char</code></td>
<td>When set to 1, alternate units dimensioning is enabled, otherwise it is disabled. Default is 0.</td>
</tr>
<tr>
<td><code>dimaltd</code></td>
<td><code>char</code></td>
<td>Sets the number of alternate units decimal places. Default is 2.</td>
</tr>
<tr>
<td><code>dimaltf</code></td>
<td><code>char</code></td>
<td>Sets the alternate units scale factor. Default is 25.4.</td>
</tr>
<tr>
<td>Variable</td>
<td>(Version)</td>
<td>Description</td>
</tr>
<tr>
<td>-------------------</td>
<td>-----------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>double dimaltrnd;</td>
<td>(2000+)</td>
<td>Determines rounding of alternate units. Default is 0.0.</td>
</tr>
<tr>
<td>short dimaltd;</td>
<td>(13+)</td>
<td>Sets tolerance decimal places on alternate units dimension. Default is 2.</td>
</tr>
<tr>
<td>char dimaltttz;</td>
<td>(13+)</td>
<td>Controls zero suppression for tolerance values. 1 suppresses, 0 does not. Default is 0.</td>
</tr>
<tr>
<td>short dimaltu;</td>
<td>(13+)</td>
<td>Controls the format of alternate units for all units in the dimension style family (not including angular). 1==scientific 2==decimal 3==engineering 4==architectural 5==fractional Default is 2.</td>
</tr>
<tr>
<td>char dimaltz;</td>
<td>(13+)</td>
<td>Controls zero suppression for alternate dimensions. 1 suppresses, 0 does not. Default is 0.</td>
</tr>
<tr>
<td>char dimapost[16];</td>
<td>(2.6+)</td>
<td>Sets a dimension prefix or suffix for alternate dimensions (except angular). See dimpost. Default is &quot;&quot;.</td>
</tr>
<tr>
<td>char dimaso;</td>
<td>(2.6+)</td>
<td>Controls whether dimensions are associative or not. 0==not associative, 1==associative. Default is 1.</td>
</tr>
<tr>
<td>double dimasz;</td>
<td>(2.5+)</td>
<td>Controls dimension arrowhead size and hookline size. Default is 0.18.</td>
</tr>
<tr>
<td>short dimaunit;</td>
<td>(13+)</td>
<td>Controls the angle format for angular dimensions. 0==decimal degrees 1==degrees/minutes/seconds 2==radians 3==radians 4==surveyor's units. Default is 0.</td>
</tr>
<tr>
<td>short dimazin;</td>
<td>(2000+)</td>
<td>Controls suppression of zeros for angular dimensions: 0 = Displays all leading and trailing zeros. 1 = Suppresses leading zeros in decimal dimensions. 2 = Suppresses trailing zeros in decimal dimensions. 3 = Suppresses leading and trailing zeros. Default is 0.</td>
</tr>
<tr>
<td>char dimblk[AD_MAX_STRLLEN];</td>
<td>(2.5-14)</td>
<td>Names a block to be drawn in place of the arrowhead at the ends of dimensions and leaders. This field is no longer used.</td>
</tr>
<tr>
<td>AD_OBJHANDLE dimblkobjhandle;</td>
<td>(2000+)</td>
<td>Handle of block to be drawn in place of the arrowhead at the ends of dimensions and leaders. Use this field instead of dimblk.</td>
</tr>
<tr>
<td>char dimblk1[AD_MAX_STRLLEN];</td>
<td>(10-14)</td>
<td>Specifies a user-defined arrowhead block for the first end of a dimension line. Valid only if dimsah is 1. This field is no longer used.</td>
</tr>
<tr>
<td>AD_OBJHANDLE dimblk1objhandle;</td>
<td>(2000+)</td>
<td>Handle of user-defined arrowhead block for the first end of a dimension line. Valid only if dimsah is 1. Use this field instead of dimblk1.</td>
</tr>
<tr>
<td>char dimblk2[AD_MAX_STRLLEN];</td>
<td>(10-14)</td>
<td>Specifies a user-defined arrowhead block for the second end of a dimension line. Valid only if dimsah is 1. This field is no longer used.</td>
</tr>
<tr>
<td>AD_OBJHANDLE dimblk2objhandle;</td>
<td>(2000+)</td>
<td>Handle of user-defined arrowhead block for the second end of a dimension line. Valid only if dimsah is 1. Use this field instead of dimblk2.</td>
</tr>
<tr>
<td>double dimcen;</td>
<td>(2.5+)</td>
<td>Determines style of center marks for arcs or circles being dimensioned. 0.0==no center marks &gt;0.0==draw center marks</td>
</tr>
</tbody>
</table>
<0.0==draw center lines. Default is 0.0.

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>short dimclrd;</td>
<td>(11+) Sets color for dimension leader lines, arrowheads, dimension lines and leaders. Can be set to any valid color or the special colors AD_COLOR_BY_LAYER or AD_COLOR_BY_BLOCK. Default is AD_COLOR_BY_BLOCK.</td>
<td></td>
</tr>
<tr>
<td>short dimclre;</td>
<td>(11+) Sets color for dimension extension lines. Default is AD_COLOR_BY_BLOCK.</td>
<td></td>
</tr>
<tr>
<td>short dimclrt;</td>
<td>(11+) Sets color for dimension text. Default is AD_COLOR_BY_BLOCK.</td>
<td></td>
</tr>
<tr>
<td>short dimdec;</td>
<td>(13+) Controls the precision for a primary units dimension's tolerance values. Default is 4.</td>
<td></td>
</tr>
<tr>
<td>double dimdle;</td>
<td>(2.5+) Controls extension of the dimension line past the extension lines for oblique-stroke style dimensions. Default is 0.0</td>
<td></td>
</tr>
<tr>
<td>double dimdli;</td>
<td>(2.5+) Offset distance for baseline dimensions. Default is 0.38.</td>
<td></td>
</tr>
<tr>
<td>short dimdsep;</td>
<td>(2000+) Single-character decimal separator used when creating dimensions whose unit format is decimal. Default is 46 (').</td>
<td></td>
</tr>
<tr>
<td>double dimexe;</td>
<td>(2.5+) Controls extension of extension line beyond dimension line. Default is 0.18.</td>
<td></td>
</tr>
<tr>
<td>double dimexo;</td>
<td>(2.5+) Offset of extension lines from origin points. Default is 0.0625.</td>
<td></td>
</tr>
<tr>
<td>Variable Name</td>
<td>Access</td>
<td>Description</td>
</tr>
<tr>
<td>--------------------</td>
<td>------------</td>
<td>-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
</tbody>
</table>
| dimfit;            | (13+)      | Determines how text and arrowheads are placed for a dimension.  

0==between extension lines if space is available, otherwise move both out.  

1==between extension lines if space is available. If there is only enough space for the text, text goes inside and arrowheads outside. If there is not enough space for the text, both go outside.  

2==between extension lines if space is available. If there is only enough space for the text, text goes inside and arrowheads outside. If there is only enough space for the arrowheads, arrowheads go inside and text outside. Otherwise, both go outside.  

3==whatever fits "best" goes between the extension lines.  

4==creates leader lines when text will not fit between extension lines.  

Default is 3. |
| dimfrac;           | (2000+)    | Scale factor used to calculate the height of text for dimension fractions and tolerances. Default is 0. |
| dimgap;            | (11+)      | Determines the gap around dimension text when the dimension line is broken to fit the text. Also determines the gap between a leader's hookline and its annotation. Finally, dimgap is also used as a minimum size for parts of the dimension. Default is 0.09. |
| dimjust;           | (13+)      | Determines horizontal text justification.  

0==center justified between extension lines.  

1==next to first extension line.  

2==next to second extension line.  

3==above and aligned with first extension line.  

4==above and aligned with second extension line.  

Default is 0. |
| dimldrblk[AD_MAX_STRLEN ]; | (2000+)  | Specifies a user-defined arrowhead name for leaders. This field is used only for temporary internal storage, and should not be referenced. |
| dimldrblkobjhandle; | (2000+)    | Handle of user-defined arrowhead block leaders. Use this field instead of dimldrblk. |
| dimlfac;           | (2.6+)     | Global scale factor for linear dimensioning measurements. Default is 1.0. |
| dimlim;            | (2.5+)     | If 1, sets the default text to dimension limits. Otherwise 0. Default is 0. |
| dimlwd;            | (2000+)    | Dimension line linewidth: -3 = Standard.-2 = ByLayer.-1 = ByBlock.0-211 = an integer representing 100th of mm. Default is -2. |
| dimlw2;            | (2000+)    | Extension line linewidth: -3 = Standard.-2 = ByLayer.-1 = ByBlock.0-211 = an integer representing 100th of mm. Default is -2. |
| dimpost[16];       | (2.6+)     | Sets a dimension prefix or suffix for the measurement text. You delineate the break between prefix and suffix by placing "<>"
between them, e.g. pre<>post. Default is "".

<table>
<thead>
<tr>
<th>variable</th>
<th>version</th>
<th>description</th>
</tr>
</thead>
<tbody>
<tr>
<td>double dimrnd;</td>
<td>(2.5+)</td>
<td>Sets a rounding value for dimensions. Default is 0.0.</td>
</tr>
<tr>
<td>char dimsah;</td>
<td>(10+)</td>
<td>Determines whether normal or user-defined arrowhead blocks are used.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0==normal arrowheads</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1==user-defined arrowheads. Default is 0.</td>
</tr>
<tr>
<td>Variable</td>
<td>Description</td>
<td></td>
</tr>
<tr>
<td>---------------------</td>
<td>-----------------------------------------------------------------------------</td>
<td></td>
</tr>
<tr>
<td>double dimscale;</td>
<td>Overall scale factor applied to dimension variables for sizes, distances and offsets. Also scales leaders. If 0.0, a value is computed based on the current scaling between modelspace and paperspace. In paperspace, or in modelspace if paperspace is not being used, the factor used is 1.0. If &gt;0.0, a scale factor is computed that causes text sizes, arrowhead sizes, and other scaled distances to plot properly. Default is 1.0.</td>
<td></td>
</tr>
<tr>
<td>char dimsd1;</td>
<td>If 1, suppresses first dimension line. Default is 0.</td>
<td></td>
</tr>
<tr>
<td>char dimsd2;</td>
<td>If 1, suppresses second dimension line. Default is 0</td>
<td></td>
</tr>
<tr>
<td>char dimse1;</td>
<td>If 1, suppresses first extension line. Default is 0</td>
<td></td>
</tr>
<tr>
<td>char dimse2;</td>
<td>If 1, suppresses second extension line. Default is 0</td>
<td></td>
</tr>
<tr>
<td>char dimsho;</td>
<td>If 1, dimensions are shown dynamically when dragged, not if 0. Default is 1.</td>
<td></td>
</tr>
<tr>
<td>char dimsoxd;</td>
<td>If 1, suppresses dimension lines outside the extension lines. Default is 0.</td>
<td></td>
</tr>
<tr>
<td>char dimtad;</td>
<td>Determines text vertical position relative to the dimension line. 0==Centered between extension lines. 1==above dimension line, except when not horizontal and text inside the dimension lines is set to be horizontal. <em>dimgap</em> is used to determine the distance from the dimension line to the baseline of the lowest line of dimension text. Default is 0.</td>
<td></td>
</tr>
<tr>
<td>short dimtdec;</td>
<td>Number of decimal places for tolerance value of a primary units dimension. Default is 4.</td>
<td></td>
</tr>
<tr>
<td>double dimtfac;</td>
<td>Scale factor for tolerance value height relative to dim text height. Default is 1.0.</td>
<td></td>
</tr>
<tr>
<td>char dimtih;</td>
<td>If 1, text between extension lines is always drawn horizontally. If 0, text is aligned with the dimension line. Default is 1.</td>
<td></td>
</tr>
<tr>
<td>char dimtix;</td>
<td>Controls text between extension lines. If 0, radius and angular dimensions have their text placed outside of the arc or circle. If 1, forces text between extension lines even if it would normally have been placed outside. Default is 0.</td>
<td></td>
</tr>
<tr>
<td>double dimtm;</td>
<td>Sets minimum tolerance limit for dimension text. Default is 0.0.</td>
<td></td>
</tr>
<tr>
<td>short dimtmove;</td>
<td>Dimension text movement rules: 0 = Moves the dimension line with dimension text. 1 = Adds a leader when dimension text is moved. 2 = Allows text to be moved freely without a leader. Default is 0.</td>
<td></td>
</tr>
<tr>
<td>char dimtofl;</td>
<td>If 1, places dimension lines between extension lines even if text is outside. Default is 0.</td>
<td></td>
</tr>
<tr>
<td>char dimtoh;</td>
<td>Sets position of dimension text outside the extension lines. 0==text aligned with dimension line, 1==text drawn horizontally. Default is 1.</td>
<td></td>
</tr>
<tr>
<td>char dimtol;</td>
<td>If 1, dimension tolerances are appended to dimension text. Default is 0.</td>
<td></td>
</tr>
<tr>
<td>char dimtolj;</td>
<td>Controls vertical justification for tolerance values relative to the dimension text. 0==bottom, 1==middle, 2==top. Default is 1.</td>
<td></td>
</tr>
<tr>
<td>double dimtp;</td>
<td>Sets maximum tolerance limit for dimension text. Default is 0.0.</td>
<td></td>
</tr>
<tr>
<td>double dimtsz;</td>
<td>Sets size of oblique strokes. 0.0==arrows drawn, &gt;0.0==draws oblique strokes at this size <em>dimscale</em>. Default is 0.0.</td>
<td></td>
</tr>
<tr>
<td>Variable</td>
<td>Class</td>
<td>Description</td>
</tr>
<tr>
<td>-------------------</td>
<td>-------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>double dimtvp</td>
<td>(10+)</td>
<td>Sets the vertical position of dim text either above or below the dimension line. Default is 0.0.</td>
</tr>
<tr>
<td>double dimtxt</td>
<td>(2.5+)</td>
<td>Dimension text height if the current dimstyle has no fixed text height. Default is 0.18.</td>
</tr>
<tr>
<td>char dimtzn</td>
<td>(13+)</td>
<td>If 1, zeroes are suppressed for tolerance values. Default is 0.</td>
</tr>
<tr>
<td>short dimunit</td>
<td>(13+)</td>
<td>Determines the units format for all types except angular. 1==scientific 2==decimal 3==engineering 4==architectural 5==fractional. Default is 2.</td>
</tr>
<tr>
<td>char dimupt</td>
<td>(13+)</td>
<td>Sets positioning mode for user-positioned text. 0==user controls only the dim line location. 1==controls both text position and dim line location. Default is 0.</td>
</tr>
<tr>
<td>char dimzin</td>
<td>(2.5+)</td>
<td>Determines suppression of inches part of a feet-and-inches dimension. 0==suppresses zero feet and zero inches. 1==includes zero feet and zero inches. 2==includes zero feet and suppresses zero inches. 3==includes zero inches and suppresses zero feet. Default is 0.</td>
</tr>
<tr>
<td>char dispsilh</td>
<td>(13+)</td>
<td>If 1, silhouette curves of body objects are displayed in wire-frame mode. Default is 0.</td>
</tr>
<tr>
<td>short dragmode</td>
<td>(2.5+)</td>
<td>Controls drag mode. 0==no dragging, 1==on if requested, 2==auto. Default is 2.</td>
</tr>
<tr>
<td>Variable</td>
<td>Description</td>
<td>Default</td>
</tr>
<tr>
<td>--------------------</td>
<td>-----------------------------------------------------------------------------</td>
<td>---------</td>
</tr>
<tr>
<td>short dwgcodepage;</td>
<td>Index of the system code page for this drawing. The corresponding strings are:</td>
<td></td>
</tr>
<tr>
<td></td>
<td>0: &quot;undefined&quot;</td>
<td></td>
</tr>
<tr>
<td></td>
<td>1: &quot;ascii&quot;</td>
<td></td>
</tr>
<tr>
<td></td>
<td>2: &quot;iso8859-1&quot;</td>
<td></td>
</tr>
<tr>
<td></td>
<td>3: &quot;iso8859-2&quot;</td>
<td></td>
</tr>
<tr>
<td></td>
<td>4: &quot;iso8859-3&quot;</td>
<td></td>
</tr>
<tr>
<td></td>
<td>5: &quot;iso8859-4&quot;</td>
<td></td>
</tr>
<tr>
<td></td>
<td>6: &quot;iso8859-5&quot;</td>
<td></td>
</tr>
<tr>
<td></td>
<td>7: &quot;iso8859-6&quot;</td>
<td></td>
</tr>
<tr>
<td></td>
<td>8: &quot;iso8859-7&quot;</td>
<td></td>
</tr>
<tr>
<td></td>
<td>9: &quot;iso8859-8&quot;</td>
<td></td>
</tr>
<tr>
<td></td>
<td>10: &quot;iso8859-9&quot;</td>
<td></td>
</tr>
<tr>
<td></td>
<td>11: &quot;dos437&quot;</td>
<td></td>
</tr>
<tr>
<td></td>
<td>12: &quot;dos850&quot;</td>
<td></td>
</tr>
<tr>
<td></td>
<td>13: &quot;dos852&quot;</td>
<td></td>
</tr>
<tr>
<td></td>
<td>14: &quot;dos855&quot;</td>
<td></td>
</tr>
<tr>
<td></td>
<td>15: &quot;dos857&quot;</td>
<td></td>
</tr>
<tr>
<td></td>
<td>16: &quot;dos860&quot;</td>
<td></td>
</tr>
<tr>
<td></td>
<td>17: &quot;dos861&quot;</td>
<td></td>
</tr>
<tr>
<td></td>
<td>18: &quot;dos863&quot;</td>
<td></td>
</tr>
<tr>
<td></td>
<td>19: &quot;dos864&quot;</td>
<td></td>
</tr>
<tr>
<td></td>
<td>20: &quot;dos865&quot;</td>
<td></td>
</tr>
<tr>
<td></td>
<td>21: &quot;dos869&quot;</td>
<td></td>
</tr>
<tr>
<td></td>
<td>22: &quot;dos932&quot;</td>
<td></td>
</tr>
<tr>
<td></td>
<td>23: &quot;mac-roman&quot;</td>
<td></td>
</tr>
<tr>
<td></td>
<td>24: &quot;big5&quot;</td>
<td></td>
</tr>
<tr>
<td></td>
<td>25: &quot;ksc5601&quot;</td>
<td></td>
</tr>
<tr>
<td></td>
<td>26: &quot;johab&quot;</td>
<td></td>
</tr>
<tr>
<td></td>
<td>27: &quot;dos866&quot;</td>
<td></td>
</tr>
<tr>
<td></td>
<td>28: &quot;ansi_1250&quot;</td>
<td></td>
</tr>
<tr>
<td></td>
<td>29: &quot;ansi_1251&quot;</td>
<td></td>
</tr>
<tr>
<td></td>
<td>30: &quot;ansi_1252&quot;</td>
<td></td>
</tr>
<tr>
<td></td>
<td>31: &quot;gb2312&quot;</td>
<td></td>
</tr>
<tr>
<td></td>
<td>32: &quot;ansi_1253&quot;</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Default is 12.</td>
<td></td>
</tr>
<tr>
<td>double elevation;</td>
<td>Current 3D elevation for entities. Default is 0.0.</td>
<td></td>
</tr>
<tr>
<td>double endcaps;</td>
<td>Lineweight endcaps setting for new objects: 0 == none; 1 == round; 2 == angle; 3 == square. Default is 0.</td>
<td></td>
</tr>
<tr>
<td>double extmin[3];</td>
<td>Lower left point of drawing extents. Default is 1.0e+20, 1.0e+20, 1.0e+20.</td>
<td></td>
</tr>
<tr>
<td>double extmax[3];</td>
<td>Upper right point of drawing extents. Default is -1.0e+20, -1.0e+20, -1.0e+20.</td>
<td></td>
</tr>
<tr>
<td>char extnames;</td>
<td>0 == R14 compatibility, which limits names to 31 characters in length. 1 == AutoCAD 2000, where names can be up to 255 characters in length. Default is 1.</td>
<td></td>
</tr>
<tr>
<td>double facetres;</td>
<td>Smoothness of shaded and hidden-line removed objects. Valid values are 0.01 to 10.0. Default is 0.5.</td>
<td></td>
</tr>
<tr>
<td>short fastzoom;</td>
<td>Sets vector generation for circles. Default is 1.</td>
<td></td>
</tr>
<tr>
<td>double filletrad;</td>
<td>Current fillet radius. Default is 0.0.</td>
<td></td>
</tr>
<tr>
<td>short fillmode;</td>
<td>0==objects are not filled, 1==objects are filled. Default is 1.</td>
<td></td>
</tr>
<tr>
<td>Variable</td>
<td>Type</td>
<td>Description</td>
</tr>
<tr>
<td>--------------------------</td>
<td>--------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>char fingerprintguid[39];</td>
<td>(2000+)</td>
<td>Set at creation time, uniquely identifies a particular drawing.</td>
</tr>
<tr>
<td>short flatland;</td>
<td>(10+)</td>
<td>1==emulate Release 9, 0==don't. Default is 0.</td>
</tr>
<tr>
<td>double frontz;</td>
<td>(10+)</td>
<td>Front clipping plane offset from target plane. Default is 0.0.</td>
</tr>
<tr>
<td>short gridmode;</td>
<td>(2.5+)</td>
<td>1==grid on, 0==grid off. Default is 0.</td>
</tr>
<tr>
<td>double gridunit[2];</td>
<td>(2.5+)</td>
<td>Sets grid spacing on X and Y. Default is 0.0, 0.0.</td>
</tr>
<tr>
<td>short handles;</td>
<td>(10+)</td>
<td>If 1, handles are enabled. This should always be 1 for R13 files. Default 1</td>
</tr>
<tr>
<td>AD_OBJHANDLE handseed</td>
<td>(10+)</td>
<td>Handle seed. Default is all zeroes (NULLOBJHANDLE)</td>
</tr>
<tr>
<td>double hdrdoub[4];</td>
<td>(13+)</td>
<td>Unknown header doubles. Default 412148564080.0,1.0,1.0,1.0</td>
</tr>
<tr>
<td>long hdrlong[2];</td>
<td>(13+)</td>
<td>Unknown header longs. Default 24L, 0L.</td>
</tr>
<tr>
<td>short hdrshort;</td>
<td>(13+)</td>
<td>Unknown header short. Default 0.</td>
</tr>
<tr>
<td>char hdrstr[4][33];</td>
<td>(13+)</td>
<td>Unknown header strings. Default &quot;&quot;.</td>
</tr>
<tr>
<td>double hyperlinkbase[256]</td>
<td>(2000+)</td>
<td>Path for all relative hyperlinks in the drawing. If &quot;&quot;, the drawing path is used. Default &quot;.&quot;.</td>
</tr>
<tr>
<td>double insbase[3];</td>
<td>(2.5+)</td>
<td>Insertion base point of file (X,Y and Z). Default is 0.0, 0.0, 0.0.</td>
</tr>
<tr>
<td>short insunits;</td>
<td>(2000+)</td>
<td>Default drawing units for AutoCAD DesignCenter blocks:0 == Unitless; 1 == Inches; 2 == Feet; 3 == Miles; 4 == Millimeters; 5 == Centimeters; 6 == Meters; 7 == Kilometers; 8 == Microinches; 9 == Mils; 10 == Yards; 11 == Angstroms; 12 == Nanometers; 13 == Microns; 14 == Decimeters; 15 == Decameters; 16 == Hectometers; 17 == Gigameters; 18 == Astronomical units; 19 == Light years; 20 == Parsecs. Default is 1.</td>
</tr>
<tr>
<td>short isolines;</td>
<td>(2.5+)</td>
<td>Number of isolines per surface on objects. Default is 4.</td>
</tr>
<tr>
<td>short joinstyle;</td>
<td>(2000+)</td>
<td>Lineweight joint setting for new objects:0==none; 1== round; 2 == angle; 3 == flat. Default is 9.</td>
</tr>
<tr>
<td>double lenslength;</td>
<td>(10+)</td>
<td>Lens length used in perspective viewing for the current viewport. Default is 50.0.</td>
</tr>
<tr>
<td>short limcheck;</td>
<td>(2.5+)</td>
<td>1==no creation of objects outside limits. 0==allow creation of objects outside limits. Default is 0.</td>
</tr>
<tr>
<td>double limmin[2];</td>
<td>(2.5+)</td>
<td>Lower left limits point. Default is 0.0, 0.0.</td>
</tr>
<tr>
<td>double limmax[2];</td>
<td>(2.5+)</td>
<td>Upper right limits point. Default is 0.0, 0.0.</td>
</tr>
<tr>
<td>double ltscale;</td>
<td>(2.5+)</td>
<td>Global linetype scale. Default is 1.0.</td>
</tr>
<tr>
<td>short lunits;</td>
<td>(2.5+)</td>
<td>Controls linear units mode. 1==scientific 2==decimal 3==engineering 4==architectural 5==fractional. Default is 2</td>
</tr>
<tr>
<td>short luprec;</td>
<td>(2.5+)</td>
<td>Linear units precision. Default is 4.</td>
</tr>
<tr>
<td>short lwdisplay;</td>
<td>(2000+)</td>
<td>Controls the display of lineweights on the Model or Layout tab:0</td>
</tr>
</tbody>
</table>
short maxactv; (11+) Maximum number of active viewports. Default is 16.

unsigned short measurement 14+ Controls hatch pattern and linetype files used when a drawing is opened.
0==use files indicated by ANSIHatch and ANSILinetype registry settings. (English)
1==use files indicated by ISOHatch and ISOLinetype registry settings. (Metric)

char menu[512]; (2.5+) Currently loaded menu. Default is "acad".

short mirrtext; (9+) Sets text mirror method. 0==keep text orientation. 1==mirror the text. Default is 1.

short olestartup; (2000+) Controls whether the source application of an embedded OLE object loads when plotting. Default is 0.

short orthomode; (2.5+) 1==force draw at right angles. 0==don't. Default is 0.

short osmode; (2.5+) Controls object snap mode. Default is 0. These values may be ORed together.
0 == none
1 == endpoint
2 == midpoint
4 == center
8 == node
16 == quadrant
32 == intersection
64 == insertion
128 == perpendicular
256 == tangent
512 == nearest
1024 == quick
2048 == apparent intersection

short pdmode; (2.5+) Sets point display mode.
0 == a point.
1 == nothing.
2 == plus sign through the point.
3 == an X through the point.
4 == a vertical tick mark upward from the point.

You may add to any of these the values 32, 64, or 96 to draw a shape around the point; 32 draws a circle, 64 draws a square, and 96 draws both a circle and a square. Default is 0.

double pdsize; (2.5+) Controls point display size. 0 creates a point at 5% of the graphics area height. >0.0 indicates an absolute size. <0.0 indicates a percentage of the viewport size. Default is 0.0.

double pelevation; (11+) Default elevation for paperspace entities. Default is 0.0.

char pellipse; (13+) If 0, true ellipses are created by the AutoCAD ELLIPSE command. If 1, polyline representations are used. Default is 0.

double pextmin[3]; (11+) Paperspace extents lower left point.

double pextmax[3]; (11+) Paperspace extents upper right point.

char pickstyle; (13+) Sets mode for group selection and associative hatch selection.
0==no group or hatch selection
1==group selection
2==hatch selection
3==both
<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>double pinsbase[3]</code></td>
<td>(11+) Paperspace insbase. Default is 0.0, 0.0, 0.0.</td>
</tr>
<tr>
<td><code>short plimcheck</code></td>
<td>(11+) Limcheck for paperspace. Default is 0.</td>
</tr>
<tr>
<td><code>double plimmin[2]</code></td>
<td>(11+) Limits lower left corner for paperspace. Default is 0.0, 0.0.</td>
</tr>
<tr>
<td><code>short plinegen</code></td>
<td>(11+) Determines whether a linetype pattern in a polyline is continuous through the vertices. 1==yes, 0==no. Default is 1.</td>
</tr>
<tr>
<td><code>double plinewid</code></td>
<td>(2.5+) Default polyline width. Default is 0.0.</td>
</tr>
<tr>
<td><code>char proxygraphics</code></td>
<td>13+ Called saveimages in R13. Controls save of wireframe images of proxy (application-defined) entities with the entities in the file. 0==the application's definition of the entity controls whether or not an image is saved 1==an image is always saved 2==no images are saved Default is 0</td>
</tr>
<tr>
<td><code>short psltscale</code></td>
<td>(11+) Determines whether linetypes are scaled based on paperspace. 0==not scaled, 1==scaled. Default is 0.</td>
</tr>
<tr>
<td><code>char pstylemode</code></td>
<td>(2000+) Indicates whether the current drawing is in a Color-Dependent or Named Plot Style mode: 0 == Uses color-dependent plot style tables in the current drawing 1 == Uses named plot style tables in the current drawing. Default is 1.</td>
</tr>
<tr>
<td><code>double psvpscale</code></td>
<td>(2000+) View scale factor for new viewports: 0 == Scaled to fit. &gt;0 == Scale factor (a positive real value). Default is 0.0.</td>
</tr>
<tr>
<td><code>double pucsorg[3]</code></td>
<td>(11+) UCS origin in paperspace. Default is 0,0,0.</td>
</tr>
<tr>
<td><code>double pucsorgback[3]</code></td>
<td>(2000+) Point which becomes the new UCS origin after changing paper space UCS to ‘BACK’ when PUCSBASE is set to WORLD. Default is 0,0,0.</td>
</tr>
<tr>
<td><code>double pucsorgbottom[3]</code></td>
<td>(2000+) Point which becomes the new UCS origin after changing paper space UCS to ‘BOTTOM’ when PUCSBASE is set to WORLD. Default is 0,0,0.</td>
</tr>
<tr>
<td><code>double pucsorgfront[3]</code></td>
<td>(2000+) Point which becomes the new UCS origin after changing paper space UCS to ‘FRONT’ when PUCSBASE is set to WORLD. Default is 0,0,0.</td>
</tr>
<tr>
<td><code>double pucsorgleft[3]</code></td>
<td>(2000+) Point which becomes the new UCS origin after changing paper space UCS to ‘LEFT’ when PUCSBASE is set to WORLD. Default is 0,0,0.</td>
</tr>
<tr>
<td><code>double pucsorgright[3]</code></td>
<td>(2000+) Point which becomes the new UCS origin after changing paper space UCS to ‘RIGHT’ when PUCSBASE is set to WORLD. Default is 0,0,0.</td>
</tr>
<tr>
<td><code>double pucsorgtop[3]</code></td>
<td>(2000+) Point which becomes the new UCS origin after changing paper space UCS to ‘TOP’ when PUCSBASE is set to WORLD. Default is 0,0,0.</td>
</tr>
<tr>
<td><code>short pucsorthoview</code></td>
<td>(2000+) Orthographic view type of paper space UCS: 0 == UCS is not orthographic; 1 == Top; 2 == Bottom; 3 == Front; 4 == Back; 5 == Left; 6 == Right. Default is 0.</td>
</tr>
<tr>
<td><code>double pucsxdir[3]</code></td>
<td>(11+) UCS X direction in paperspace. Default is 1,0,0.</td>
</tr>
<tr>
<td><code>double pucsydir[3]</code></td>
<td>(11+) UCS Y direction in paperspace. Default is 0,1,0.</td>
</tr>
<tr>
<td><code>short qtextmode</code></td>
<td>(2.5+) 1==draw text as boxes, 0==draw text as text. Default is 0.</td>
</tr>
<tr>
<td>Variable</td>
<td>Version</td>
</tr>
<tr>
<td>----------</td>
<td>---------</td>
</tr>
<tr>
<td>regenmode;</td>
<td>2.5+</td>
</tr>
<tr>
<td>shadedge;</td>
<td>(11+)</td>
</tr>
<tr>
<td>shadedef;</td>
<td>(11+)</td>
</tr>
<tr>
<td>sketchinc;</td>
<td>(2.5+)</td>
</tr>
<tr>
<td>skpoly;</td>
<td>(2.5+)</td>
</tr>
<tr>
<td>snapang;</td>
<td>(2.5+)</td>
</tr>
<tr>
<td>snapbase[2];</td>
<td>(2.5+)</td>
</tr>
<tr>
<td>snapop;</td>
<td>(2.5+)</td>
</tr>
<tr>
<td>snapmode;</td>
<td>(2.5+)</td>
</tr>
<tr>
<td>snapstyle;</td>
<td>(2.5+)</td>
</tr>
<tr>
<td>snapunit[2];</td>
<td>(2.5+)</td>
</tr>
<tr>
<td>splframe;</td>
<td>(9+)</td>
</tr>
<tr>
<td>splinesegs;</td>
<td>(9+)</td>
</tr>
<tr>
<td>splinetype;</td>
<td>(10+)</td>
</tr>
<tr>
<td>stylesheet[512];</td>
<td>(2000+)</td>
</tr>
<tr>
<td>surfinc1;</td>
<td>(10+)</td>
</tr>
<tr>
<td>surfinc2;</td>
<td>(10+)</td>
</tr>
<tr>
<td>surfop;</td>
<td>(10+)</td>
</tr>
<tr>
<td>surfu;</td>
<td>(10+)</td>
</tr>
<tr>
<td>surfv;</td>
<td>(10+)</td>
</tr>
<tr>
<td>target[3];</td>
<td>(10+)</td>
</tr>
<tr>
<td>tdcreate[2];</td>
<td>(2.5+)</td>
</tr>
<tr>
<td>tdindwrg[2];</td>
<td>(2.5+)</td>
</tr>
<tr>
<td>tducreate[2];</td>
<td>(2000+)</td>
</tr>
<tr>
<td>tduupdate[2];</td>
<td>(2.5+)</td>
</tr>
<tr>
<td>tdusrimer[2];</td>
<td>(2.5+)</td>
</tr>
<tr>
<td>tduupdate[2];</td>
<td>(2000+)</td>
</tr>
<tr>
<td>textqlty;</td>
<td>(13+)</td>
</tr>
<tr>
<td>textsize;</td>
<td>(2.5+)</td>
</tr>
<tr>
<td>thickness;</td>
<td>(2.5+)</td>
</tr>
<tr>
<td>tilemode;</td>
<td>(11+)</td>
</tr>
<tr>
<td>Variable</td>
<td>Version</td>
</tr>
<tr>
<td>-----------------</td>
<td>---------</td>
</tr>
<tr>
<td>double tracewid;</td>
<td>(2.5+)</td>
</tr>
<tr>
<td>short treedepth;</td>
<td>(11+)</td>
</tr>
<tr>
<td>Variable</td>
<td>(Type)</td>
</tr>
<tr>
<td>--------------------------</td>
<td>--------</td>
</tr>
<tr>
<td>short ucsflag;</td>
<td>(10+)</td>
</tr>
<tr>
<td>short ucsicon;</td>
<td>(10+)</td>
</tr>
<tr>
<td>char ucspname[32];</td>
<td>(10+)</td>
</tr>
<tr>
<td>double ucsorg[3];</td>
<td>(10+)</td>
</tr>
<tr>
<td>double ucsorgback[3];</td>
<td>(2000+)</td>
</tr>
<tr>
<td>double ucsorgbottom[3];</td>
<td>(2000+)</td>
</tr>
<tr>
<td>double ucsorgfront[3];</td>
<td>(2000+)</td>
</tr>
<tr>
<td>double ucsorgleft[3];</td>
<td>(2000+)</td>
</tr>
<tr>
<td>double ucsorgright[3];</td>
<td>(2000+)</td>
</tr>
<tr>
<td>double ucsorgtop[3];</td>
<td>(2000+)</td>
</tr>
<tr>
<td>short ucsorthoview;</td>
<td>(2000+)</td>
</tr>
<tr>
<td>double ucsxdir[3];</td>
<td>(10+)</td>
</tr>
<tr>
<td>double ucsydir[3];</td>
<td>(10+)</td>
</tr>
<tr>
<td>short unitmode;</td>
<td>(11+)</td>
</tr>
<tr>
<td>short useri[5];</td>
<td>(2.5+)</td>
</tr>
<tr>
<td>double userr[5];</td>
<td>(2.5+)</td>
</tr>
<tr>
<td>short usrtimer;</td>
<td>(2.5+)</td>
</tr>
<tr>
<td>char versionguid[39];</td>
<td>(2000+)</td>
</tr>
<tr>
<td>Variable</td>
<td>Size</td>
</tr>
<tr>
<td>-------------------</td>
<td>------</td>
</tr>
<tr>
<td>double viewctr[3];</td>
<td>(2.5+)</td>
</tr>
<tr>
<td>double viewsize;</td>
<td>(2.5+)</td>
</tr>
<tr>
<td>double viewtwisthdr;</td>
<td>(10+)</td>
</tr>
<tr>
<td>short visretain;</td>
<td>(11+)</td>
</tr>
<tr>
<td>double vpointhdr[3];</td>
<td>(2.5+)</td>
</tr>
<tr>
<td>short worldview;</td>
<td>(10+)</td>
</tr>
<tr>
<td>char xedit;</td>
<td>(2000+)</td>
</tr>
<tr>
<td>AD_OBJHANDLE curdimstyleobjhandle;</td>
<td>(13+)</td>
</tr>
<tr>
<td>AD_OBJHANDLE curdimtextstyleobjhandle;</td>
<td>(2.5+)</td>
</tr>
<tr>
<td>AD_OBJHANDLE curlayerobjhandle;</td>
<td>(2.5+)</td>
</tr>
<tr>
<td>AD_OBJHANDLE curltypeobjhandle;</td>
<td>(2.5+)</td>
</tr>
<tr>
<td>AD_OBJHANDLE curmlstyleobjhandle;</td>
<td>(13+)</td>
</tr>
<tr>
<td>AD_OBJHANDLE curpucsobjhandle;</td>
<td>(11+)</td>
</tr>
<tr>
<td>AD_OBJHANDLE curtextstyleobjhandle;</td>
<td>(2.5+)</td>
</tr>
<tr>
<td>AD_OBJHANDLE curucsobjhandle;</td>
<td>(10+)</td>
</tr>
<tr>
<td>AD_OBJHANDLE curucsbaseobjhandle;</td>
<td>(2000+)</td>
</tr>
<tr>
<td>AD_OBJHANDLE</td>
<td>AD_OBJHANDLE</td>
</tr>
<tr>
<td>-----------------------</td>
<td>--------------------------------------------------------</td>
</tr>
<tr>
<td>curucorthorefobjhandle;</td>
<td>curpucsbasobjhandle;</td>
</tr>
<tr>
<td>If model space UCS is orthographic, this is the name of the UCS that the orthographic UCS is relative to. If blank, UCS is relative to WORLD.</td>
<td>UCS that defines the origin and orientation of orthographic paper space UCS settings.</td>
</tr>
</tbody>
</table>
ENTITY DATA:

Entities are the things that are drawn in AutoCAD.

OpenDWG Toolkit returns data on entities in a structure, AD_ENT_HDR, and a union AD_ENT. These are the header and body of an entity, respectively. The entity header contains data that is common to all entities; the body contains data specific to the particular entity type.

ENTITY HEADER:

Here is the definition for an entity header:

typedef struct adenhdstru {
    unsigned short enttype;                         (2.5+)
    AD_OBJHANDLE enthandle;                        (10+ main ents only, 11+ all)
    AD_OBJHANDLE entlayerobjhandle;                (2.5+)
    AD_OBJHANDLE enttypeobjhandle;                 (2.5+)
    AD_OBJHANDLE xdicobjhandle;                    (14+)
    double entthickness;                           (2.5+)
    short entcolor;                                (2.5+)
    double extrusion[3];                           (10+)
    double entltscale;                             (13+)
    unsigned char entflags;                        (11+, extru (10+), invisible (13+)
    short linewidth;                               (2.5+)
    long userdata;                                 (13+)
    long numreactors;                              (13+)
    AD_VMA DDR xdblob;                             (11+)
    AD_VMADDR reactorblob;                         (13+)
} AD_ENT_HDR,*PAD_ENT_HDR;

enttype is the type of entity.

The type of entity is determined in one of two ways. enttype may be set to one of the defined constants listed below:

    AD_ENT_LINE, AD_ENT_POINT, AD_ENT_CIRCLE, AD_ENT_SHAPE, AD_ENT_ELLIPSE,
    AD_ENT_SPLINE, AD_ENT_TEXT, AD_ENT_ARC, AD_ENT_TRACE, AD_ENT_REGION, AD_ENT_SOLID,
    AD_ENT_BLOCK, AD_ENT_ENDBLK, AD_ENT_INSERT, AD_ENT_ATTDEF, AD_ENT_ATTRIB,
    AD_ENT_SEQEND, POLYAD_ENT_LINE, AD_ENT_VERTEX, AD_ENT_LINE3D, AD_ENT_FACE3D,
    AD_ENT_DIMENSION, AD_ENT_VIEWPORT, AD_ENT_SOLID3D, AD_ENT_RAY, XAD_ENT_LINE,
    AD_ENT_MTEXT, AD_ENT_LEADER, AD_ENT_TOLERANCE, AD_ENT_MLINE, AD_ENT_BODY.

However, there are also entity types (starting with R13) called "proxy" entities, which have values which vary from drawing to drawing. These types are tested for by getting the appropriate enttype from a function. The functions are:

adLwpLineEnttype(AD_DB_HANDLE handle);          for lightweight polyline entities
adHatchEnttype(AD_DB_HANDLE handle);            for hatch entities
adImageEnttype(AD_DB_HANDLE handle);            for image entities
adOle2FrameEnttype(AD_DB_HANDLE handle);        for Ole2 entities
adWipeoutEnttype(AD_DB_HANDLE handle);          for wipeout entities
adArcAlignedTextEnttype(AD_DB_HANDLE handle);   for arc aligned text entities
adRTextEnttype(AD_DB_HANDLE handle);            for rtext entities
Finally, the entity can also be a generic proxy entity; this is tested using the macro
AD_IS_A_GENERIC_PROXYENT(a).

- **enthandle** is the AD_OBJHANDLE for this entity which uniquely identifies it.
- **entlayerobjhandle** is the AD_OBJHANDLE for the entity's layer.
- **enttypeobjhandle** is the AD_OBJHANDLE for the linetype with which this entity is to be drawn.
- **xdicobjhandle** is the AD_OBJHANDLE for the ACAD_XDICTIONARY object associated with this entity.
- **entthickness** is the thickness of this entity in the extrusion direction.
- **entcolor** is the color of this entity, which is either an AutoCAD color number 1..255 or one of the defined
  constants AD_COLOR_BLACK or AD_COLOR_BROWN.
- **extrusion** is the vector indicating the normal to the entity, and thus the direction along which it will be
  extruded if its thickness is nonzero.
- **entltscale** is the scale at which this entity's linetype is applied. Default is 1.0.
- **entflags** contains various flags indicating the entity's state. These flags should be tested and set using
  the following defined macros:

  Is entity in paperspace?
  test:  adEntIsInPaperspace(adenhd->entflags)
  set: adSetEntIsInPaperspace(adenhd->entflags)
        adSetEntIsInModelspace(adenhd->entflags)

  Is entity's extrusion valid (i.e. not 0,0,1)?
  test:  adEntHasExtrusion(adenhd->entflags)
  set: adSetEntHasExtrusion(adenhd->entflags)
        adSetEntHasNoExtrusion(adenhd->entflags)

  Is entity invisible?
  test:  adEntIsInvisible(adenhd->entflags)
  set:  adSetEntIsInvisible(adenhd->entflags)
        adSetEntIsNotInvisible(adenhd->entflags)

- **lineweight** is the lineweight value of the entity. Default: -1.
- **userdata** is a long integer for use by the programmer. This value is not altered by OpenDWG Toolkit; it is
  simply stored in virtual memory, and retrieved when an entity is read. It is not saved in the output file.
- **xdblob** is the blob containing the entity's extended entity data, if any. If none, xdblob will be equal to
  AD_VMNULL.
- **numreactors** is the number of persistent reactors attached to this entity.
- **reactorblob** is the blob containing the persistent reactors, if any.

**ENTITY BODY:**

The AD_ENT union is a union of structures, each of which is appropriate for a particular entity type.
Descriptions of the entity data for each entity type are listed below. In each description, we give the defined
constant to which adenhd->enttype would be equal for this entity type, the name of the structure definition for
this entity type, and the name of this structure in the AD_ENT union. These descriptions assume the following
variable definitions:

PAD_ENT aden;
PAD_ENT_HDR adenhd;
**3D FACE:**

Defined constant adenhd->enttype==AD_ENT_FACE3D.
Structure definition AD_FACE3D.
Found in AD_ENT union structure face3d (e.g. aden->face3d).
Earliest AutoCAD version for this entity: 2.6

A 3D face is a 3D planar region bounded by four points.

```c
typedef struct adface3dstru {
    double pt0[3];
    double pt1[3];
    double pt2[3];
    double pt3[3];
    short faceflag;
} AD_FACE3D, *PAD_FACE3D;
```

*pt0, pt1, pt2 and pt3* are the bounding points of the face.
*faceflag* is a bit coded field indicating which segments of the face, if any, are invisible. It can be tested by ANDing with the defined constants

- AD_FACE3D_FACE1INVIS (first segment is invisible)
- AD_FACE3D_FACE2INVIS (second segment is invisible)
- AD_FACE3D_FACE3INVIS (third segment is invisible)
- AD_FACE3D_FACE4INVIS (fourth segment is invisible)

**3D LINE:**

Defined constant adenhd->enttype==AD_ENT_LINE3D.
Structure definition AD_LINE3D.
Found in AD_ENT union structure line3d (e.g. aden->line3d).
Valid for AutoCAD versions: 2.6+

A 3D line is an entity which only appeared in two versions of AutoCAD (9 and 10). It is a line with each endpoint specified with 3 dimensions. Versions of AutoCAD beyond 10 simply used LINE entities for this purpose. Unless you have a specific need to emit Release 9 or Release 10 format with 3D lines in the drawings, we suggest that you use LINE entities instead.

```c
typedef struct adline3dstru {
    double pt0[3];
    double pt1[3];
} AD_LINE3D, *PAD_LINE3D;
```

*pt0 and pt1* are the endpoints of the line.
ACIS DATA:

Defined constant adenhd->enttype==AD_ENT_SOLID3D, AD_ENT_REGION, or AD_ENT_BODY.
Structure definition AD_ACIS.
Found in AD_ENT union structure acisobj (e.g. aden->acisobj).
Valid for AutoCAD versions: 13+

typedef struct adacisstru {
    double pt0[3];
    AD_VMADDR ldblob;
    AD_VMADDR imgdata;
    char objcamefromdwgfile;
} AD_ACIS, *PAD_ACIS;

Starting with AutoCAD Release 13, Autodesk included a solids modeler called ACIS in AutoCAD. Autodesk embeds the textual data (called SAT data) within the DWG or DXF file that ACIS uses to perform its function. Autodesk encrypts this data; OpenDWG Toolkit decrypts it.

Although there are three different entity types (AD_ENT_SOLID3D, AD_ENT_REGION, and AD_ENT_BODY), since they are all stored in the same format OpenDWG Toolkit uses a common structure, AD_ACIS, to return them.

Note the following important limitation regarding ACIS data:
AutoCAD occasionally stores a wireframe image of ACIS objects in DWG files. OpenDWG Toolkit has been unable to get a complete decryption of this data.

As a result, OpenDWG Toolkit cannot originate ACIS data for a DWG file; that is, you cannot take ACIS data, perhaps from some external source, and write it out into a DWG file, without the related image, and create a valid DWG file. OpenDWG Toolkit will issue a warning in this instance. You can write ACIS data into a DXF file, however.

This limitation also holds for reading a DXF file and writing a DWG file. The image data is not stored in the DXF file, so OpenDWG Toolkit does not have access to it.

If you have read an existing DWG file, however, OpenDWG Toolkit will carry the image data associated with the ACIS object along with it, and will write it out correctly into an output DWG or DXF file.

pt0 is the point at which the entity is inserted.
ldblob contains the textual data describing the entity. Read the data from this blob using adReadAcisString.
imgdata is a blob containing data generated by AutoCAD which describes the wireframe image of the entity.
objcamefromdwgfile is a flag indicating whether this ACIS object came from a DWG file, and thus whether or not it can validly be written out by OpenDWG Toolkit to a new DWG file.
ARC:

Defined constant adenthd->enttype==AD_ENT_ARC.
Structure definition AD_ARC.
Found in AD_ENT union structure arc (e.g. aden->arc).
Valid for AutoCAD versions: 2.5+

typedef struct adarcstru {
    double pt0[3];
    double radius;
    double stang;
    double endang;
} AD_ARC,*PAD_ARC;

pt0 is the center point of the arc.
radius is the radius of the arc.
stang is the start angle of the arc.
endang is the end angle of the arc.

Arcs are always drawn counterclockwise from the start angle to the end angle.

ARCALIGNEDTEXT:

Function to retrieve enttype adenthd->enttype== adArcAlignedTextEnttype(handle);
Structure definition AD_ARCALIGNEDTEXT.
Found in AD_ENT union structure arcalignedtext (e.g. aden->arcalignedtext).
Valid for AutoCAD versions: 2000+

typedef struct adarcalignedtextstru {
    char text[1024];
    char fontname[AD_MAX_STRLEN];
    char bigfontname[AD_MAX_STRLEN];
    AD_OBJHANDLE textstyleobjhandle;
    double pt0[3];
    double radius;
    double widthfactor;
    double height;
    double charspacing;
    double offsetfromarc;
    double rightoffset;
    double leftoffset;
    double startangle;
    double endangle;
    short charorder;
    short direction;
    short alignment;
    short side;
    short bold;
    short italic;
    short underline;
    short charset;
    short pitch;
    short fonttype;
    long color;
    short wizardflag;
    AD_VMADDR grblob;
    AD_OBJHANDLE arcobjhandle;
} AD_ARCALIGNEDTEXT, *PAD_ARCALIGNEDTEXT;

text indicates the text.
fontname indicates the font name.
bigfontname indicates the big font name.
textstyleobjhandle indicates the text style.
pt0 indicates the center point.
radius indicates the radius of the arc.
widthfactor indicates the width factor.
height indicates the text height.
charspacing indicates the character spacing.
offsetfromarc indicates the offset from the arc.
rightoffset indicates the right offset.
leftoffset indicates the left offset.
startangle indicates the start angle in radians.
endangle indicates the end angle in radians.
charorder indicates the character order, where 0 == normal and 1 == reverse.
direction indicates the direction flag.
alignment indicates the alignment.
side indicates the side flag.
bold indicates the bold flag.
italic indicates the italic flag.
underline indicates the underline flag.
charset indicates the character set value.
pitch indicates the pitch and family values.
fonttype indicates the font type, where 0 == TTF, and 1 == SHX.
color indicates the color index.
wizardflag indicates the wizard flag.
grblob contains 310 data.
arcobjhandle indicates the arc entity handle.

ATTDEF:

Defined constant adenhd->enttype==AD_ENT_ATTDEF.
Structure definition AD_ATTDEF.
Found in AD_ENT union structure attdef (e.g. aden->attdef).
Valid for AutoCAD versions: 2.5+

typedef struct adattdefstru {
    double pt0[3];
    char attflag;
    short fieldwidth;
    AD_TDATA tdata;
    char prompt[513];
    char tag[513];
    char defaultval[513];
} AD_ATTDEF,*PAD_ATTDEF;

An attribute definition is a definition of an attribute to be inserted when a block is inserted. When a block containing attribute definitions is inserted, the user is prompted for the values of those attributes, and attribute entities are placed in the database following the insert entity, and the list is terminated with a sequence end entity.

pt0 is the insertion point for the attribute.
attflag is a bit coded flag containing parameters for this attribute definition. They are: invisibility, constant (attribute always has the same value), verify, and preset. These attributes can be tested by ANDing the attflag with the defined constants AD_ATTDEF_INVIS, AD_ATTDEF_CONST, AD_ATTDEF_VERIFY, AD_ATTDEF_PRESET.
fieldwidth is unused and should be set to 0.
tdata is a substructure containing parameters defining how the attribute definition, and the attribute made from it, should be drawn. See the TEXT entity for a description of this structure.

prompt is the text prompt given to the user when this attribute is being inserted into the drawing.
tag is the attribute tag, which is the unique identifying string for this attribute.
defaultval is the default value, if any, for this attribute.

ATTRIBUTE:

Defined constant adenhd->enttype==AD_ENT_ATTRIB.
Structure definition AD_ATTRIB.
Found in AD_ENT union structure attrib (e.g. aden->attrib).
Valid for AutoCAD versions: 2.5+

typedef struct adattribstru {
  double pt0[3];
  char  attflag;
  short fieldwidth;
  AD_TDATA tdata;
  char  tag[513];
  char  attval[2049];
} AD_ATTRIB,*PAD_ATTRIB;

An attribute is a piece of arbitrary text data that follows an insert entity.

pt0 is the insertion point for the attribute.
attflag is a bit coded flag containing parameters for this attribute. They are: invisibility, constant (attribute always has the same value), verify, and preset. These attributes can be tested by ANDing the attflag with the defined constants AD_ATTRIB_INVIS, AD_ATTRIB_CONST, AD_ATTRIB_VERIFY, and AD_ATTRIB_PRESET.
fieldwidth is unused and should be set to 0.
tdata is a substructure containing parameters defining how the attribute definition, and the attribute made from it, should be drawn. See the TEXT entity for a description of this structure.
tag is the attribute tag, which is the unique identifying string for this attribute.
attval is the value of the attribute.

BLOCK:

Defined constant adenhd->enttype==AD_ENT_BLOCK.
Structure definition AD_BLOCK.
Found in AD_ENT union structure block (e.g. aden->block).
Valid for AutoCAD versions: 2.5+

typedef struct adblockstru {
  double base[3];
  char  name2[AD_MAX_STRLEN];
  char  xrefpath[512];
} AD_BLOCK,*PAD_BLOCK;

A block entity is only found as the first entity of a block.

base is the base point for the block.
name2 is the name of the block.
xrefpath is the path to the external reference file, if this block is an external reference.
BODY: See ACIS DATA

CIRCLE:

Defined constant adenhd->enttype==AD_ENT_CIRCLE.
Structure definition AD_CIRCLE.
Found in AD_ENT union structure circle (e.g. aden->circle).
Valid for AutoCAD versions: 2.5+

typedef struct adcirclestru {
    double pt0[3];
    double radius;
} AD_CIRCLE, *PAD_CIRCLE;

pt0 is the center point of the circle.
radius is the radius.

CONSTRUCTION LINE:

Defined constant adenhd->enttype==AD_ENT_XLINE.
Structure definition AD_XLINE.
Found in AD_ENT union structure xline (e.g. aden->xline).
Valid for AutoCAD versions: 13+

typedef struct adxlinestru {
    double pt0[3];
    double unitvec[3];
} AD_XLINE, *PAD_XLINE;

A construction line is an infinitely long line with one fixed point and a direction. The construction line extends to infinity in the defined direction and its exact opposite.

pt0 is the fixed point on the construction line.
unitvec is a vector indicating the direction of the construction line on X, Y and Z.
DIMENSION:

Defined constant adenhd->enttype==AD_ENT_DIMENSION.
Structure definition AD_DIM.
Found in AD_ENT union structure dim (e.g. aden->dim).
Valid for AutoCAD versions: 2.6+

typedef struct addimensionstru {
  short  dimcloneinfopresent;
  double dimcloneinspt[3];
  unsigned char   dimflag;
  double arcdefpt[3];
  double defpt2[3];
  double defpt3[3];
  double defpt4[3];
  double leaderlen;
  double dimrotang;
  double dimtextrot;
  double dimoblique;
  double dimtextmidpt[3];
  double dimlinedefpt[3];
  short  attachpt;
  short  linespacingstyle;
  double linespacingfactor;
  double actualmeasurement;
  AD_OBJHANDLE blockheaderobjhandle;
  AD_OBJHANDLE dimstyleobjhandle;
  double reserved;
  char   dimtext[1024];
} AD_DIM, *PAD_DIM;

A dimension consists of the dimension entity, which contains the control parameters for the dimension, and a block definition which contains the entities defining the visual representation of the block. This block definition should be an AutoCAD “anonymous” block and have the name “*D”.

dimcloneinfopresent indicates that the dimcloneinspt is valid.
dimcloneinspt is the insertion point for clones of this dimension. (12,22,32 in DXF).
dimflag indicates the type of the dimension, and is a bit tricky to unravel. To get the dimension type, AND dimflag with 7, then compare it to the following defined constants:

AD_DIMTYPE_ROTATED     (linear dimension, possible rotated)
AD_DIMTYPE_ALIGNED     (aligned between 2 points)
AD_DIMTYPE/angular     (dimensions an angle)
AD_DIMTYPE_DIAMETER    (diameter)
AD_DIMTYPE_RADIUS      (radius)
AD_DIMTYPE_DIM LAN Đầu 3PT  (3 point angular dimension)
AD_DIMTYPE_ORDINATE   (ordinate dimension)

Two of the other bits in this flag are bit coded; you should test these by ANDing dimflag with either of the following defined constants:

AD_DIMTYPE_LANGUAGBIDIM (this ordinate dimension is in the X direction)
AD_DIMTYPE_TEXTPOS     (indicates the dim text is at a user-defined location).
arcdefpt defines the dimension arc for an angular dimension. This is equivalent to the 16,26,36 group in DXF.

defpt2 is a definition point. This is equivalent to the 13,23,33 group in DXF.
defpt3 is a definition point. This is equivalent to the 14,24,34 group in DXF.
defpt4 is a definition point. This is equivalent to the 15,25,35 group in DXF.
leaderlen is the length of the dimension leader.
dimrotang is the rotation angle for the dimension in radians.
dimtextrot is the text rotation angle.
dimoblique is the obliquing angle for the dimension.
dimtextmidpt is the midpoint of the text. (11,21,31 in DXF).
dimlinedefpt is the definition point for the dimension line. (10,20,30 in DXF).
ucsxangle is the angle of the UCS when the dimension was created.
attachpt is the attachment point of dimension, defined as
  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.
linespacingstyle is either 1 (at least), or 2 (exact).
linespacingfactor is the percentage of default line spacing to be applied. Valid values are from 0.25 to 4.0.
actualmeasurement is read-only value used by AutoCAD.
blockheaderobjhandle is the object handle for the block which contains the entities composing the image of this dimension.
dimstyleobjhandle is the object handle of the dimension style for this dimension.
reserved is the overall rotation of the dimension.
dimtext is the value of the text string for this dimension.

For rotated (linear) dimensions, the first extension line is specified by defpt2. The second extension line is specified by defpt3, and the dimension line is specified by dimlinedefpt.

For angular dimensions, defpt2 and defpt3 are the endpoints of the first extension line. dimlinedefpt and defpt4 are the endpoints of the second extension line. arcdefpt specifies the dimension line arc.

For angular 3 point dimensions, defpt4 is the vertex of the angle. defpt2 is the endpoint of the first extension line. defpt3 is the endpoint of the second extension line. dimlinedefpt specifies the arc for the dimension line.

For diameter dimensions, defpt4 is the point selected on the circle or arc being dimensioned. dimlinedefpt is the point on the circle exactly across from the selection point.

For ordinate dimensions, defpt3 is the point which was selected. defpt4 is the point indicating the endpoint of the leader.

ELLIPSE:

Defined constant adenhd->enttype==AD_ENT_ELLIPSE.
Structure definition AD_ELLIPSE.
Found in AD_ENT union structure ellipse (e.g. aden->ellipse).
Valid for AutoCAD versions: 13+
typedef struct adellipsestru {
    double pt0[3];
    double pt1offset[3];
    double minortomajorratio,startparam,endparam;
} AD_ELLIPSE,*PAD_ELLIPSE;

pt0 is the center point of the ellipse. pt1offset is the offset to pt1, the second defining point, from pt0. minortomajorratio is the ratio of the minor axis to the major axis. startparam and endparam are the start and end angles for the ellipse, allowing creation of an elliptical arc rather than a full ellipse.
ENDBLOCK:

Defined constant adenhd->enttype==AD_ENT_ENDBLK.
Structure definition: NONE
Not in AD_ENT union.
Valid for AutoCAD versions: 2.5+

There is no data for an endblk.

HATCH:

Function to retrieve enttype adenhd->enttype==adHatchEnttype(handle).
Structure definition: AD_HATCH
Found in AD_ENT union structure hatch (e.g. aden->hatch)
Valid for AutoCAD versions: 14+

typedef struct adhatchstru {
    double pt0[3];
    char name[AD_MAX_STRLEN];
    unsigned short associative,solidfill;
    AD_VMADDR pathblob;
    long numpaths;
    unsigned short style;
    unsigned short patterntype;
    double angle;
    double scaleorspacing;
    unsigned short doublehatch;
    unsigned short numdeflines;
    AD_VMADDR hatchblob,boundaryobjblob;
    double pixelsize;
    AD_VMADDR seedpointblob;
    long numseedpoints;
} AD_HATCH, *PAD_HATCH;

A hatch is a crosshatched area, which can be solid filled. It is a generative element, meaning that it is recomputed when drawn, unlike previous AutoCAD hatches which were stored as inserts of blocks containing the entities composing the hatch.

pt0, according to the DXF documentation, has its X and Y coordinates always 0.0 and it's Z coordinate indicates the elevation.

name is the hatch pattern name.

associative is 1 if the hatch is associative, otherwise 0.

solidfill is 1 if the hatch is a solid fill, 0 if it is a pattern fill.

numpaths indicates the number of boundary paths in the hatch.

style indicates the hatch style. 0 means hatch only the "odd parity" area, and is called the "Normal" style. 1 means hatch only the outermost area, and is called the "Outer" style. 2 means
pattern type indicates the type of pattern. 0 means user-defined, 1 means predefined, and 2 means custom.

angle is the hatch pattern angle in radians for pattern fill hatches.

scale or spacing indicates the hatch pattern scale or spacing for pattern fill hatches.

double hatch is 1 if the hatch is to be double hatched, otherwise 0.

num definition lines is the number of pattern definition lines in the hatch.

hatch blob contains the pattern data.

boundary obj blob contains the boundary objects.

pixel size indicates the size of the pixels in AutoCAD units.

seed point blob contains the seed points for the hatch.

num seed points indicates the number of seed points in the hatch.

Reading the boundaries:

The boundary obj blob contains the boundary objects. These are edges, which are either polylines, lines, arcs, ellipses, or splines, which compose the boundary of the hatch. There may be multiple boundary paths; the number of paths is indicated by num paths. Reading these boundaries is a fairly involved process; prnstuff.h contains a full example of reading hatch boundaries.

Reading the pattern data:

The hatch blob contains the hatch pattern data. num definition lines indicates the number of definition lines in the hatch pattern.

To read the hatch blob, perform the following num definition lines times:

- adReadBlobDouble() to get the line angle.
- adReadBlob2Double() to get the pattern through point, indicating a point through which the pattern passes.
- adReadBlob2Double() to get the pattern offset.
- adReadBlobShort() to get the number of dash length items for this definition.

Repeat "number of dash length items" times:

- adReadBlobDouble() to get the dash item, which is the length of a dash

Again, there is a full example of reading hatch boundaries in prnstuff.h.

Reading the seed points:

The seed point blob holds the seed points for the hatch pattern. num seed points indicates the number of seed points in the hatch. To read the blob, simply use adReadBlobDouble() to read the x and y coordinates, and repeat that num seed points times.

There is a full example of reading hatch boundaries in prnstuff.h.
Function to retrieve enttype adenhd->enttype==adImageEnttype(handle);
Structure definition AD_IMAGE.
Found in AD_ENT union structure image (e.g. aden->image).
Valid for AutoCAD versions: 14+

typedef struct adimagestru {
  long classversion;
  double pt0[3];
  double size[2];
  double uvec[3],vvec[3];
  unsigned short clipping, brightness, contrast, fade, clipboundtype;
  unsigned short displayprops;
  long numclipverts;
  double rectclipvert0[2], rectclipvert1[2];
  AD_VMADDR polyclipvertblob;
  AD_VMADDR grblob;
  AD_OBJHANDLE imagedefobjhandle, imagedefreactorobjhandle;
} AD_IMAGE, *PAD_IMAGE;

An image is a raster image displayed within a drawing. The image data is not actually stored in the DWG file; rather a pointer to the raster file containing the image is kept in the DWG file.

classversion is the version of the image class; currently 0.
pt0 is the insertion point of the text.
size is the image size and pixels.
uvec is the u vector for any single pixel
vvec is the v vector for a single pixel
clipping is 1 if clipping is on, 0 if it is off
brightness is the brightness of the image; the range of values is 0 to 100, with a default value of 50.
contrast is the contrast of the image; the range of values is 0 to 100, with a default value of 50.
fade is the fade value of the image; the range of values is 0 to 100 with a default value of 50.
clipboundtype indicates the type of clipping boundary used, if any. 1 means rectangular, 2 means polygonal.
displayprops is the image display properties, bitcoded:
  1==Show image
  2==Show image when not aligned with screen
  4==use clipping boundary
  8==transparency on
numclipverts is the number of clipping boundary vertices in the polyclipvertblob.
rectclipvert0 and rectclipvert1 are the corners of the clipping rectangle, if clipboundtype is set as rectangular.
polyclipvertblob contains the clip boundary vertices if the clipboundtype is polygonal. To read these vertices, simply use adReadBlob2Double() to read numclipverts vertices.
grblob is an image blob for the image entity. This usually contains a crosshatch pattern.
imagedefobjhandle is the AD_OBJHANDLE of the image def object associated with this image.
imagedefreactorobjhandle is the AD_OBJHANDLE of the image def object associated with this image.

We do not yet provide a full example of image entity creation.

**INSERT:**

Defined constant adenhd->enttype==AD_ENT_INSERT.
Structure definition AD_INSERT.
Found in AD_ENT union structure insert (e.g. aden->insert).
Valid for AutoCAD versions: 2.5+

An insert is an instance of a block in the drawing. The insert entity indicates that a block is in a given position, with the given scaling, rotation, etc. If there are attributes connected to this insert, the attribsfollow field should be set to 1, and the AD_ENT_ATTRIB entities should follow this entity. At the end of the sequence of AD_ENT_ATTRIBs should then come an AD_ENT_SEQEND entity.

typedef struct adinsertstru {
    short attribsfollow;
    double pt0[3];
    double xscale;
    double yscale;
    double zscale;
    double rota;
    double rowdist;
    double coldist;
    short numrows;
    short numcols;
    AD_OBJHANDLE blockheaderobjhandle;
} AD_INSERT,*PAD_INSERT;

attribsfollow indicates that attribute entities follow this insert entity.
pt0 is the insertion point for the inserted block.
xscale is the scaling in the x direction.
yscale is the scaling in the y direction.
zscale is the scaling in the z direction.
rota is the rotation angle of the block in radians.

The next four fields pertain to blocks which were inserted with the "MINSERT" command, which inserts an array of the block.

rowdist is the distance between rows. (0 for no array)
coldist is the distance between columns. (0 for no array)
numrows is the number of rows. (1 for no array)
numcols is the number of columns. (1 for no array)

blockheaderobjhandle is the AD_OBJHANDLE of the block header for the inserted block.
LEADER:

Defined constant adenhd->enttype==AD_ENT_LEADER.
Structure definition AD_LEADER.
Found in AD_ENT union structure leader (e.g. aden->leader).
Valid for AutoCAD versions: 13+

type def struct adleaderstru {
    unsigned short numpoints;
    unsigned char pathtype,arrowheadon,hooklineonxdir,hashookline;
    unsigned short annotype,colorforbyblock;
    double annoht,annowid;
    double txtoffset[3];
    double xdir[3];
    double offsettoblkinspt[3];
    AD_OBJHANDLE annohandle, dimstyleobjhandle;
    AD_VMADDR ldblob;
} AD_LEADER, *PAD_LEADER;

A leader is a sequence of line segments with an arrowhead at one end and, typically, text at the other. Note that the text is a separate entity, and is NOT part of the leader itself.

numpoints is the number of points defining the leader.
pathtype is the type of path the leader follows, and is equal to one of the following defined constants:

AD_LEADER_PATH_STRAIGHT (straight lines)
AD_LEADER_PATH_SPLINE (spline)

arrowheadon is 1 if the arrowhead is on, 0 if it is off.
hooklineonxdir is 1 if the hook line (or spline end tangent) is in the same direction as the x direction; 0 if it is not.
hashookline is 1 if the leader has a hook line.
annotype indicates the type of annotation, and is equal to one of the following defined constants:

AD_LEADER_ANNO_TEXT (annotation is mtext)
AD_LEADER_ANNO_TOLERANCE (annotation is a tolerance entity)
AD_LEADER_ANNO_BLOCK (annotation is a block reference)
AD_LEADER_ANNO_NONE (no annotation)

colorforbyblock is the color to use if the leader is drawn with BYBLOCK color.
annoht is the height of the associated mtext entity.
annowid is the width of the associated mtext entity.

txtoffset is the offset of the last leader vertex from the annotation placement point.
xdir is a vector indicating the horizontal direction of the text.
offsettoblkinspt is the offset of the last leader vertex from block reference insertion point.
annohandle is a reference to the associated annotation entity.
dimstyleobjhandle is the AD_OBJHANDLE of the dimension style which gives some of the draw parameters for this leader.
ldblob contains the points that compose the leader. You can read points from this blob using adReadLeaderPoint.

A number of parameters of the leader are stored in the leader's dimension style. Specifically, the fields dimst.dimblk1 and dimst.dimblk2 will indicate the block to be drawn for the leader's arrowhead. Typically these are set to one of several standard blocknames for blocks which AutoCAD constructs for leaders. They are:

.Closed unfilled arrow
.Dot a filled dot
_NONE    no arrowhead
_OBLIQUE  slash
_OPEN    unfilled, unclosed arrowhead
_ORIGIN  open dot
_OPEN90  right-angle style arrowhead

The arrowhead size is also stored in the dimension style, in \texttt{dimst.dimasz}, and the \texttt{dimst.dimgap} gap variable is also used.

\textbf{LIGHTWEIGHT POLYLINE:}

Function to retrieve \texttt{enttype)==adLwplineEnttype(handle);}
Structure definition \texttt{AD_LWPLINE}.
Found in \texttt{AD_ENT union structure lwpline (e.g. aden->lwpline)}.
Valid for AutoCAD versions: 14+

\begin{verbatim}
typedef struct adlwplinestru {
  long numpoints;
  unsigned short flag;
  double elevation;
  double constantwidth;
  AD_VMADDR ldblob;
} AD_LWPLINE, *PAD_LWPLINE;
\end{verbatim}

A lightweight polyline is Autodesk's method for getting away from the overloaded polyline/vertex/endseq method of data storage. That method incurs substantial overhead due to layers, linetypes, handles, elevation etc. being stored for every single vertex in the polyline. The lightweight polyline stores only a subset of the information contained in a "regular" polyline. Like AutoCAD Release 14, OpenDWG Toolkit automatically converts polylines to lightweight polylines when their characteristics allow it. This behavior can be changed with the function \texttt{adConvertPlinesOnLoad();} passing in a 1 causes conversion to take place, 0 suppresses conversion.

\textbf{numpoints} is the number of points (vertices) in the lightweight polyline.
\textbf{flag} is a bit coded flag which can be tested by anding with the following constants:

\begin{verbatim}
AD_LWPLINE_HAS_CONSTANT_WIDTH
AD_LWPLINE_HAS_BULGES
AD_LWPLINE_HAS_WIDTHS
AD_LWPLINE_CONTINUELT
AD_LWPLINE_PLINEGEN
AD_LWPLINE_IS_CLOSED
\end{verbatim}

\textbf{elevation} is the Z elevation of the polyline.
\textbf{constantwidth} is set to the value of the polyline's constant width, if \texttt{aden->lwpline.flag & AD_LWPLINE_HAS_CONSTANT_WIDTH} is nonzero.
\texttt{ldblob} contains the vertex data.

\textbf{Reading lightweight polyline vertex data:}

To read data from the ldblob, the proper sequence of operations is as follows:

Repeat \textbf{numpoints} times:
  \texttt{adReadBlob2Double} to get the vertex point.
  if (aden->lwpline.flag & AD_LWPLINE_HAS_BULGES)
adReadBlobDouble() to get the bulge;
if (aden->lwpline.flag & AD_LWPLINE_HAS_WIDTHS)
adReadBlob2Double() to get the widths;
end repeat

LINE:

Defined constant adenhd->enttype==AD_ENT_LINE.
Structure definition AD_LINE.
Found in AD_ENT union structure line (e.g. aden->line).
Valid for AutoCAD versions: 2.5+

typedef struct adlinestru {
   double pt0[3];
   double pt1[3];
} AD_LINE, *PAD_LINE;

pt0 and pt1 are the endpoints of the line.

MULTI-LINE:

Defined constant adenhd->enttype==AD_ENT_MLINE.
Structure definition AD_MLINE.
Found in AD_ENT union structure mline (e.g. aden->mline).
Valid for AutoCAD versions: 13+

typedef struct admlinestru {
   double scale;
   short justification;
   short openclosed, linesinstyle;
   unsigned short numverts;
   double basept[3];
   AD_OBJHANDLE mlinestyleobjhandle;
   AD_VMADDR ldblob;
} AD_MLINE, *PAD_MLINE;

scale is the scale of the multi-line.
justification is the multi-line’s justification, and is equal to one of the following defined constants:

   AD_MLINE_JUST_TOP
   (justified from top)
   AD_MLINE_JUST_ZERO
   (justified from center)
   AD_MLINE_JUST_BOTTOM
   (justified from bottom)

openclosed indicates whether the multi-line is open or closed, and is equal to one of the following defined constants:

   AD_MLINE_OPEN
   AD_MLINE_CLOSED

linesinstyle indicates the number of lines in the corresponding multi-line style.
umverts is the number of vertices that compose this multi-line.
basept is the base point for the multi-line.
mlinestyleobjhandle indicates the multi-line style with which this entity is drawn.
ldblob is the blob which holds the elements of the multi-line. Follow the example in adexr.c or adexrw.c to see how to read this blob.

MULTI-LINE TEXT (MTEXT):

Defined constant adenhd->enttype==AD_ENT_MTEXT.
Structure definition AD_MTEXT.
Found in AD_ENT union structure mtext (e.g. aden->mtext).
Valid for AutoCAD versions: 13+

typedef struct admtextstru {
    double pt0[3];
    double xdir[3];
    double ht, refrectwid;
    char attachpt;
    char drawdir;
    double boxht, boxwid;
    short linespacingstyle;
    double linespacingfactor;
    AD_OBJHANDLE shapefileobjhandle;
    AD_VMADDR ldblob;
    char textstr[257];
} AD_MTEXT, *PAD_MTEXT;

Multi-line text is a long text entity which can extend onto multiple lines.

pt0 is the insertion point of the text.
xdir is a vector indicating the horizontal direction of the text.
ht is the starting height of the text.
refrectwid is the width of a single line of text; AutoCAD will break the line and start a new line when the text reaches this width.
attachpt is the attachment point of the text, and is equal to one of the following defined constants:

AD_MTEXT_ATTACH_TOPLEFT
AD_MTEXT_ATTACH_TOPCENTER
AD_MTEXT_ATTACH_TOPRIGHT
AD_MTEXT_ATTACH_MIDDLELEFT
AD_MTEXT_ATTACH_MIDDLERIGHT
AD_MTEXT_ATTACH_BOTTOMLEFT
AD_MTEXT_ATTACH_BOTTOMCENTER
AD_MTEXT_ATTACH_BOTTOMRIGHT

drawdir indicates the direction in which the text is drawn, and is equal to one of the following defined constants.

AD_MTEXT_DRAWDIR_LTOR (left to right)
AD_MTEXT_DRAWDIR_RTOL (right to left)
AD_MTEXT_DRAWDIR_TTOB (top to bottom)
AD_MTEXT_DRAWDIR_BTOT (bottom to top)

boxht is the current height of the box enclosing the mtext. See the next entry.
boxwid is the current width of the box enclosing the mtext. It seems that the width and height can safely be set to 0.0.

linespacingstyle is the current linespacing style, either 1 (at least), or 2 (exact).
linespacingfactor is the percentage of default line spacing to be applied, ranging from 0.25 to 4.00.
**shapefileobjhandle** indicates the starting shape file (style) for this text item.

**ldblob** and **textstr** hold the actual text. OpenDWG Toolkit uses the following strategy. If the length of the mtext is 256 bytes or less, it is simply stored in the **textstr** field. If the string is longer, it is stored in **ldblob**. When reading the mtext, you can simply test to see if the **ldblob** field is AD_VMNULL or not to determine the source of the text. When creating mtext, you should be sure to follow the strategy we have described; store the text in **textstr** if it is 256 bytes in length or less, otherwise store it in **ldblob**.

To read data from **ldblob**, use **adReadMtextBlock**.

**OLE2FRAME:**

Function to retrieve enttype adenhd->enttype==adOle2frameEnttype();
Structure definition AD_OLE2FRAME.
Found in AD_ENT union structure ole2frame (e.g. aden->ole2frame).
Valid for AutoCAD versions: 13+

```c
typedef struct adole2framestru {
    short flag;
    short mode;
    char  rawchar;
    long  databytes;
    double pt0[3],pt1[3];
    AD_VMADDR datablob;
} AD_OLE2FRAME,*PAD_OLE2FRAME;
```

An ole2frame is an embedded OLE 2 frame from some application outside AutoCAD.

**flag** indicates the ole2frame version.
**mode** indicates the tile mode of the ole2frame.
**rawchar** is an unknown value that is normally 1.
**databytes** indicates the number of bytes in the datablob.
**pt0** and **pt1** are the corners of the ole2frame in WCS.
**datablob** is the blob containing the ole2frame information.

AutoCAD emits a certain amount of data to DXF which is actually stored inside the binary OLE data; we do not parse out this data.

**POINT:**

Defined constant adenhd->enttype==AD_ENT_POINT.
Structure definition AD_POINT.
Found in AD_ENT union structure point (e.g. aden->point).
Valid for AutoCAD versions: 2.5+

```c
typedef struct adpointstru {
    double pt0[3];
    double ucsxangle;
} AD_POINT,*PAD_POINT;
```

**pt0** is the point at which this point resides.
**ucsxangle** is the rotation angle around the z axis which is to be applied to the point. This rotation would only be visible if PDMODE and PDSIZE in the header were appropriately set to cause points to be drawn as, for instance, a box with an x through it.
POLYLINE:

Defined constant adenhd->enttype==AD_ENT_POLYLINE.
Structure definition AD_PLINE.
Found in AD_ENT union structure pline (e.g. aden->pline).
Valid for AutoCAD versions: 2.5+

typedef struct adpolylinestru {
   double z1;
   double startwidth;
   double endwidth;
   short  fittype;
   short  meshmdensity;
   short  meshndensity;
   short  meshm;
   short  meshn;
   unsigned short  polyflag;
} AD_PLINE,*PAD_PLINE;

A polyline is a connected sequence of vertices. A polyline may have width; in fact, each vertex can have both a startwidth and an endwidth. Segments of the polyline (drawn between vertices) can also be curved. Polylines are also used to store meshes, which come of two types, a polyline mesh which is a grid-type mesh, and a polyface mesh, which is an arbitrarily bounded mesh. A polyline should start with an AD_ENT_POLYLINE entity, followed by 2 or more AD_ENT_VERTEX entities, and end with an AD_ENT_SEQEND entity.

z1 is the default elevation for the polyline.
startwidth is the default start width for vertices in this polyline.
endwidth is the default end width for vertices in this polyline.
fittype is the type of curve or smooth surface type, if any. It is not bit-coded, but simply equal to one of the defined constants AD_PLINE_SURFTYPE_NONE (no curve), AD_PLINE_SURFTYPE_QUAD (quadratic B-spline), AD_PLINE_SURFTYPE_CUBIC (cubic B-spline), or AD_PLINE_SURFTYPE_BEZIER (bezier).

meshmdensity is the smooth surface density in the M direction.

meshndensity is the smooth surface density in the N direction.

meshm is the number of vertices in the M direction.

meshn is the number of vertices in the N direction.

polyflag contains parameters regarding this polyline. It is a bit-coded flag which can be tested by ANDing with the following defined constants:

AD_PLINE_CLOSED (polyline is closed, or surface closed in M direction)
AD_PLINE_CURVEFIT (curve fit has been applied)
AD_PLINE_SPLINEFIT (spline fit has been applied)
AD_PLINE_3DPLINE (polyline is a 3D polyline)
AD_PLINE_3DMESH (polyline is a 3D polygon mesh)
AD_PLINE_CLOSEDN (surface is closed in the N direction)
AD_PLINE_POLYFACEMESH (polyline is a polyface mesh)
AD_PLINE_CONTINUELT (linetype pattern to be generated continuously through the vertices)

PROXY ENTITY:

Defined constant adenhd->enttype==PROXYENT.
Structure definition AD_PROXYENT.
Found in AD_ENT union structure proxy (e.g. aden->proxyent).
Valid for AutoCAD versions: 13+

typedef struct adproxyentstru {
    short entclassid;
    short appclassid;
    long entdatabits;
    long numobjids;
    long objectdrawingformat;
    short origdataformat;
    AD_VMADDR entdatablob;
    AD_VMADDR objidblob;
    AD_VMADDR grblob;
} AD_PROXYENT,*PAD_PROXYENT;

Proxy entities are programmer-defined entities residing in the DWG or DXF file. These are custom entities which have behavior defined/controlled by application programs. They are pretty tricky to deal with. You can test whether or not an entity is a proxy by using the AD_IS_A_PROXY(adenhd->enttype) macro.

entclassid is the entity class id for the proxy entity.
appclassid is the id for the application class for this proxy entity.
entdatabits is the amount of valid data, in bits, in the entdatablob.
umobjids is the number of object ids stored in the objidblob.
objectdrawingformat contains the version (low word) and maintenance release version (high word) of the object.
origdataformat is the original data format, either 0 for dwg or 1 for dxf.
entdatablob holds the entity's local data.
objidblob is the set of object ids which are associated with this entity.
grblob is the blob containing the graphic data representation, if any, for this proxy entity.

The objidblob may be read using adReadProxyEntityObjid.
The entdatablob may be read using the various adReadProxyDataXXX functions, but only if the definition is known. There are no signposts in the data to indicate its structure; you must know the exact sequence of data and make the calls correctly to read the data from this blob.

The grblob may be read using adReadGrblobData. This function returns an AD_GR_DATA structure containing the next item of graphic data representation for the proxy entity. The structure definitions are given below:

typedef struct adgrcirclestru {
    double pt0[3],radius,normal[3];
} AD_GR_CIRCLE,*PAD_GR_CIRCLE;

typedef struct adgrcir3ptstru {
    double pt0[3],pt1[3],pt2[3];
} AD_GR_CIR3PT,*PAD_GR_CIR3PT;

typedef struct adgrcircarcestru {
    double pt0[3],radius,normal[3],startvector[3],sweepangle;
    long arctype;
} AD_GR_CIRCARC,*PAD_GR_CIRCARC;

typedef struct adgrcircarc3ptstru {
    double pt0[3],pt1[3],pt2[3];
    long arctype;
} AD_GR_CIRCARC3PT,*PAD_GR_CIRCARC3PT;

typedef struct adgrplinestru {
    long numpts;
} AD_GR_PLINE,*PAD_GR_PLINE;

typedef struct adgrpolygonstru {
    long numpts;
} AD_GR_POLYGON,*PAD_GR_POLYGON;
typedef struct adgrvertexstru {
    double pt0[3];
} AD_GR_VERTEX,*PAD_GR_VERTEX;

typedef struct adgrmeshstru {
    long rows,cols;
    double *vertex;
    unsigned long edgeprimflag,faceprimflag,vertprimflag;
    short *edgecolorid,*edgelayerid,*edgelinetypeid;
    long *edgegsmarker,*edgevisibility;
    short *facecolorid,*facelayerid;
    long *facenormal;
    long *facevisibility;
    double *vertexnormal;
    long vertexorientation;
} AD_GR_MESH,*PAD_GR_MESH;

typedef struct adgrshellstru {
    long numpts;
    double *vertex;
    long numfaceentries;
    long *faceentry;
    unsigned long edgeprimflag,faceprimflag,vertprimflag;
    short *edgecolorid,*edgelayerid,*edgelinetypeid;
    long *edgegsmarker,*edgevisibility;
    short *facecolorid,*facelayerid;
    long *facenormal;
    long *facevisibility;
    double *vertexnormal;
    long vertexorientation;
} AD_GR_SHELL,*PAD_GR_SHELL;

typedef struct adgrtextstru {
    double pt0[3];
    double normal[3],direction[3],height,widthfactor,oblique;
    char textstr[512];
} AD_GR_TEXT,*PAD_GR_TEXT;

typedef struct adgrtext2stru {
    double pt0[3];
    double normal[3],direction[3];
    char textstr[512];
    char raw;
    double height,widthfactor,oblique;
    double xdir[3];
    char fontfile[128];
} AD_GR_TEXT2,*PAD_GR_TEXT2;

typedef struct adgrxlinestru {
    double pt0[3],pt1[3];
} AD_GR_XLINE, *PAD_GR_XLINE;

typedef struct adgrraystru {
    double pt0[3],pt1[3];
} AD_GR_RAY, *PAD_GR_RAY;
typedef struct ad2grdatastru {
    long grtype;
    union grbodyunion {
        AD_GR_CIRCLE circle;
        AD_GR_CIR3PT circle3pt;
        AD_GR_CIRCARC circulararc;
        AD_GR_CIRCARC3PT circulararc3pt;
        AD_GR_PLINE pline;
        AD_GR_POLYGON polygon;
        AD_GR_MESH mesh;
        AD_GR_SHELL shell;
        AD_GR_TEXT text;
        AD_GR_TEXT2 text2;
        AD_GR_XLINE xline;
        AD_GR_RAY ray;
        long colorindex;
        long layerindex;
        long linetypeindex;
        long markerindex;
        long fillon;
    } grbody;
} AD_GR_DATA,*PAD_GR_DATA;

grtype indicates the type of graphic data returned, and is equal to one of the defined constants:

**AD_GREAT_CIRCLE** (circle)

fields in grdata.grbody.circle:

```c
double pt0[3]; center
double radius; radius
double normal[3]; entity normal
```

**AD_GREAT_CIRCLE3PT** (3 point circle)

fields in grdata.grbody.circle3pt:

```c
double pt0[3],pt1[3],pt2[3]; definition points
```

**AD_GREAT_CIRCULARARC** (circular arc)

fields in grdata.grbody.circulararc:

```c
double pt0[3]; center
double radius; radius
double normal[3]; entity normal
double startvector[3]; direction of start angle
double sweepangle; size of angle swept
long arctype; arc type -- see below
```

**AD_GREAT_CIRCULARARC3PT** (circular arc 3 point)

fields in grdata.grbody.circulararc3pt:

```c
double pt0[3],pt1[3],pt2[3]; definition points
long arctype; arc type
```
Arc types: 0==simple arc, 1==arc sector, 2==arc chord.
Arc sector draws lines from the endpoints of the arc to its center.
Arc chord draws a line connecting the arc's endpoints.

**AD_GRENT_POLYLINE** (polyline)

fields in grdata.grbody.pline:

long numpts; number of vertices

**AD_GRENT_POLYGON** (polygon)

fields in grdata.grbody.polygon:

long numpts; number of vertices

**AD_GRENT_MESH** (mesh)

/* see the process in prnstuff.h for how to read meshes */

fields in grdata.grbody.mesh:

long rows, cols; number of rows, columns
double *vertex; vertex points for mesh, stored as:
vertex[0] is first X coord
vertex[1] is first Y coord
vertex[2] is first Z coord
vertex[3] is second X coord
etc.

unsigned long edgeprimflag; flag indicating whether the edges
have primitive data assigned.
unsigned long faceprimflag; flags indicating whether the faces have
primitive data assigned.
unsigned long vertprimflag; flags indicating whether the vertices have
primitive data assigned.
short *edgecolorid; colors assigned to edges.
short *edgelayerid; layers assigned to edges.
short *edgelinetypeid; linetypes assigned to edges.
long *edgegsmarker; gs markers assigned to edges.
long *edgevisibility; visibilities assigned to edges.
short *facecolorid; colors assigned to faces.
short *facelayerid; layers assigned to faces.
long *facelayerid; linetypes assigned to faces.
long *facegsmarker; gs markers assigned to faces.
double *facenormal; normals assigned to faces, stored as:
normal[0] is first X coord
normal[1] is first Y coord
normal[2] is first Z coord
normal[3] is second X coord
etc.

long *facevisibility; visibilities assigned to faces.
double *vertexnormal; normals assigned to vertices.
long vertexorientation; orientation assigned to vertices.

**AD_GRENT_SHELL** (shell)

/* see the process in prnstuff.h for how to read shells */

fields in grdata.grbody.shell:

long numpts; number of points
double *vertex; vertex points for mesh, stored as:
vertex[0] is first X coord
vertex[1] is first Y coord
vertex[2] is first Z coord
vertex[3] is second X coord
e tc.

long numfaceentries; number of face def entries in **faceentry**
long *faceentry; face def entries
unsigned long edgeprimflag; flag indicating whether the edges have primitive data assigned.
unsigned long faceprimflag; flags indicating whether the faces have primitive data assigned.
unsigned long vertprimflag; flags indicating whether the vertices have primitive data assigned.
short *edgecolorid; colors assigned to edges.
short *edgelayerid; layers assigned to edges.
short *edgelinetypeid; linetypes assigned to edges.
long *edgegsmarker; gs markers assigned to edges.
long *edgevisibility; visibilities assigned to edges.
short *facecolorid; colors assigned to faces.
short *facelayerid; layers assigned to faces.
long *facegsmarker; gs markers assigned to faces.
double *facenormal; normals assigned to faces, stored as:
normal[0] is first X coord
normal[1] is first Y coord
normal[2] is first Z coord
normal[3] is second X coord
e tc.

long *facevisibility; visibilities assigned to faces.
double *vertexnormal; normals assigned to vertices.
long vertexorientation; orientation assigned to vertices.
**AD_GREAT_TEXT**

(text type 1)

Fields in grdata.grbody.text:

- `double pt0[3]`: insertion point
- `double direction[3]`: x direction for text
- `double height`: height
- `double widthfactor`: width factor
- `double oblique`: obliquing angle
- `char textstr[512]`: the text string

**AD_GREAT_TEXT2**

(text type 2)

Fields in grdata.grbody.text2:

- `double pt0[3]`: insertion point
- `double direction[3]`: x direction
- `char textstr[512]`: text string
- `char raw`: 1==do not interpret %% codes, 0==do interpret %% codes
- `double height`: text height
- `double widthfactor`: width factor
- `double oblique`: obliquing angle
- `char fontfile[128]`: font file

**AD_GREAT_XLINE**

(construction line)

Fields in grdata.grbody.xline:

- `double pt0[3],pt1[3]`: definition points

**AD_GREAT_RAY**

(ray)

Fields in grdata.grbody.ray:

- `double pt0[3],pt1[3]`: definition points

**AD_GREAT_SUBENT_COLOR**

(subentity color)

Fields in grdata.grbody: `<< -- note`

- `long colorindex`: index of the color
AD_GRENT_SUBENT_LAYER (subentity layer)
fields in grdata.grbody: << -- note
long layerindex;  index of the layer

AD_GRENT_SUBENT_LINETYPE (subentity linetype)
fields in grdata.grbody: << -- note
long linetypeindex;  index of the linetype

AD_GRENT_SUBENT_MARKER (subentity marker)
fields in grdata.grbody: << -- note
long markerindex;  index of the marker

AD_GRENT_SUBENT_FILLON (subentity fill on)
fields in grdata.grbody: << -- note
long fillon;  index of the color

A number of these entities, including AD_GRENT_POLYLINE, AD_GRENT_MESH, and AD_GRENT_SHELL, have further data which must be read using the appropriate OpenDWG Toolkit function calls. See prnstuff.h for a complete example of proxy entity graphic reading.

RAY:

Defined constant adenhd->enttype==AD_ENT_RAY.
Structure definition AD_RAY.
Found in AD_ENT union structure ray (e.g. aden->ray).
Valid for AutoCAD versions: 13+

typedef struct adraystru {
  double pt0[3];
  double unitvec[3];
} AD_RAY,*PAD_RAY;

A ray is an infinitely long line with one fixed endpoint and a direction.

pt0 is the endpoint of the ray.
unitvec is a vector indicating the direction of the ray on X, Y and Z.
REGION: See ACIS DATA

RTEXT:

Function to retrieve enttype adenhd->enttype==adRtextEnttype(handle);
Structure definition AD_RTEXT.
Found in AD_ENT union structure rtext(e.g. aden->rtext).
Valid for AutoCAD versions: 2000+

```c
typedef struct adrtextstru {
    double pt0[3];
    double rotang;
    double height;
    AD_OBJHANDLE textstyleobjhandle;
    short typeflag;
    AD_VMADDR grblob;
    char contents[1024];
} AD_RTEXT, *PAD_RTEXT;
```

pt0 indicates the insertion point of the rtext.
rotang indicates the rotation angle in radians.
height indicates the text height.
textstyleobjhandle indicates the text style.
typeflag indicates the type flags, where 1 == inline mtext, and 2 == external file.
grblob contains 310 data.
contents contains either an external file name or inline mtext, depending on the value of typeflag.

SEQEND:

Defined constant adenhd->enttype==AD_ENT_SEQEND.
Structure definition: none.
Not in AD_ENT union.
Valid for AutoCAD versions: 2.5+

A sequence end entity indicates the end of one of two types of sequences; either an INSERT ATTRIB
ATTRIB ... ATTRIB SEQEND sequence, or a POLYLINE VERTEX VERTEX ... VERTEX SEQEND sequence.
A sequence end entity has no data.

SHAPE:

Defined constant adenhd->enttype==AD_ENT_SHAPE.
Structure definition AD_SHAPE.
Found in AD_ENT union structure shape (e.g. aden->shape).
Valid for AutoCAD versions: 2.5+

```c
typedef struct adshapestru {
    double pt0[3];
    double scale;
    unsigned char whichshape;
    double rotang;
    double widthfactor;
```
double oblique;
AD_OBJHANDLE shapefileobjhandle;
} AD_SHAPE, *PAD_SHAPE;

A shape is an AutoCAD entity which is defined by an integer-based format and whose definition lives in an external file whose extension is .SHX. At one time shapes were used more extensively because their integer format allowed them to be computed more quickly than normal AutoCAD entities or blocks; this has become less important as computers have gotten faster. However, shapes live on. They are also used as the basis for AutoCAD text. See the description of shapefiles for information on how to make a shapefile entry (which is required if shapes are to be used).

pt0 is the insertion point for the shape.
scale is the scaling to be applied in both X and Y directions.
whichshape is the index of the shape in the shape file.
rotang is the rotation angle in radians.
widthfactor is an additional multiplication applied to the x scale only which makes the shapes either thinner or fatter.

oblique is an obliquing angle to be applied to the shape, which causes it to "lean" either to the right or left.
shapefileobjhandle is the AD_OBJHANDLE for the shapefile entry which indicates from which shapefile this shape is to be read/drawn.

**SOLID:**

Defined constant adenhd->enttype==AD_ENT_SOLID.
Structure definition AD_SOLID.
Found in AD_ENT union structure solid (e.g. aden->solid).
Valid for AutoCAD versions: 2.5+

typedef struct adsolidstru {
    double pt0[3];
    double pt1[3];
    double pt2[3];
    double pt3[3];
} AD_SOLID, *PAD_SOLID;

A solid is a filled region whose defining points are pt0, pt1, pt2, and pt3.

**SOLID3D:** See ACIS DATA.

**SPLINE:**

Defined constant adenhd->spline==AD_ENT_SPLINE.
Structure definition AD_SPLINE.
Found in AD_ENT union structure spline (e.g. aden->spline).
Valid for AutoCAD versions: 13+

typedef struct adsplinestru {
    short degree;
    short flag;
    unsigned short numknots,numctlpts,numfitpts;
    double knottol,ctltol,fittol;
    double starttangent[3],endtangent[3];

degree is the degree of the spline.
flag is a bit-coded flag that contains information about the kind of spline, and can be ANDed with the following defined constants:

- AD_SPLINE_CLOSED (spline is closed)
- AD_SPLINE_PERIODIC (spline is periodic)
- AD_SPLINE_RATIONAL (spline is rational; i.e. has weights)
- AD_SPLINE_PLANAR (spline is planar)
- AD_SPLINE_LINEAR (spline is linear)

numknots is the number of knots stored in the ldblob.
numctlpts is the number of control points stored in the ldblob.
umfitpts is the number of fit points stored in the ldblob.
knottol is the knot tolerance.
controltol is the control point tolerance.
fittol is the fit point tolerance.
starttangent is the start tangent for the spline.
endtangent is the end tangent.

ldblob is the blob containing the spline's control points, knots, and weights. Read this blob using adReadSplineCtlpt, adReadSplineFitpt, adReadSplineKnot, and adReadSplineWeight.

It is very important to understand that control points and knots are not always present in AutoCAD splines. In fact, the default AutoCAD spline has only fit points. OpenDWG Toolkit will provide you with whatever data is stored in the file; it is not making a mistake if it tells you there were no control points or knots. If it says this, then none were stored in the file. Note that when AutoCAD makes a DXF file of a DWG file which OpenDWG Toolkit reports as not having control points and knots, this DXF file will have control points and knots. This is because AutoCAD is calculating these points and knots and adding them to the DXF file; it is not because OpenDWG Toolkit is failing to report their presence in the original DWG file. They are not there.

To read the spline blob successfully, you should follow the read order specified in adexr.c or adexrw.c, using the functions mentioned above in the ldblob description.

TEXT:

Defined constant adenhd->enttype==AD_ENT_TEXT.
Structure definition AD_TEXT.
Found in AD_ENT union structure text (e.g. aden->text).
Valid for AutoCAD versions: 2.5+

typedef struct adtextstru {
  double pt0[3];
  AD_TDATA tdata;
  char   textstr[2048];
} AD_TEXT,*PAD_TEXT;

A text entity is a single line of text.

pt0 is the lower left point for the text. Note that this is NOT the point around which the text is centered, middled, etc. It is literally the lower left point for the text. AutoCAD uses this point to draw the text. The center,
The middle etc. point is kept in tdata->secondtextloc. You can use adComputeNewTextParms to calculate a correct new pt0 if necessary.

textstr is the text contained in the string.

tdata is a substructure which contains parameters regarding how the text is to be rendered to the screen.

The AD_TDATA structure looks like this:

```c
typedef struct adtdatastru {
    double height;
    double rotang;
    double widthfactor;
    double oblique;
    char generationflag;
    char justification;
    char vertalignment;
    double secondtextloc[2];
    AD_OBJHANDLE shapefileobjhandle;
} AD_TDATA, *PAD_TDATA;
```

height is the text height.

rotang is the rotation angle of the text in radians.

widthfactor is an additional scaling applied in the x direction which makes the text either fatter or thinner.

oblique is an obliquing angle to be applied to the text, which causes it to “lean” either to the right or left.

generationflag is a bit-coded flag indicating special generation for the text. The default is normal (left to right, right-side up). The two other possibilities are backward and upside-down; these can be tested by ANDing generationflag with the defined constants AD_TEXT_GEN_BACKWARD and AD_TEXT_GEN_UPSIDEDOWN.

justification contains a value indicating the manner in which the text is justified. The valid justifications are left, center, right, aligned, middle, and fit. This is not a bit-coded flag; it simply contains a value equal to one of the defined constants AD_TEXT_JUST_LEFT, AD_TEXT_JUST_CENTER, AD_TEXT_JUST_RIGHT, AD_TEXT_JUST_ALIGNED, AD_TEXT_JUST_MIDDLE, or AD_TEXT_JUST_FIT.

vertalignment indicates the vertical alignment of the text. Text may be aligned vertically from the baseline, bottom, middle, or top, as indicated by the defined constants AD_TEXT_VALIGN_BASELINE, AD_TEXT_VALIGN_BOTTOM, AD_TEXT_VALIGN_MIDDLE and AD_TEXT_VALIGN_TOP.

secondtextloc is the second alignment point for text which requires a justification other than from the lower left corner. This could be the center point, right point, middle point, or the other point (along with pt0) in the pair between which the text is either aligned or fit.

shapefileobjhandle indicates the shapefile (style) entry used to determine the font from which this text is drawn.

**TOLERANCE:**

Defined constant adenh->enttype==AD_ENT_TOLERANCE.

Structure definition AD_TOLERANCE.

Found in AD_ENT union structure tolerance (e.g. adenh->tolerance).

Valid for AutoCAD versions: 13+

```c
typedef struct adtolerancestru {
    double pt0[3];
    double xdir[3];
    AD_OBJHANDLE dimstyleobjhandle;
    double textht;
    char textstr[1024];
} AD_TOLERANCE, *PAD_TOLERANCE;
```

pt0 is the insertion point of the tolerance entity.

xdir is the horizontal direction of the tolerance.
**dimstyleobjhandle** indicates the dimension style which contains parameters controlling this tolerance entity.

**textth** is the height of the text in the tolerance.

**textstr** contains the tolerance's text.

**TRACE:**

Defined constant adenhd->enttype==AD_ENT_TRACE.
Structure definition AD_TRACE.
Found in AD_ENT union structure trace (e.g. aden->trace).
Valid for AutoCAD versions: 2.5+

```c
typedef struct adtracestru {
    double pt0[3];
    double pt1[3];
    double pt2[3];
    double pt3[3];
} AD_TRACE, *PAD_TRACE;
```

A trace is a solid filled path, composed of four-sided figures whose defining points are \(pt0, pt1, pt2,\) and \(pt3.\)

**VERTEX:**

Defined constant adenhd->enttype==AD_ENT_VERTEX.
Structure definition AD_VERTEX.
Found in AD_ENT union structure vertex (e.g. aden->vertex).
Valid for AutoCAD versions: 2.5+

```c
typedef struct advertexstru {
    double pt0[3];
    unsigned short vertexflag;
    double startwidth;
    double endwidth;
    short  startwidthvalid;
    short  endwidthvalid;
    double bulge;
    double tangentdir;
    short  polyfacevertcount;
    short  polyfacevert[4];
} AD_VERTEX, *PAD_VERTEX;
```

A vertex is a definition point for a polyline.

\(pt0\) is the location of the vertex.

**startwidth** is the starting width for the segment extending from this vertex to the next one.

**endwidth** is the ending width for the segment extending from this vertex to the next one.

**startwidthvalid** is 1 if the start width is valid, 0 if not.

**endwidthvalid** is 1 if the end width is valid, 0 if not.

**bulge**, if nonzero, is the tangent of one fourth of the included angle for an arc segment. It is negative if the arc direction is clockwise, positive if the direction is counterclockwise. A value of 0 for the bulge means the segment is straight; a value of 1 means the segment is a semicircle.
**vertexflag** is a bit-coded flag containing parameters for this vertex. It can be tested by ANDing with the following defined constants:

- **AD_VERTEX_FROMFIT** (vertex added by curve fitting)
- **AD_VERTEX_HASTANGENT** (vertex has a curve fit tangent)
- **AD_VERTEX_SPLINEVERTEX** (vertex is part of a spline-fit curve)
- **AD_VERTEX_SPLINEFRAME** (vertex is part of a spline frame)
- **AD_VERTEX_3DPLINE** (a 3D polyline vertex)
- **AD_VERTEX_3DMESHVERT** (a 3D mesh vertex)
- **AD_VERTEX_FACELIST** (a list of vertices for a polyface)

**tangentdir** is the starting tangent direction for a polyline.

**polyfacevertcount** is the number of valid entries in the **polyfacevert** array.

**polyfacevert** is an array of entries indicating the indices of the vertices making up a single face of a polyface mesh.

A polyface mesh is a special kind of mesh which is also stored in the POLYLINE/VERTEX format. Vertex points for the polyface mesh are indicated by the 64 bit being set in the flag field of a VERTEX entity. This can be tested by taking (aden->vertex.flag & **AD_VERTEX_3DMESHVERT**); nonzero means it is a mesh vertex. A VERTEX entity which contains a face definition can be detected by taking (aden->vertex.flag & **AD_VERTEX_FACELIST**); nonzero means it is a vertex list. The vertex list consists of indices of the vertices which make up the polyface mesh face; if a vertex index is negative, the edge of the face going from that vertex to the next is invisible. The number of valid entries in the vertex list is given by **polyfacevertcount**.

**VIEWPORT ENTITY:**

Defined constant adenhd->enttype==**AD_ENT_VIEWPORT**.
Structure definition **AD_VPENT**.
Found in **AD_ENT** union structure vpent (e.g. aden->vpent).
Valid for AutoCAD versions: 11+

```c
typedef struct advpentstru {
    AD_VMADDR frozenlayerblob;
    short  numfrozenlayers;
    short  flag;
    short  stackval;
    double cen[3];
    double width;
    double height;
    short  id;
    char  purgedflag;
    short  xrefindex;
    AD_OBJHANDLE objhandle;
    AD_OBJHANDLE clipboundobjhandle;
    unsigned long statusflags;
    short  rendermode;
    char  stylesheet[AD_MAX_STRLEN];
    short  ucsvp;
    short  ucsiconatucsorigin;
    double ucsorg[3];
    double ucsxdir[3];
    double ucsydir[3];
    short  ucsorthoviewtype;
    double ucslelevation;
    AD_OBJHANDLE ucsobjhandle;
    AD_OBJHANDLE baseucsobjhandle;
    double viewtarget[3];
    double viewdir[3];
    double viewtwist;
```
double viewheight;
double viewcenter[2];
double viewlenslength;
double viewfrontclip;
double viewbackclip;
short viewcirclezoom;
double viewsnapdist[2];
double viewgriddist[2];
double viewsnapangle;
double viewsnapbase[2];
char name[AD_MAX_STRLEN];
} AD_VPENT, *PAD_VPENT;
A viewport entity is an entity which defines a region in paperspace through which modelspace entities are seen.

- **frozenlayerblob** contains the handles of the layers that are frozen in this viewport.
- **numfrozenlayers** indicates the number of handles in **frozenlayerblob**.
- **flag** has no useful values.
- **stackval** is the viewport status field. A value of -1 indicates that the viewport is on, but not currently being displayed, either because the maximum has been reached, or the viewport is offscreen. 0 indicates that the viewport has been turned off, and a positive value indicates that the viewport is on, and gives the position of this viewport entity in the viewport stack.
- **cen** is the center of the viewport in AutoCAD space.
- **width** is the width of the viewport in AutoCAD units.
- **height** is the height of the viewport in AutoCAD units.
- **id** is the viewport id.
- **purgedflag** is 1 if the viewport has been purged, 0 otherwise.
- **xrefindex** is -1 if this viewport does not come from an xref, otherwise it indicates the xref from which it came. Although this is clearly what the value means, we do not currently believe that it is possible for a viewport entity to come in from an xref.
- **objhandle** is used internally by OpenDWG Toolkit and should be ignored.
- **clipboundobjhandle** is the handle of the object that defines the clip boundary for the viewport.
- **statusflags** is a bit field that indicates various properties of the viewport entity. It can be anded with the following constants:
  - `AD_VPENT_STATUS_PERSPECTIVEMODE` (perspective mode enabled)
  - `AD_VPENT_STATUS_FRONTCLIP` (front clipping enabled)
  - `AD_VPENT_STATUS_BACKCLIP` (back clipping enabled)
  - `AD_VPENT_STATUS_UCSFOLLOW` (ucs follow enabled)
  - `AD_VPENT_STATUS_NOTATEYE` (enables front clipping not at eye)
  - `AD_VPENT_STATUS_UCSICON` (enables ucs icon visibility)
  - `AD_VPENT_STATUS_UCSATORIGIN` (enables ucs at origin)
  - `AD_VPENT_STATUS_FASTZOOM` (enables fast zoom)
  - `AD_VPENT_STATUS_SNAPON` (enables snap mode)
  - `AD_VPENT_STATUS_GRIDON` (enables grid mode)
  - `AD_VPENT_STATUS_SNAPSTYLEISO` (enables isometric snap style)
  - `AD_VPENT_STATUS_HIDDENINPLOT` (enables hide plot mode)
  - `AD_VPENT_STATUS_SNAPISOPAIRTOP` (enables isopair top)
  - `AD_VPENT_STATUS_SNAPISOPAIRRIGHT` (enables isopair right, but if both isopair top and isopair right are enabled, this indicates isopair left)
  - `AD_VPENT_STATUS_ZOOMLOCKING` (enables zoom locking)
  - `AD_VPENT_STATUS_ALWAYS` (always enabled)
  - `AD_VPENT_STATUS_NONRECTCLIP` (enables non-rectangular clipping)
  - `AD_VPENT_STATUS_OFF` (turns viewport off)
- **rendermode** indicates the render mode for the viewport.
- **stylesheet** is the plot style sheet name assigned to this viewport.
- **ucsvp** is the ucs per viewport flag.
- **ucsciconatucsoorigin** is the display ucs icon at origin flag.
- **ucso** is the ucs origin.
- **ucsxdir** is the ucs x axis.
- **ucsydir** is the ucs y axis.
- **ucsorthoviewtype** is the ucs orthographic view type.
- **ucselevation** is the ucs elevation.
- **ucsoobjhandle** is the ucs object handle.
- **baseucsoobjhandle** is the base ucs object handle.
- **viewtarget** is the view target point.
- **viewdir** is the view direction vector.
- **viewtwist** is the view twist angle.
viewheight is the view height.
viewcenter is the view center point.
viewlenslength is the view lens length.
viewfrontclip is the view front clip plane z value.
viewbackclip is the view back clip plane z value.
viewcirclezoom is the circle zoom percent.
viewsnapdist is the snap spacing.
viewgriddist is the grid spacing.
viewsnapangle is the snap angle.
viewsnapbase is snap base point.
name is the name of the view.

In AutoCAD versions R14 and earlier, information about the viewport itself was stored (and only Autodesk knows why) in extended data attached to the viewport entity. This data has the following format:

1001 group: application handle (shd be for application "ACAD")
1000 group: start of viewport data – string "MVIEW"
1002 group: start of window data – value is '{'
1070 group: version number of xdata, value 16
1010 group: target point for view
1010 group: direction vector
1040 group: twist angle
1040 group: height of view
1040 group: center point x
1040 group: center point y
1040 group: lens length for perspective
1040 group: front clip z
1040 group: back clip z
1070 group: view mode
1070 group: circle zoom percent
1070 group: fastzoom
1070 group: ucsicon
1070 group: snap mode (on/off)
1070 group: grid mode (on/off)
1070 group: snap style
1070 group: snap iso pair
1040 group: snap angle
1040 group: snap base point x
1040 group: snap base point y
1040 group: snap spacing x
1040 group: snap spacing y
1040 group: grid spacing x
1040 group: grid spacing y
1070 group: flag indicating "hidden in plot"
1002 group: '{' – beginning of frozen layer list. Can be an empty list.
1003 group: layer frozen in this vport. May be any number of these entries.
1002 group: '}' – end of frozen layer list
1002 group: '}' – end of viewport data

The Toolkit removes this extended data and places the contents directly into the AD_VPENT structure when loading R14 and earlier files, so that the data in AD_VPENT structures is in the same format for all versions. The Toolkit also creates this extended data when saving back to a version earlier than AutoCAD 2000. When creating viewport entities, the user should place all data directly into the AD_VPENT structure, and not add the MVIEW extended data section.
WIPEOUT ENTITY:

Function to retrieve enttype adenhd->enttype==adWipeoutEnttype(handle);
Structure definition AD_WIPEOUT.
Found in AD_ENT union structure wipeout (e.g. aden->wipeout).
Valid for AutoCAD versions: 2000+

typedef struct adwipeoutstru {
    long classversion;
    double pt0[3];
    double size[2];
    double uvec[3],vvec[3];
    unsigned short clipping,brightness,contrast,fade,clipboundtype;
    unsigned short displayprops;
    long numclipverts;
    double rectclipvert0[2],rectclipvert1[2];
    AD_VMADDR polyclipvertblob;
    AD_VMADDR grblob;
    AD_OBJHANDLE imagedefobjhandle,imagedefreactorobjhandle;
} AD_WIPEOUT, *PAD_WIPEOUT;

A wipeout entity is similar to an image entity, except that it takes on the color of the background.

See AD_IMAGE for a description of the data members.

EXTENDED OBJECT DATA:

Extended data is a mechanism provided by AutoCAD to add up to 16K of data to objects. Extended data is returned by OpenDWG Toolkit in the AD_XD structure. The xddxfnumber field indicates the type of extended data stored therein, and is equal to one of these defined constants:

AD_XD_STRING
AD_XD_APPOBJHANDLE
AD_XD_CONTROL
AD_XD_LAYEROBJHANDLE
AD_XD_BINDATA
AD_XD_OBJHANDLE
AD_XD_REALS
AD_XD_WORLDPOS
AD_XD_WORLDDISP
AD_XD_WORLDDIR
AD_XD_REAL
AD_XD_DIST
AD_XD_SCALE
AD_XD_INT
AD_XD_LONG

Generally, extended data is stored starting with an AD_XD_APPOBJHANDLE group indicating the application which owns this data, then a control item with '{' as the control character, then whatever extended data is stored by this application, then a control item with '}' as the control character. There can be multiple applications, each with its own extended data block, within a set of extended data. In addition, AutoCAD does NOT seem to enforce the '{' and '}' bracketing described in its documentation.

typedef union xdunion {
    double xd3dpt[3]; /* 3d point */
    AD_OBJHANDLE xdappobjhandle; /* objhandle of regapp */
}
char xdbindata[128]; /* binary data */
char xdcontrol; /* { or } */
AD_OBJHANDLE xdhandle; /* database handle */
double xddist; /* distance */
short xdint; /* integer */
AD_OBJHANDLE xdentlayerobjhandle; /* layer object handle */
long xdlong; /* long integer */
double xdreal; /* real number */
double xdscale; /* scale factor */
char xdstring[2048]; /* string */
double xdworlddir[3]; /* world direction xyz */
double xdworlddisp[3]; /* world displacement xyz */
double xdworldpos[3]; /* world position xyz */
} AD_XD_DATA, *PAD_XD_DATA;

typedef struct adxdstru {
 short xddxfnumber; /* the DXF group code */
 short xdbindatalength; /* size of binary data */
 AD_XD_DATA xddata;
} AD_XD,*PAD_XD;
#endif

The field contents are self-explanatory, except for the xdcontrol field, whose valid values are '{' and '}', and xdbindatalength, which indicates the amount of valid binary data stored in the xdbindata field for an extended data item of type AD_XD_BINDATA.
TABLE ENTRIES

This section discusses the various structures defined for table entries in an AutoCAD file. Most table entries have a bit-coded flag field whose values are the same from table entry to table entry, and can be tested by ANDing with the following defined constants:

- AD_ENTRY_XREF_DEPENDENT (entry is dependent on an xref)
- AD_XREF_RESOLVED (entry is resolved from an xref)
- AD_REFERENCED (entry is currently referenced)

Note that the AD_XREF_RESOLVED and AD_REFERENCED flags may not be reliable, so we do not suggest using them. They can be safely set to 0 when writing.

BLOCK HEADER:

Valid AutoCAD versions for this table entry: 2.5+

typedef struct adblkhstru {
    AD_OBJHANDLE ownerobjhandle; (2000+)
    AD_OBJHANDLE layoutobjhandle; (2000+)
    AD_VMADDR previewblob; (2000+)
    char flag;
    char name[AD_MAX_STRLEN];
    char purgedflag;
    short xrefindex; (11+)
    AD_OBJHANDLE objhandle;
    char xrefblkisoverlaid; (13+)
    AD_VMADDR xdblob; (13+)
    AD_VMADDR entitylist;
    char ad3vectorize; (AD internal)
    AD_OBJHANDLE xdicobjhandle; (14+)
    long numreactors;
    AD_VMADDR reactorblob;
    char description[512];
} AD_BLKH,*PAD_BLKH;

ownerobjhandle is the handle of the owner of this block.
layoutobjhandle is the handle of the layout object associated with this block.
previewblob contains the preview data for the block (preview data is optional).
flag is the block flag. In addition to the standard flag values above, this flag can be ANDed with the following constants for testing:

- AD_BLOCK_ANONYMOUS (an anonymous block which was generated by dimensioning, hatching, etc.: name starts w/ '*')
- AD_BLOCK_XREF (block is an external reference)

name is the name of the block.
purgedflag is 1 if the block has been purged, otherwise 0.
xrefindex is the index of the xref in the block header table from which this block header comes, if it is dependent on an xref. -1 indicates it is not from an xref.
objhandle is the object handle for this blockheader.
xrefblkisoverlaid is 1 if this is an "overlaid" xref, or 0 if it is an "attached" xref.
xdblob contains any extended data associated with this blockheader.
entitylist is the list of entities that compose this block.
ad3vectorize is a flag used by AD 3 to turn vectorization of this block on or off.
xdicobhandle is an the AD_OBJHANDLE of an extended dictionary, if any.
numreactors is the number of persistent reactors attached to this blockheader.
reactorblob is the blob containing the reactors, if any.
description contains an optional block description.

CLASS:

Valid AutoCAD versions for this table entry: 13+

With Release 13 Autodesk introduced the concept of classes defined within the drawing file. The structure is given below:

typedef struct adclassstru {
 unsigned short classnum;
 char classdxfname[AD_MAX_STRLEN];
 char cplusplusclassname[AD_MAX_STRLEN];
 char appname[AD_MAX_STRLEN];
 short version;
 short wasaproxy,itemclassid;
} AD_CLASS, *PAD_CLASS;

classnum is the number for this class.
classdxfname is the name of the class.
cplusplusclassname is the name within C++.
appname is the name of the application to which this class belongs.
version is the version of the class.
wasaproxy is a field which seems always to be 0.
itemclassid is the number which is placed in instances of this class for identification.

DIMENSION STYLE:

Valid AutoCAD versions for this table entry: 11+

Dimension styles are collections of dimension control variables which are used to control the appearance of dimensions. For descriptions of the meanings of the individual dimxxx fields, please see the header section where the header dim variables are defined.

typedef struct addimstylestru {
 AD_OBJHANDLE ownerobjhandle;
 short xrefindex;
 char flag;
 char purgedflag;
 char name[AD_MAX_STRLEN];
 short dimadec;
 char dimalt;
 char dimaldt;
 double dimaltf;
 double dimaltrnd;
 char dimapost[AD_MAX_STRLEN];
 double dimasz;
 double dimazin;
 AD_OBJHANDLE dimblkobjhandle;
 AD_OBJHANDLE dimblk1objhandle;
 AD_OBJHANDLE dimblk2objhandle;
 double dimcen;
 short dimclrd;
 short dimclre;
short dimclrt;
double dimdle;
double dimdli;
double dimexe;
double dimexo;
double dimgap;
double dimfac;
char dimlim;
char dimpost[AD_MAX_STRLEN];
double dimrnd;
char dimsa;
double dimscale;
char dimsel;
char dimse2;
double dimtfac;
char dimtih;
double dimtm;
char dimtoh;
char dimtof1;
char dimtol;
double dimtp;
double dimtsz;
double dimtxt;
double dimtpv;
char dimsox;
char dimtad;
char dimtx;
char dimz;
short dimalttd; (13+)
char dimalttz; (13+)
short dimalu; (13+)
char dimaltz; (13+)
short dimaunit; (13+)
short dimdec; (13+)
char dimfit; (13+)
char dimjust; (13+)
char dimsd1; (13+)
char dimsd2; (13+)
short dimtdec; (13+)
char dimtolj; (13+)
char dimtjin; (13+)
short dimunit; (13+)
char dimupt; (13+)
short dimfrac;
short dimdsep;
short dimtmove;
AD_OBJHANDLE dimldrblock;
short dimlwd;
short dimlwe;
AD_OBJHANDLE objhandle, shapefileobjhandle;
AD_VMADDR xdblob; (13+)
long numreactors;
AD_VMADDR reactorblob;
} AD_DIMSTYLE,*PAD_DIMSTYLE;

ownerobjhandle is the handle of the owner object for this dimstyle.
xrefindex is the index of the external reference from which this dimstyle comes, if any. -1 indicates that
dimstyle is not from an external reference.
flag has the standard flag values.
purgedflag is 1 if the dimstyle has been purged, otherwise 0.
name is the name of this dimension style.
objhandle is the AD_OBJHANDLE for this dimension style.
shapefileobjhandle indicates the shape file (style) used for text drawn with this dimension style.
xdblob contains the extended data for this dimstyle, if any.
umreactors is the number of persistent reactors attached to this dimstyle.
reactorblob is the blob containing the reactors, if any.
**LAYER:**

Valid AutoCAD versions for this table entry: 2.5+

```c
typedef struct adlaystru {
    AD_OBJHANDLE ownerobjhandle;
    char   purgedflag;
    short  xrefindex;  // (11+)
    short  color;
    char   flag;
    char name[AD_MAX_STRLEN];
    AD_OBJHANDLE linetypeobjhandle;
    AD_OBJHANDLE objhandle;
    AD_OBJHANDLE xdicobjhandle;
    AD_VMADDR xdblob;  // (13+)
    long numreactors;
    AD_VMADDR reactorblob;
    short plotflag;
    short lineweight;
    AD_OBJHANDLE plotstyleobjhandle;
} AD_LAY,*PAD_LAY;
```

- **ownerobjhandle** is the handle of the owner object for this layer.
- **purgedflag** is 1 if this layer has been purged, otherwise 0.
- **xrefindex** indicates the index of the xref in the blockheader list from which this layer comes. If the layer is from this file the index is -1.
- **color** is the color for entities on this layer, negative if the layer is turned off.
- **flag** has the standard values, plus several bit codes which can be tested by ANDing with the following defined constants:
  - `AD_LAYER_FROZEN` (layer is frozen)
  - `AD_LAYER_VPFROZEN` (layer is frozen in new viewports)
  - `AD_LAYER_LOCKED` (layer is locked)
- **name** is the name of the layer.
- **linetypeobjhandle** is the object handle for this layer’s linetype.
- **objhandle** is the object handle for this layer.
- **xdicobjhandle** is the handle of the ACAD_XDICTIONARY object associated with this layer.
- **xdblob** is the extended data, if any, attached to this layer. It is equal to AD_VMNULL if there is no extended data.
- **numreactors** is the number of persistent reactors attached to this layer.
- **reactorblob** is the blob containing the reactors, if any.
- **plotflag** is the plotting flag for this layer (0 means do not plot).
- **lineweight** is the lineweight for this layer.
- **plotstyleobjhandle** is handle for the PlotStyleName object associated with this layer.

**LINETYPE:**

Valid AutoCAD versions for this table entry: 2.5+

```c
typedef struct adltpstru {
    AD_OBJHANDLE ownerobjhandle;
    char   purgedflag;
```
short xrefindex;           (11+)
char name[AD_MAX_STRLEN];
char text[512];
char alignflag;
char flag;
short numlinetypesegs;
double patternlength;
struct adltpsegstru segdata[15];   (varies)
AD_OBJHANDLE objhandle;
AD_OBJHANDLE xdicobjhandle;
char ltypetextstr[256];
AD_VMADDR xdblob;               (13+)
long numreactors;
AD_VMADDR reactorblob;
} AD_LTYPE,*PAD_LTYPE;

ownerobjhandle is the owner handle for the linetype.
purgedflag is 1 if the linetype has been purged, otherwise 0.
xrefindex is the index of the xref in the block header list from which this linetype comes, -1 if it is defined in this drawing.
name is the name of the linetype.
text is a text picture of the linetype.
alignflag is always 'A'.
flag has the standard flag values.
numlinetypesegs is the number of valid segments in the segdata array.
patternlength is the overall length of the linetype pattern.
segdata is an array containing the data for each segment of the linetype. The array is an array of type
struct adltpsegstru, shown below:

struct adltpsegstru {
    double val;                  (2.5+)
    short complexshapecode;      (13+)
    double scale;                (13+)
    double rotang;               (13+)
    double xoffset;              (13+)
    double yoffset;              (13+)
    AD_OBJHANDLE shapeentryobjhandle;   (13+)
    short shapeflag;             (13+)
    unsigned short stroffset;    (13+)
} ;

val is the length of this linetype segment, >0.0 means draw a segment, <0.0 means move but do not draw. 0.0 draws a dot.
complexshapecode is the shape to draw for this segment, if any, otherwise 0.
scale is the overall scale of the linetype
rotang is the rotation angle in radians for a text element in the linetype
xoffset is the x offset of a text element.
yoffset is the y offset of a text element.
shapeentryobjhandle indicates the shape file (style) to be used to draw a text element for this linetype.
shapeflag is a bit-coded flag. If the 2 bit is set, stroffset is valid (see below). If 1 is set, there is a text component to this segment of the linetype.
stroffset is the offset into the ltypetextstr array of the text string to be drawn for this segment, valid if shapeflag & 2.

objhandle is the AD_OBJHANDLE for this linetype.
xdicobjhandle is the AD_OBJHANDLE for the ACAD_XDICTIONARY object associated with this linetype.
ltypetextstr contains strings to be drawn into segments of this linetype as indicated by shapeflag and stroffset.
xdblob contains the extended data, if any, attached to this linetype.
umreactors is the number of persistent reactors attached to this linetype.
**reactorblob** is the blob containing the reactors, if any.

**REGISTERED APPLICATION:**

Valid AutoCAD versions for this table entry: 11+

```c
typedef struct adappstru {
    AD_OBJHANDLE ownerobjhandle;
    short xrefindex;
    char purgedflag;
    char flag;
    char name[AD_MAX_STRLEN];
    AD_OBJHANDLE objhandle;
    AD_OBJHANDLE xdicobjhandle;
    AD_VMADDR reactorblob;  // (13+)
    long numreactors;
    AD_VMADDR reactorblob;
} AD_APP,*PAD_APP;
```

**ownerobjhandle** is the owner object handle for this regapp.

**xrefindex** is the index of the xref in the block header table from which this registered application came. It is -1 if the application was defined in this file.

**purgedflag** is 1 if this entry was purged, otherwise 0.

**flag** has the standard flag values.

**name** is the name of the application.

**objhandle** is the AD_OBJHANDLE for this object.

**xdicobjhandle** is the AD_OBJHANDLE for the ACAD_XDICTIONARY object associated with this regapp.

**xdblob** contains the extended entity data, if any, attached to this object.

**numreactors** is the number of persistent reactors attached to this regapp.

**reactorblob** is the blob containing the reactors, if any.

**SHAPEFILE (STYLE):**

Valid AutoCAD versions for this table entry: 2.5+

AutoCAD uses shapefile entries for both text styles and shape files to be loaded for use in the file. Probably the predominant use is for text styles.

```c
typedef struct adshptbstru {
    AD_OBJHANDLE ownerobjhandle;
    char bigfontname[256];
    char purgedflag;
    short xrefindex;
    char flag2;
    char flag;
    char name[AD_MAX_STRLEN];
    char file[256];
    double angle;
    double height;
    double width;
    double lastshapeheight;
    AD_OBJHANDLE objhandle;
    AD_OBJHANDLE xdicobjhandle;
    AD_VMADDR xdblob;  // (13+)
    long numreactors;
    AD_VMADDR reactorblob;
```
AD_SHPTB, *PAD_SHPTB;

ownerobjhandle is the owner object handle for this shapefile.
bigfontname is the name of the bigfont file, if any, used for this style.
purgedflag is 1 if this shapefile is purged, otherwise 0.
xrefindex is the index of the xref in the block header table from which this shapefile came. It is -1 if the application was defined in this file.

flag2 contains the text generation flags. This is a bit-coded flag which can be tested by ANDing with the following defined constants:

AD_SHAPEFILE_GEN_BACKWARD
AD_SHAPEFILE_GEN_UPSIDEDOWN

flag contains the standard flag values; in addition certain bits can be tested by ANDing with the following defined constants:

AD_SHAPEFILE_SHAPES (used for shapes, NOT a style definition)
AD_SHAPEFILE_VERTICAL (draw vertically)

AD_SHAPEFILE_SHAPES is set if this shapefile entry is used to draw AutoCAD shapes; it is not set if this is a text style from which text entities are being drawn.

name is the name of the shapefile.
file is the font or shape file used for this entry.
angle is the oblique angle for this shapefile.
height is the default height for this shapefile.
width is the default widthfactor for this shapefile.
lastshapeheight is the last height used.
objhandle is the AD_OBJCHANDLE for this shapefile.
xdicobjhandle is the AD_OBJCHANDLE for the ACAD_XDICTIONARY object associated with this shapefile.
xdblob contains the extended data, if any, attached to this shapefile.
numreactors is the number of persistent reactors attached to this shapefile.
reactorblob is the blob containing the reactors, if any.

USER COORDINATE SYSTEM:

Valid AutoCAD versions for this table entry: 10+

A user coordinate system is defined by its origin, X direction, and Y direction.

typedef struct aducsstru {
    AD_OBJCHANDLE ownerobjhandle;
    char   purgedflag;
    short  xrefindex;       (11+)
    char   flag;
    char   name[AD_MAX_STRLEN];
    double org[3];
    double xdir[3];
    double ydir[3];
    AD_OBJCHANDLE objhandle;
    AD_OBJCHANDLE xdicobjhandle;
    AD_VMADDR xdblob;       (13+)
    long numreactors;
    AD_VMADDR reactorblob;
    short orthoviewtype;
    short orthographictype;
    double elevation;
} AD_UCS;
ownerobjhandle is the owner object handle for this UCS.

purgedflag is 1 if this UCS has been purged, otherwise 0.

xrefindex is the index of the block header for the xref from which this UCS came, -1 if the UCS is defined in this file.

flag has the standard flag values.

name is the name of the UCS.

org is the origin of the UCS.

xdir is the X direction.

ydir is the Y direction.

objhandle is the AD_OBJHANDLE for this UCS.

xdicobjhandle is the AD_OBJHANDLE for the ACAD_XDICTIONARY object associated with this UCS.

xdblob contains the extended data, if any, attached to this shapefile.

numreactors is the number of persistent reactors attached to this UCS.

reactorblob is the blob containing the reactors, if any.

orthoviewtype is the orthographic type of UCS. 0 == UCS is not orthographic, 1 == Top, 2 == Bottom, 3 == Front, 4 == Back, 5 == Left, 6 == Right.

orthographictype is the orthographic type of the UCS, 1 == Top, 2 == Bottom, 3 == Front, 4 == Back, 5 == Left, 6 == Right.

elevation is the elevation of the UCS.

orthoviewrefobjhandle is the handle of the base UCS if this is an orthographic.

lasthandle contains an unknown handle value.

orthovieworigin contains the origin for this orthographic type relative to this UCS.
typedef struct adviewstru {
    AD_OBJHANDLE ownerobjhandle;
    char purgedflag;
    short xrefindex;            (11+)
    double backclip;
    double center[2];
    double dir[3];
    char  flag;
    char flag2;
    double frontclip;
    double ht;
    double lenslength;
    char  name[AD_MAX_STRLEN];
    double target[3];          (13+)
    double twist;
    short viewmode;
    double width;
    short rendermode;
    short isassociateducs;
    double ucsorg[3];
    double ucsxdir[3];
    double ucsydir[3];
    short ucsorthoviewtype;
    double ucselevation;
    AD_OBJHANDLE ucsobjhandle;
    AD_OBJHANDLE baseucsobjhandle;
    AD_OBJHANDLE objhandle;
    AD_OBJHANDLE xdicobjhandle;
    AD_VMADDR xdblob;          (13+)
    long numreactors;
    AD_VMADDR reactorblob;
} AD_VIEW, *PAD_VIEW;

ownerobjhandle is the owner object handle for this view.
purgedflag is 1 if this view has been purged, otherwise 0.
xrefindex is the index of the block header for the xref from which this view came, otherwise -1 if it was defined in this file.
backclip is the offset of the back clipping plane from the target plane.
center is the center of this view.
dir is the view direction for this view.
flag has the standard flag values; in addition bit 1 is set if this is a paperspace view. This can be tested by ANDing with AD_VIEW_ISPAPERSPACE.
flag2 appears only in older files and can be set to 0.
frontclip is the offset of the front clipping plane from the target plane.
ht is the view height.
lenslength is the view lens length.
name is the name of the view.
target is the target point.
twist is the twist angle.
viewmode is the view mode (see the VIEWMODE header variable).
width is the view width.
rendermode is the render mode for the view, 0 == 2D optimized, 1 == wireframe, 2 == hidden line, 3 == flat shaded, 4 == gouraud shaded, 5 == flat shaded with wireframe, 6 == Gouraud shaded with wireframe.
isassociateducs is 1 if there is a UCS associated with this view, 0 otherwise.
ucscorg is the UCS origin.
ucsxdir is the UCS x axis.
ucsydir is the UCS y axis.
ucsorthoviewtype is the UCS orthographic type (see AD_UCS values).
ucsellevation is the UCS elevation.
ucsobjhandle is the handle of the named UCS.
baseucsobjhandle is the handle of the base UCS if the UCS is orthographic.
objhandle is the AD_OBJHANDLE for this view.
xdicobjhandle is the AD_OBJHANDLE for the ACAD_XDICTIONARY object associated with this view.
xdblob contains the extended data, if any, attached to this view.
numreactors is the number of persistent reactors attached to this view.
reactorblob is the blob containing the reactors, if any.

VPORT:

Valid AutoCAD versions for this table entry: 10+

Not to be confused with paperspace viewport entities, these are viewports in modelspace that look at the modelspace entities.

typedef struct advportstru {
    AD_OBJHANDLE ownerobjhandle;
    char  purgedflag;
    short xrefindex;  // (11+)
    char  flag;
    char  name[AD_MAX_STRLEN];
    double ll[2];
    double ur[2];
    double snapbase[2];
    double snapdist[2];
    double griddist[2];
    double center[2];
    double viewdir[3];
    double target[3];
    double viewtwist;
    double viewheight;
    short viewmode;
    short zoomperc;
    short fastzoom;
    short ucsicon;
    double aspect;
    double backclip;
    double frontclip;
    short gridmode;
    double lenslength;
    short snapisopair;
    short snapmode;
    double snaprot;
    short snapstyle;
    short rendermode;
    short ucsvp;
    double ucsorg[3];
    double ucsxdir[3];
    double ucsydir[3];
    short ucsorthoviewtype;
    double ucselevation;
    AD_OBJHANDLE ucsobjhandle;
    AD_OBJHANDLE baseucsobjhandle;
    AD_OBJHANDLE objhandle;
    AD_OBJHANDLE xdicobjhandle;
    AD_VMADDR xdblob;  // (13+)
    long numreactors;
    AD_VMADDR reactorblob;
} AD_VPORT,*PAD_VPORT;
ownerobjhandle is the owner object handle for the viewport.
purgedflag is 1 if this viewport has been purged, otherwise 0.
xrefindex is the index of the block header for the xref from which this viewport came, -1 if the viewport was defined in this drawing.
flag has the standard values.
name is the name of the viewport.
ll is the lower left corner of the viewport.
ur is the upper right corner of the viewport.
snapbase is the snap base point.
snapdist is the snap distance (X and Y).
griddist is the distance between grid points.
center is the center of the viewport.
viewdir is the view direction vector for the viewport.
target is the target point for the viewport.
viewtwist is the view twist.
viewheight is the height of this view.
viewmode is the viewmode. See viewmode in the header definition for an explanation.
zoomperc is the circle zoom percentage.
fastzoom is the fast zoom setting.
ucsicon determines the display of the ucsicon. See ucsicon in the header definition.
aspect is the aspect ratio for this viewport.
backclip is the offset of the back clipping plane from the target point.
frontclip is the offset of the front clipping plane from the target point.
gridmode is the grid mode. See gridmode in the header definition.
lenlength is the lens length for perspective views.
napisopaip -- see the header for a definition.
napmod -- see the header for a definition.
naprot -- see the header for a definition.
napstyl -- see the header for a definition.
rendersmode -- see viewport render mode.
ucsvp is 1 if the viewport stores its own UCS, otherwise 0.
ucscrg is the UCS origin.
ucsexdir is the UCS x axis.
ucseydir is the UCS y axis
ucseothovewtype – see AD_UCS.
ucselevalation is the UCS elevation.
ucsbobjhandel is the handle of the named UCS.
baseucsoobjhandel is the handle of the base UCS if the UCS is orthographic.
objhandel is the AD_OBJHANDLE for this viewport.
xdicobjhandel is the AD_OBJHANDLE for the ACAD_XDICTIONARY object associated with this viewport.
xdblob holds the extended data, if any, attached to this viewport.
numreactors is the number of persistent reactors attached to this viewport.
reactorblob is the blob containing the reactors, if any.

OBJECTS SECTION

This section discusses the entries in the OBJECTS section of the AutoCAD DWG file. All objects, like entities, have an object header and object body.
OBJECT HEADER

The object header looks like this:

```c
typedef struct adobjhdrstru {
    AD_OBJHANDLE  objhandle;
    unsigned short objtype;
    AD_VMADDR     xdblob;
    AD_VMADDR     reactorblob;
    long           numreactors;
    AD_OBJHANDLE  ownerobjhandle;
} AD_OBJ_HDR, *PAD_OBJ_HDR;
```

- `objhandle` is the object handle for this object.
- `objtype` is the object type for this object. This is 500+the index of the class in the class list.
- `xdblob` is the extended object data, if any, attached to this object.
- `reactorblob` is the set of persistent reactors, if any, attached to this object.
- `numreactors` is the number of reactors in the reactorblob.
- `ownerobjhandle` is the owner object handle for this object.

Like entities, some objects have identification numbers that are fixed, and we can test of set these simply by comparing with the defined constants:

- AD_OBJ_DICTIONARY
- AD_OBJ_GROUP
- AD_OBJ_MLINESTYLE

Most objects, though, have objtypes which vary with the drawing. The types for these objects are determined by calling a function which returns the object type. For instance, for an IDBUFFER object, you would call

```c
myobjtype=adIdbufferObjtype(handle);
```

to get the objtype for an IDBUFFER.
OBJECT BODY

The AD_OBJ union is a union of structures, each of which is appropriate for a particular object type. Descriptions of the object data for each object type are listed below. In each description, we give the defined constant (or function call return) to which adobhd->objtype would be equal for this object type, the name of the structure definition for this object type, and the name of this structure in the AD_OBJ union. These descriptions assume the following variable definitions:

PAD_OBJ adob;
PAD_OBJ_HDR adobhd;

DICTIONARY:

Defined constant adobhd->objtype==AD_OBJ_DICTIONARY.
Structure definition AD_DIC.
Found in AD_OBJ union structure dic (e.g. adob->dic).
Valid for AutoCAD versions: 13+

Dictionary objects are found in the object list for a drawing. They are containers for pairs of string/AD_OBJHANDLE which indicate other objects.

typedef struct addicstru {
    short numdicitems;
    AD_VMADDR itemblob;
    SHORT hardownerflag; (2000+)
    short cloningflag; (2000+)
} AD_DIC, *PAD_DIC;

numdicitems is the number of items in this dictionary object.
itemblob contains the dictionary items for this object.
hardownerflag is 1 if the dictionary elements are to be treated as hard owned.
cloningflag is the duplicate record cloning flag.

Dictionary items are read from the blob using adReadDicItem. The structure for a dictionary item is given below.

typedef struct addicitemstru {
    char str[AD_MAX_STRLEN];
    AD_OBJHANDLE itemhandle;
} AD_DICITEM, *PAD_DICITEM;

str is the description of the item.
itemhandle is the AD_OBJHANDLE for the object indicated by this item.

Reactors are read from the blob using adReadBlobObjhandle.
DICTIONARYVAR:

Function to retrieve obtype adobhd->obtype==adDictionaryvarObjtype(handle);
Structure definition AD_DICTIONARYVAR.
Found in AD_OBJ union structure dic (e.g. adob->dictionaryvar).
Valid for AutoCAD versions: 14+

Dictionaryvars are found in the object list for the drawing. They are used to store named values in the database as system variables, without need to continue to change the drawing header. As of 1/98, the variables which may be found here are DIMADEC, DIMDSEP, INDEXCTL, PROJECTNAME, and XCLIPFRAME.

typedef struct addictionaryvarobjstru {
    short intval;
    char str[256];
} AD_DICTIONARYVAR,*PAD_DICTIONARYVAR;

DICTIONARYWDFLT:

Defined constant adobhd->obtype==AD_OBJ_DICTIONARYWDFLT.
Structure definition AD_DICTIONARYWDFLT.
Found in AD_OBJ union structure dic (e.g. adob->dictionarywdflt).
Valid for AutoCAD versions: 2000+

Dictionarywdflt objects are found in the object list for a drawing.

typedef struct addictionarywdfltobjstru {
    unsigned short numitems;
    short cloningflag;
    AD_VMADDR itemblob;
    AD_OBJHANDLE defaultobjectobjhandle;
} AD_DICTIONARYWDFLT,*PAD_DICTIONARYWDFLT;

numitems is the number of items in this dictionarywdflt object.
cloningflag indicates the duplicate record cloning flag.
itemblob contains the items for this object.
defaultobjectobjhandle is the default object handle.

GROUP:

Defined constant adobhd->obtype==AD_OBJ_GROUP
Structure definition AD_GROUP.
Found in AD_OBJ union structure grp (e.g. adob->grp).
Valid for AutoCAD versions: 13+

A group is a collection of objects. It is found in the object list.

typedef struct adgrpstru {
    char str[AD_MAX_STRLEN];
    long numgrouphandles;
    short unnamed,selectable;
    AD_VMADDR entryblob;
} AD_GRP,*PAD_GRP;

str is the name of the group.
numgrouphandles is the number of AD_OBJHANDLEs in entryblob.
**unnamed** is 1 if the group has no name, otherwise 0.

**selectable** is 1 if the group is selectable, otherwise 0.

**entryblob** contains the AD_OBJHANDLEs indicating the items in the group. Read this blob using `adReadGroupEntryHandle()`.
**IDBUFFER:**

Function to retrieve objtype adobhd->objtype==adIdbufferObjtype(handle);
Structure definition AD_IDBUFFER
Found in AD_OBJ union structure idbuffer (e.g. adob->idbuffer).
Valid for AutoCAD versions: 14+

An idbuffer contains information related to external references. It contains a list of entries for a given external reference. These objects are created and maintained automatically by adCreateXrefBlock()

```c
typedef struct adidbufferobjstru {
    long numobjids;
    AD_VMADDR objidblob;
} AD_IDBUFFER, *PAD_IDBUFFER;
```

cnumobjids is the number of object ids in the objid blob.
objidblob contains a list of entries for a given external reference.

**IMAGEDEF:**

Function to retrieve objtype adobhd->objtype==adImagedefObjtype(handle);
Structure definition AD_IMAGEDEF
Found in AD_OBJ union structure imagedef (e.g. adob->imagedef).
Valid for AutoCAD versions: 14+

An imagedef contains information related to image entities. No further definition is available at this time.

```c
typedef struct adimagedefobjstru {
    long classversion; /* 0 == R14 version */
    char filepath[512];
    double size[2];
    double pixelsize[2];
    short imageisloaded, resunits;
} AD_IMAGEDEF, *PAD_IMAGEDEF;
```

classversion is the version of this class. 0 means the R14 version.
filepath is the path to the raster file.
size is the size of the image in world coordinates.
pixelsize is the default size of one pixel in AutoCAD units.
imageisloaded is 1 if the image is loaded, 0 if not.
resunits is the units of resolution. 0==no units, 2==centimeters, 5==inches.
**IMAGEDEFREACTOR:**

Function to retrieve objtype adobhd->objtype==adImagedefreactorObjtype(handle);
Structure definition AD_IMAGEDEFREACTOR
Found in AD_OBJ union structure imagedefreactor (e.g. adob->imagedefreactor).
Valid for AutoCAD versions: 14+

An imagedefreactor contains information related to image entities. No further definition is available at this time.

```c
typedef struct adimagedefobjstru {
    long classversion; /* 2==R14 */
    AD_VMADDR reactorblob;
} AD_IMAGEDEFREACTORBLOB,
    *PAD_IMAGEDEFREACTORBLOB;
```

classversion is the class version.
reactorblob is the blob containing the reactors.

**LAYER_INDEX:**

Function to retrieve objtype adobhd->objtype==adLayerindexObjtype(handle);
Structure definition AD_LAYERINDEX
Found in AD_OBJ union structure layerindex (e.g. adob->layerindex).
Valid for AutoCAD versions: 14+

No further definition is available at this time.

```c
typedef struct adlayerindexstru {
    long numentries;
    long timestamp1,timestamp2;
    AD_VMADDR datablob;
    AD_VMADDR handleblob;
} AD_LAYERINDEX, *PAD_LAYERINDEX;
```

**LAYOUT OBJECT:**

Function to retrieve objtype adobhd->objtype==adLayoutObjtype(handle);
Structure definition AD_LAYOUT
Found in AD_OBJ union structure layout (e.g. adob->layout).
Valid for AutoCAD versions: 2000+

A layout object contains information related to AutoCAD layouts.

```c
typedef struct adlayoutobjstru {
    char name[AD_MAX_STRLEN];
    short flag;
    short taborder;
    AD_PLOTSETTINGS plotsettings;
    double limmin[2];
    double limmax[2];
    double inspt[3];
    double extmin[3];
```
double extmax[3];
double elev;
double ucsorg[3];
double ucsxdir[3];
double ucsydir[3];
short ucsorthoviewtype;
AD_OBJHANDLE pspaceblockrec;
AD_OBJHANDLE lastactivevport;
AD_OBJHANDLE ucsobjhandle;
AD_OBJHANDLE baseucsobjhandle;
} AD_LAYOUT,*PAD_LAYOUT;

name is the layout name. flag is a bitfield where & 1 indicates the PSLTSCALE value for this layout when this layout is current, and & 2 indicates the LIMCHECK value for this layout when this layout is current.
taborder indicates the tab position of this layout.
plotsettings contains the plotsettings information.
limmin contains the minimum limits for this layout.
limmax contains the maximum limits for this layout.
inspt contains the insertion point for this layout.
min contains the minimum extents for this layout.
max contains the maximum extents for this layout.
elev contains the elevation for this layout.
ucsort contains the UCS origin.
ucsxdir contains the UCS x-axis.
ucsydir contains the UCS y-axis.
ucsort2type indicates the orthographic type of the UCS, where 0 == UCS is not orthographic; 1 == Top; 2 == Bottom; 3 == Front; 4 == Back; 5 == Left; 6 == Right.
pspaceblockrec indicates the handle of the associated paperspace block.
lastactivevport indicates the handle of the last viewport that was active when this layout was current.
ucsobjhandle indicates the UCS object handle.
baseucsobjhandle indicates the object handle of the base UCS object.

MULTI-LINE STYLE:

Defined constant adobhd->objtype==AD_OBJ_MLSTYLE.
Structure definition AD_MLSTYLE.
Found in AD_OBJ union structure mlstyle (e.g. adob->mlstyle).
Valid for AutoCAD versions: 13+

A multi-line style defines the way a multi-line is drawn.
typedef struct admlstylestru {
  char  name[AD_MAX_STRLEN];
  unsigned short  flag;
  char  desc[256];
  unsigned short  fillcolor;
  double startang,endang;
  unsigned char linesinstyle;
  AD_VMADDR  seginfoflob;
} AD_MLSTYLE,*PAD_MLSTYLE;

name is the name of the mlstyle.
flag is a bit-coded field which can be tested by ANDing with the following defined constants:

AD_MLSTYLE_FILL_ON  (fill is on)
AD_MLSTYLE_SHOW_MITERS  (show miters)
AD_MLSTYLE_START_SQUARE  (start with a square cap)
AD_MLSTYLE_START_INNER_ARC  (start with an inner arc cap)
AD_MLSTYLE_START_ROUND  (start with an outer round cap)
AD_MLSTYLE_END_SQUARE  (end with a square cap)
AD_MLSTYLE_END_INNER_ARC  (end with an inner arc cap)
AD_MLSTYLE_END_ROUND  (end with an outer round cap)

desc is a description of the mlstyle.
fillcolor is the color with which to fill the multi-line.
startang is the start angle for the style.
endang is the end angle for the style.
linesinstyle is the number of parallel lines in this style.
seginfoflob contains the data regarding each segment of the multiline style. You can read this data using adReadMlstyleSeginfo.

typedef struct admlstyleseginfostru {
  short segcolor;
  short  ltindex;
  double offset;
} AD_MLSTYLESEGINFO,*PAD_MLSTYLESEGINFO;

segcolor is the color of the segment.
ltindex is the index of the linetype in the linetype list in which this segment is to be drawn.
offset is the offset of this segment from the previous one.

OBJECTPTR:

Function to retrieve objtype adobhd->objtype==adObjectptrObjtype(handle);
Structure definition AD_OBJECTPTR
Found in AD_OBJ union structure objectptr (e.g. adob->objectptr).
Valid for AutoCAD versions: 14+

No further definition is available at this time.
typedef struct adobjectptrstru {
    AD_VMADDR datablob;
    long databits;
} AD_OBJECTPTR, *PAD_OBJECTPTR;

PLACEHOLDER:

Function to retrieve objtype adobhd->objtype==adPlaceholderObjtype(handle);
Structure definition AD_PLACEHOLDER
Found in AD_OBJ union structure placeholder (e.g. adob->placeholder).
Valid for AutoCAD versions: 2000+

No further definition is available at this time.

typedef struct adplaceholderobjstru {
    short unused;
} AD_PLACEHOLDER, *PAD_PLACEHOLDER;

PLOTSETTINGS OBJECT:

Function to retrieve objtype adobhd->objtype==adPlotsettingsObjtype(handle);
Structure definition AD_PLOTSETTINGS
Found in AD_OBJ union structure layout (e.g. adob->plotsettings).
Valid for AutoCAD versions: 2000+

A plotsettings object contains information related to AutoCAD plotsettings objects.

typedef struct adplotsettingsobjstru {
    char pagesetupname[AD_MAX_STRLEN];
    char printerorconfigfilename[AD_MAX_STRLEN];
    char papersize[AD_MAX_STRLEN];
    char plotviewname[AD_MAX_STRLEN];
    double leftmargin;
    double bottommargin;
    double rightmargin;
    double topmargin;
    double paperwidth;
    double paperheight;
    double origin[2];
    double windowmin[2];
    double windowmax[2];
    double realworldunits;
    double drawingunits;
    short plotlayoutflag;
    short plotpaperunits;
    short plotrotation;
    short plottype;
    char currentstylesheet[AD_MAX_STRLEN];
    short scaletype;
    double scalefactor;
    double paperimageorigin[2];
pagesetupname indicates the page setup name.
prinrorconfigfilename indicates the printer configuration file name.
papersize indicates the paper size.
plotviewname indicates the plot view name.
lefftmargin indicates the size of the left margin in millimeters.
bottommargin indicates the size of the bottom margin in millimeters.
rightmargin indicates the size of the right margin in millimeters.
topmargin indicates the size of the top margin in millimeters.
paperwidth indicates the paper width in millimeters.
paperheight indicates the paper height in millimeters.
origin indicates the plot origin.
windowmin indicates the lower left window corner.
windowmax indicates the upper right window corner.
realworldunits indicates the real world units.
drawingunits indicates the drawing units.
plotlayoutflag indicates the plot layout flag, where 1 == PlotViewportBorders, 2 == ShowPlotStyles, 4 == PlotCentered, 8 == PlotHidden, 16 == UseStandardScale, 32 == PlotPlotStyles, 64 == ScaleLineweights, 128 == PrintLineweights, 512 == DrawViewportsFirst, 1024 == ModelType, 2048 == UpdatePaper, 4096 == ZoomToPaperOnUpdate, 8192 == Initializing, 16384 == PrevPlotInit.
plotpaperunits indicates the plot paper units, where 0 == Plot in inches, 1 == Plot in millimeters, 2 == Plot in pixels.
plotrotation indicates plot rotation, where 0 == No rotation, 1 == 90 degrees, counterclockwise, 2 == Upside down, 3 == 90 degrees clockwise.
plottype indicates plot type, where 0 == Last screen display, 1 == Drawing extents, 2 == Drawing limits, 3 == Specified view, 4 == Specified window, 5 == Layout information.
currentstylesheet indicates the current style sheet.
scaletype indicates the standard scale type.
scalefactor contains the floating point scale factor.
paperimageorigin indicates the paper image origin.

PROXY OBJECT:

Valid AutoCAD versions for this table entry: 13+

Similar to proxy entities, these are instances of a class and reside in the object list for a drawing.

typedef struct adproxyobjstru {
  long  databits;
  long  numobjids;
  AD_VMADDR datablob;
  AD_VMADDR objidblob;
  long objectdrawingformat;   (2000+)
  short origdataformat;        (2000+)
} AD_OBJ_PROXY, *PAD_OBJ_PROXY;

databits is the number of valid data bits in the datablob.
numobjids is the number of object ids in the objidblob.
datablob holds the object's data. This data can be read with the adReadProxyDataXXXXX functions.
objidblob holds the object ids related to this object.
objectdrawingformat contains AcDbDwgVersion as the low word, and MaintenanceReleaseVersion as the high word.
origdataformat is original data format of the object, 0 for DWG and 1 for DXF.
RASTERVARIABLES OBJECT:

Function to retrieve objtype adobhd->objtype==adRastervariablesObjtype(handle);
Structure definition AD_RASTERVARIABLES
Found in AD_OBJ union structure rastervariables (e.g. adob->rastervariables).
Valid for AutoCAD versions: 14+

A rastervariables object contains information related to image entities.

typedef struct adrastervariablesobjstru {
    long classversion; /* 0==R14 version */
    short displayframe;
    short displayquality;
    short units;
} AD_RASTERVARIABLES, *PAD_RASTERVARIABLES;

classversion is the version of the class.
displayframe indicates whether the frame around the image should be displayed. 1==yes, 0==no.
displayquality indicates the display quality. 0==draft, 1==high.
units indicates the AutoCAD units for inserting images. This indicates the value of one AutoCAD unit for
the purpose of inserting and scaling images.
    0: None
    1: Millimeter
    2: Centimeter
    3: Meter
    4: Kilometer
    5: Inch
    6: Foot
    7: Yard
    8: Mile

SORTENTSTABLE:

Function to retrieve objtype adobhd->objtype==adSortentstableObjtype(handle);
Structure definition AD_SORTENTSTABLE
Found in AD_OBJ union structure sortentstable (e.g. adob->sortentstable).
Valid for AutoCAD versions: 14+

This table is used by the DRAWORDER command to associate handles in the drawing with handles that
indicate the order in which entities are to be drawn. Normally entities are drawn in ascending order by handle;
this list give an alternative ordering which is used when DRAWORDER is in use.

typedef struct adsortentstablestru {
    AD_OBJHANDLE owner;
    long numentries;
    AD_VMADDR sorthandleblob,handleblob;
} AD_SORTENTSTABLE, *PAD_SORTENTSTABLE;

owner is the owner dictionary.
numentries is the number of entries in the sorthandleblob and the
handleblob
sorthandleblob is a blob full of handles indicating the order of drawing.
handleblob contains the handles corresponding to the sorthandleblob's handles.

SPATIALFILTER OBJECT:

Function to retrieve objtype adobhd->objtype==adSpatialfilterObjtype(handle);
Structure definition AD_SPATIALFILTER
Found in AD_OBJ union structure spatialfilter (e.g. adob->spatialfilter).
Valid for AutoCAD versions: 14+

A spatialfilter object contains information relating to clipped external references.

typedef struct adspatialfilterobjstru {
    unsigned short numpts;
    AD_VMADDR ptblob;
    double extrusion[3];
    double clipboundorigin[3];
    short displayboundary, frontclipon, backclipon;
    double frontclip, backclip;
    double inverseblocktransmat[4][3], clipboundtransmat[4][3];
} AD_SPATIALFILTER, *PAD_SPATIALFILTER;

numpts is the number of points in the ptblob.
ptblob is a blob containing point entries.
extrusion is the extrusion vector for this filter.
clipboundorigin1 is the origin of the local coordinate system for the clip boundary.
displayboundary displays boundary if 1, no display if 0.
frontclipon is 1 if front clip is on, 0 if off.
backclipon is 1 if back clip is on, 0 if off.
frontclip holds the front clip distance.
backclip holds the back clip distance.
inverseblocktransmat is the inverse of the block transformation matrix for this object.
clipboundtransmat is the transformation matrix to clip boundary coordinates.

SPATIALINDEX:

Function to retrieve objtype adobhd->objtype==adSpatialindexObjtype(handle);
Structure definition AD_SPATIALINDEX
Found in AD_OBJ union structure spatialindex (e.g. adob->spatialindex).
Valid for AutoCAD versions: 14+

No further definition is available at this time.

typedef struct adspatialindexstru {
    long timestamp1, timestamp2;
    AD_VMADDR datablob;
    long databits;
} AD_SPATIALINDEX, *PAD_SPATIALINDEX;

VBAPROJECT OBJECT:

Function to retrieve objtype adobhd->objtype==adVbaProjectObjtype(handle);
Structure definition AD_VBAPROJECT
typedef struct advbaprojectobjstru {
   long datalength;
   AD_VMADDR datablob;
} AD_VBAPROJECT, *PAD_VBAPROJECT;

datalength is the number of data bytes in the datablob.

datablob holds the VB data.

XRECORD OBJECT:

Function to retrieve objtype adobhd->objtype==adXrecordObjtype(handle);
Structure definition AD_XRECORD
Found in AD_OBJ union structure xrecord (e.g. adob->xrecord).
Valid for AutoCAD versions: 13+

An xrecord object contains arbitrary data. Xrecord data can be read using adReadXrecordItem().

typedef struct adxrecordobjstru {
   long databits;
   AD_VMADDR datablob;
   short cloningflag;   (2000+)
} AD_XRECORD, *PAD_XRECORD;

databits is the number of data bits in the datablob.
datablob holds the xrecord data.
cloningflag is the cloning flag for the xrecord.

See adexr.c for an example of reading Xrecord data.

WIPEOUTVARIABLES OBJECT:

Function to retrieve objtype adobhd->objtype==adWipeoutVariablesObjtype(handle);
Structure definition AD_WIPEOUTVARIABLES
Found in AD_OBJ union structure wipeoutvariables (e.g. adob->wipeoutvariables).
Valid for AutoCAD versions: 2000+

A wipeoutvariables object contains information related to wipeout entities.

typedef struct adwipeoutvariablesobjstru {
   short dxf70;
} AD_WIPEOUTVARIABLES,*PAD_WIPEOUTVARIABLES;

dxf70 is the DXF 70 value.
11. ERROR CODES

OpenDWG Toolkit, through the \texttt{adError()} function, provides a mechanism for determining the cause of errors which occur during a program's run. The simplest way to interpret the meaning of the value returned by \texttt{adError()} is to use \texttt{adErrorStr(adError())}, which returns a string which is a description of the error. Below is a list of the error strings returned by \texttt{adErrorStr} and some more description of the possible causes of these errors.

"Attempt to add duplicate handle"

A program has attempted to add an item to the OpenDWG Toolkit database which has an AD_OBJCHANDLE which is already used by another object in the database.

"Attempt to add NULL objhandle"

An attempt has been made to add an object to the OpenDWG Toolkit database whose object handle is NULL (all 0s).

"Attempt to allocate 0 bytes"

There has been an attempt to allocate an item of size 0 bytes from the virtual memory subsystem.

"Attempt to delete from null list"

There has been an attempt to delete from a list that has no entries.

"Attempt to free a free VM pointer"

There has been an attempt to free a block of virtual memory which is already free. This error is usually an artifact of some other error, such as a file with bad data or adding items to the file with duplicate handles.

"Attempt to free a non-malloced VM pointer"

There has been an attempt to free a block of virtual memory which is not allocated.

"Attempt to free a null VM pointer"

There has been an attempt to free a block of virtual memory whose address is AD_VMNULL.

"Attempt to insert non-existent block"

There has been an attempt to add an insert to the database whose block referent does not exist.

"Attempt to read from non-existent VM block"

There has been an attempt to read from a virtual memory block which does not exist.

"Attempt to read from null blob"

There has been an attempt to read from a blob which is NULL.

"Attempt to write to null blob"

There has been an attempt to write to a blob which is NULL.

"Attempt to write unknown Xrecord type"

There has been an attempt to write an Xrecord type which is not recognized by OpenDWG Toolkit.

"Bad class definition"

\texttt{adDwgSmellsBad} detected a bad class definition in a DWG file.

"Bad entity"
adDwgSmellsBad detected a bad entity in a DWG file.

"Bad object"

adDwgSmellsBad detected a bad object in a DWG file.

"Bad second header"

adDwgSmellsBad detected a bad second header in a DWG file.

"Bad vmmalloc magic number"

OpenDWG Toolkit has detected an inconsistency in the virtual memory subsystem.

"Block entity invalid"

adDwgSmellsBad detected a bad entity in a block in a DWG file.

"Block header invalid"

adDwgSmellsBad detected a bad blockheader in a DWG file.

"Block header not found"

OpenDWG Toolkit could not find the requested block header.

"Btree error"

There has been an error in OpenDWG Toolkit's OBJHANDLE index.

"Can't add default objects"

OpenDWG Toolkit was unable to add the default objects to a newly created file. This is probably because of corruption or some kind of memory allocation error.

"Can't add group"

OpenDWG Toolkit was unable to add a GROUP object to the file.

"Can't allocate "

Various memory allocation failures.

"Can't create dxf temp data list"

A memory allocation failed when OpenDWG Toolkit attempted to create a virtual memory list for temporary OpenDWG Toolkit bookkeeping data for a DXF file.

"Can't create idbuffer class"

OpenDWG Toolkit was unable to add an idbuffer class to the file.

"Can't determine file type"

OpenDWG Toolkit was unable to determine the type of file to be opened.

"Can't find block for entity read"

During a call to adEntityList, OpenDWG Toolkit could not find the requested block.

"Can't find entity to replace"

A request was made to replace an entity that OpenDWG Toolkit could not find in the database.

"Can't find group dictionary"
OpenDWG Toolkit was unable to find the group dictionary during adCreateGroup().

"Can't find leader dimstyle"
OpenDWG Toolkit could not find the dimension style indicated for a leader entity.

"Can't find tolerance dimstyle"
OpenDWG Toolkit could not find the dimension style indicated for a tolerance entity.

"Can't get current mlstyle"
OpenDWG Toolkit could not find the current multi-line style.

"Can't init file system"
OpenDWG Toolkit could not initialize its file system, probably due to insufficient memory.

"Can't malloc for raster file write"
OpenDWG Toolkit could not allocate temporary storage for write of a raster file.

"Can't open BMP output file"
OpenDWG Toolkit could not open the requested output file during adExtractBmpToFile().

"Can't open external reference"
OpenDWG Toolkit could not open an external reference (AD3 only).

"Can't open file"
OpenDWG Toolkit was unable to open the requested file.

"Can't open WMF output file"
OpenDWG Toolkit could not open the requested output file during adExtractWmfToFile().

"Can't open xref temp file"
OpenDWG Toolkit was unable to open an xref temporary file.

"Can't read header block"
OpenDWG Toolkit could not read the header block from virtual memory.

"Can't replace dictionary object"
OpenDWG Toolkit was unable to replace a dictionary object.

"Can't replace entity"
OpenDWG Toolkit was unable to replace an entity.

"Can't replace objects of different types"
OpenDWG Toolkit was unable to replace one object with another because their types were not identical.

"Can't resolve block header"
OpenDWG Toolkit could not resolve (convert to R13 DWG format) a block header.

"Can't resolve dimension style"
OpenDWG Toolkit could not resolve (convert to R13 DWG format) a dimension style.
"Can't resolve extended data application index"
OpenDWG Toolkit could not resolve (convert to R13 DWG format) an extended data application index.

"Can't resolve extended data entity layer index"
OpenDWG Toolkit could not resolve (convert to R13 DWG format) an extended data entity layer index.

"Can't resolve header dim text style"
OpenDWG Toolkit could not resolve (convert to R13 DWG format) the default dimension text style from the file's header.

"Can't resolve header dimstyle"
OpenDWG Toolkit could not resolve (convert to R13 DWG format) the current dimension style.

"Can't resolve header layer"
OpenDWG Toolkit could not resolve (convert to R13 DWG format) the current layer.

"Can't resolve header linetype"
OpenDWG Toolkit could not resolve (convert to R13 DWG format) the current linetype.

"Can't resolve header mlstyle"
OpenDWG Toolkit could not resolve (convert to R13 DWG format) the current multi-line style.

"Can't resolve header pspace ucs"
OpenDWG Toolkit could not resolve (convert to R13 DWG format) the current paperspace UCS.

"Can't resolve header text style"
OpenDWG Toolkit could not resolve (convert to R13 DWG format) the current text style.

"Can't resolve header ucs"
OpenDWG Toolkit could not resolve (convert to R13 DWG format) the current UCS.

"Can't resolve layer"
OpenDWG Toolkit could not resolve (convert to R13 DWG format) a layer entry.

"Can't resolve linetype"
OpenDWG Toolkit could not resolve (convert to R13 DWG format) a linetype entry.

"Can't resolve text style/shape file"
OpenDWG Toolkit could not resolve (convert to R13 DWG format) a text style/shapefile entry.

"Can't seek entity"
OpenDWG Toolkit was unable to seek a requested entity.

"Can't seek group dictionary"
OpenDWG Toolkit was unable to seek to the group dictionary.

"Can't set current node"
OpenDWG Toolkit was unable to set the current node for a list.

"Can't set write buffer"
A call to setvbuf failed during open of a file for write.

"Cannot process large proxy entity"

OpenDWG Toolkit was unable to process a proxy entity with data larger than 64K.

"Character not in font"

`adComputeNewTextParms()` was unable to find a character in the width table.

"Couldn’t find vpenheader handle"

OpenDWG Toolkit could not find the OBJHANDLE for a viewport entity header.

"Did not find class to delete"

An attempt was made to delete a class that OpenDWG Toolkit could not find.

"Did not find object to delete"

An attempt was made to delete an object that OpenDWG Toolkit could not find.

"Dimstyle invalid"

`adDwgSmellsBad` found an invalid dimension style in a DWG file.

"Dimstyle not found"

OpenDWG Toolkit was unsuccessful in finding a requested dimension style.

"Disk full"

The target disk is full.

"Dropped shape entity from file"

OpenDWG Toolkit was unable to identify a shape in a file when it was loaded. This is probably because ODT could not find the related .SHX file. As a result, during an attempt to save a file, OpenDWG Toolkit reports that it had to drop the shape entity from the file.

"Duplicate handle in file"

During save, OpenDWG Toolkit detected that the file had a duplicated handle.

"Entity invalid"

`adDwgSmellsBad` found an invalid entity in a DWG file.

"Entity not found in list"

A requested entity was not found in the list.

"Error allocating **"

Various memory allocation errors.

"Error closing file"

OpenDWG Toolkit encountered an error while closing a file.

"Error deleting **"

OpenDWG Toolkit was unable to delete the requested item.

"Error flushing object to VM"
OpenDWG Toolkit was unable to complete write of a block to virtual memory.

"Error freeing **
OpenDWG Toolkit encountered an error freeing the virtual memory for this item.

"Error in DXF block"
OpenDWG Toolkit detected an error in the BLOCKS section of a DXF file.

"Error in DXF entity"
OpenDWG Toolkit detected an error in the ENTITIES section of a DXF file.

"Error in DXF header"
OpenDWG Toolkit detected an error in the HEADER section of a DXF file.

"Error in DXF object"
OpenDWG Toolkit detected an error in the OBJECTS section of a DXF file.

"Error in DXF table"
OpenDWG Toolkit detected an error in the TABLES section of a DXF file.

"Error in header"
addDwgSmellsBad detected an error in the header of a DWG file.

"Error in raster data"
OpenDWG Toolkit detected an error in some raster data (BMP or WMF).

"Error in xdata read malloc"
OpenDWG Toolkit was unable to allocate memory for extended data read.

"Error in xdata write malloc"
OpenDWG Toolkit was unable to allocate memory for extended data write.

"Error loading **
OpenDWG Toolkit encountered an error during load of this item.

"Error on block entity read"
OpenDWG Toolkit encountered an error while trying to load this item.

"Error on entity add during replace"
OpenDWG Toolkit was unable to add an item to the database during adReplaceEntity().

"Error on layer read start"
OpenDWG Toolkit encountered an error while trying to load this item.

"Error on vpent list update"
OpenDWG Toolkit encountered an error while updating the internal viewport entity list.

"Error reading header"
OpenDWG Toolkit encountered an error while trying to load this item.
"Error reading layer"
OpenDWG Toolkit encountered an error while trying to load this item.

"Error removing purged items"
OpenDWG Toolkit encountered an error while trying to remove purged items.

"Error resolving header"
OpenDWG Toolkit was unable to resolve (convert to R13 DWG format) the header.

"Error resolving viewport entities"
OpenDWG Toolkit was unable to resolve (convert to R13 DWG format) a viewport entity.

"Error writing dictionary object"
OpenDWG Toolkit encountered an error while writing a dictionary object.

"Error writing header"
OpenDWG Toolkit encountered an error while writing a file header.

"Error writing proxy object"
OpenDWG Toolkit encountered an error while writing a proxy object.

"File already has BMP"
An attempt was made to add a BMP image to a file that already has one.

"File already has WMF"
An attempt was made to add a WMF image to a file that already has one.

"File has no BMP"
An attempt was made to extract a BMP image to a file that has none.

"File has no WMF"
An attempt was made to extract a WMF image to a file that has none.

"Font not in table"
During `adComputeNewTextParms`, OpenDWG Toolkit was unable to find a requested font in the width table.

"General failure"
OpenDWG Toolkit failure. Usually due to unserialized downloaded software.

"Header size invalid"
`adDwgSmellsBad` detected an invalid header in a DWG file.

"Incomplete ACIS entry (no image)"
An attempt was made to write an ACIS entity (SOLID3D, REGION, or BODY), which had no image data attached, to a DWG file.

"Incorrect init file version"
An attempt was made to use an invalid version of ADINIT.DAT with OpenDWG Toolkit. Be sure to use the proper version for the version of ODT you are using.

"Invalid DWG map info"

adDwgSmellsBad detected an invalid map in a DWG file.

"Invalid DXF version"

An attempt was made to open a DXF file whose version is not supported by OpenDWG Toolkit.

"Layer invalid"

adDwgSmellsBad detected an invalid layer in a DWG file.

"Layer not found"

OpenDWG Toolkit was unable to find a requested layer.

"Linetype invalid"

adDwgSmellsBad detected an invalid linetype in a DWG file.

"Linetype not found"

OpenDWG Toolkit was unable to find a requested linetype.

"Maximum VM memory blocks exceeded"

The maximum number of virtual memory blocks has been exceeded.

"No blockheaders in list"

No blockheaders are present in the blockheader list.

"No BMP present in this file"

There is no BMP attached to this file.

"No classes in list"

There are no entries in this list.

"No dimstyles in list"

There are no entries in this list.

"No entities found for block"

The block being acted upon has no entities.

"No error"

There was no error.

"No layers in list"

There are no entries in this list.

"No linetypes in list"

There are no entries in this list.

"No pspace vport (id 1) found"

No paperspace viewport was found when one was required.
"No objects in list"
   There are no entries in this list.

"No registered apps in list"
   There are no entries in this list.

"No shapefiles in list"
   There are no entries in this list.

"No ucss in list"
   There are no entries in this list.

"No viewports in list"
   There are no entries in this list.

"No views in list"
   There are no entries in this list.

"No WMF present in this file"
   There is no WMF data attached to this file.

"Not enough memory to draw leader"
   OpenDWG Toolkit was unable to allocate temporary storage for a leader draw.

"Not enough memory to draw spline"
   OpenDWG Toolkit was unable to allocate temporary storage for a spline draw.

"Not enough memory to start VM"
   OpenDWG Toolkit could not start the virtual memory subsystem.

"Object map write error"
   OpenDWG Toolkit had trouble writing an R13+ DWG file. Normally this is due to either a duplicate or NULL object handle in the file.

"Save attempted to incompatible version"
   OpenDWG Toolkit could not save the database to a version that was later than the version originally passed to adNewFile.

"Regapp not found"
   A requested registered application could not be found.

"Registered application invalid"
   adDwgSmellsBad detected an invalid registered application in a DWG file.

"Shape file invalid"
   adDwgSmellsBad detected an invalid shapefile entry in a DWG file.

"Shapefile not found"
   A requested shapefile could not be found.
"Too many VM page frames"
An attempt was made to allocate more than the maximum number of virtual memory page frames.

"Too many VM pages"
An attempt was made to allocate more than the maximum number of virtual memory pages.

"UCS invalid"
adDwgSmellsBad detected an invalid UCS in a DWG file.

"Ucs not found"
A requested UCS could not be found.

"Unable to allocate VM memory"
OpenDWG Toolkit was unable to allocate memory to start the virtual memory manager.

"Unable to open init file"
OpenDWG Toolkit was unable to open ADINIT.DAT.

"Unexpected end of file in DXF"
OpenDWG Toolkit reached the end of a DXF file unexpectedly.

"Unexpected end of file"
OpenDWG Toolkit reached the end of a file unexpectedly.

"Unknown Xrecord type read"
An Xrecord entry of unknown type was read.

"Unsupported AutoCAD version"
During a file open an attempt was made to open a DWG file whose version is not supported by OpenDWG Toolkit.

"Unsupported DXF version"
During a file open an attempt was made to open a DXF file whose version is not supported by OpenDWG Toolkit.

"Unsupported file write version"
An attempt was made to write a file in a format not supported by OpenDWG Toolkit.

"View invalid"
adDwgSmellsBad detected an invalid view in a DWG file.

"View not found"
A requested view could not be found.

"Viewport entity header invalid"
adDwgSmellsBad detected an invalid viewport entity header in a DWG file.

"Viewport invalid"
adDwgSmellsBad detected an invalid viewport in a DWG file.

"Viewport not found"
A requested vport could not be found.

"VM file write failed"

A write to the virtual memory system failed.
12. FREQUENTLY ASKED QUESTIONS

1) How do I tell if a polyline vertex has width?

If the startwidthvalid field of a vertex is 1, you should use the startwidth value associated with the vertex. If not, you should use the default value stored in the polyline entity that started the polyline. Similarly for endwidth and endwidthvalid.

2) Why aren't my Watcom builds working properly?

It may be that you need to increase the stack size to something north of 40K.

3) Why do I get garbage results from OpenDWG Toolkit structures?

The most probable reason is that you are using a non-default structure alignment. If you do this, unless you take some other action the data in OpenDWG Toolkit structures will not be readable, since the compiler will be assuming that OpenDWG Toolkit was compiled with an alignment different than the one actually used.

We discuss how to handle this in the section STRUCTURE ALIGNMENT ISSUES.

Another possibility is that you may have downloaded new libraries without downloading nonlibs.zip, and thus you are using out-of-date headers that may not be correct for the library.

4) What does the void fun _((short handle, long longnum)) construction mean?

We use a method that allows us to have function prototypes for the platforms we support which have them, and to remove the prototypes for the platforms that do not. By defining

```c
#ifdef AD_PROTOTYPES
#define _(a) a
#else
#define _(a) ()
#endif
```

the arguments are present if AD_PROTOTYPES is defined, otherwise not.

6) Which libraries are for Win 95?

It depends on whether you are compiling for 16 bit or 32 bit. Select the appropriate library by looking at the list earlier in this manual, or in LIBINFO.TXT, and choose either a 16 bit or 32 bit library as appropriate.

7) Why aren't the changes I'm making to the file taking effect?

Either because you are not executing adSaveFile() or you are reading into structures but not putting the information back using one of the adReplace*() functions.

8) Can OpenDWG Toolkit be run on a thread?
We don't recommend it. It certainly is not re-entrant; if you must try to run OpenDWG Toolkit on a thread you should make sure that you do not re-initialize it over and over, and that not more than one process is in OpenDWG Toolkit at a time.

9) Why don't anonymous blocks have identifying numbers? (*X3 or whatever)

The numbers are simply an artificial method that AutoCAD uses to distinguish between blocks. When using OpenDWG Toolkit you should use the blockheader object handle and the blockheaderobjhandle field of the insert structure to refer to a particular block.

10) How do I use OpenDWG Toolkit with the VC debugger without a debug version of OpenDWG Toolkit? I am having strange lockup problems when I try to run a debug version of my program with OpenDWG Toolkit in VC.

Are you getting an error message that says (something like) "Functions in LIBC conflict with other previously linked functions – use /nodefaultlib"? If so, this is the problem. When you are creating a debug version of your program, it is important to make sure that you are not linking in both the debug and non-debug versions of the C runtime library. OpenDWG Toolkit will always want the non-debug version, so you must force the linker not to use it by using the /nodefaultlib:libc switch. Other possibilities for the library that needs to be excluded are libcmt and msvcr71 if you are compiling for either multi-threaded or dll use.

11) How do I use OpenDWG Toolkit with Borland C++ Builder?

You can use the BC 5 libraries, but C++ builder has a different default structure alignment, so you need to add the pragmas shown below surrounding the #include statements:

```
#pragma option -a1 /* that's a-one, not a-ell */
#include "ad2.h"
#include "ad3.h"
#pragma option -a4
```
INDEX

3D Face, 252
3D Line, 252

Aborting load, 39, 41
ACIS data, 253
AD_ACIS, 253
AD_APP, 294
AD_ARC, 254
AD_ATTDEF, 255
AD_ATTRIB, 256
AD_BKIH, 289
AD_BLOCK, 256
AD_CIRCLE, 257
AD_CLASS, 290
AD_COLOR_BYBLOCK, 236
AD_COLOR_BYLAYER, 236
AD_DIC, 301, 302
AD_DICTIONARYVAR, 302
AD_DIM, 258
AD_DIMSTYLE, 291
AD_DWGHDR, 235
AD_ELLIPSE, 260
AD_ENT, 251
AD_ENT_HDR, 250
AD_FACE3D, 252
AD_GRP, 302
AD_HATCH, 261
AD_IDBUFFER, 304
AD_IMAGE, 263
AD_IMAGEDEF, 304
AD_IMAGEDEFREACTORBLOB, 305
AD_INSERT, 264
AD_IS_A_PROXY, 271
AD_LAY, 292
AD_LEADER, 265
AD_LINE, 267
AD_LINE3D, 252
AD_LTYPE, 293
AD_LWPLINE, 266
AD_MLINE, 267
AD_MLSTYLE, 307
AD_MLSTYLESEGINFO, 307
AD_MTEXT, 268
AD_OBJ, 301
AD_OBJ_HDR, 300
AD_OBJ_PROXY, 309
AD_OBJHANLEDS, 26
AD_OLE2FRAME, 269
AD_FLINE, 270
AD_POINT, 269
AD_PROTOTYPES, 31
AD_PROXYENT, 271
AD_RASTERVARIABLES, 310
AD_RAY, 277
AD_SHAPE, 279
AD_SHPTB, 295
AD_SOL, 279
AD_SPATIALFILTER, 311
AD_SPLINE, 280
AD_TDATA, 281
AD_TEXT, 280
AD_TOLERANCE, 282
AD_TRACE, 282
AD_TYPEDOBJHANDLE, 26
AD_UCS, 296
AD_VERTEX, 282
AD_VIEW, 297
AD_VFENT, 284
AD_VPORT, 299
AD_XD, 288
AD_X_DATA, 288
AD_XLINE, 257
AD_XRECORD, 312
adAcceptR14BetaFiles, 53
adAddBlockHeader, 53
adAddBmp, 54
adAddBmpFromBuffer, 54
adAddClass, 55
adAddDimStyle, 55
adAddEntityToList, 57
adAddEntityToListAfter, 57
adAddLayer, 58
adAddLinetype, 59
adAddObject, 60
adAddReactorToEntity, 61
adAddRegApp, 61
adAddShapeFile, 62
adAddUcs, 62
adAddView, 63
adAddVport, 64
adAdVersion, 64
adAllowInvalidDxfVersions, 65
adArcAlignedTextEnttype, 65
adAsciiCodePageToShort, 66
adAsciiHexToBinary, 66
adAsciiObjHandleToBinary, 66
adAudit, 67
adBinaryHexToAscii, 68
adBinaryObjHandleToAscii, 68
adBlobSeek, 69
<table>
<thead>
<tr>
<th>Function Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>adBlobSize</td>
<td>blob size</td>
</tr>
<tr>
<td>adBlobSizeFromVmaddr</td>
<td>blob size from vmaddr</td>
</tr>
<tr>
<td>adBlobTell</td>
<td>blob tell</td>
</tr>
<tr>
<td>adBmpSize</td>
<td>bmp size</td>
</tr>
<tr>
<td>adBmpSizeFromFile</td>
<td>bmp size from file</td>
</tr>
<tr>
<td>adCalcPlineArcCenter</td>
<td>calculate pline arc center</td>
</tr>
<tr>
<td>adCloseAd2</td>
<td>close ad2</td>
</tr>
<tr>
<td>adCloseFile</td>
<td>close file</td>
</tr>
<tr>
<td>adCodePageCharToUnicode</td>
<td>code page char to unicode</td>
</tr>
<tr>
<td>adComputeBulge</td>
<td>compute bulge</td>
</tr>
<tr>
<td>adConvertPlinesOnLoad</td>
<td>convert plines on load</td>
</tr>
<tr>
<td>adCreateBlob</td>
<td>create blob</td>
</tr>
<tr>
<td>adCreateGroup</td>
<td>create group</td>
</tr>
<tr>
<td>adCreateXrefBlock</td>
<td>create xref block</td>
</tr>
<tr>
<td>adCtrlObjhandlePtr</td>
<td>control objhandle ptr</td>
</tr>
<tr>
<td>adDecodeAcadDate</td>
<td>decode acad date</td>
</tr>
<tr>
<td>adDecodeAcadTime</td>
<td>decode acad time</td>
</tr>
<tr>
<td>adDefaultExamineFile</td>
<td>default examine file</td>
</tr>
<tr>
<td>adDeleteBlob</td>
<td>delete blob</td>
</tr>
<tr>
<td>adDeleteBlockheader</td>
<td>delete blockheader</td>
</tr>
<tr>
<td>adDeleteBmp</td>
<td>delete bmp</td>
</tr>
<tr>
<td>adDeleteClass</td>
<td>delete class</td>
</tr>
<tr>
<td>adDeleteDimstyle</td>
<td>delete dimstyle</td>
</tr>
<tr>
<td>adDeleteEntity</td>
<td>delete entity</td>
</tr>
<tr>
<td>adDeleteLayer</td>
<td>delete layer</td>
</tr>
<tr>
<td>adDeleteLinetype</td>
<td>delete linetype</td>
</tr>
<tr>
<td>adDeleteObject</td>
<td>delete object</td>
</tr>
<tr>
<td>adDeleteRegapp</td>
<td>delete regapp</td>
</tr>
<tr>
<td>adDeleteShapefile</td>
<td>delete shapefile</td>
</tr>
<tr>
<td>adDeleteUcs</td>
<td>delete ucs</td>
</tr>
<tr>
<td>adDeleteView</td>
<td>delete view</td>
</tr>
<tr>
<td>adDeleteVport</td>
<td>delete vport</td>
</tr>
<tr>
<td>adDeleteWmf</td>
<td>delete wmf</td>
</tr>
<tr>
<td>adDictionaryvarObjtype</td>
<td>dictionary var obj type</td>
</tr>
<tr>
<td>adDictionarywdfliObjtype</td>
<td>dictionary wdf li obj type</td>
</tr>
<tr>
<td>adDwgSmellsBad</td>
<td>dwg smells bad</td>
</tr>
<tr>
<td>adDxfQualityFlags</td>
<td>dxf quality flags</td>
</tr>
<tr>
<td>adEncodeAcadDate</td>
<td>encode acad date</td>
</tr>
<tr>
<td>adEncodeAcadTime</td>
<td>encode acad time</td>
</tr>
<tr>
<td>adEntHasExtrusion</td>
<td>ent has extrusion</td>
</tr>
<tr>
<td>adEntIsInPaperspace</td>
<td>ent is in paperspace</td>
</tr>
<tr>
<td>adEntIsInvisible</td>
<td>ent is invisible</td>
</tr>
<tr>
<td>adEntityList</td>
<td>entity list</td>
</tr>
<tr>
<td>adEntityListIndex</td>
<td>entity list index</td>
</tr>
<tr>
<td>adEntityListLast</td>
<td>entity list last</td>
</tr>
<tr>
<td>adEntityListNext</td>
<td>entity list next</td>
</tr>
<tr>
<td>adEntityListPrev</td>
<td>entity list prev</td>
</tr>
<tr>
<td>adEntityListSize</td>
<td>entity list size</td>
</tr>
<tr>
<td>adError</td>
<td>error</td>
</tr>
<tr>
<td>adErrorClear</td>
<td>error clear</td>
</tr>
<tr>
<td>adErrorObjhandle</td>
<td>error obj handle</td>
</tr>
<tr>
<td>adErrorStr</td>
<td>error str</td>
</tr>
<tr>
<td>adExistsBlobFromFileToBuffer</td>
<td>exists blob from file to buffer</td>
</tr>
<tr>
<td>adExtractBmpToBuffer</td>
<td>extract bmp to buffer</td>
</tr>
<tr>
<td>adExtractBmpToFile</td>
<td>extract bmp to file</td>
</tr>
<tr>
<td>adFileDisplayableInPaperspace</td>
<td>file displayable in paperspace</td>
</tr>
<tr>
<td>adFileHasBmp</td>
<td>file has bmp</td>
</tr>
<tr>
<td>adFileHasWmf</td>
<td>file has wmf</td>
</tr>
<tr>
<td>adFileSavedInModelSpace</td>
<td>file saved in model space</td>
</tr>
<tr>
<td>adFileTypeLoaded</td>
<td>file type loaded</td>
</tr>
<tr>
<td>adFileVersionLoaded</td>
<td>file version loaded</td>
</tr>
<tr>
<td>adFindBlockheaderByName</td>
<td>find blockheader by name</td>
</tr>
<tr>
<td>adFindDimstyleByName</td>
<td>find dimstyle by name</td>
</tr>
<tr>
<td>adFindLayerByName</td>
<td>find layer by name</td>
</tr>
<tr>
<td>adFindLinetypeByName</td>
<td>find linetype by name</td>
</tr>
<tr>
<td>adFindRegappByName</td>
<td>find regapp by name</td>
</tr>
<tr>
<td>adFindShapefileByName</td>
<td>find shapefile by name</td>
</tr>
<tr>
<td>adFindUcsByName</td>
<td>find ucs by name</td>
</tr>
<tr>
<td>adFindViewByName</td>
<td>find view by name</td>
</tr>
<tr>
<td>adGenerateObjhandle</td>
<td>generate obj handle</td>
</tr>
<tr>
<td>adGetBlockHandle</td>
<td>get block handle</td>
</tr>
<tr>
<td>adGetBlockheader</td>
<td>get blockheader</td>
</tr>
<tr>
<td>adGetClass</td>
<td>get class</td>
</tr>
<tr>
<td>adGetDimstyle</td>
<td>get dimstyle</td>
</tr>
<tr>
<td>adGetEntity</td>
<td>get entity</td>
</tr>
<tr>
<td>adGetLayer</td>
<td>get layer</td>
</tr>
<tr>
<td>adGetLayerState</td>
<td>get layer state</td>
</tr>
<tr>
<td>adGetLinetype</td>
<td>get linetype</td>
</tr>
<tr>
<td>adGetObject</td>
<td>get object</td>
</tr>
<tr>
<td>adGetRegapp</td>
<td>get regapp</td>
</tr>
<tr>
<td>adGetShapefile</td>
<td>get shapefile</td>
</tr>
<tr>
<td>adGetUcs</td>
<td>get ucs</td>
</tr>
<tr>
<td>adGetView</td>
<td>get view</td>
</tr>
<tr>
<td>adGetVport</td>
<td>get vport</td>
</tr>
<tr>
<td>adHanclear</td>
<td>hanclear</td>
</tr>
<tr>
<td>adHancmp</td>
<td>hancmp</td>
</tr>
<tr>
<td>adHasPrimTraits</td>
<td>has prim traits</td>
</tr>
<tr>
<td>adHatchEnttype</td>
<td>hatch ent type</td>
</tr>
<tr>
<td>adHeaderPointer</td>
<td>header pointer</td>
</tr>
<tr>
<td>adIdbufferObjtype</td>
<td>id buffer obj type</td>
</tr>
<tr>
<td>adImagedefObjtype</td>
<td>imagedef obj type</td>
</tr>
<tr>
<td>adImagedefreactorObjtype</td>
<td>imagedef reactor obj type</td>
</tr>
<tr>
<td>adImageEnttype</td>
<td>image ent type</td>
</tr>
<tr>
<td>adIncrementObjhandle</td>
<td>increment obj handle</td>
</tr>
<tr>
<td>adNewFile</td>
<td>new file</td>
</tr>
<tr>
<td>adNextClassNum</td>
<td>next class num</td>
</tr>
<tr>
<td>adNumBlockheaders</td>
<td>num block headers</td>
</tr>
<tr>
<td>adNumClasses</td>
<td>num classes</td>
</tr>
<tr>
<td>adNumDimstyles</td>
<td>num dim styles</td>
</tr>
<tr>
<td>adNumEntities</td>
<td>num entities</td>
</tr>
</tbody>
</table>
adNumLayers, 127
adNumLinetypes, 128
adNumObjects, 128
adNumRegapps, 129
adNumsFromFaceEntryList, 129
adNumShapefiles, 129
adNumUcs, 130
adNumValidCodePages, 130
adNumViews, 131
adNumVports, 131
adObjectptrObjtype, 132
adObjhandleValidChars, 132
adOle2frameEnttype, 133
adPlaceholderObjtype, 133
adPrimsHaveColors, 133
adPrimsHaveLayers, 134
adPrimsHaveLinetypes, 135
adPrimsHaveMarkers, 135
adPrimsHaveNormals, 136
adPrimsHaveOrientation, 136
adPrimsHaveVisibilities, 137
adRastervariablesObjtype, 137
adReadAcisString, 137
adReadBlob2Double, 138
adReadBlob3Double, 139
adReadBlobByte, 140
adReadBlobBytes, 140
adReadBlobDouble, 141
adReadBlobLong, 141
adReadBlobObjhandle, 142
adReadBlobShort, 142
adReadBlobString, 143
adReadDicItem, 143
adReadExtendeddata, 144
adReadGrblobData, 145
adReadGrblobVertexPt, 145
adReadGrshellFaceEntries, 146, 220
adReadHeaderBlock, 146
adReadMlstyleSeginfo, 147
adReadMtextBlock, 147
adReadDicItem, 143
adReadProxyDataPoint2d, 158
adReadProxyDataPoint3d, 159
adReadProxyDataScale3d, 159
adReadProxyDataString, 160
adReadProxyDataUInt16, 160
adReadProxyDataUInt32, 161
adReadProxyDataUInt8, 161
adReadProxyDataVector2d, 162
adReadProxyDataVector3d, 163
adReadProxyDataVoid2d, 163
adReadProxyDataVoid3d, 163
adReadProxyObjectObjid, 164
adReadProxyObjectObjid, 164
adReadXrecordItem, 153
adReplaceBlockheader, 165
adReplaceDimstyle, 165
adReplaceEntity, 166
adReplaceLayer, 167
adReplaceLinetype, 167
adReplaceObject, 168
adReplaceRegapp, 169
adReplaceShapefile, 169
adReplaceUcs, 170
adReplaceView, 170
adReplaceVport, 171
adRewindBlob, 171
adRtextEnttype, 172
adSaveFile, 172
adSeekBlockheader, 173
adSeekDimstyle, 173
adSeekEntity, 174
adSeekLayer, 174
adSeekLinetype, 175
adSeekObject, 175
adSeekRegapp, 176
adSeekShapefile, 177
adSeekUcs, 177
adSeekView, 178
adSeekVport, 178
adSetAd2CloseInitFileFn, 179
adSetAd2CriticalErrorFn, 179
adSetAd2LoadometerFn, 182
adSetAd2OpenInitFileFn, 183
adSetAd2ReadInitFileCharFn, 183
adSetAd2SaveometerFn, 184
adSetAd2DxfscanometerFn, 180
adSetAd2ExamineEntityLoadFn, 181
adSetAd2ExamineShapefileFn, 39, 182
adSetAd2LoadometerFn, 182
adSetAd2OpenInitFileFn, 183
adSetAd2ReadInitFileCharFn, 183
adSetAd2ReadInitFileCharsFn, 183
adSetAd2SaveometerFn, 184
adSetAd2WarningFn, 184
adSetCallbackUserdata, 185
adsetDefaultDimstyle, 185
adsetDefaultDimstyleFromHeader, 186
adsetDefaultLayer, 186
adsetDefaultLinetype, 187
adsetDefaultMlstyle, 187
adSetDefaultRegapp, 187
adSetDefaultShapefile, 188
adSetDefaultUcs, 189
adSetDefaultView, 189
adSetDefaultVport, 190
adSetDxfWriteCompleteness, 190
adSetEntHasExtrusion, 251
adSetEntHasNoExtrusion, 251
adSetEntIsInModelspace, 251
adSetEntIsInPaperspace, 251
adSetEntIsInvisible, 251
adSetEntIsNotInvisible, 251
adSetEntityDefaults, 191
adSetHandseed, 191
adSetHeaderDefaults, 192
adSetLayerState, 192
adSetNewFileStartHandle, 193
adSetObjectDefaults, 193
adSetupDwgRead, 194
adSetupDxfRead, 194
adShadowObjhandlePtr, 195
adShortCodePageToAscii, 196
adSortentstableObjtype, 197
adSpatialfilterObjtype, 197
adSpatialindexObjtype, 198
adStartBlobAppend, 198
adStartBlobRead, 199
adStartBlobWrite, 199
adStartBlockheaderGet, 200
adStartClassGet, 200
adStartDimstyleGet, 201
adStartEntityGet, 201
adStartGrblobDataRead, 201
adStartGrblobDataWrite, 202
adStartLayerGet, 203
adStartLinetypeGet, 203
adStartObjectGet, 204
adStartProxyDataRead, 207
adStartProxyDataWrite, 207
adStartRegappGet, 204
adStartShapefileGet, 205
adStartUcsGet, 205
adStartViewGet, 206
adStartVportGet, 206
adStopLoading, 208
adTextBoundingBox, 208
adUnicodeToCodePageChar, 209
adValHdrStr, 209
AdvancedOpenDWG Toolikitfeatures, 35
adVbaProjectObjtype, 210
adWipeoutEnttype, 210
adWipeoutVariablesObjtype, 211
adWriteAcisString, 211
adWriteBlob2Double, 213
adWriteBlob3Double, 213
adWriteBlobByte, 214
adWriteBlobBytes, 214
adWriteBlobDouble, 215
adWriteBlobLong, 216
adWriteBlobObjhandle, 216
adWriteBlobShort, 217
adWriteBlobString, 217
adWriteDictItem, 218
adWriteExtendedData, 218
adWriteGrblobData, 219
adWriteGrblobVertexPt, 220
adWriteHeaderBlock, 221
adWriteMlstyleSeginfo, 221
adWriteMtextBlock, 223
adWriteProxyDataBinary, 224
adWriteProxyDataBool, 224
adWriteProxyDataChar, 225
adWriteProxyDataDouble, 225
adWriteProxyDataInt16, 226
adWriteProxyDataInt32, 226
adWriteProxyDataObjhandle, 227
adWriteProxyDataPoint2d, 227
adWriteProxyDataPoint3d, 228
adWriteProxyDataScale3d, 228
adWriteProxyDataString, 229
adWriteProxyDataTypedObjhandle, 229
adWriteProxyDataUint16, 230
adWriteProxyDataUint32, 230
adWriteProxyDataUint64, 230
adWriteProxyDataVector2d, 231
adWriteProxyDataVector3d, 231
adWriteProxyDataVoidstarData, 232
adWriteProxyEntityObjid, 233
adWriteProxyObjectObjid, 233
adWriteXrecordItem, 223
adXrecordObjtype, 234
Aligning text, 40
Arc, 254
Arcalignedtext, 254
Attdef, 255
Attribute, 256

BLOBs, 26
Block Entity, 256
Block Header, 289
Blocks, 27
Body, 257
Byblock color, 236
Bylayer color, 236
Circle, 257
Class, 28, 290
Colors, 236
Construction line, 257
Creating files. See adNewFile
Critical errors, 37
Data organization, 25
Data reference, 235
DEC Alpha Libraries, 16
Dictionary, 301
Dictionaryvar, 302
Dictionarywdflt, 302
Dimension, 258
Dimension style, 290
Dimstyles, 28
DWG, 19
DXF, 19

Ellipse, 259
Endblock, 261
Entities, 27
Entity
  3D Face, 252
  3D Line, 252
  ACIS data, 253
  arc, 254
  Arcalignedtext, 254
  attdef, 255
  attribute, 256
  block, 256
  body, 257
  circle, 257
  construction line, 257
  dimension, 258
  ellipse, 259
  endblock, 261
  hatch, 261
  image, 263
  insert, 264
  leader, 265
  lightweight polyline, 266
  line, 267
  multi-line, 267
  multi-line text (mtext), 268
  ole2frame, 269
  point, 269
  polyline, 270
  proxy, 270
  ray, 277
  region, 278
  rtext, 278
  seqend, 278
  shape, 278
  solid, 279
  solid3D, 279
  spline, 279
  text, 280
  tolerance, 281
  trace, 282
  vertex, 282
  viewport, 283
  Wipeout, 286
Entity body, 251
Entity data, 250
Entity header, 250
  entcolor, 251
  entflags, 251
  enthandle, 251
  entlayerobjhandle, 251
  entltscale, 251
  enttypeobjhandle, 251
  entthick, 251
  enttype, 250
  extrusion, 251
  linweight, 251
  numreactors, 251
  reactorblog, 251
  userdata, 251
  xdblob, 251
  xdicobjhandle, 251
Error codes, 313
Error handling, 36
  Critical errors, 37
  General, 36
  Warning errors, 37
Example programs, 20
Extended data, 29, 287

File handles, 26
FILE I/O AND MEMORY MANAGEMENT, 35
Frequently asked questions, 325
Function prototypes, 20
Function reference, 53
Functions
  by category, 45

Group, 302
Handles, 26
Hatch, 261
Header, 27, 235
  acadmaintver, 235
  acadver, 235
  angbase, 235
  angdir, 235
  attdia, 235
  attmode, 235
  attreq, 235
  aunits, 235
  auprec, 235
  axismode, 235
  axisunit, 235
  backz, 235
  blipmode, 235
<table>
<thead>
<tr>
<th>Function</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>cecolor, 236</td>
<td></td>
</tr>
<tr>
<td>celscale, 236</td>
<td></td>
</tr>
<tr>
<td>celweight, 236</td>
<td></td>
</tr>
<tr>
<td>cepsntype, 236</td>
<td></td>
</tr>
<tr>
<td>chamfera, 236</td>
<td></td>
</tr>
<tr>
<td>chamferc, 236</td>
<td></td>
</tr>
<tr>
<td>chamferd, 236</td>
<td></td>
</tr>
<tr>
<td>cmiljust, 236</td>
<td></td>
</tr>
<tr>
<td>cmlscale, 236</td>
<td></td>
</tr>
<tr>
<td>coords, 236</td>
<td></td>
</tr>
<tr>
<td>curdimstyleobjhandle, 248</td>
<td></td>
</tr>
<tr>
<td>curdimtextstyleobjhandle, 248</td>
<td></td>
</tr>
<tr>
<td>curltypeobjhandle, 248</td>
<td></td>
</tr>
<tr>
<td>curpucsbaseobjhandle, 249</td>
<td></td>
</tr>
<tr>
<td>curpucsobjhandle, 248</td>
<td></td>
</tr>
<tr>
<td>curpucsorthorefofbjhandle, 249</td>
<td></td>
</tr>
<tr>
<td>curtextstyleobjhandle, 248</td>
<td></td>
</tr>
<tr>
<td>curucbaseobjhandle, 248</td>
<td></td>
</tr>
<tr>
<td>curucobjhandle, 248</td>
<td></td>
</tr>
<tr>
<td>curucsorthorefofbjhandle, 249</td>
<td></td>
</tr>
<tr>
<td>cyclecurr, 236</td>
<td></td>
</tr>
<tr>
<td>cyclekeep, 236</td>
<td></td>
</tr>
<tr>
<td>delobj, 236</td>
<td></td>
</tr>
<tr>
<td>dimadec, 236</td>
<td></td>
</tr>
<tr>
<td>dimalt, 236</td>
<td></td>
</tr>
<tr>
<td>dimaltd, 236</td>
<td></td>
</tr>
<tr>
<td>dimaltf, 236</td>
<td></td>
</tr>
<tr>
<td>dimaltrnd, 237</td>
<td></td>
</tr>
<tr>
<td>dimaltld, 237</td>
<td></td>
</tr>
<tr>
<td>dimalttz, 237</td>
<td></td>
</tr>
<tr>
<td>dimaltu, 237</td>
<td></td>
</tr>
<tr>
<td>dimaltz, 237</td>
<td></td>
</tr>
<tr>
<td>dimapost, 237</td>
<td></td>
</tr>
<tr>
<td>dimaso, 237</td>
<td></td>
</tr>
<tr>
<td>dimasz, 237</td>
<td></td>
</tr>
<tr>
<td>dimaunit, 237</td>
<td></td>
</tr>
<tr>
<td>dimazin, 237</td>
<td></td>
</tr>
<tr>
<td>dimblk, 237</td>
<td></td>
</tr>
<tr>
<td>dimblk1objhandle, 237</td>
<td></td>
</tr>
<tr>
<td>dimblk2objhandle, 237</td>
<td></td>
</tr>
<tr>
<td>dimblkobjhandle, 237</td>
<td></td>
</tr>
<tr>
<td>dimcen, 237</td>
<td></td>
</tr>
<tr>
<td>dimclrd, 237</td>
<td></td>
</tr>
<tr>
<td>dimclre, 238</td>
<td></td>
</tr>
<tr>
<td>dimclrt, 238</td>
<td></td>
</tr>
<tr>
<td>dimdec, 238</td>
<td></td>
</tr>
<tr>
<td>dimdle, 238</td>
<td></td>
</tr>
<tr>
<td>dimdli, 238</td>
<td></td>
</tr>
<tr>
<td>dimdsep, 238</td>
<td></td>
</tr>
<tr>
<td>dimexe, 238</td>
<td></td>
</tr>
<tr>
<td>dimexo, 238</td>
<td></td>
</tr>
<tr>
<td>dimfit, 239</td>
<td></td>
</tr>
<tr>
<td>dimfrac, 239</td>
<td></td>
</tr>
<tr>
<td>dimgap, 239</td>
<td></td>
</tr>
<tr>
<td>dimjust, 239</td>
<td></td>
</tr>
<tr>
<td>dimldrlblkobjhandle, 239</td>
<td></td>
</tr>
<tr>
<td>dimlfrac, 239</td>
<td></td>
</tr>
<tr>
<td>dimlim, 239</td>
<td></td>
</tr>
<tr>
<td>dimlw2, 239</td>
<td></td>
</tr>
<tr>
<td>dimlw3, 239</td>
<td></td>
</tr>
<tr>
<td>dimlwd, 239</td>
<td></td>
</tr>
<tr>
<td>dimpost, 239</td>
<td></td>
</tr>
<tr>
<td>dimprnd, 239</td>
<td></td>
</tr>
<tr>
<td>dimsah, 239</td>
<td></td>
</tr>
<tr>
<td>dimscale, 240</td>
<td></td>
</tr>
<tr>
<td>dimsdl1, 240</td>
<td></td>
</tr>
<tr>
<td>dimsdl2, 240</td>
<td></td>
</tr>
<tr>
<td>dimse1, 240</td>
<td></td>
</tr>
<tr>
<td>dims2, 240</td>
<td></td>
</tr>
<tr>
<td>dimsho, 240</td>
<td></td>
</tr>
<tr>
<td>dimsoxd, 240</td>
<td></td>
</tr>
<tr>
<td>dimtad, 240</td>
<td></td>
</tr>
<tr>
<td>dimtdec, 240</td>
<td></td>
</tr>
<tr>
<td>dimtfac, 240</td>
<td></td>
</tr>
<tr>
<td>dimth, 240</td>
<td></td>
</tr>
<tr>
<td>dimtix, 240</td>
<td></td>
</tr>
<tr>
<td>dimtm, 240</td>
<td></td>
</tr>
<tr>
<td>dimtmmove, 240</td>
<td></td>
</tr>
<tr>
<td>dimtof, 240</td>
<td></td>
</tr>
<tr>
<td>dimtoh, 240</td>
<td></td>
</tr>
<tr>
<td>dimtol, 240</td>
<td></td>
</tr>
<tr>
<td>dimtolj, 240</td>
<td></td>
</tr>
<tr>
<td>dimtp, 240</td>
<td></td>
</tr>
<tr>
<td>dimtsz, 240</td>
<td></td>
</tr>
<tr>
<td>dimtvp, 241</td>
<td></td>
</tr>
<tr>
<td>dimtxt, 241</td>
<td></td>
</tr>
<tr>
<td>dimtzn, 241</td>
<td></td>
</tr>
<tr>
<td>dimunit, 241</td>
<td></td>
</tr>
<tr>
<td>dimupt, 241</td>
<td></td>
</tr>
<tr>
<td>dimzin, 241</td>
<td></td>
</tr>
<tr>
<td>dispplh, 241</td>
<td></td>
</tr>
<tr>
<td>dragmode, 241</td>
<td></td>
</tr>
<tr>
<td>dwgcodepage, 242</td>
<td></td>
</tr>
<tr>
<td>elevation, 242</td>
<td></td>
</tr>
<tr>
<td>endcaps, 242</td>
<td></td>
</tr>
<tr>
<td>extmax, 242</td>
<td></td>
</tr>
<tr>
<td>extmin, 242</td>
<td></td>
</tr>
<tr>
<td>extnames, 242</td>
<td></td>
</tr>
<tr>
<td>factres, 242</td>
<td></td>
</tr>
<tr>
<td>fastzoom, 242</td>
<td></td>
</tr>
<tr>
<td>filletrad, 242</td>
<td></td>
</tr>
<tr>
<td>fillmode, 242</td>
<td></td>
</tr>
<tr>
<td>fingerprintguid, 242</td>
<td></td>
</tr>
<tr>
<td>flatland, 242</td>
<td></td>
</tr>
<tr>
<td>frontz, 242</td>
<td></td>
</tr>
<tr>
<td>gridmode, 243</td>
<td></td>
</tr>
<tr>
<td>gridunit, 243</td>
<td></td>
</tr>
<tr>
<td>handles, 243</td>
<td></td>
</tr>
<tr>
<td>handseed, 243</td>
<td></td>
</tr>
<tr>
<td>hrdoub, 243</td>
<td></td>
</tr>
</tbody>
</table>
Imagedef, 304
Imagedefreactor, 305
Include files, 20
INITIALIZATION DATA, 21
INSERT, 264
Layer, 292
AYER_INDEX, 305
Layers, 28
Layout object, 305
LEADER, 265
Libraries, 15
Lightweight polyline, 266
Line, 267
Linetype, 28, 293
Linked lists, 26
Load File. See adLoadFile
Load, aborting, 39
Load/save, monitoring, 39

Macintosh Libraries, 16
Modifying entities. See adReplaceEntity
Multi line style, 306
Multi-line, 267
Multi-line text (mtext), 268

Object body, 301
Object handles, 26
Object header, 300
OBJECTPTR, 307
Objects, 29
Objects Section, 299
Ole2frame, 269

Placeholder object, 308
Plotsettings object, 308
Point, 269
Polyline, 270
Positioning - text, 40
Prototypes, 31
Proxy, 270
Proxy object, 309

Rastervariables object, 310
Ray, 277
Reading data. See adReadxxxxFunctions
Region, 278
Registered application, 29, 294
Rtext, 278

Saving Files. See adSaveFile
secondtextloc, 40
Seqend, 278
Shape, 278
Shape file callback, 39

Shapefile, 28, 294
Solid, 279
Solid3D, 279
SORTENTSTABLE, 310
Spatialfilter object, 311
SPATIALINDEX, 311
Spline, 279
Stack usage, 21
Structure alignment issues, 21
Supported environments, 19

Table, 27, 289
class, 290
classes, 28
dimension style, 290
dimstyles, 28
extended data, 29
group, 302
layer, 292
layers, 28
linetype, 28, 293
multi-line style, 306
objects, 29
proxy object, 309
registered application, 29, 294
shapefile, 28, 294
user coordinate system, 28, 295
view, 28, 297
vport, 28, 298

Taxonomy of OpenDWG Toolkit functions, 45
Text, 280
Text alignment, 40
Tolerance, 281
Trace, 282
TROUBLESHOOTING TIPS, 23
Typed object handle, 26

UCS, 28, 295
Unicode, 41
Unix Libraries, 16
User coordinate system, 295
User Coordinate System, 28

Vbaproject object, 311
Vertex, 282
Vision dependencies, 51
View, 28, 297
Viewport, 283
Vport, 28, 298

Warning errors, 37
Windows Libraries, 15
Wipeout, 286
Wipeoutvariables object, 312
Xrecord object, 312

Zombie. See Proxy