AutoCAD Map 3D 2009 AutoLISP Reference

AutoCAD Map 3D AutoLISP functions cover data extension (drawing sets, queries, object data... the "ADE" functionality), map plotting, and topology.

For broader coverage of AutoCAD Map 3D functionality, use the .NET API, which is included in the AutoCAD Map 3D SDK. The SDK is a separate installation from AutoCAD Map 3D itself. You can download it from the Autodesk web site.

Note The AutoLISP API remains available to support legacy applications that use it. For all new development, the .NET API is the best choice.

To download the AutoCAD Map 3D SDK

- 1. Open the <u>Autodesk Home Page</u> (it opens in a new window) and click Search.
- 2. Search for "ObjectARX for Autodesk Map 3D".
- 3. Click "Developer Center ObjectARX for Autodesk Map 3D".
- 4. Click "License And Download".

Three Functional Groups

AutoCAD Map AutoLISP comprises three functional groups:

Data Extension Functions Plotting Functions Topology Functions

Before You Begin

To develop AutoLISP applications for AutoCAD Map, you should be familiar with AutoCAD, the AutoCAD Map user interface, and the AutoLISP API for AutoCAD.

Related Topics

Using AutoLISP Functions Numeric Identifiers Typographic Conventions

What's New

No new features or enhancements have been added to AutoLISP for AutoCAD Map 3D 2008.

Data Extension Function Catalog

Function Synopsis

Data Extension functions listed alphabetically.

alias | altp | ds | dwg | edit | ent | err | expr | key | od os | pref | proj | ql | qry | rt | save | sql | ss | user | ver ade aliasadd ade aliasdelete

ade_aliasgetlist ade_aliasupdate

ade_altpclear ade_altpdefine ade_altpdelprop ade_altpgetprop ade_altplist ade_altpsetprop

ade_dsattach ade_dsdetach ade_dsisnested ade_dslist ade_dsproplist

ade_dwgactivate ade_dwgactualpath ade_dwgaliaspath ade_dwgattriblist ade_dwgdeactivate ade_dwggetid ade_dwggetsetting ade dwghaslocks ade dwgindex ade dwgindexdef ade dwgisactive ade dwgistoplevel ade dwgproplist ade dwgquickview ade dwgselectdlg ade dwgsetof ade dwgsetsetting ade dwgunlock ade dwgzoomextents ade editdefcen ade editlockederased ade editislocked ade editlocked ade_editlockobjs ade editnew ade_editunlockobjs ade entsetlocation ade errclear ade errcode ade_errgetlevel ade_errmsg ade_errpush ade_errpushstatement ade_errqty ade errsetlevel ade_errshowdlg ade errstatement ade_errtype ade_expreval ade keycolumnlist ade odaddfield

ade odaddrecord ade odattachrecord ade oddefinetab ade oddeletefield ade oddeletetab ade oddelrecord ade odfreerec ade odgetfield ade odgetrecfield ade odgetrecord ade odgettables ade odmodifyfield ade odmodifytab ade_odnewrecord ade_odpresetfield ade_odrecordqty ade odsetfield ade odtabledefn ade odtablelist ade_osfexpand ade prefgetval ade prefsetval ade_projgetctgyname ade_projgetinfo ade_projgetwscode ade_projlistcrdsysts ade_projlistctgy ade_projptbackward ade projptforward ade projsetdest ade projsetsrc ade projsetwscode ade_qldelctgy

<u>ade_qldelquery</u> <u>ade_qlgetctgyinfo</u> ade_qlgetqryinfo ade_qllistctgy ade_qlloadqry ade_qlqrygetid ade_qlsetctgyname ade_qlsetquery ade gryclear ade grydefine ade gryexecute ade grygetcond ade_qrygetdwgandhandle ade grygetentlist ade grygetreptransform ade_qrygroup ade_qryhandent ade_qrylist ade_qrysave ade_qrysetaltprop ade_qrysetcond ade_qrysetreptransform ade_qrysettype ade_qryungroup ade rtdefrange ade rtdeltable ade rtgetid ade_rtgetprop ade rtlist ade saveobjs ade savetodwg ade_sqlgetenvstring ade ssfree ade_userget ade_usergetrights ade userlist

<u>ade_userset</u> <u>ade_usersetrights</u>

ade_version

Data Extension Function Synopsis

Function Catalog

Data Extension functions listed by functional group.

Coordinate Transformation Functions

Drawing Functions

Drawing Set Functions

Drive Alias Functions

Error Message Functions

Expression Evaluation Function

Object Data Functions

Object Editing Functions

Object Saving Functions

Option Functions

Property Alteration Functions

Query Functions

Query Library Functions

Range Table Functions

SQL Environment Functions

Text Label Function

User Security Functions

Other Functions

Plotting Function Catalog

Function Synopsis

Plotting functions listed alphabetically.

<u>dwg</u> | <u>plt</u> | <u>topo</u>

map_dwgbreakobj map_dwgtrimobj

map_pltblkatts map_pltblklist map_pltblkvps map_pltcleanup map_pltcurrdef map_pltcurrdel map_pltcurrget map_pltcurrsave map_pltcurrset map_pltdefdelete map_pltdefget map_pltdeflist map_pltdefread map_pltdefsave map_pltdefvalid map_pltdefverify map_pltdisplay map_pltexecute map_pltinit map_pltplot map pltrestore

map_topoaudit
map_topoclose
map_topocomplete
map_topostat

Plotting Function Synopsis

Function Catalog

Plotting functions listed by functional group.

Boundary Functions
Plotting Functions
Topology Functions

Topology Function Catalog

Function Synopsis

Topology functions listed alphabetically.

ac | ana | clean | edit | elem | info | iter | mnt | qry | trace | var

tpm acclose tpm acexist tpm_acload tpm_acopen tpm_acqty tpm_acunload tpm_acupgradeopen tpm_anabuffer tpm_anadissolve tpm_anaoverlay tpm_cleanactionlistdel tpm_cleanactionlistgetat tpm_cleanactionlistins tpm_cleanactionlistqty tpm_cleanalloc tpm_cleananchorss tpm_cleancancel tpm_cleancomplete tpm_cleancreatedss tpm_cleanend tpm_cleanerrorcur tpm_cleanerrordraw tpm cleanerrorfix

tpm_cleanerrorget tpm_cleanerrormark tpm cleanerrorset tpm cleanfree tpm_cleangroupdraw tpm_cleangroupfix tpm cleangroupmark tpm cleangroupnext tpm cleangroupqty tpm_cleangroupsubtype tpm_cleangrouptype tpm_cleanincludess tpm cleaninit tpm_cleaninitanchorset tpm_cleanmodifiedss tpm_cleanprofileload tpm_cleanprofilesave tpm_cleanstart tpm cleanunchangedss tpm editaddelem tpm editdelelem tpm_editmodelem tpm_editupdelem tpm elemadj

tpm_elemfind tpm_elemget tpm_elemid tpm_elemqty tpm_elemss

tpm_infobuildvar tpm_infocomplete tpm_infocorrect tpm_infocurrent tpm_infodesc tpm_infomodified tpm_infoname tpm_infostatus tpm_infotype tpm_infoversion tpm_iterdesc tpm_itername tpm_iternext tpm_iterstart tpm_iterstop tpm_itertype tpm_iterversion tpm_mntbuild tpm_mnterase tpm_mntrebuild tpm_mntrename tpm_qrygetresdesc tpm_qrygetrestopo tpm_qrygettoponame tpm_qrysetrestopo tpm_qrysettoponame tpm_tracealloc tpm_tracebestroute tpm_tracebestroutescan tpm_tracebestrouteval tpm_traceelemedit tpm_traceelemget tpm_traceelemid tpm_traceflood tpm_tracefree tpm_traceqty tpm_tracesetmaxres tpm_tracesetminres tpm_traceshort tpm_traceshortscan

tpm_traceshortval

tpm_varalloc

tpm_varfree tpm_varget tpm_varlist tpm_varset

Topology Function Synopsis

Function Catalog

Topology functions listed by functional group.

Access Functions Analyzing Functions Building and Erasing Functions Cleanup Functions Editing Functions Element Information Functions Iterating Functions Network Tracing Functions Topology Information Functions Topology Query Functions

Configuration Variables

Configuration variables include three subsets:

<u>Cleanup variables</u>, properties for cleanup models.

<u>Cleanup action variables</u>, properties for cleanup actions.

Topology variables, properties for topologies.

To allocate a set of configuration variables, regardless of which subset you will be using, use <u>tpm_varalloc</u>, which returns an <u>ade_id</u> for the set that it creates.

To free a set of configuration variables, use <u>tpm_varfree</u>.

To get the value of a configuration variable, use <u>tpm_varget</u>.

To set the value of a configuration variable, use <u>tpm_varset</u>.

To get the properties of a given cleanup action, first allocate a new set of variables, and then use <u>tpm_cleanactionlistgetat</u>.

To get the properties of a given topology, first allocate a new set of variables, and then use <u>tpm_infobuildvar</u>.

To list all the values of a given set of variables, use <u>tpm_varlist</u>.

Topology variables store properties for topologies, which are created by <u>tpm_mntbuild</u> and the topology analysys functions, <u>tpm_ana[xx]</u>.

Topology variables are a subset of the <u>configuration variables</u> data structure.

BUILD_TOL	Tolerance for topology building, buffering, and other processes (positive real). Default = 0.01
CNTR_COLOR	Color for new centroids, an AutoCAD color index (integer). Negative integer = Current color Default = -1 See <u>Color Index Colors</u> .
CNTR_LAYER	Layer name for new centroids (string). nil = Current layer Default = ''''
CNTR_TYPE	Entity type of new centroids (integer). 1 = Point (default)

	2 = Block
CNTR_BLOCK	Block name for new centroids (string). Default = ""
CREATE_CNTR	Flag for centroids generation (integer). 1 = Generate (default) 0 = Do not generate
CREATE_MARKERS	<pre>Whether errors should be marked with persistent markers as drawn by tpm_cleanerrormark (integer). 0 = Do not mark (default) 1 = Mark Marker shapes are set by MARKER_HEIGHT and the [XX]_MARKER cleanup variables.</pre>
CREATE_NODE	Flag for nodes generation (integer). 1 = Generate (default) 0 = Do not generate
CREATE_VIEW	Whether errors should be marked with temporary markers as drawn by <u>tpm_cleanerrordraw</u> 0 = Do not mark 1 = Mark (default)
DEF_OFFSET	Default offset (<mark>real</mark>).

	Use when offset value cannot be calculated for an object, for example, if the object lacks necessary <u>object data</u> . Default = 1.0
DUPLICATE_CENTROID_COLOR	Marker color for duplicate centroid errors, an AutoCAD color index (integer). Default = 1 (red) See <u>Color Index Colors</u> .
DUPLICATE_CENTROID_MARKER	Marker shape for duplicate centroid errors (integer). See <u>Marker Shapes</u> . Default = 4 (square)
IGNORE_INCOMPLETE_AREA	What to do if links are encountered that do not belong to any polygon (integer). 1 = Ignore them 0 = Cancel topology creation process (default)
INCOMPLETE_AREA_COLOR	Marker color for incomplete area errors, an AutoCAD color index (integer). Default = 2 (yellow) See <u>Color Index Colors</u> .
INCOMPLETE_AREA_MARKER	Marker shape for incomplete area errors (integer). See <u>Marker Shapes</u> . Default = 2 (triangle)
INTERSECTION_COLOR	Marker color for intersection

	errors, an AutoCAD color index (<mark>integer</mark>). Default = 3 (green) See <u>Color Index Colors</u> .
INTERSECTION_MARKER	Marker shape for intersection errors (integer). See <u>Marker</u> <u>Shapes</u> . Default = 1 (octagon)
MISSING_CENTROID_COLOR	Marker color for missing centroid errors, an AutoCAD color index (integer). Default = 4 (cyan) See <u>Color Index Colors</u> .
MISSING_CENTROID_MARKER	Marker shape for missing centroid errors (integer). See <u>Marker Shapes</u> . Default = 3 (rhombus)
NODE_BLOCK	Block name for new nodes (string). Default = ""
NODE_COLOR	Color for new nodes, an AutoCAD color index (integer): Negative integer = The current color Default = -1 See <u>Color Index Colors</u> .
NODE_LAYER	Layer name for new nodes (string): nil The current layer Default = ""

NODE_TYPE	Entity type of new nodes (integer): 1 Point (default) 2 Block
STOP_AT_MISSING_CNTR	What to do if a polygon has no centroid (integer): 0 Create a centroid (default) 1 Cancel topology creation process
STOP_AT_MULTIPLE_CNTR	 What to do if a polygon has more than one centroid (integer): 0 Designate one centroid 1 Cancel topology creation process (default)

Cleanup Variables

Cleanup variables store properties for cleanup models.

Cleanup variables are a subset of the <u>configuration variables</u> data structure.

Note Data types cited below are AutoLISP types. The AutoLISP real corresponds to the C++ double. Similarly, integer corresponds to long.

ANCHOROBJS_FEATURES	<pre>!! New !! Classified objects to include, a comma-separated list of feature names (string), such as "Roads, Rivers, Streets". Default = "*" (all features).</pre>
ANCHOROBJS_LAYERS	Anchored entities specified by tpm_cleaninitanchorset will be anchored only if they reside on layers specified here, with multiple layer names separated by commas. For example, "0,Layer1,Layer2". Default = "*" (any layer)

APPARENT_INTERSECTION_COLOR	<pre>!! New !! Color for apparent-intersection markers, an AutoCAD color index (integer). See <u>Color Index Colors</u>. Default = 7 (white)</pre>
APPARENT_INTERSECTION_MARKER	<pre>!! New !! Marker shape for apparent-intersection errors (integer). See Marker Shapes. Default = 2 (triangle)</pre>
ARC_TYPE	Convert arcs to (integer): 0 = Arc (default) 1 = 2D polyline
CIRCLE_TYPE	Convert circles to (integer): 0 = Circle (default) 1 = 2D polyline 2 = Arc
CLEAN_TOL	Cleanup tolerance (positive <mark>real</mark>). Default = 0.01
CLUSTER_COLOR	Color for cluster markers, an AutoCAD color index (integer). See <u>Color Index</u> <u>Colors</u> . Default = 5 (blue)
CLUSTER_MARKER	Marker shape for cluster errors (integer). See <u>Marker Shapes</u> . Default = 4 (square)

CONVERT	Flag for the original entities conversion (integer). 0 = Do not convert (default) 1 = Convert
CORRIDOR	Corridor width (positive real). Default = 0.01 Note If you are using an explicit cleanup action list (specifying cleanup actions using tpm_cleanactionlistins) this setting is ignored.
CROSS_COLOR	Color for cross markers, ar AutoCAD color index (integer). See <u>Color Index</u> <u>Colors</u> . Default = 2 (yellow)
CROSS_MARKER	Marker shape for cross errors (integer). See <u>Marker Shapes</u> . Default = 1 (octagon)
DANGL_COLOR	Color for dangling node markers, an AutoCAD color index (integer). See <u>Color Index Colors</u> . Default = 1 (red)
DANGL_MARKER	Marker shape for dangling node errors (integer). See <u>Marker Shapes</u> .

	Default = 1 (octagon)
DEL_MARKER	Whether to delete previous error markers (integer). 0 = Do not delete 1 = Delete (default)
DUPL_COLOR	Color for duplicates markers, an AutoCAD color index (integer). See <u>Color Index Colors</u> . Default = 6 (magenta)
DUPL_MARKER	Marker shape for duplicate errors (<mark>integer</mark>). See <u>Marker Shapes</u> . Default = 1 (octagon)
ENT_PROCESS	 Flag for original entities processing (integer). 1 = Convert original 2 = Create new and keep original 3 = Create new and delete original Default = 0
GENERALIZE	Flag for generalization (link simplication) (integer). 0 = Do not generalize (default) 1 = Generalize Note If you are using an explicit cleanup action list (specifying cleanup actions

	using <u>tpm_cleanactionlistins</u>) this setting is ignored.
INCLUDE_LINEAROBJS	 !! New !! Whether to include linear objects while deleting duplicates (integer). 0 = Do not include. 1 = Include. Default = 1.
INCLUDE_POINTS	 !! New !! Whether to include points while deleting duplicates or snapping clustered nodes (integer). 0 = Do not include. 1 = Include. Default = 1.
INCLUDE_BLOCKS	 !! New !! Whether to include blocks while deleting duplicates or snapping clustered nodes (integer). 0 = Do not include. 1 = Include. Default = 0.
INCLUDE_TEXT	 !! New !! Whether to include text while deleting duplicates (integer). 0 = Do not include. 1 = Include. Default = 0.

INCLUDE_MTEXT	 !! New !! Whether to include mtext while deleting duplicates (integer). 0 = Do not include. 1 = Include. Default = 1.
INCLUDE_ROTATION	<pre>!! New !! Whether to include rotation while deleting duplicates (integer). 0 = Do not include. 1 = Include. Default = 0.</pre>
INCLUDE_ZVALUES	 !! New !! Whether to include z-values while deleting duplicates (integer). 0 = Do not include. 1 = Include. Default = 0.
SNAP_TO_NODE	 !! New !! Whether to snap to nodes or links (integer). 0 = Snap to nodes. 1 = Snap to links. Default = 0.
INCLUDEOBJS_AUTOSELECT	How entities are specified for cleanup 0 = Select entities manually (that is, by passing a

	selection set to <u>tpm_cleaninit</u>) 1 = Select all entities in the drawing (in which case the selection set passed to <u>tpm_cleaninit</u> , if any, is ignored)
INCLUDEOBJS_FEATURES	<pre>!! New !! Classified objects to include, a comma-separated list of features (string), such as "Roads, Rivers, Streets". Default = "*" (all features).</pre>
INCLUDEOBJS_LAYERS	Entities specified by tpm_cleaninit will be cleaned only if they reside on layers specified here, with multiple layer names separated by commas. For example, "0,Layer1,Layer2". Default = "*" (any layer)
LINE_TYPE	Convert lines to (integer) 0 = Line (default) 1 = 2D polyline
LINK_COLOR	Color for new links, an AutoCAD color index (integer). See <u>Color Index</u> <u>Colors</u> . Negative integer = Current color

	Default = -1
LINK_CORRECT	Flag for link error correction (integer). 0 = Do not correct 1 = Correct (default) Note If you are using an explicit cleanup action list (specifying cleanup actions using tpm_cleanactionlistins) this setting is ignored.
LINK_ERROR	Link error types (integer), sum of the desired options. 0 = None 1 = Short 2 = Cross 4 = Undershoot 8 = Duplicate Default = 15
LINK_LAYER	Layer name for new links (string). nil = Current layer Default = ""
MAINTAIN_MARKERS	Flag to maintain markers when cleanup ends 0 = Do not maintain 1 = Maintain markers This variable affects only errors that have not been fixed. If an error has been fixed, its marker is deleted at the end of the cleanup

	process no matter what (by tpm_cleanend).
MARKER_HEIGHT	The height of error markers, a percent of screen height (positive real). Default = 5.0
NODE_CORRECT	Flag for node error correction (integer). 0 = Do not correct 1 = Correct (default) Note If you are using an explicit cleanup action list (specifying cleanup actions using tpm_cleanactionlistins) this setting is ignored.
NODE_ERROR	Node error types (integer), sum of the desired options. 0 = None 16 = Cluster 32 = Pseudo 64 = Dangling Default = 48
POLY3D_TYPE	Convert 3D polylines to (integer). 0 = 3D polyline (default) 1 = 2D polyline
PSEUDO_COLOR	Color for pseudo node markers, an AutoCAD

	color index (<mark>integer</mark>). See <u>Color Index Colors</u> . Default = <mark>6</mark> (magenta)
PSEUDO_MARKER	Marker shape for pseudo node errors (<mark>integer</mark>). See <u>Marker Shapes</u> . Default = 2 (triangle)
SHORT_COLOR	Color for short markers, ar AutoCAD color index (positive integer). See <u>Color Index Colors</u> . Default = 1 (red)
SHORT_MARKER	Marker shape for short errors (<mark>integer</mark>). See <u>Marker Shapes</u> . Default = 1 (octagon)
UNDER_COLOR	Color for undershoot markers, an AutoCAD color index (integer). See <u>Color Index Colors</u> . Default = <mark>3</mark> (green)
UNDER_MARKER	Marker shape for undershoot errors (integer). See <u>Marker</u> <u>Shapes</u> . Default = 1 (octagon)
WEED_DISTANCE	<pre>!! New !! Weeding distance (real). Default = 15.0</pre>
WEED_ANGLE	!! New !! Weed distance

	(real). Default = 4.0
WEED_SUPPLEMENT_DISTANCE	<pre>!! New !! Weed supplement distance (real). Default = 100.0</pre>
WEED_SUPPLEMENT_BULGE	<pre>!! New !! Weed supplement bulge (real). Default = 1.0</pre>

Cleanup action variables store properties for cleanup actions, which are assigned to action lists by <u>tpm_cleanactionlistins</u>.

Action lists are a feature of cleanup models. Cleanup action variables are a subset of the <u>configuration variables</u> data structure.

ARC_TYPE	With the Simplify Objects cleanup action (clean group type 128) <i>only</i> , whether to create arcs (integer). 1 = Create arcs (default) 0 = Do not create arcs For any action other than 128, Simplify Objects, ARC_TYPE does not matter.
CLEAN_TOL	Cleanup tolerance (positive real). Default = 0.01 CLEAN_TOL affects the following actions only: 8 = Delete Duplicates 1 = Erase Short Objects 4 = Extend Undershoots 16 = Snap Clustered Nodes 64 = Erase Dangling Objects 128 = Simplify Objects With the following actions, the CLEAN_TOL value does not matter: 2 = Break Crossing Objects
	32 = Dissolve Pseudo Nodes 256 = Zero Length Objects
---------	--
CONVERT	With the Extend Undershoots cleanup action (clean group type4) <i>only</i> , whether to break the target object (insert a new node) where the extended undershoot intersects it (integer). 1 = Break target (default) 0 = Do not break target For any action other than 4, Extend Undershoots, CONVERT does not matter.

Error Codes

The following errors codes are native to AutoCAD Map.

Custom applications can define their own error codes. See <u>ade_errpush</u>.

You can get an error's error code by passing its stack index (0 = first error) to <u>ade_errcode</u>.

Error Code Index				
€15	<u>1200</u>	<u>2000</u>	<u>2500</u>	<u>3000</u>
<u>0</u>	<u>1300</u>	<u>2100</u>	<u>2600</u>	<u>3100</u>
<u>200</u>	<u>1500</u>	<u>2200</u>	<u>2700</u>	<u>3200</u>
<u>1000</u>	<u>1800</u>	<u>2300</u>	<u>2800</u>	<u>4000</u>
<u>1100</u>	<u>1900</u>	<u>2400</u>	<u>2900</u>	

Expression

1 5	kErrXEDValueFail	Internal error
1 4	kErrRngTabEvalFail	Internal error
1 3	kErrNoRngTabFound	Internal error
1 2	kErrNoRngTabLibFound	Internal error
� 11	kErrLispValueFail	Internal error

1 0	kErrGetAttrFail	Internal error
1 09	kErrGetSQLFail	Internal error
\$ 08	kErrGetEEDFail	Internal error
1 07	kErrInvalidProperty	Internal error
\$ 06	kErrExpInvalidOperand	Internal error
1 05	kErrExpADS	Internal error
1 04	kErrExpNoMemforOperand	Internal error
1 03	kErrExpMathOverFlow	Internal error
1 02	kErrExpInvalidOper	Internal error
@ 01	kErrExpSyntaxErr	Internal error

For more Expression errors, which begin at 2900, click ■.

Common Usage

00	kAdeOk	General ADE return value: call to ADE object is successful
01	kAdeErr	General ADE error: call to ADE object is not successful
02	kAdeBadInput	Invalid function argument
03	kAdeObjectNotFound	
04	kAdeOutOfMemory	
05	kAdeObjNotInitialized	
06	kAdeWrongType	
07	kAdeWrongProject	

08 kAdeEO	В
-----------	---

09 kAdeADSError

Invalid function argument type

- 10 kAdeAdsNameConvertionFails
- 11 kAdeWrongArgument
- 12 kAdeWriteBufFails
- 13 kAdeReadBufFails
- 14 kAdeXDataCorrupted
- 15 kAdeNoEnvironment
- 16 kAdeUsrBreak
- 17 kAdeUncomparable

18	kAdePermissionDenied	No permission to perform some action. When using the following commands, the message has different meanings: ADESAVEOBJS = Can't be executed in demonstration mode; ADEDRAWINGS = User has no rights to update the set; ADESELOBJS, ADDREMOBJS = User has no rights to edit; ADEDWGMAINT = User needs to be superuser to remove the foreign locks; MAPOPTIONS: System Preferences = Can't set "Force User Login" flag and change "Object Locking" flag if drawing set contains active or locked drawings. Deactivate or unlock the drawings before trying to set these preferences.
----	----------------------	--

19 kAdeWrongSyntax

20	kAdeDuplicate	
21	kAdeInvalidPathOrFileName	
22	kAdeInvalidVersion	
23	kAdeFileIOFatalError	

External Subsystems

200	kAdeAcDbError	
201	kAdeIRDNotInitialized	
202	kAdeIRDError	
203	kAdeASENotInitialized	
204	kAdeASEError	
205	kAdeASIError	
206	kAdeAsiNotInitialized	
207	kAdeAsiConnectToEnvironmentFailed	

Transactional Manager

1000	kErrClosed	Repeated attempts to close previously closed ADE object. Call support.
1001	kErrWasErased	Attempt to work with erased ADE object. Call support.
1002	kErrOpenForRead	Attempt to update ADE objects opened for read. Call support.
1003	kErrOpenForWrite	Attempts to get multiple access to ADE object opened for write. Call support.
1004	kErrWrongMode	Incorrect mode of the ADE objects should be

		opened. Call support.
1005	kErrClone	Exception at the time of cloning ADE objects. Call support.
1006	kErrResponse	Incorrect attempt to start ADE transaction. Call support.
1007	kErrObjIsAbsent	Attempt to work with erased ADE object. Call support.
1008	kErrAccess	Type of work with ADE does not correspond to its status. Call support.
1009	kErrMultipleUsage	Attempts to get multiple access to ADE object opened for write. Call support.
1010	kErrUpgrade	Incorrect attempt to update opening mode of the ADE object. Call support.
1011	kErrNotClosed	Object was not closed during current ADE transaction. Call support.

CAdeList

1100	kAdeGetIteratorFails	
1101	kAdeListIsEmpty	

CAdeListIterator

1102	kErrListEnd	
1103	kErrListObjectIsAbsent	

Drawing

1200	kAdeQueriedEntity	Ignoring queried entity for
		saving selection. When using

		ADESAVEOBJS, the message means queried objects that are selected for save back are ignored.
1201	kAdeDwgNotActive	Attempt to read the object locked from an inactive drawing. When using ADEWHOHASIT, the message means the drawing from which the object was queried is no longer active. ADE is unable to determine if the object selected is currently locked. Activate the specified drawing and re- enter the command.
1202	kAdeReadDwgFileFails	ADE fails to read external drawing. If using ADEDRAWINGS, this message means that ADE is unable to read the specified drawing. Perhaps the drawing doesn't exit or the specified file is not a valid AutoCAD Drawing File. Check the error message stack for more information.
1203	kAdeMultipleUsers	ADE fails to access external drawing. If using ADEDRAWINGS, this message means ADE is unable to access the specified drawing because of file locking problems. Check the error message stack for more information.

1204	kAdeEntityLockingFails	ADE fails to lock. If using ADESELOBJS, this message means ADE is unable to lock an object. The object may already be locked. Use the ADEWHOHASIT to determine if the object is locked, perhaps by another user. Check the error message stack for more information.
1205	kAdeUnlockedEntity	Entity is unlocked. If using ADEWHOHASIT, this message indicates that the selected object is not currently locked.
1206	kAdeLockedEntity	Entity has been locked by another ADE user. If using ADESELOBJS, this message indicates that the object is already locked by another user. Use the ADEWHOHASIT command to identify the user.
1207	kAdeAlreadyInSaveSet	
1208	kAdeAlreadyNotInSaveSet	
1209	kAdeMultipleReaders	
1210	kAdeOpenPrefDictionaryFails	ADE fails to open ADE preferences dictionary. Call support.
1211	kAdeSavePrefDictionaryFails	ADE fails to save ADE preferences dictionary. Call support.

1212	kAdeRestoreDSetFails	ADE fails to restore the drawing set. Call support.
1213	kAdeOpenDSetDictionaryFails	ADE fails to open ADE drawing set dictionary. Call support.
1214	kAdeSaveDSetDictionaryFails	ADE fails to save ADE drawing set in dictionary. Call support.
1215	kAdeOpenQueryDictionaryFails	ADE fails to open ADE query library dictionary. Call support.
1216	kAdeSaveQueryDictionaryFails	ADE fails to save ADE query library in dictionary. Call support.
1217	kAdeOpenRTableDictionaryFails	ADE fails to open ADE range table dictionary. Call support.
1218	kAdeSaveRTableDictionaryFails	ADE fails to save ADE range table in dictionary. Call support.
1219	kAdeRestoreRTableDictionaryFails	ADE fails to restore ADE range table in dictionary. Call support.
1220	kAdeOpenDocViewDictionaryFails	ADE fails to open ADE Doc View information dictionary. Call support.
1221	kAdeSaveDocViewDictionaryFails	ADE fails to save ADE Doc View information in dictionary. Call support.
1222	kAdeOpenKeyViewDictionaryFails	ADE fails to open ADE Key View information dictionary
1223	kAdeSaveKeyViewDictionaryFails	ADE fails to save ADE Key

		View information in dictionary. Call support.
1224	kAdeSaveProjectionFails	ADE fails to save projection code in the drawing. Call support.
1225	kAdeCopyHardPointerFails	ADE fails to apply property alteration for some symbol table. Call support.
1226	kAdeDwgToBeReloaded	
1227	kAdeDwgHasBeenModified	There were objects queried from the drawing that will be treated as new objects. If using ADEDRAWINGS, this message means that when a drawing from which objects have been queried is detached, ADE converts the objects into newly created objects. When you use the ADEWHOHASIT command to see the origin of these objects, it says that they have not been queried.
1228	kAdeOnLockedLayer	Objects from a locked layer have been selected. If using ADESELOBJS, ADEREMOBJS, or ADESAVEOBJS, this message means you selected objects from a locked layer and they cannot be added to, saved to, or removed from the save set or saved to source.
1229	kAdeDwgSaveFales	ADE fails to save the source

		 drawing. If using ADESAVEOBJS, ADESELOBJS, ADEREMOBJS, OPEN, NEW, or QUIT, or if you're configuring options, or modifying objects, you may get this message. Call support. NOTE: Use this message exactly as spelled here.
1230	kAdeDwgLocksLeft	ADE fails to remove object locks (if present) by the end of ADE session. If using OPEN, NEW, or QUIT, or if you're configuring options, this message means the drawing might be locked by another ADE user.
1231	kAdeLinkWillBeLost	ADE does not save links between queried objects and source drawings between ADE sessions. Detach source drawing with queried objects. Use the SAVE command with queried objects or no objects in the save set.
1232	kAdeDwgDiskFull	
1233	kAdeDwgHasQueriedObject	
1234	kAdeDwgUnlockedFile	

Current Session

1300 kAdeGetCPointFails = 1300

ADE fails to get Text location for

		the entity. Call support.
1301	kAdeSetCPointFails	ADE fails to store Text location for the entity. Call support.
1302	kAdeTextInsPointMissed	Missing Text insert point. Call support.
1303	kAdeTextAllignPointMissed	Missing alignment point for Aligned and Fit Text. Call support.
1304	kAdeTextHeightMissed	Missing Text height. Call support.
1305	kAdeTextStringMissed	Missing Text string. Call support.
1306	kAdeIndexUpdateFails	Exception in Index generation or regeneration. If using ADESAVEOBJS, ADESELOBJS, ADEREMOBJS, or ADEDWGMAINT, this message means entity modification occurred. Call support.
1307	kAdeIgnorePreview	ADE custom object is selected to add to save set or to save. If using ADESAVEOBJS, ADESELOBJS, or ADEREMOBJS this message has the following meaning: When doing a Preview Query, ADE creates a special object called a PREVIEW object, used to display the queried objects. PREVIEW cannot be saved back to a source drawing. When a user selects this object for adding to the save set or when saving back, ADE detects this and prevents the operation.
1308	kAdeIgnorePSpace	

1309	kAdeEraseIRDObjectFails	ADE fails to erase the Object Data Table. Call support.
1310	kAdeRenameIRDObjectFails	ADE fails to rename the Object Data Table. Call support. If using ADEDEFDATA, this message means ADE can't rename Object Data Table. Perhaps Object Data of the same name already exists in the drawing or Object Data Table definitions bearing the old name are different in the source drawings. Check the error message for more information.
1311	kAdeAlterIRDObjectFails	ADE fails to alter Object Data Table. If using ADEDEFDATA, this message means ADE can't alter Object Data Table. Perhaps Object Data Table definitions are different in the source drawings. Check the error message stack for more information.
1312	kAdeIrdDuplicateTableName	Object Data Table with specified name already exists. ADEDEFDATA - all Object Data table names must be unique. Duplicate names are not allowed.
1313	kAdeIrdNotIdenticaFormat	Conflict in Object Data Table definition. If using ADEDEFDATA, this message means that perhaps Object Data Table definitions are different in the source drawings. NOTE: Use this message exactly as spelled here.

1314	kAdeBHatchUnit	ADE treats bhatch and its boundary as one unit. If using ADESELOBJS or ADEREMOBJS, this message has the following meaning: When adding an object to the save set, ADE checks if this object is part of a hatch boundary. If it is, all other objects that form the boundary are added to or removed from the save set. When saving back bhatched areas, ADE always treats the boundary as one object.
1315	kAdeReQuery	ADE queried one or more objects twice. ADE does not support UNDO for this operation. If using ADEQUERY or ADERUNXQUERY this message has the following meaning: If a drawing file has been modified by another ADE user and if a queried object matches another query, ADE removes the old copy and queries a new copy. This operation can't be undone.
1316	kAdeCantAccessFont	
1317	kAdeCantAccessImageFile	
1318	kAdeUnableRedefineXrefBlock	

Drawing Set

1500 kAdeRestoreDrawingSetFails

ADE fails to read Drawing Set from DWG file. You encounter this message during ADE

		initialization or when using Open drawing file or ADEDRAWINGS. If using ADEDRAWINGS - attach drawing, the message means "activate on attach" is ON or ade_dsattach(), ade_dswcattach() drawing is corrupted or old ADE version is in use.
1501	kAdeDuplicatedDrawing	Attempt to attach the same drawing twice. If using ADEDRAWINGS, this error appears when a drawing with the same name has already been attached to the work session.
1502	kAdeNestedDrawing	Prohibited attempt to edit properties of the drawing in the nested drawing set. If using ADEDRAWINGS, this message means a user is not allowed to modify the transformation and save back extents of nested drawings in the work session. These properties can only be modified for top level drawings.
1503	kAdeActivateDrawingFails	ADE fails to activate source drawing. ADE was unable to activate a drawing Perhaps the drawing does not exist or it is locked by another user, or the current user doesn't have permission to read the specified drawing file. Check the error message stack for more information.

1504	kAdeDeactivateDrawingFails	ADE fails to deactivate source drawing. ADE is unable to deactivate a drawing. Perhaps the drawing is locked by another user, the drawing no longer exists, or there are locked objects in the drawing. Check the error message stack for more information.
1505	kAdeLongDrawingDescription	Specified drawing description exceeds 133 symbols. Shorten description.
1506	kAdeEntityHasBeenLocked	Drawing with locked entities can't be deactivated. ADEDRAWINGS - the specified drawing cannot be deactivated because it contains locked objects. Remove locks and deactivate.
1507	kAdePreviewNotSupported	ADE 2.0 ignores these objects: Tolerance, Body, Ellipse, 3dSolid, Region, and Mline. If using ADEQUERY (Preview), ADEQVIEWDWGS, or ADEKEYVIEW, this message means ADE does not support a Preview Query of Tolerance, Body, Ellipse, 3dSolid, Region and Mline.
1508	kAdeAliasIsInUse	A drive alias of the same name already exists.
1509	kAdeActivateDrawingCancelled	The drawing activation operation failed.
1510	kAdeAttachDrawingCancelled	The drawing attachment operation failed.

Feature Alteration

1800	kAdeInvalidFeatureType	Attempt to set an invalid property alteration type. If calling API functions ade_altpsetprop or ade_altpdefine, check for a mistake in property type.
1801	kAdeNoListId	Property alteration internal list is invalid. Call support.
1802	kAdeInvalidExpType	Attempt made to set an invalid property alteration expression type. Call support.
1803	kAdeNoExpression	Property alteration internal object is invalid. Call support.
1804	kAdeTextCreationFailed	Property alteration was unable to create a new text object. This message occurs when using queries that alter properties. Check expressions in the text property alteration definition.
1805	kAdeHatchCreationFailed	Property alteration was unable to create a new hatch object. This message occurs when using queries that alter properties. Check expressions in the hatch property alteration definition.
1806	kAdeInvalidColor	Invalid color passed to property alteration. This message occurs when using queries that alter properties. Check expressions evaluates to a valid AutoCAD color.
1807	kAdeInvalidLayer	Invalid layer name. This message occurs when using queries that alter properties. Check expressions evaluates to a valid AutoCAD layer.

1808	kAdeInvalidStyle	Invalid style name. This message occurs when using queries that alter properties. Check expressions evaluates to a valid style.
1809	kAdeInvalidJustification	The expression for justification in a text property alteration did not evaluate to a valid justification. This message occurs when using queries that alter properties. Check expressions evaluates to a valid AutoCAD justification.
1810	kAdeInvalidScale	
1811	kAdeNoRangeId	ADE internal object is invalid. Call support.

Mapping

1900	kErrMapCoincPoint	Coincident points. If using ADERSHEET or ADETRANSFORM this message means either old or new points are coincident. They must be different.
1901	kErrMapWrongScale	Invalid scale. Call support.
1902	kErrMapTransform	Can't transform entity. ADERSHEET, ADETRANSFORM, ADEQUERY, ADESAVEOBJS An error appeared at the time of entity transformation. It is high- level error. There must be another error in the stack with more specific information.

1903	kErrMapWrongExtents	Invalid entity extents. If you're using ADETEXTLOC, ADERSHEET, ADEQUERY, ADESAVEOBJS, or calling AcDbEntity::getGeomExtents() method you may get this message.
1904	kErrMapWrongPoints	Invalid points number. If you're using ADERSHEET, this message means that the numbers of old and new points are different, or less than 2. Dialog doesn't allow this.
1905	kErrMapWrongSelSet	Invalid selection set. Call support.
1906	kErrMapWrongEntityName	Invalid entity name. The entity is open. For example, it has been received from the API.
1907	kErrMapOpenEntity	Can't open entity. The entity is open. For example, another application opened the entity.
1908	kErrMapUpgradeEntity	Can't upgrade open. Entity modification occurred.
1909	kErrMapMoveStretchPoints	Can't modify stretch points. Call support.
1910	kErrMapEntityPoint	Can't modify entity points. Call support.
1911	kErrMapCmdecho	Can't change CMDECHO variable. Using ADEFILLPOLYG may produce this message.

1912	kErrMapCecolor	Can't change CECOLOR variable. Using ADEFILLPOLYG may produce this message.
1913	kErrMapHatch	Error in hatch command. Using ADEFILLPOLYG may produce this message.
1914	kErrMapWrongIntersectForPoints	Can't find intersection. Using ADEQUERY or calling AcDbentity::IntersectWith() method may produce this message.
1915	kErrMapWrongHandle	Wrong entity handle. Call support.
1916	kErrMapNotPolyline	Entity isn't polyline. Call support.
1917	kErrMapIterator	Can't create iterator. Call support.
1918	kErrMapWriteXData	Can't write Xdata. Ensure that Xdata size is 16 KB or less.
1919	kErrMapBuffer	Can't create buffer. If you're using ADEQUERY, to make a location query using a bufferfence, you may get this message.
1920	kErrMapStretchPoints	Can't get stretch points. If you're using ADETRANSFORM or ADEQUERY or calling AcDbEntity::getStretchPoints() you may get this message.

Lopology

2000	kErrTopInvalidName	Invalid topology name. Occurs during topology creation.
2001	kErrTopExist	Topology already exists. Occurs during topology creation.
2002	kErrTopBuildNet	Error building network topology. Occurs during topology creation.
2003	kErrTopBuildPolygon	Error building polygon topology. Occurs during topology creation.
2004	kErrTopBuildNode	Can't create node. Occurs during topology creation.
2005	kErrTopBuildArc	Can't create link. Occurs during topology creation.
2006	kErrTopBuildCntr	Can't create centroid. Occurs during topology creation.
2007	kErrTopAPIReg	Can't register topology API. Occurs during ADE loading.
2008	kErrTopFuncNotAvail	Function isn't available. Occurs if you're using topology functions of the API.
2009	kErrTopWriteData	Error writing Xdata. Occurs during topology creation and modification.
2010	kErrTopNotExist	Topology doesn't exist. Occurs if you're using topology functions of the API.
2011	kErrTopOverlayType	Wrong overlay type.
2012	kErrTopMakeLayer	Can't create new layer. Using ADEDWGCLEAN produces this

		message.
2013	kErrTopBlockNotExist	Block doesn't exist.
2014	kErrTopNotOpenForWrite	Topology isn't open for write. Occurs when editing topology.
2015	kErrTopOpenIrdTable	Can't open object data table. Occurs when loading and editing topology.
2016	kErrTopWrongIrdAttr	Invalid object data table. Occurs when loading and editing topology.
2017	kErrTopLoaded	Topology is already loaded. Occurs when loading topology.
2018	kErrTopIncompleteElem	Incomplete topology element. Occurs when editing topology.
2019	kErrTopInvalidColor	Invalid color number. Occurs when using ADEDWGCLEAN and creating topology.
2020	kErrTopInvalidFlag	Invalid flag. Occurs when using ADEDWGCLEAN.
2021	kErrTopInvalidTolerance	Invalid tolerance. Occurs when using ADEDWGCLEAN and creating topology.
2022	kErrTopInvalidCorridor	Invalid corridor width. Occurs when using ADEDWGCLEAN.
2023	kErrTopInvalidOffset	Invalid offset. Occurs when using buffering.
2024	kErrTopInvalidHeight	Invalid marker height. Occurs when using ADEDWGCLEAN.
2025	kErrTopInvalidMarkerType	Invalid marker type. Occurs when

		using ADEDWGCLEAN.
2026	kErrTopInvalidEntityType	Invalid type for new entities. Occurs when using ADEDWGCLEAN.
2027	kErrTopInvalidErrorType	Invalid error type. Occurs when using ADEDWGCLEAN.
2028	kErrTopIntersection	Intersections detected. Occurs when creating and editing polygon topology.
2029	kErrTopOverlayItself	Can't overlay topology with itself.
2030	kErrTopSourceDwgAccess	Can't access source drawing.
2031	kErrTopSourceDwgNotActive	Source drawing isn't active.
2032	kErrTopSourceDatabaseAccess	Can't access source drawing database.
2033	kErrTopSourceObjectId	Can't get object ID by handle in source drawing database.
2034	kErrTopNotLoaded	Topology isn't loaded.
2035	kErrTopImplicitNode	Node object doesn't exist in node topology.
2036	kErrTopMisplacedNode	Wrong node coordinates.
2037	kErrTopUnreferencedNode	Node isn't referenced in links.
2038	kErrTopUnexistentNode	Link references nonexistent node.
2039	kErrTopMismatchStartNode	Link has invalid ID at the start node.
2040	kErrTopMismatchEndNode	Link has invalid ID at the end node.

2041	kErrTopMisplacedCentroid	Wrong centroid coordinates.
2042	kErrTopMismatchLeftPoly	Link has invalid ID for the left polygon.
2043	kErrTopMismatchRightPoly	Link has invalid ID for the right polygon.
2044	kErrTopUnexistentCentroid	Centroid isn't inside polygon.
2045	kErrTopMultiplyCentroid	Polygon has several centroids inside.
2046	kErrTopWrongPolyQty	Some polygons are incorrect.
2047	kErrTopMismatchPolyArea	Incorrect polygon area.
2048	kErrTopMismatchPolyPerimeter	Incorrect polygon perimeter.
2049	kErrTopOpenSourceDwgTopo	Topology loaded from source drawings can't be open for write.
2050	kErrTopOpenTempTopo	Temporary topology can't be open for write.
2051	kErrTopIdNotExist	Current drawing doesn't have OD table with information about last ID.
2052	kErrTopEmpty	Can't create or load empty topology.
2053	kErrTopWasModified	Topology objects were modified by AutoCAD commands.
2054	kErrTopMultiple	Object belongs to multiple topologies and can't be erased.
2055	kErrTopCalculateOffset	Can't calculate offset. Use default.
		Occurs when using buffering.
2056	kErrTopZeroOffset	Occurs when using buffering. Zero offset. Can't build buffer.

2057	kErrTopDifferentOffset	Offset has different sign for some objects. Can't build buffer.
2058	kErrTopInvalidSelSet	Invalid selection set. Occurs when using the API.
2059	kErrTopCleanNotInit	Cleanup model isn't initialized. Occurs when using the API.
2060	kErrTopCleanNoGroup	There is no current group. Occurs when using the API.
2061	kErrTopCleanInvalidIndex	Invalid error index. Occurs when using the API.
2062	kErrTopCleanNoError	Current error isn't set. Occurs when using the API.
2063	kErrTopTraceLinkNotExist	Link doesn't exist in tracing model. Occurs when using the API.
2064	kErrTopTraceNodeNotExist	Node doesn't exist in tracing model. Occurs when using the API.
2065	kErrTopTraceNoPath	Result path isn't calculated. Occurs when using the API.
2066	kErrTopTraceInvalidIndex	Invalid element index. Occurs when using the API.
2067	kErrTopInvalidExpression	Can't process ADE expression. Occurs when using overlay, buffer, dissolve, or tracing command.
2068	kErrTopLockedTable	Can't write into topology OD table. Occurs when using dissolve command.
2069	kErrTopCreateTable	Can't create OD table. Occurs when using topology creation,

		overlay, buffer, or dissolve commands.
2070	kErrTopCreateTableColumn	Can't add column to OD table. Occurs when using topology creation, overlay, buffer, or dissolve commands.
2071	kErrTopTraceNodesEqual	Start and end nodes are the same. Occurs doing shortest path tracing.
2072	kErrTopTracePathNotExist	Empty path. Occurs during shortest path tracing.
2073	kErrTopTraceFloodNotExist	Empty path. Occurs when tracing floods.
2074	kErrTopRenameDisabled	Can't rename topology, because current drawing has queried objects with OD.
2075	kErrTopDeleteDisabled	Can't delete topology, because current drawing has queried objects with OD.
2076	kErrTopInvalidExtents	

Topology API

2100	kAdeTopApiErrWrongInput	Missing or invalid parameter.
2101	kAdeTopApiWrongId	Invalid ID.

Tracing

2150	k A deTonSnrFrr	Tracing error Occurs during topology tracing
2100	M luc ropopriir	frucing cirol. Occurs during topology trucing.

Ouerv Definition

~--*j* ---

2200	kErrUnexpectedBuffChar	Invalid character encountered while reading the query definition from the drawing. Options are a) Recover the drawing b) Define and save a new query definition in the drawing.
2201	kErrInvalidIndex	An invalid line number was specified for either grouping or ungrouping of query lines. Specify the correct line number for grouping or ungrouping the lines.
2202	kErrInvalidQueryLine	One or more query lines have been incorrectly defined. May occur when you incorrectly place a parenthesis or an operator in a query line.
2203	kErrInvalidName	Either a query or a query category name is invalid. Ensure that the query or category name conform to the AutoCAd symbol name specifications.
2204	kErrEntryAlreadyExists	Either the query or the query category name already exists in the query library. Ensure that the query name is unique within the query library.
2205	kErrEntryInOtherCategory	The query name specified already exists in another category in the query library. Ensure that the query name is unique within the query library.
2206	kErrEntryAndFileAlreadyExist	The file name specified for saving

		the external query already exists. Choose a different file name.
2207	kErrASIConnectFailed	The connection to the ASI environment required for the SQL query was not made. Use ASE to connect to the environment before attempting to perform the SQL query.
2208	kErrASIStmtPrepareFailed	The call to CAsiExecStm:: Prepare failed. correct the table name or the SQL statement specified.
2209	kErrASICsrAllocFailed	The call to CAsiCsr::Allocate failed. Look at the ASI error displayed.
2210	kErrASICsrOpenFailed	The call to CAsiCsr::Open failed. Look at the ASI error displayed.
2211	kErrInvalidDOName	An invalid Environment, Schema or Catalog name was specified. Set the correct Environment, Catalog, and Schema names.
2212	kErrLPInitFailed	The call to CAseLinkPath::init failed. Look at the ASE error displayed.
2213	kErrColNotFound	Used for the SQL Order-by dialog now obsolete. Call support.
2214	kErrQDefNotInTM	The CAdeQueryDef object was not appended to the Transaction Manager. Internal error.
2215	kErrQryDefnExists	A query definition already exists and a new one cannot be loaded. Clear the existing query definition

		before loading a new one.
2216	kErrInvalidOperator	Invalid operator defined in query definition. The specified comparison operator is incompatible with operand types. Do not use > with the point type. Check the query definition and change either the operator or operand type.
2217	kErrInvalidPtrnOperator	Invalid operator defined in query definition for pattern value. If value operand is defined as pattern, only "=" comparison operator can be used. Check the query definition and change either operator or operand value.
2218	kErrInvalidField	Non-existent object data field specified. This error occurs when the user specifies the wrong object data field name for a table (if there is no such field in the specified table) in the query definition and executes the query. Check query definition and tables and correct the mistake.
2219	kErrInvalidNotBranch	
2220	kErrInvalidBranch	
2221	kErrUndefinedValue	
2222	kErrInvalidLocationType	
2223	kErrCantLoadExternQuery	

Ouerv Manager

▼ - *J* - *O* -

2300	kErrIntersectFailed	A call to CAseLinkSel:: intersectPartialKey failed. Look at the ASE error displayed.
2301	kErrNoTemplate	The query type was specified as report but no report options were defined. Define report options.
2302	kErrASIStoreValueFailed	A call to CAsiData ::storeValue failed. Look at the ASI error displayed.
2303	kErrASIGetValueFailed	A call to CAsiData ::getValue failed. Look at the ASI error displayed.

Utility

2400	kErrLicFatal	Fatal error in ADE license. Call support.
2401	kErrFileNotFound	Can't find associated document. Occurs when using ADEDOCVIEW.
2402	kErrPathNotFound	Can't find executable file. Occurs when using ADEDOCVIEW.
2403	kErrBadFormat	Syntax error in the command line. Occurs when using ADEDOCVIEW.
2404	kErrConvtErr	Error converting ADE 1.0 data to ADE 2.0 data. Occurs when using ADECONVERT.

Data Dialogs

2450	kErrIRDMismatch	
2451	kErrIRDInvalidName	
2452	kErrIRDTableExists	
2453	kErrInvalidTableName	

2454	kErrInvalidAttrName	
2455	kErrTopoName	
2456	kErrQueriedAndNotNew	
2457	kErrNotAdministrator	

GenLink

2500	kErrTagNotFound	
2501	kErrTagValueAbcent	
2502	kErrIllegalFormat	
2503	kErrColMoreThanOne	

Environment

2600	kErrInitEnv	An error occurred during the initialization of ADE. The cause of this error may be due to errors in loading/initializing ADE user preferences, system preferences, log file, or user list.
2601	kErrCantFindAdeExePath	
2602	kErrINIWrite	
2603	kErrInvalidUserName	The user name specified does not exist in the user list. Use a user name that already exists in the user list or define a new one using User Administration.
2604	kErrLoadUserList	
2605	kErrSaveUserList	
2605	kErrSaveUserList	

2606	kErrInvalidPswd	The password specified does not match
		the one specified in the user list for this
		user. Use the correct password.

Rx

2700	kErrRxAseLoad	ASE isn't loaded Can't initialize ASE API.
2701	kErrRxAseInit	Object Data module isn't loaded.
2702	kErrRxIrdLoad	Can't initialize Object Data API.
2703	kErrRxIrdInit	Specified coordinate system category not found in the library. Call support.

Projection

2800	kErrNoProjCatFound	Specified coordinate system category not found in the library.Call support.
2801	kErrNoDatumFound	Specified coordinate system datum not found. Call support.
2802	kErrNoElipFound	Ellipsoid not found in the ellipsoid list. Call support.
2803	kErrNoCoordFound	Specified Coordinate System not found. Call support.
2804	kErrFaileOpenDatumFile	Can't open projection .mp3 file. Occurs when loading ADE.
2805	kErrFaileOpenElipFile	Can't open ellipse file. Occurs when loading ADE.
2806	kErrNoneCoord	Internal code to set "None" projection to the drawing. Call support.

Expression

2900	kErrNoExpressionFound	Empty expression is specified. Call support.
2901	kErrGetPropFail	This is an internal code to show that entity has no specified property. Call support.
2902	kErrExpEvalFail	ADE fails to evaluate expression. Occurs when executing query with property alteration and executing a property query.
2903	kErrExpMissingQuote	Quotes mismatched in SQL expression. Occurs when executing a SQL query and a query with SQL property alteration. Also occurs when using ADECONVERT.
2904	kErrExpMissingCParen	Parenthesis mismatched. Occurs when executing a query with feature alteration and executing a property query.
2905	kErrExpExceedThreeOper	More than three operands are specified. Occurs when executing a query with property alteration and executing a property query.
2906	kErrRngTabNameExist	Range table with specified name already exists. Call support.
2907	kErrLpnInvalid	
2908	kErrLpnNotFound	
2909	kErrRangeInvalidElse	

For more Expression errors, which begin at \clubsuit 15, click **\square**.

Index

3000	kErrInvalidIndexVersion	The version of the index in the drawing is invalid. The options are: 1)Regenerate the index using drawing maintenance 2)Remove the index using the index removal utility and then re-generate the index.
3001	kErrIndexOutOfDate	The index in the drawing is out-of- date. Regenerate the index using drawing maintenance.
3002	kErrTypeAllObjects	
3003	kErrTypeNoOneObject	

Validation

3100	kErrWrongSymbolName	
3101	kErrWrongSymbol	
3102	kErrWrongStrLength	
3103	kErrDirDoesNotExist	
3104	kErrDirReadOnly	
3105	kErrAccessDenied	
3106	kErrFileDoesNotExist	
3107	kErrFileAlreadyExists	
3108	kErrFileOpenFailed	
3109	kErrFileReadOnly	
3110	kErrInvalidString	
3111	kErrOutOfRange	

3112	kErrWrongColor	
3113	kErrIncorrectParameters	One of ADE validation methods recognized incorrect input parameters. This error is an internal ADE error.
3114	kErrFileOpenLimit	
3115	kErrShareViolation	
3116	kErrNetAccessDenied	
3117	kErrPathDoesNotExist	

File Locking

3200	kErrDwkFileDoesNotExist	ADE lock file is locked. Occurs when using ADEDRAWINGS and ADEQUERY commands and when ADE is running in a multi- user environment.
3201	kErrOpenDwkFileFailed	ADE was unable to open the .DWK lock file. Call support.
3202	kErrFileLockedByAcad	Attempt to remove a user who does not exist from the lock file. Call support.
3203	kErrOldMapLockFile	ADE was unable to create the .DWK lock file. Call support.
3204	kErrFileIsNotDwk	ADE tried to lock a file for write that was already locked for read. Call support.
3205	kErrSpecifiedUserDoesNotExist	ADE tried to lock a file for read that was already locked for write. Occurs during query operations in a multi-user environment.

3206	kErrCreateDwkFileFailed	ADE tried to open and read a file that was not a valid .DWK file. Call support.
3207	kErrFileIsLockedForRead	ADE was unable to unlock the lock file. Call support.
3208	kErrFileIsLockedForWrite	ADE tried to attach a file that is already open by AutoCAD. Occurs when using ADEDRAWINGS with ATTACH operations if the file is open in an AutoCAD project.
3209	kErrInvalidLockStateSpecified	ADE internal object is invalid. Call support.
3210	kErrNotOwnerOfWLH	ADE tried to remove a write lock when the user did not have a write lock. Call support.
3211	kErrUserIsNotWriter	
3212	kErrUserIsNotReader	
3213	kErrUserHasReadLock	
3214	kErrLockFileIsFull	
3215	kErrDwgFileDoesNotExist	ADE tried to unlock a file but the .DWK file was missing. Occurs if the .dwk file was erased after a file was attached.
3216	kErrNotAnADELockFile	ADE internal object is invalid. Call support.
3217	kErrFileMayHaveBeenModified	Existing .DWK file does not belong to ADE. the lock file exists and can be read by ADE, but ADE does not own the file.
ADE tried to remove a lock file but it was not found. Occurs if the .dwk file was erased after a file was attached.

Unicode Support

4000	kAdeUnicodeInsufficientBufferToConvert	
4001	kAdeUnicodeInvalidFlagsToConvert	
4002	kAdeUnicodeInvalidParameterToConvert	
4003	kAdeUnicodeNoTranslation	
4004	kAdeUnicodeCodePageNotAvailable	

Double-Byte Support

4005	kAdeNoMBCSAllowed	
------	-------------------	--

Error Types

Errors are grouped by error type.

You can get an error's error type by passing its stack index (0 = first error) to <u>ade_errtype</u>.

00	kAdeNoMessage	Error condition does not exist.
01	kAdeWarning	ADE (AutoCAD Data Extension) execution warning
02	kAdeError	ADE execution error
03	kAseWarning	ASE (AutoCAD SQL Extension) execution warning
04	kAseError	ASE execution error
05	kAcWarning	AutoCAD execution warning
06	kAcError	AutoCAD execution error
07	kAsiWarning	ASI (AutoCAD SQL Interface) execution warning
08	kAsiError	ASI execution error.
09	kIRDWarning	Extended object data (Xdata) warning
10	kIRDError	Xdata error
13	kMapError	AutoCAD Map execution error

14	kMentorError	
15	kApplicationError	Operating system level application error
16	kDiagMessage	Diagnostic message returned.

Converting point coordinates from one geo-referenced system to another.

Keep in mind that any Cartesian coordinate pair you select in a geo-referenced coordinate system corresponds to a point on the surface of the earth. This fact defines a relation between the coordinate pairs in one coordinate system and the coordinate pairs in any other, so long as the point in question actually exists in both systems. In other words, so long as the coordinate systems have a region of intersection, and the point in question is in it.

To convert the coordinates of a point from one geo-referenced coordinate system to another

- 1. Define a "source" coordinate system with <u>ade_projsetsrc</u>.
- 2. Define a "destination" coordinate system with <u>ade_projsetdest</u>.
- 3. Pass a coordinate pair to <u>ade_projptforward</u>.

The function assumes that the coordinate pair you pass to it is a point in the source system, and it returns the corresponding coordinate pair in the destination system. If there is no corresponding coordinate pair, it returns nil.

To convert in the other direction, use <u>ade_projptbackward</u>.

You can specify coordinate triplets, but if you do, the Z value is ignored.

Plotting a Map

Plotting is the last stage in presenting map data.

The following sample outlines the steps required to set up and plot a map set.

To plot a map

1. Set the Plotter name for Layout1 using the AutoCAD ActiveX interface.

```
(vl-load-com)
(setq acadObject (vlax-get-Acad-object))
(setq acadDocument (vla-get-ActiveDocument acadObject))
(setq activeLayoutObject (vla-Get-ActiveLayout acadDocument))
(setq pSpace (vla-get-PaperSpace acadDocument))
(setq layout (vla-get-layout pSpace))
(vla-put-configname layout "\\\PlotServer\\Hp Laserjet 4100 Series Pc
```

2. Add a drive alias to the list so that AutoCAD Map can find the drawings used to execute the plot.

(ade_aliasadd "LISP_SAMPLE" "C:\\Lisp\\Drawings\\PlotMapSet")

3. Define a boundary drawing or use an existing one. This example uses an existing drawing named System-Grid.dwg.

The boundary drawing must contain at least one closed polyline with object data. Each closed polyline is a division of the map, and the sheet

name is attached to it as object data.

4. Attach the drawings for this plot set and the boundary drawing.

(ade_dsattach "LISP_SAMPLE:\\Landbase-Shared.dwg") (ade_dsattach "LISP_SAMPLE:\\Water-Facilities.dwg") (ade_dsattach "LISP_SAMPLE:\\System-Grid.dwg")

5. Define a "Draw" mode query which selects "ALL" objects in the attached drawings.

```
(ade_qryclear)
(ade_qrydefine
 '("" "" "Location" ("All") "" ))
(ade_qrysettype "draw")
```

6. Save the query internal to the project.

(ade_qrysave "PlotQueries"
 '(("name" "TruckBook-100") ("saveoption" 2)))

- 7. Define a layout block, (title block) or use an existing one. The layout block must contain at least one viewport and optionally a reference viewport. If the layout block contains an attribute which represents a mapname or some other type of area naming convention, that name can be mapped from the boundary drawing to this attribute, in fact any object data associated to a boundary can be mapped to an attribute on the layout block. This sample does not contain code for creating or defining this layout block.
- 8. Create a new plot set.

(map_pltCurrDef "TruckBook-100")

9. Define Plot Set variables

Define the drawing set:

(setq dwgs_list
(list "LISP_SAMPLE:\\Landbase-Shared.dwg" "LISP_SAMPLE:\\Wat
(map_pltCurrSet "dwgs" dwgs_list)

Define the plotter name via the layout:

```
(map_pltCurrSet "pnam" "Layout1")
```

Define the layout block name:

```
(map_pltCurrSet "block" "TITLE-A")
```

Define the layout blocks viewport layer:

```
(map_pltCurrSet "vlayer" "TB-BORDER")
```

Define the object data table containing boundary information:

(map_pltCurrSet "bnamt" "MAPSYS-MAP")

Define the boundary field name:

(map_pltCurrSet "bnamf" "MAPNAME")

Define the boundary object data field to block attribute in layout mapping:

```
(map_pltCurrSet "atts" "MAPNAME")
```

Define the layout blocks reference viewport flag:

```
(map_pltCurrSet "kflg" nil)
```

Define the layout blocks key viewport layer:

```
(map_pltCurrSet "klayer" "")
```

Define the display layers in the reference viewport:

```
(map_pltCurrSet "kdispl" "")
```

Define the layers to freeze in main viewport:

```
(map_pltCurrSet "vdispl" "0")
```

Define the query category:

```
(map_pltCurrSet "qcat" "PlotQueries")
```

Define the query name:

```
(map_pltCurrSet "qnam" "TruckBook-100")
```

Define the boundary drawing:

```
(map_pltCurrSet "bdwg" "LISP_SAMPLE:\\System-Grid.dwg")
```

Define the boundary layer:

```
(map_pltCurrSet "blyr" "MAPEDGE")
```

Define the boundary field name:

```
(map_pltCurrSet "bodfs" "MAPNAME")
```

Specify the boundaries:

(setq bndry_list (list "101" "102"))
(map_pltCurrSet "bnds" bndry_list)

Define the plot to specified scale flag:

```
(map_pltCurrSet "sflg" nil)
```

Define the plot scale:

```
(map_pltCurrSet "scale" "1:750")
```

Define the clip objects against boundary flag:

```
(map_pltCurrSet "clip" T)
```

Define the plot the boundary flag:

```
(map_pltCurrSet "pbnd" T)
```

10. Append the current plot set definition to the plot set list.

```
(map_pltCurrSave)
```

11. Write the current plot set definition to the plot definition dictionary within the current project.

(map_pltDefSave)

12. Perform a cursory check to see if the required attributes have been set, this is optional.

```
(if (not (map_pltDefValid "TruckBook-100"))
  (prompt "\nERROR: Plot set invalid.")
)
```

13. Perform a more extensive check that makes sure that no errors occur when plotting takes place, this is optional.

(map_pltdefverify "TruckBook-100")

14. Initialize the plotting environment. See (<u>map_pltInit</u>) for more detail on this function.

(map_pltInit)

15. Display a single sheet within the plot set based on a boundary value, then plot if successful. Note, you must always call map_pltdisplay() before calling map_pltPlot().

```
(map_pltDisplay "101")
;Visually check the plot set display. If it is correct, plot it.
(initget "Yes No")
(setq answer
    (getkword "\nDisplay is correct Yes/No ?"))
(if (or (null answer) (equal answer "Yes"))
    (map_pltPlot))
```

16. Restore the display environment to its state before the last execution of the map_pltDisplay().

(map_pltRestore)

17. Restore the data extension options and AutoCAD settings altered by map_pltInit().

(map_pltCleanup)

Completing a Topology

This procedure creates a partial topology and uses <u>map_topocomplete</u> to add the missing elements to complete it.

To complete a topology

- 1. Create a polygon topology and save it to a file.
- 2. Start a new project and attach the drawing using <u>ade_aliasadd</u>, <u>ade_dsattach</u>, and <u>ade_dwgactivate</u>.
- 3. Query in some of the topology's objects to create a partial topology. Use <u>query functions</u>.

```
(ade_qryclear) ; clear all queries
(ade_qrysettype "draw") ; draw query results
(setq pt1 (list 2.9123 8.8513 0.0))
(setq pt2 (list 5.9134 1.1634 0.0))
(setq pt3 (list 10.7931 1.7354 0.0))
(setq pt4 (list 10.0371 8.8742 0.0))
(setq qt4 (list 10.0371 8.8742 0.0))
(setq qry_id (ade_qrydefine "" "" "location" lst ""))
(if (null qry_id )
  (prompt "\nERROR: Query definition failed.")
  (progn
      (prompt "\nQuerying in part of the polygon topology.")
      (setq result (ade_qryexecute))
      (if (null result)
```

```
(prompt "\nExecution of query failed.")
)
)
```

4. Load the topology into memory from the project drawing.

(setq result (tpm_acload name nil))

5. Open the topology with read access.

```
(setq tpm_id (tpm_acopen "test" nil))
```

6. Test the topology for correctness and completeness.

```
(if (tpm_infocorrect tpm_id)
  (prompt "\nTopology is correct.")
  (prompt "\nTopology is not correct.")
)
(if (tpm_infocomplete tpm_id)
  (prompt "\nTopology is complete.")
  (prompt "\nTopology is not complete.")
)
```

7. Optionally, you can get topology statistics. You can add code to display the statistics.

```
(setq lst (map_topostat tpm_id))
(if (null lst)
  (prompt "\nERROR: Unable to get statistics.")
  (progn
     (prompt "\nNumber of nodes: ")
```

```
(princ (cdr (assoc "node_count" lst)))
 (prompt "\nNumber of links: ")
 (princ (cdr (assoc "link_count" lst)))
)
```

8. Close the topology.

)

(tpm_acclose tpm_id)

9. Use <u>map_topocomplete</u> to bring in the missing objects to complete the topology.

(map_topocomplete "test")

- 10. Repeat steps 5 through 7 to test the topology for completeness.
- 11. Unload the topology. >

(tpm_acunload "test")

Drawing Cleanup

You must clean drawing objects before building a topology with them.

Drawing cleanup extends undershoots, snaps clustered nodes, removes duplicates, simplifies linear objects, and corrects other errors.

Let's look at this in two phases, preparing the cleanup model, which ends with a call to tpm_cleaninit, and executing the cleanup, which begins with a call to tpm_cleanstart.

To prepare the cleanup model

1. Allocate memory for the cleanup model. Use <u>tpm_cleanalloc</u>.

(setq clean_id (tpm_cleanalloc))

2. Allocate memory for <u>cleanup variables</u>, which specify properties for the cleanup process. The variables are initialized to their default values. Use <u>tpm_varalloc</u>.

(setq clean_var_id (tpm_varalloc))

If you will be specifying an explicit list of cleanup actions (you create and manage this list with calls to <u>tpm_cleanactionlistins</u> and related functions), also allocate memory for <u>cleanup action variables</u>, which specify properties for individual actions. Again use <u>tpm_varalloc</u>.

(setq action_var_id (tpm_varalloc))

3. Get a selection set of objects to be cleaned (the *include* set).

```
(prompt "\nSelect objects to clean.")
(setq ss_clean (ssget))
```

You can also get a selection set of objects to be anchored (the *anchor* set). Anchored objects are not repositioned by the cleanup process, but remain fixed while others are repositioned around them.

```
(prompt "\nSelect objects to anchor.")
(setq ss_anchor (ssget))
```

The **ssget** function prompts the user to select objects and returns a selection set.

4. Set cleanup variables using <u>tpm_varset</u> with the <u>clean_var_id</u> that you allocated in step 2. A few of these variables specify cleanup actions, but most of them specify how cleanup actions will be performed.

```
; set some cleanup variables
(tpm_varset clean_var_id "MAINTAIN_MARKERS" 1)
(tpm_varset clean_var_id "CLEAN_TOL" 4.21)
(tpm_varset clean_var_id "ANCHOROBJS_LAYERS" "Layer1")
```

Before setting cleanup variables, you can load a cleanup profile if you saved one previously, and in that way set many variables at once. Use tpm_cleanprofileload.

```
(setq result
  (tpm_cleanprofileload clean_var_id "C:\\profile.dpf"))
```

If you specify an explicit list of cleanup actions, note that those will be the only actions performed. Cleanup actions specified by the variables **NODE_ERROR**, **LINK_ERROR**, and **GENERALIZE** will be ignored, as well as any setting specific to them only, such as **CORRIDOR**'s, which defines the tolerance for **GENERALIZE**.

Using an action list is the best way to specify cleanup actions, because you can specify the order in which they execute, and you can include the same action more than once. Using variables to specify cleanup actions is an older technique, which is still supported for the sake of older scripts, but it is deprecated from AutoCAD Map 6 onward.

Note When you insert the Simplify Objects action (clean group type 128), it is always listed first, and you cannot insert it more than once.

With an explicit list of cleanup actions, note that certain individual actions can have individual tolerance settings (and in some cases, other settings also). See <u>Cleanup Action Variables</u>. When you are about to insert an action into the action list, you can use <u>tpm_varset</u> with the action_var_id that you allocated in step 2 to set variables for this action before calling <u>tpm_cleanactionlistins</u>. You can continually reset and reuse the same set of cleanup action variables with each action that you insert.

; insert a cleanup action into the action list

; first set a tolerance for this action

(tpm_varset action_var_id "CLEAN_TOL" 2.2)

; with the action list referenced by clean_var_id...

; * insert at the first position (position 0)

- ; * insert Erase Short Objects (action 1)
- ; * with the options referenced by action_var_id

(tpm_cleanactionlistins clean_var_id 0 1 action_var_id)

At any point while you are setting cleanup variables, or after you have finished, you can save the current cleanup profile using tpm_cleanprofilesave.

(setq result
 (tpm_cleanprofilesave clean_var_id "C:\\profile.dpf"))

Note that saved profiles are XML files. You can view or edit them in a text editor as you can with saved queries (which are AutoLISP scripts). See <u>Editing Query Files</u>.

5. Call <u>tpm_cleaninit</u> to add cleanup variables and the selection set of objects to clean to the cleanup model.

(setq result
 (tpm_cleaninit clean_id clean_var_id ss_clean))

If you have collected a selection set of objects to be anchored, first call <u>tpm_cleaninitanchorset</u> before calling <u>tpm_cleaninit</u>.

(setq result
 (tpm_cleaninitanchorset clean_id clean_var_id ss_anchor))

The cleanup model is now complete.

To execute the cleanup

1. Begin the cleanup process with <u>tpm_cleanstart</u>.

(setq result (tpm_cleanstart clean_id))

2. Execute cleanup actions (process cleanup groups) until cleanup is complete. With each cleanup group, with each error, mark and fix it.

(tpm_cleangroupnext clean_id)

```
(while (not (tpm_cleancomplete clean_id))
; count errors in this group
(setq i (tpm_cleangroupqty clean_id))
; process each error
(while (>= (setq i (1- i)) 0)
; with the current error
(tpm_cleanerrorcur clean_id i)
; mark it
(tpm_cleanerrormark clean_id)
; clean it
(tpm_cleanerrorfix clean_id)
)
(tpm_cleangroupnext clean_id)
)
```

3. Update the drawing with <u>tpm_cleanend</u>.

```
(tpm_cleanend clean_id)
```

To clear the cleanup model without updating the drawing, use <u>tpm_cleancancel</u>.

Building a Topology

The following steps describe building a topology.

The resulting topology is loaded but not open.

Note For simplicity, cleaning drawing objects before building the topology is omitted. See <u>Drawing Cleanup</u>.

To build a topology

1. Allocate memory for <u>topology variables</u>. The variables are initialized to their default values.

(setq var_id (tpm_varalloc))

2. Set the topology type, name, and description. This example code prompts for the topology type.

```
(initget "noDe Network Polygon")
(setq typ (getkword "Select topology type
      (noDe/Network/Polygon) <Exit>: "))
(if (null typ)
      (prompt "\nNo topology type entered.")
)
```

3. The next step is to include objects for the topology (nodes, links, centroids), depending on the topology type. This code repeats three times to include each object type.

After prompting for node objects, as shown here, this example prompts for link and centroid objects.

Set up a filter to get the objects you want. Some automatic filtering is performed by **ssget**, which places the objects in an AutoCAD selection set. Note that you must include different object types for each type of topology.

```
; Node objects are POINT, TEXT, and INSERT
 (cond
   ((= indx 1)
     (setq filter (list (cons -4 "<OR")
                      (cons 0 "POINT")
                      (cons 0 "TEXT")
                      (cons 0 "INSERT")
                      (cons -4 "OR>")
                  )
     )
     (prompt "\nSelect node objects.")
     (setq ss_nod (ssget filter))
   )
  ; Link objects are LINE, PLINE, ARC, and CIRCLE
   ((and (= indx 2))
         (or (= typ "NETWORK")
            (= typ "POLYGON")
         )
    )
     (setq filter (list (cons -4 "<OR")
                      (cons 0 "LINE")
                      (cons 0 "PLINE")
                      (cons 0 "ARC")
                      (cons 0 "CIRCLE")
                      (cons -4 "OR>")
                 )
     (prompt "\nSelect link objects.")
```

4. Build the type of topology you selected in step 2, using <u>tpm_mntbuild</u>. This sample uses a conditional operation to build the topology.

For type 1, a node topology, tpm_mntbuild includes only nodes.

```
(cond
 ((= typ "NODE")
    (setq result (tpm_mntbuild var_id name desc 1 ss_nod))
 )
```

For type 2, a network topology, it includes nodes and links.

```
((= typ "NETWORK")
  (setq result (tpm_mntbuild var_id name desc 2 ss_nod ss_lnk))
)
```

For type 3, a polygon topology, it includes nodes, links, and centroids.

```
((= typ "POLYGON")
  (setq result
     (tpm_mntbuild var_id name desc 3 ss_nod ss_lnk ss_ctr))
```

If no errors occur, the topology is now complete.

Opening a Topology

The following numbered steps describe how to load and open a topology for read access, get information about the topology, and then close it.

To open a topology

1. Prompt for the topology name.

(setq name (getstring "\nEnter the topology name"))

2. First, use <u>Topology Access functions</u> to see if the topology is already loaded and, if not, load it.

Check to see if the topology is loaded with <u>tpm_acexist</u>.

```
(setq result (tpm_acexist name T T))
(if result
  (prompt "\nTopology is already loaded.")
)
```

Using T for both the **source** and **loaded** parameters causes tpm_acexist to check for topologies in both current and source drawings that are already loaded in memory.

3. If it is not loaded, load it with <u>tpm_acload</u>.

(setq result (tpm_acload name))

You can add code here to handle errors or announce successful loading.

Open the topology for read access. The tpm_acopen function opens a topology and creates a new topology_ID (*) that provides access to it. Using the nil value for the write_access parameter sets access to read.

(setq tpm_id (tpm_acopen name nil))

5. Use <u>Topology Information functions</u> to get information about the topology.

Get the description of the topology with tpm_infodesc.

```
(prompt (strcat "\nTopology desc: "
    (tpm_infodesc tpm_id)))
```

Get the type of the topology with <u>tpm_infotype</u>.

(prompt (strcat "\nTopology type: "
 (itoa (tpm_infotype tpm_id))))

Test the topology to see if it is correct with <u>tpm_infocorrect</u>.

(if (tpm_infocorrect tpm_id)
 (prompt "\nTopology is correct.")
 (prompt "\nTopology is not correct."))

Test the topology to see if it is complete with tpm_infocomplete.

(if (tpm_infocomplete tpm_id)

(prompt "\nTopology is complete.")
(prompt "\nTopology is not complete."))

Get the version of the topology with tpm_infoversion.

(prompt (strcat "\nTopology version: " (tpm_infoversion tpm_id)))

6. Close the topology.

(tpm_acclose tpm_id)

When you use tpm_acopen to open or test the status of a topology, always close the topology with tpm_acclose. Otherwise, you run the risk of leaving the topology open with multiple IDs pointing to it.

Editing Query Files

If you have queries that you use often, you can save them in query files, which you can modify and reuse as you need them. Modifying an existing query rather than creating a new one can save you time. This section shows the contents of some typical query files.

Location Query Property Queries Location Query with Property Alteration Location Query with Property Alteration: Before and After Complex Query Colors can be specified or returned as AutoCAD color indexes (ACIs) or true colors.

By *true colors* we mean 24-bit color: three RGB components, 8 bits each, with no alpha component (that is, no transparency value).

ACI Colors

The valid ACI formats are

Color Indexes, integer strings from 0 through 256. For example, "123". Note that indexes 0 and 256 do not specify colors literally, as 1 through 255 do, but logically. See "Logical Colors" below.

Color Names, which correspond to indexes 1 through 7. The color names are red, yellow, green, cyan, blue, magenta, and white. For example, "yellow" (always double-quoted).

Logical Colors, which correspond to indexes 0 and 256. The logical colors are ByBlock and ByLayer. For example, "ByBlock" (always double-quoted). Note that ByBlock and ByLayer can return true colors or ACIs.

For more information about ACI colors, see <u>Color Index Colors</u>.

True Colors

The valid true-color formats are

RGB Triplets, where each component is an integer from 0 through 255. For example, "255,0,0". RGB triplets are wrapped in double quotes except when they are used in query conditions, in which case they must always be wrapped in escaped double quotes ('\''). See "Color Patterns" below.

Color Book Colors, such as "Pantone, 123 CVC", a composite of two comma-separated names representing a Color Book and a color within it. Color book strings are wrapped in double quotes except when they are used in query conditions, in which case they must always be wrapped in escaped double quotes ('\''). See "Color Patterns" below. And no matter where they are used, color book strings must always be wrapped in escaped double quotes if they contain certain special characters. If you are unsure if a color book string contains special characters, there is no harm wrapping it in escaped double quotes just to be sure.

Expressions, such as ".COLOR" or ".TRUECOLOR" (always doublequoted). ".COLOR" always returns an ACI color. If the selected object's color is a true color it returns the nearest ACI equivalent. ".TRUECOLOR" returns a true color if the selected object's color is a true color, or an ACI if its color is an ACI. Note that ".TRUECOLOR", and other expressions that can return true colors, return in valid format only if the type argument of (ade_expreval) is "string".

Color Patterns, comma-separated lists of colors in any of the valid formats, including ACI colors, always double-quoted. Color patterns are used to express multiple color conditions in compact format. Consider the color pattern "red, green". The pseudocode expression, color = "red,green", is logically equivalent to (color = "red") OR (color = "green"). Similarly, color <> "red,green" is logically equivalent to (color <> "red") AND (color <> "green"). Because color patterns are commaseparated lists, Color Book colors and RGB colors in query conditions are always bounded by escaped double quotes ('\'') because they are themselves comma-separated. For example, the following color pattern includes six colors: three ACIs, one RGB, and one Color Book color.

"12,34,56,\"12,34,56\",\"Pantone, 123 CVC\""

Note You can use wildcard characters when you specify a match string for Color Book colors (but not for RGB colors). For this reason, any wildcard character in a Color Book string that is meant to be taken literally must be

escaped using a backquote, "`". For example, the "." character in the following string, normally a wildcard matching any non-alphanumeric character, is meant as a literal: "My`.Colors, Hot".

Property Alteration Definition Samples

The following examples compose property alteration definitions that use range tables.

```
(defun c:run_altpsample1 ()
 ; Range table for Color is:
 ; < 15 "green"
 ; < 20 "yellow"
 ; otherwise "red"
 (ade_rtdefrange
   "COLOR_RANGE" "Color Range Table"
   '(
     ("<" 15 "green")
     ("<" 20 "yellow")
     ("otherwise" "" "red")
   )
 )
 ; Range table for Text Value is:
 ; < 15 "Small"
  < 20 "Medium"
 ; otherwise "Large"
 (ade_rtdefrange
 "TEXT_RANGE" "Text Value Range Table"
   '(
```

```
("<" 15 "Small")
("<" 20 "Medium")
("otherwise" "" "Large")
)
```

)

; The layer of each object will be used to determine what value ; from the range table is to be used:

; Clear out any existing alteration definitions: (ade_altpclear)

```
; Next define the color for the objects being created:
(ade_altpdefine "color" "(range .layer COLOR_RANGE)")
```

```
; Next define the text object to be created for each entity:
(ade_altpdefine
 "textobject"
 '(
   ("color" "(range .layer COLOR_RANGE)")
   ("textvalue" "(range .layer TEXT_RANGE)")
 )
)
; For fill, use the following property alteration functions
(ade_altpdefine
 "hatch"
 '(
   ("pattern" "(range .layer TEXT_RANGE)")
   ("color" "(range .layer COLOR_RANGE)")
 )
)
(princ)
```

```
(defun c:run_altpsample2 ()
 ; Range table for pattern is:
 (ade_rtdefrange
   "PATTERN_RANGE" "Hatch Pattern Range Table"
   '(
     ("<" 15 "Earth")
     ("<" 20 "Grass")
     ("otherwise" "" "Swamp")
   )
 )
 ; Range table for Color is:
 (ade_rtdefrange
   "COLOR_RANGE" "Color Range Table"
   '(
     ("<" 15 "green")
     ("<" 20 "yellow")
     ("otherwise" "" "red")
   )
  )
 ; Range table for scale is:
 (ade_rtdefrange
   "SCALE_RANGE" "Hatch Scale Range Table"
   '(
     ("<" 15 "25")
     ("<" 20 "50")
     ("otherwise" "" "100")
   )
 )
 ; Range table for Angle is:
 (ade_rtdefrange
   "ANGLE_RANGE" "Hatch Angle Range Table"
   '(
     ("<" 15 "45")
```

```
("<" 20 "90")
     ("otherwise" "" "0")
   )
  )
 ; The layer of each object will be used to determine which value
 ; from the range table is to use
 ; Clear out any existing alteration definitions:
 (ade_altpclear)
 ; For fill, use the following property alteration functions
 (ade_altpdefine
   "hatch"
   '(
     ("pattern" "(range .layer PATTERN_RANGE)")
     ("scale" "(range .layer SCALE_RANGE)")
     ("rotation" "(range .layer ANGLE_RANGE)")
     ("color" "(range .layer COLOR_RANGE)")
   )
 )
 (princ)
)
```

```
(defun c:run_altpsample3 ()
```

```
; Range table for symbol name is:
(ade_rtdefrange
"SYMBOL_RANGE" "Symbol Name Range Table"
'(
  ("<" 15 "Sym1")
  ("<" 20 "Sym2")
  ("otherwise" "" "Sym3")
)
```

```
; Range table for scale is:
(ade_rtdefrange
  "SCALE_RANGE" "Symbol Scale Range Table"
  '(
   ("<" 15 "5")
   ("<" 20 "10")
   ("otherwise" "" "50")
 )
)
; Range table for Color is:
(ade_rtdefrange
  "COLOR_RANGE" "Color Range Table"
  '(
   ("<" 15 "green")
   ("<" 20 "yellow")
   ("otherwise" "" "red")
 )
)
; The layer of each object will be used to determine which
; value from the range table to use
; Clear out any existing alteration definitions:
(ade_altpclear)
; For fill, use the following property alteration functions:
(ade_altpdefine "blockname" "(range .layer SYMBOL_RANGE)")
(ade_altpdefine "scale" "(range .layer SCALE_RANGE)")
(ade_altpdefine "color" "(range .layer COLOR_RANGE)")
```

```
(princ)
```

)

More Samples

There are dozens of Visual LISP samples in the Sample folder of your AutoCAD Map installation.
Using AutoLISP Functions

To write AutoLISP programs, use the Visual LISP IDE. You do not need a compiler, because AutoCAD Map includes an interpreter that processes AutoLISP source code directly.

Optional Parameters

If an AutoLISP function has optional parameters, they are enclosed in square brackets in the function prototype. If a parameter is optional, you can omit its argument if no arguments follow.

A-lists

The term a-list used in AutoLISP function descriptions denotes an association list, also called a dotted pair. It looks like this:

(property . value)

For example, in the expression

```
(ade_altpdefine "textobject"
    '( ("color" . "yellow")
        ("textvalue" . ".Layer") ) )
```

the sub-expression ("color" . "yellow") is an a-list, and so is ("textvalue" .

".Layer").

Invoking Data Extension Commands

You can invoke almost any Data Extension command using the AutoLISP expression, (command "_.ADE[XX]" ...). The exceptions are ADEDEFCRDSYS, ADEZEXTENTS, ADEEDITDATA, and ADEATTACHDATA. You cannot invoke these commands from within ADSRX or Visual LISP. For the ADEKEYVIEW command, only the Redisplay option is supported when ADSRX or Visual LISP is active.

Incomplete Execution of Functions

If an AutoLISP function that performs a sequence of tasks fails partway through its execution and returns an error code, the tasks it completed correctly are undone. For example, <u>ade_qrysetcond</u> modifies a query condition. If you call this function and it is not able to modify every component of the condition, the components it did modify are returned to their original states.

AutoLISP functions work with numeric identifiers (IDs). Their type is real.

For example, when you use <u>ade_dslist</u> to list the drawings attached to the current drawing, the function returns a list of IDs, not drawing names. To get a drawing name, you call <u>ade_dwggetsetting</u> with two arguments: "dwgname" (the setting you want to know about) and the drawing ID.

Topology function documentation uses a number of different kinds of IDs. See <u>Topology IDs</u> for an annotated list. Although these IDs are all the same type, ade_id, we give them different names in the documentation according to their purpose.

Typographic Conventions

This documentation uses the following typographic conventions.

Text element	Description	Example
bold sans serif	Text you enter	At the command prompt, enter (ade_dslist) .
italic	Names of files and directories	c:\map\map_api.hlp
monospace font	API proper names, inline sample code	The expression ("color"."yellow")
[] square brackets	Optional parameters in function prototypes	(tpm_iterstart [source] [loaded])

Typographic Conventions

Note All file names and directory paths in AutoCAD Map are case sensitive.

tpm_cleanactionlistgetat <u>Cleanup Functions</u>

Gets the cleanup action at a given list position.

(tpm_cleanactionlistgetat clean_var_id index action_var_id)

Returns a cleanup action as a clean group type (integer) or 0 on error: see tpm_cleangrouptype for a list of types.

clean_var_id	Cleanup variables ID (real) returned by <u>tpm_varalloc</u>
index	List position to access (integer)
action_var_id	Cleanup action variables ID (real) returned by <u>tpm_varalloc</u>

The **clean_var_id** argument references properties for the cleanup operation that you are preparing to initiate (see <u>Cleanup Variables</u>). These properties include the action list.

The index argument is a zero-based position in the action list. A value greater than or equal to the list size or less than 0 returns an error.

The action_var_id argument references properties affecting the specific cleanup action that you are getting (see <u>Cleanup Action Variables</u>). Use <u>tpm_varget</u> or <u>tpm_varlist</u> to read them after calling tpm_cleanactionlistgetat.

tpm_cleanactionlistins <u>Cleanup Functions</u>

Inserts a cleanup action in the action list.

(tpm_cleanactionlistins clean_var_id index action action_var_id)

Returns T or nil.

clean_var_id	Cleanup variables ID (real) returned by <u>tpm_varalloc</u>
index	Where to insert in the list (integer)
action	Cleanup action to insert (integer), a clean group type: see <u>tpm_cleangrouptype</u> for a list of types.
action_var_id	Cleanup action variables ID (real) returned by tpm_varalloc

The **clean_var_id** argument references properties for the cleanup operation that you are preparing to initiate (see <u>Cleanup Variables</u>). These properties include the action list.

The index argument is a zero-based position in the action list, or -1 for the last position. A value greater than or equal to the list size or less than -1 is taken as -1.

Note When you insert the Simplify Objects action (clean group type 128), it is always listed first, and you cannot insert it more than once.

The action_var_id argument references properties affecting the specific

cleanup action that you are inserting (see <u>Cleanup Action Variables</u>). Use <u>tpm_varset</u> to set them before calling <u>tpm_cleanactionlistins</u>.

ADSRX Equivalent

```
int
tpm_cleanactionlistins
    ade_id clean_var_id,
    long index,
    int action,
    ade_id action_var_id);
```

Returns **RTNORM** or an error code.

tpm_cleangrouptype

Determines the type of the current group.

(tpm_cleangrouptype clean_id)

Returns a type code or nil.

clean_id Model ID returned by <u>tpm_cleanalloc</u>

Type Codes

1	Erase short objects
2	Break crossing objects
4	Extend undershoots
8	Delete duplicates
16	Snap clustered nodes
32	Dissolve pseudo nodes
64	Erase dangling objects
128	Simplify objects
256	Zero length objects
512	Apparent intersections

1024 Weed polylines

tpm_cleancreatedss <u>Cleanup Functions</u>

Gets created entities following a drawing cleanup.

(tpm_cleancreatedss clean_id)

Returns a selection set.

clean_id Cleanup model ID (real), returned by tpm_cleanalloc.

This function returns entities that were created during the cleanup process.

Note Call this function after calling tpm_cleanend, which concludes the cleanup process. If you call this function earlier, it returns a selection set from the previous cleanup or the empty selection set.

tpm_cleanmodifiedss <u>Cleanup Functions</u>

Gets changed entities following a drawing cleanup.

(tpm_cleanmodifiedss clean_id)

Returns a selection set.

clean_id Cleanup model ID (real), returned by <u>tpm_cleanalloc</u>.

This function returns members of the <u>Include</u> set that were changed during the cleanup process.

Note Call this function after calling tpm_cleanend, which concludes the cleanup process. If you call this function earlier, it returns a selection set from the previous cleanup or the empty selection set.

tpm_cleanunchangedss <u>Cleanup Functions</u>

Gets unchanged entities following a drawing cleanup.

(tpm_cleanunchangedss clean_id)

Returns a selection set.

clean_id Cleanup model ID (real), returned by tpm_cleanalloc.

This function returns members of the <u>Include</u> set that were not changed during the cleanup process. Note that it does not return members of the <u>Anchor</u> set, which are unchanged by definition.

Note Call this function after calling tpm_cleanend, which concludes the cleanup process. If you call this function earlier, it returns a selection set from the previous cleanup or the empty selection set.

Calculates the best round-trip route.

(tpm_tracebestroute tpm_id trace_id node0 node1 ... noden)

Returns a topology ID (real) representing the best route or nil.

tpm_id	Topology ID (real) representing the network you are analyzing
trace_id	Tracing model ID (real) returned by tpm_tracealloc
node0	Element ID (real) of the start and end node
node1 noden	Element IDs (real) of the nodes to visit

The best route topology, whose ID this function returns if successful, is assigned an arbitrary name and is open for read. To get its name, use <u>tpm_infoname</u>. To change its name, use <u>tpm_mntrename</u>.

For the best route trace to succeed, the total calculated resistance cannot be greater than the value set for the maximum resistance or less than the value set for the minimum resistance. See <u>tpm_tracesetmaxres</u> and <u>tpm_tracesetminres</u>. The accumulated resistance value is the total resistance of the nodes and links that make up the best route.

Typical usage:

(setq bestroute (tpm_tracebestroute tpm_id trace_id 7.0 1.0 4.0 10.0 5.0 6.0 9.0))

where 7.0 is the element ID of the start and end node and the remaining arguments are element IDs of the nodes to visit. (Element IDs are returned by tpm_traceelemid.)

ADSRX Equivalent

```
ade_id
tpm_tracebestroute
ade_id tpm_id,
ade_id trace_id,
struct resbuf *nodes);
```

Returns a topology ID or ADE_NULLID.

The list of nodes is implemented as a **resbuf** chain, which you can create like this:

```
struct resbuf *nodes; // node list
struct resbuf rb0, rb1, ..., rbN; // resbuf elements
ade_id node0, node1, ..., nodeN; // nodes to analyze
// get tracing element IDs for the nodes
// that we will analyze and assign them to
// node0, node1, ..., nodeN
...
rb0.restype = RTLONG;
rb0.resval.rlong = node0;
rb0.rbnext = &rb1;
rb1.restype = RTLONG;
```

```
rb1.resval.rlong = node1;
rb1.rbnext = &rb2;
...
rbN.restype = RTLONG;
rbN.resval.rlong = nodeN;
rbN.rbnext = NULL;
nodes = &rb0;
```

tpm_tracebestroutescan <u>Network Tracing Functions</u>

Gets the element ID of a link or node in the best route.

(tpm_tracebestroutescan trace_id flag)

Returns an element ID (real) or nil.

trace_idTracing model ID (real) returned by tpm_traceallocflagPath element code (integer):
0 = Current element
1 = First element
2 = Last element
3 = Next element
4 = Previous element

First use <u>tpm_tracebestroute</u> to calculate the best route.

ADSRX Equivalent

ade_id tpm_tracebestroutescan ade_id trace_id, int flag);

Returns a element ID or ADE_NULLID.

tpm_tracebestrouteval Network Tracing Functions

Calculates the resistance of the best route.

(tpm_tracebestrouteval trace_id)

Returns a resistance value (real) or nil.

trace_id Tracing model ID (real) returned by tpm_tracealloc

First use <u>tpm_tracebestroute</u> to calculate the best route.

ADSRX Equivalent

int
tpm_tracebestrouteval
 ade_id trace_id,
 ads_real *resist);

Returns **RTNORM** or an error code.

resist Resistance

The ADSRX function passes the resistance of the best route through a parameter instead of returning it as the AutoLISP function does.

Color Index Colors

Valid color index values are integers from 0 through 256. You can also use color names or the logical colors, ByBlock and ByLayer.

AutoCAD Map also supports true color.

Named Colors

Indexes 1 through 7 are the *named colors*. You can specify any of these colors by name or by index. Index 7, the color named white, displays as white or black depending on background color.

Index	Name
1	Red
2	Yellow
3	Green
4	Cyan
5	Blue
6	Magenta
7	White

Note You can specify ACI colors in query conditions using color names or color indexes. But if you retrieve such a condition, ACI colors are always reported as

color indexes, because that is how they are stored, even if they were originally specified with color names. However, this is not the case with color patterns. If you retrieve a condition using a color pattern, whether it contains color indexes, color names, or both, the color pattern is reported as it was originally specified.

Other Colors

Indexes 8 and 9 (a dark gray and a light gray), together with the named colors, are collectively the *standard colors*. Indexes 0 and 256 (the *logical colors*, ByBlock and ByLayer) invoke the relevant block and layer colors respectively. Note that ByBlock and ByLayer can return ACIs or true colors.

For more information

- 1. In AutoCAD Map, click Format > Color.
- 2. In the Select Color dialog box, click Help.

ade_prefgetval

Gets an AutoCAD Map option setting.

(ade_prefgetval variable)

Returns an option setting or nil.

variable Option name (string). See the Options tables below.

The function return value depends on which option you specify. The tables below show option names and return values, organized by option type.

Work Session Options

RestoreLastActiveDwgsOnStartup	T or nil.
ActivateDwgsOnAttach	T or nil.
DontAddObjectsToSaveSet	T or nil.
MarkObjectsForEditingWithoutPrompting	T or nil.
LogFileActive	T or nil.
LogFileName	File name (string). For example, "ade.log".
LogMessageLevel	0, 1, or 2.

Query Options

QueryFileDirectory.	Path (string). For example, "c:\\data\\qry".
CaseSensitiveMatch.	T or nil.
SaveCurrQueryInSession.	T or nil.
MkSelSetWithQryObj	T or nil.
DefaultJoinOperator	1 = OR, 2 = AND.
ColorForAdd	Color (string).
ColorForRemove	Color (string).
BlockLocnForQuery	1 = insertion point, 2 = bounding box.
TextLocnForQuery	1 = insertion point, $2 =$ bounding box.
ShowBlockAsInsPt	T or nil.
ShowImageAsBoundary	T or nil.
CreateAssociativeHatchObjects	T or nil.
ReferenceBoundaryForAreaLocation	T or nil.

Save Back Options

RedefineBlockDefinitions	T or nil.
RedefineLayerDefinitions	T or nil.
RedefineTextStyleDefinitions	T or nil.
RemoveUnusedGroups	T or nil.

EraseSavedBackObjects	T or nil.
RemoveLockAfterSave	T or nil.
CreateHistoryFileOfChanges	T or nil.
CreateBackupFileOfSourceDwg	T or nil.

External Database Options

NoOfSQLConditionsInHistory	integer.
DisplayTabsInSingleView	T or nil.
OpenDataViewReadOnly	T or nil.
SaveDataViewFmtChanges	T or nil.
ReconnectDbOnWSOpen	T or nil.
ShowFullDBPath	T or nil.
KeepDataViewOnTop	T or nil.
dbfDatabases	string, one of the following: "Prompt", "DB3", "DB4", "DB5", "FOX2.0", "FOX2.5", or "FOX2.6".
xlsDatabases	string, one of the following: "Prompt", "Excel3", "Excel4", "Excel5", or "Excel7".
dbDatabases	string, one of the following: "Prompt", "Paradox3.0", "Paradox4.0", or "Paradox5.0".

Coordinate Transformation Options

AdjustSizesAndScalesForChangesInUnits	T or nil.
AdjustRotationsForMapDistortions	T or nil.
AdjustSizesAndScalesForMapDistortions	T or nil.
AdjustElevations	T or nil.
AdjustZeroRotationObjects	T or nil.

System Options

AccessWorkCenter	T or nil.
CheckoutDirectory	Path (string). For example, "c:\\data\\dwg" or "" if none.
PreserveAWCFiles	T or nil.
ForceUserLogin	T or nil.
EnableObjectLocking	T or nil.
ReadPrefFromINI	T or nil.
NumberofOpenDwgs	integer
DoublePrec	real, 0 or greater, but less than 1.

The "ForceUserLogin" and "DoublePrec" system options cannot be modified unless your end user has superuser privileges.

If "DoublePrec" is set to 0, the behavior of data extension queries is the same as before introducing this option. The "DoublePrec" option has no user interface equivalent.

Workspace Options

CheckClasses	T or nil.
CheckDrawings	T or nil.
CheckQueryLibrary	T or nil.
CheckDatabases	T or nil.
CheckTables	T or nil.
CheckQueries	T or nil.
CheckTopologies	T or nil.
CheckLPNs	T or nil . Note that link path names (LPNs) have been replaced by link templates in AutoCAD Map.
ShowOPMOnStartup	T or nil.
ShowWSpaceOnStartup	T or nil.
WSpaceDockingView	T or nil.
WSpaceWindowRect	A list of four values (integer) that define the left, top, right, and bottom of the window rectangle.

Database tables and database query categories are visible in the workspace only if "CheckTables" and "CheckQueries" are set to T and "CheckDatabases" is set to T also.

The following workspace options are read only. That is, they can be used only with <u>ade_prefgetval</u> to determine if a category is visible in the workspace.

Read-Only Workspace Options

ClassesVisible	T or nil.
DrawingsVisible	T or nil.
QueryLibraryVisible	T or nil.
DatabasesVisible	T or nil.
TablesVisible	T or nil.
QueriesVisible	T or nil.
TopologiesVisible	T or nil.
LPNsVisible	T or nil.

ade_prefsetval

Sets an AutoCAD Map option.

(ade_prefsetval variable value)

Returns T or nil.

variable Option name (string)

value Value appropriate for the given option (type varies).

See <u>ade_prefgetval</u> for a list of option names and values.

The following example sets "ColorForAdd" to "red".

(ade_prefsetval "ColorForAdd" "red")

ade_altpdefine Property Alteration Functions

Creates a property alteration expression.

(ade_altpdefine property value)

Returns a property alteration expression ID or nil.

property Property to alter (string). See the Alterable Properties table below
 value New value (type varies), or a range table expression (string) that determines the new value. See <u>Using a Range Table</u> later in this topic.

A list of one or more property alteration expressions constitutes a property alteration definition. If there is a current property alteration definition when you create a property alteration expression, the new expression is added to it. When you execute a Draw query, each queried entity is altered in accord with the current property alteration definition.

The following table lists the alterable properties:

Alterable Properties

blockname	Block name (string)
color	Color (string)
elevation	Z coordinate (point) in the user coordinate system

height	Text height (<mark>real</mark>)
layer	Layer name (string)
linetype	Line type (string)
rotation	Rotation (real)
scale	Scaling factor (real). For example, 1.2 = 120%
style	Text style (string)
width	Line width (real)
textvalue	Text value (string)
thickness	Thickness (real)
hatch	List of dotted pairs that define the hatch properties. See Hatch properties below
textobject	List of dotted pairs that define the text object properties. See Text object properties below

To add a hatch pattern to each queried entity, as long as it is a closed polygon, specify "hatch" for the property argument. The value argument is then a list of dotted pairs. Each dotted pair is composed of a hatch property and a string value.

Hatch Properties

pattern	Hatch pattern name (string)
scale	Scaling factor (string). For example, "1.2" = 120%
rotation	Rotation of the hatch pattern (string)
layer	Name of the layer that contains the hatch pattern (string)

color Hatch pattern <u>color</u> (string)

To create a text object for each queried entity, specify "textobject" for the **property** argument. The **value** argument is then a list of dotted pairs. Each dotted pair is composed of a text object property and a string value. The value element in the dotted pair can be an explicit value or a range table expression that determines a value.

Text Object Properties

textvalue	Text to display (string)
height	Text height (string)
inspt	Point where text is inserted (expression as a string)
justify	Text alignment (string). For example, "center".
style	Text style (string)
layer	Name of the layer on which the text object resides (string)
color	Text <u>color</u> (string)
rotation	Rotation of the text object (string)

The property alteration expression defined by

(ade_altpdefine "color" "red")

changes the color of each queried entity to red.

The property alteration expression defined by

(ade_altpdefine "textobject"
 '(("color" . "yellow") ("textvalue" . ".Layer")))

creates a text object for each queried entity. Each text object is yellow, and its text value is the layer on which the entity resides.

A longer property alteration expression for a text object:

```
(ade_altpdefine "textobject"
(list
  (cons "Textvalue" ".Layer")
  (cons "Justify" "MIDDLE")
  (cons "Inspt" ".CENTER")
  (cons "Style" "STANDARD")
  (cons "Height" "1.0")
  (cons "Rotation" "0.0")
  (cons "Color" "BYLAYER")
))
```

A property alteration expression for a hatch:

```
(ade_altpdefine "hatch"
(list
  (cons "Pattern" "USER")
  (cons "Scale" "1.0")
  (cons "Rotation" "45.0")
  (cons "Layer" "Query_Hatch")
  (cons "Color" "BYLAYER")
))
```

Using a Range Table

Instead of supplying an explicit value argument when you call ade_altpdefine, you can supply a range table expression that references an existing range table. Note that this expression is a string value. It must be enclosed in quotes.

A range table expression has the following format:

(range reference rtname)

Range	Table	Expression	Parameters
	1010		1 un uniteter o

range	The range keyword. All range table expressions begin with range . It is not quoted.
reference	Reference property, such as .Color or .Layer . It is not quoted.
rtname	Range table name. Can be up to 31 characters long. Must be unique, contain no spaces, and start with an alphanumeric character. It is not quoted.

The range table expression uses its included range table to process the value of the reference property and return a new property value. For example, the following code (1) uses <u>ade_rtdefrange</u> to define a range table, (2) references the range table in a range table expression, and then (3) supplies the range table expression as the value argument in an <u>ade_altpdefine</u> call.

First we define a range table:

(ade_rtdefrange "rt_def" "Change all except red to yellow" '(("=" 1 1) ("OTHERWISE" "" 2)))

Then we reference this range table in a range table expression, which asserts that the range table will look at the **.Color** property of each queried entity to determine if the entity's color will be altered and what color it will be.

```
(setq propVal "(range .Color rt_def)")
```

Finally, we use the range table expression instead of an explicit property alteration value in a call to ade_altpdefine.

```
(ade_altpdefine "color" propVal)
```

The next time a Draw query is executed with Property Alteration in effect, the color of each queried entity is altered depending on its current color and in accordance with the rules embedded in the range table and its enclosing range table expression.

The following example uses real values:

(ade_rtdefrange "rt_def" "Set rotation" '(("=" 45. 90.) ("OTHERWISE" "" 45.)))

For another example, you could rewrite the "textobject" example cited earlier to use a range table.

```
(ade rtdefrange "labelWaterOnly" ""
    '(("=" "Water" "Water")("OTHERWISE" "" ""))
)
(ade_altpdefine "textobject"
    '(("color" . "yellow")
        ("textvalue" . "(range .Layer myRangeTable)"))
)
```

The following example executes a location query based on a circle defined by the user. It includes property alteration based on a range table. The color of each queried entity, if it is not already red, is changed to yellow.

```
(ade_qryclear)
(ade_qrysettype "draw")
(ade_dwgzoomextents)
```

```
(prompt "\nQuery LOCATION by CIRCLE: ")
(setq c_cen (getpoint "\nCenter of circle: ")
   c_radp (getpoint c_cen "\nRadius of circle: ")
   c_rad (distance c_cen c_radp)
   qry_cond (list "circle" "inside" c_cen c_rad)
   qry_id (ade_qrydefine "" "" ""
             "location" qry_cond "")
)
(if (null qry_id)
 (prompt "\nERROR: Query definition failed.")
 (progn
   (ade_altpclear)
   ; Define the range table
   (ade_rtdefrange "rt_def"
       "Change all except red to yellow"
       '(("=" 1 1) ("OTHERWISE" "" 2)))
   ; Reference the range table in a range table
   ; expression; note that it's all one string
   (setq propVal "(range .Color rt_def)"
   (if (or (null (ade_altpdefine "color" propVal))
           (null (ade_qrysetaltprop T))
       )
     (prompt "\nERROR: Alter properties definition
             failed.")
     (if (= 0.0 (ade_qryexecute))
       (prompt "\nERROR: No objects found.")
     ); if
   ); if
 ); progn
); if
```

ade_altpgetprop Property Alteration Functions

Gets a property alteration expression.

(ade_altpgetprop altp_id)

Returns a property alteration expression ID or nil.

altp_id Property alteration expression ID (real)

See <u>ade_altpdefine</u> for information about property alteration expressions.

ade_altpsetprop Property Alteration Functions

Modifies a property alteration expression.

(ade_altpsetprop altp_id property value)

Returns T or nil.

altp_idProperty alteration expression ID (real)propertyProperty to alter (string)valueNew value (type varies).

See <u>ade_altpdefine</u> for information about properties and values.

ade_qrydefine Query Functions

Defines a query.

(ade_qrydefine joinop bggroups not_op condtype qrycond endgroups)

Returns a condition ID or nil.

joinop	A joining operator: "and" or "or" or "" (none). If "" (none) is specified, the default joining operator is used (see <u>ade_prefgetval</u>).
bggroups	For grouping this condition with others in the query definition you are building. Use one or more open parentheses as needed, or "" (none). For example, "((".
not_op	The NOT operator, if needed: "not" or "" (none).
condtype	A condition type: "Location", "Property", "Data", or "SQL".
qrycond	A condition expression. Depends on the condition type. See <u>Condition Expressions</u> below.
endgroups	For grouping this condition with others in the query definition you are building. Use one or more close parentheses as needed, or "" (none). For example, "))".

A query definition is composed of one or more conditions, each defined by a separate ade_qrydefine call. You can group conditions by supplying
parentheses or empty strings to the **bggroups** or **endgroups** parameters as needed.

You must specify all six ade_qrydefine arguments.

Condition Expressions

The **qrycond** parameter requires a condition expression. Condition expressions are lists. What you include in the list depends on the condition type: Location, Property, Data, or SQL.

Location Expressions Property Expressions Data Expressions SQL Expressions

ade_qrygetcond

Gets a condition of the current query.

(ade_qrygetcond condition_id)

Returns a query condition or nil.

condition_id Query condition ID (real)

See <u>ade_qrydefine</u> for information about query conditions.

ade_qrysetcond

Replaces a query condition.

(ade_qrysetcond condition_id condition)

Returns T or nil.

condition_id Query condition ID (real) to replace.

condition New query condition (a list). See <u>ade_qrydefine</u>.

This function affects the current query.

You cannot alter grouping with this function. Any grouping you specify is ignored. To group or ungroup, use <u>ade_qrygroup</u> or <u>ade_qryungroup</u>.

Property Expressions

Property expressions are used as querycond arguments in ade_querydefine calls that define Property conditions.

They have the following format:

(property operator value [subclasses])

Property Expression Parameters

property	Property name (string). See the Property and Value Arguments table below.
operator	"=", ">", "<", "<=", ">=", "<>". Note that the only valid operator in a string context is "=".
value	Depends on the property argument. See the Property and Value Arguments table below.
subclasses	Optional. T or nil. The default if the argument is omitted is nil. This setting has no effect unless property is "feature". T means return all objects belonging to the feature class identified by the value argument, including objects belonging to any subclass of that feature. nil means do not include objects belonging to such a subclass.

Property and Value Arguments

Property	Value
area	Area value (string).
blockname	Block name (string).
color	<u>Color</u> (string).
elevation	Z coordinate (string).
"feature"	Feature name (string).
group	Group name (string).
layer	Layer name (string).
length	Length (<mark>string</mark>).
linetype	Line type (<mark>string</mark>).
"lineweight"	Line weight (string).
"plotstyle"	Plot style (string).
style	Text style (string).
thickness	Thickness (string).
objtype	Object type (string), or "unknown".
value	Text value (<mark>string</mark>).

Property Expression Examples

The following examples define Property conditions. The first specifies a layer:

))

And the second specifies a color:

Data Expressions

Data expressions are used as querycond arguments in ade_querydefine calls that define Data conditions.

They have the following format.

(datatype tablename.fieldname operator value [subclasses])

Data Expression Parameters

datatype	Data type to match (string): "objdata", "attrib", "aselink", "EED", or "feature".
tablename	Depends on the datatype argument. See the Tablename And Fieldname Arguments table below.
fieldname	Depends on the datatype argument. See the Tablename And Fieldname Arguments table below.
operator	Comparison operator (string): "=", ">", "<", "<=", ">=", or "<>". Note that the only valid operator in a string context is "=".
value	Value to match.
subclasses	Optional. T or nil. The default if the argument is omitted is T. This setting has no effect unless datatype is "feature". T means return objects belonging to the feature class

identified by the tablename argument, including objects belonging to any subclass of that feature. nil means do not include objects belonging to such a subclass.

The tablename and fieldname arguments depend on the datatype argument:

datatype	tablename	fieldname
objdata	Table name.	Field name.
attrib	Block name.	Attribute definition.
aselink	Link template.	Column name.
EED	RegApp name.	EED field name.
feature	Feature Class name.	Property name.

Tablename and Fieldname Arguments

Data Expression Examples

The following examples define Data conditions of various types.

```
(ade_qrydefine
    '("" "" "Data"
    ("attrib" "*.Type" "=" "c*")""))
(ade_qrydefine
    '("" "" "Data"
    ("objdata" "mytable.fl1" "=" "1")""))
(ade_qrydefine
    '("" "" "Data"
    ("aselink" "cpu_lpn.cpu" "=" "MAC2LC")""))
```

```
(ade_qrydefine
    '("" "" "Data"
    ("EED" "REGAPP.STREET" "=" "Willow")""))
(ade_qrydefine
    '("" "" "Data"
    ("EED" "REGAPP.#NUMBER" "=" "512")""))
(ade_qrydefine
    '("" "" "Data"
    ("EED" "REGAPP.&SQLLINK;" "=" "MAC2LC")""))
```

Note in the last example that **&**SQLLINK is not treated as a link template key. It is treated just like any other ADE 1.0 EED field. For example, if an object has EED such as

(-3 (1000 . "&EEDFIELD;" = "1234"))

then the Data condition to retrieve the object is written as follows:

```
("EED" "REGAPP.&EEDFIELD;" "=" "1234")
```

If the EED is defined by

(-3 (1000 . "&EEDFIELD;" = "'First', 'Last'"))

then the Data condition to retrieve the object is written as follows:

("EED" "REGAPP.&EEDFIELD;" "=" "'First', 'Last"')

In other words, everthing after the equal sign is treated as one string. That way you can use any pattern (for wcmatch) in the query.

ade_aliasadd Drive Alias Functions

Creates a drive alias.

(ade_aliasadd alias_name path_name)

Returns T or nil.

- alias_name Alias name (string). Can be up to 31 characters long. Must be unique, contain no spaces, and start with an alphanumeric character.
- path_name Drive and path to which the alias refers (string).

The function adds a new drive alias to the drive alias list. For example:

(ade_aliasadd "tutor" "c:\\tutorials\\drawings")

ade_aliasdelete

Deletes a drive alias.

(ade_aliasdelete alias_name)

Returns T or nil.

alias_name Alias name (string); can be up to 31 characters long. Must be unique, contain no spaces, and start with an alphanumeric character.

ade_aliasgetlist Drive Alias Functions

Lists all drive aliases in the project.

(ade_aliasgetlist)

Returns the list of drive aliases or nil

The list of drive aliases is a list of a-lists. Each a-list displays the alias name and the drive and path to which it refers.

ade_aliasupdate Drive Alias Functions

Changes the actual path of a drive alias.

(ade_aliasupdate alias_name path_name)

Returns T or nil.

alias_name Alias name (string) can be up to 31 characters long. Must be unique, contain no spaces, and start with an alphanumeric character.

path_name New actual path for this drive alias (string).

ade_altpclear Property Alteration Functions

Clears the current property alteration definition .

(ade_altpclear)

Returns T or nil.

A property alteration definition is a list of one or more property alteration expressions. See ade_altpdefine for information about property alteration expressions.

ade_altpdelprop Property Alteration Functions

Deletes a property alteration expression.

(ade_altpdelprop altp_id)

Returns T or nil.

altp_id Property alteration expression ID (real)

ade_altplist Property Alteration Functions

Lists the IDs of the current property alteration expressions.

(ade_altplist)

Returns a list of property alteration IDs or nil.

This list of property alteration expressions returned by this function constitutes the current property alteration definition.

ade_dsattach <u>Drawing Set Functions</u>

Attaches a drawing to the project drawing.

(ade_dsattach dwgname)

Returns the ID of the attached drawing or nil.

dwgname Path alias and file name (string)

This function returns an ID even if the drawing does not exist. A system administrator can use this function to define a drawing set before the drawing files it references are created or installed. A drawing must exist before you can make it active.

```
(ade_aliasadd "dwg_drive" "d:\\myproject\\mydir")
(setq dwg_id
    (ade_dsattach "dwg_drive:\\mydrawing.dwg")
)
; check if drawing was successfully attached
(if dwg_id ; check if it returned an ID
    (princ "\nSuccessfully attached.")
    (princ "\nDid not attach.")
)
```

ade_dsdetach Drawing Set Functions

Detaches a drawing from the project.

(ade_dsdetach dwg_id)

Returns T or nil.

dwg_id Drawing ID to detach (real)

The following code detaches the drawing attached in the ade_dsattach example:

```
(if (ade_dsdetach dwg_id)
 ; check if it returned T
  (princ "\nSuccessfully detached.")
  (princ "\nDid not detach.")
)
```

ade_dsisnested Drawing Set Functions

Checks if a drawing has nested drawings .

```
(ade_dsisnested [dwg_id])
```

Returns **T** if the drawing has drawings attached, or **nil**.

dwg_id Drawing ID (real)

Verify that the drawing in question is active before calling ade_dsisnested. It is not possible to determine if an inactive drawing has nested drawings. If dwg_id is not specified or is nil, the function checks the project drawing to see if it has drawings attached.

```
; Get IDs of drawings currently attached
(setq ds_ids (ade_dslist))
; See if the first one has nested drawings
(princ "\nFirst drawing has "
(if (ade_dsisnested (car ds_ids))
(princ "drawings attached.")
(princ "no drawings attached.")
)
```

ade_dslist <u>Drawing Set Functions</u>

Lists the drawings attached to a given drawing.

(ade_dslist [dwg_id [nested]])

Returns a list of drawing IDs or nil.

dwg_id	Drawing ID (real) or nil
nested	Drawing is nested or not. Values: T or nil

If the dwg_id argument is omitted or nil, the function returns drawing IDs for the drawings attached to the project.

If you supply a **nested** argument other than **nil**, the function includes drawings that are directly attached and all nested drawings at every level below them. Otherwise, it includes only drawings that are directly attached.

(setq ds_ids (ade_dslist))

ade_dsproplist Drawing Set Functions

Lists all values found in the drawing set for a given drawing property.

(ade_dsproplist property)

Returns a list of values or nil.

property Drawing property (string). See Drawing Properties below.

The function searches all active source drawings and returns a list of the values it finds for the given drawing property.

The following table shows property names and return values.

Drawing Properties

object_type	AutoCAD object types (string)
blockname	Block names (string)
linetype	Line type names (string)
textstyle	Text style names (string)
attrib	Attribute name tags (string)
extents	Computed extents: the lower-left and upper-right points in the set of active source drawings. For example: ((2.20286 4.99866) (20.4689 12.3563))

group	Group names (string)
layer	Layer names (string)
lpn	Link templates (<mark>string</mark>). Note that link path names (LPNs) have been replaced by link templates in AutoCAD Map.
objdata	Names of object data tables. Table names can be up to 25 characters long (string). Must be unique, contain no spaces, and start with an alphanumeric character
mlinestyle	Mline style (string)
feature	Feature name (string)
lineweight	Line weight (string)
plotstyle	Plot style (string)

This code returns a list of layers in the drawing set.

(ade_dsproplist "layer")

ade_dwgactivate

Activates a drawing.

(ade_dwgactivate dwg_id)

Returns **T**, if the drawing is already active, or **nil**.

dwg_id Drawing ID (real)

You can attach a drawing that does not yet exist, but you cannot activate it. See <u>ade_dsattach</u>.

ade_dwgactualpath

Returns the actual path for a drawing.

(ade_dwgactualpath dwg_id)

Returns the full path (without an alias) of the specified drawing or nil.

dwg_id Drawing ID (real)

ade_dwgaliaspath

Returns the alias path for a drawing.

(ade_dwgaliaspath dwg_path)

Returns the alias path of the specified drawing or nil.

dwg_path Actual path of the drawing (string)

ade_dwgattriblist

Drawing Functions

Returns a list of attribute tags for the specified block name .

(ade_dwgattriblist dwg_id block_name)

Returns a list of attribute tags or nil.

dwg_idDrawing ID (real)block_nameBlock name for which to get attribute tags (real)

This function returns a list of the attribute tags, given a block name from the specified drawing.

ade_dwgdeactivate

Deactivates a drawing.

(ade_dwgdeactivate dwg_id)

Returns T or nil.

dwg_id Drawing ID (real)

This code deactivates all the drawings in the drawing set:

(foreach dwg_id (ade_dslist) (ade_dwgdeactivate dwg_id))

This code uses the mapcar function to deactivate the drawings in the drawing set.

(mapcar 'ade_dwgdeactivate (ade_dslist))

ade_dwggetid

Gets the drawing ID of a drawing.

(ade_dwggetid dwg_pathname)

Returns a drawing ID (real) or nil.

dwg_pathname Path alias and drawing file name (string)

Code example:

(ade_aliasadd "mydwgs" "d:\\myproject\\mydrawing")
(setq dwg_id
 (ade_dwggetid "mydwgs:\\mydrawing.dwg"))

ade_dwggetsetting Drawing Functions

Gets a drawing setting value.

(ade_dwggetsetting dwg_id setting)

Returns the value of the given drawing setting or nil.

dwg_id	Drawing ID (real)
--------	----------------------------

settingDrawing setting name (string).See Drawing Setting Names below

Drawing Setting Names

Setting Name	Return Value
dwgname	Drawing name (string); a path alias and file name, such as "myfiles:\\mydwg.dwg"
dwgdesc	Drawing description (string)
t_scale	Simple transform scale (real . For example, 1.2 = 120%
t_rotate	Simple transform rotation direction. Value depends on the AutoCAD ANGDIR setting (real). Values: 0 = counterclockwise 1 = clockwise

t_xoffset	Simple transform X offset (real)
t_yoffset	Simple transform Y offset (real)
t_apply	Flag value (integer). Values: apply all simple transformations defined for the given drawing a do not apply transformations
saveback	Save back coordinates, a sequence of corner points, in this order: lower left, lower right, upper right, upper left, separated by "."

The following code gets the name of the first drawing in the list of attached drawings:

(setq dwg_id (car (ade_dslist)))
(ade_dwggetsetting dwg_id "dwgname")

The return value is a drawing path name, for example,

"c:\\drawings\\mydwg.dwg"

ade_dwghaslocks

Checks if a drawing has locked objects.

(ade_dwghaslocks dwg_id)

Returns **T** if the drawing has locked objects, or **nil**.

dwg_id Drawing ID (real)

ade_dwgindex Drawing Functions

Creates or removes indexes for a given set of drawings.

```
(ade_dwgindex dwgIds)
```

Returns T or nil.

dwgID List of drawing IDs for which indexes will be created.

Calls to <u>ade_dwgindexdef</u> add index operations to the index operation list. Calling <u>ade_dwgindex</u> executes the index operations in the list. If the index operation list is empty, <u>ade_dwgindex</u> has no effect and returns <u>nil</u>.

The following example creates location and property indexes for each attached drawing:

```
; clear the index operations list
(ade_dwgindexdef nil)
; add a 'create location index' operation to the list
(ade_dwgindexdef "location" 1)
; add a 'create property index' operation to the list
(ade_dwgindexdef "property" 1)
```

; execute the given index operations for each attached drawing (mapcar 'ade_dwgindex (ade_dslist))

ade_dwgindexdef

Drawing Functions

Adds operations to the index operations list.

(ade_dwgindexdef indextype [indexoper] [indexparams])

Returns T or nil.

indextype	Type of index (string): "location", "property", "eed", "sqllinks", "objdata", or nil, where nil means remove all.
indexoper	Operation (integer): $1 = \text{create}, 0 = \text{remove}$. Omit this argument if indextype is nil.
indexparams	List of object-data tables and fields to include in the index if indextype is " objdata ", or nil to include all. Add this argument only if indextype is " objdata ".

Calls to ade_dwgindexdef add operations to the index operations list. Calling <u>ade_dwgindex</u> executes the index operations in the list. If the index operation list is empty, ade_dwgindex has no effect and returns nil.

Examples

(ade_dwgindexdef nil) Remove all indexes.

(ade_dwgindexdef "location" 1) Create a location index.

(ade_dwgindexdef "location" 0) Remove the location index.

(ade_dwgindexdef "property" 1) Create a property index.

(ade_dwgindexdef "property" 0) Remove the property index.

Object Data Examples

(ade_dwgindexdef "objdata" 1 nil) Create an object data index that includes all object data.

(ade_dwgindexdef "objdata" 0 nil) Remove all object data from the object data index.

(ade_dwgindexdef "objdata" 1 '(("TABLE"))) Create an object data index that includes all fields in TABLE.

(ade_dwgindexdef "objdata" 0 '(("TABLE"))) Remove all fields in TABLE from the object data index.

More Object Data Examples

(ade_dwgindexdef "objdata" 1 '(("TABLE" "FIELD1")))
(ade_dwgindexdef "objdata" 0 '(("TABLE" "FIELD1")))
(ade_dwgindexdef "objdata" 1 '(("TABLE" "FIELD1" "FIELD2")))
(ade_dwgindexdef "objdata" 0 '(("TABLE" "FIELD1" "FIELD2")))
(ade_dwgindexdef "objdata" 0 '(("TABLE1")("TABLE2" "FIELD1"))
)
ade_dwgisactive

Checks if a drawing is active.

(ade_dwgisactive dwg_id)

Returns T or nil.

dwg_id Drawing ID (real)

The function returns T if the specified drawing is active. If the drawing is not active or the drawing ID is invalid, the function returns nil.

ade_dwgistoplevel <u>Drawing Functions</u>

Checks if a drawing is directly attached to the project drawing.

(ade_dwgistoplevel dwg_id)

Returns **T** if the drawing is attached directly to the project drawing, or **nil**.

dwg_id Drawing ID (real)

ade_dwgproplist

Lists all values found in a drawing for a given drawing property.

(ade_dwgproplist dwg_id property)

Returns a list of the values for the drawing property, or nil.

dwg_id	Drawing ID (real)
property	Property name (string). See Property Names below

The function searches the given drawing and returns a list of the values it finds for the given drawing property.

Property Names

Name	Return Value
object_type	AutoCAD object types (string)
blockname	Block names (string)
linetype	Line type names (string)
textstyle	Text style names (string)
attrib	Attribute tag names (string)
extents	Computed extents. The most lower-left point and the most

	upper-right point in the drawing For example: ((2.20286 4.99866) (20.4689 12.3563))
group	Group names (string)
layer	Layer names (string)
lpn	Link templates (<mark>string</mark>). Note that link path names (LPNs) have been replaced by link templates in AutoCAD Map
objdata	Names of object data tables (string)
mlinestyle	Mline style (<mark>string</mark>)
feature	Feature name (string)
lineweight	Line weight (string)
plotstyle	Plot style (string)

The following code identifies the last drawing attached to the project drawing and returns a list of its layers.

(setq dwg_id (last (ade_dslist)))
(ade_dwgproplist dwg_id "layer")

The returned list has the form

("water" "sewer" "electric")

ade_dwgquickview <u>Drawing Functions</u>

Displays a quick view of a drawing.

(ade_dwgquickview dwg_id)

Returns T or nil.

dwg_id Drawing ID (real)

The following code displays quick views of all drawings in the project.

```
(foreach dwg_id (ade_dslist)
  (if (not (ade_dwgquickview dwg_id))
      (princ (strcat "\nProblem viewing drawing: "
            (ade_dwggetsetting dwg_id "dwgname") )
      )
    )
)
```

ade_dwgselectdlg <u>Drawing Functions</u>

Displays the Select Drawings dialog box.

(ade_dwgselectdlg [parent [caption]])

Returns a list of selected drawings, each represented by its alias path (string), or nil if no drawings are selected.

- parentInteger value defining the pointer to the Select Drawings dialog
box parent window, which is expected to be represented by a
CWnd object. 0 means that the parent window is not defined.
- CaptionA string that is shown before the current directory in the caption
of the Select Drawings dialog box.

ade_dwgsetof

Identifies the drawings to which a given drawing is attached.

(ade_dwgsetof dwg_id)

Returns a list of drawing IDs (real) or nil.

dwg_id Drawing ID (real)

You cannot use this function to check if a drawing is attached to the project drawing. Use <u>ade_dwgistoplevel</u> instead. If a drawing is attached to both the project drawing and to other drawings, this function returns a list of the IDs of the other drawings only.

ade_dwgsetsetting

Drawing Functions

Sets a drawing setting value.

(ade_dwgsetsetting dwg_id proplist)

Returns **T** if successful and the drawing is active, or **nil**.

|--|

proplist List composed of a setting name and a value (string). See Setting Names below.

Setting	Names
---------	-------

Setting name	Value
dwgname	Drawing name (string); a full path name, such as c:\\drawings\\mydwg.dwg.
dwgdesc	Drawing description (string)
t_scale	Simple transform scale (real). For example, 1 . 2 = 120%
t_rotate	Simple transform rotation (real); rotation direction depends on the AutoCAD ANGDIR setting
t_xoffset	Simple transform X offset (real)

t_yoffset	Simple transform Y offset (real)
t_apply	Flag value (integer). Values: apply all simple transformations defined for the given drawing apply transformations
saveback	Save back coordinates, a sequence of corner points, in this order: lower left, lower right, upper right, upper left, separated by "."

The following code sample uses the "saveback" value of the proplist parameter—a sequence of corner points: lower left, lower right, upper right, upper left.

(setq dwg_id (car (ade_dslist))) (ade_dwgsetsetting dwg_id '(("dwgdesc" . "Sample Drawing Description"))) (ade_dwgsetsetting dwg_id '(("saveback" . ((2.20286 4.99866) (20.4689 4.99866) (20.4689 12.3563) (2.20286 12.3563)))))

ade_dwgunlock

Drawing Functions

Removes all object locks from a drawing.

(ade_dwgunlock dwg_id)

Returns T or nil.

dwg_id Drawing ID (real)

Using this function requires superuser privileges.

ade_dwgzoomextents

Drawing Functions

Zooms to the extents of the active drawings .

(ade_dwgzoomextents)

Returns T or nil.

ade_editdefcen Object Editing Functions

Defines a new label point for an object.

(ade_editdefcen ename pt)

Returns T or nil.

ename AutoCAD entity name.pt Label point, a list of real values defined in 2D or 3D point (point)

Use this function with property alteration if the current label point is not suitable for the text object you are adding.

The following example sets the label point of the last object to (5, 5).

```
(setq pt1 '(5 5))
(setq ent1 (entlast))
(ade_editdefcen ent1 pt)
(ade_expreval (entlast) ".labelpt" "point")
```

ade_editlockederased

Object Editing Functions

Gets the objects in the save set that have been erased.

```
(ade_editlockederased)
```

Returns a selection set or **nil**.

These are objects that were erased in the project drawing and are now queued for save back to source drawings.

(setq ss_erased (ade_editlockederased))

ade_editislocked

Object Editing Functions

Gets lock information about an object if it is locked.

(ade_editislocked ename)

Returns a list of lock information about the specified object if locked, or if not locked, then nil.

ename AutoCAD entity name.

The list of lock information returned by this function contains the following strings, in order:

- Login name of the user who locked the object.
- Name and path of the drawing that contains the object.
- Date the object was locked.
- Time the object was locked.
- Name and path of the project drawing.

For example:

```
( "login"
    "c:\\path\\drawing.dwg"
    "7/1/2000"
    "9:58:36 AM"
    "c:\\path\\project.dwg" )
```

You can change the format of the date and time strings through options in the International dialog box in the Microsoft Windows Control Panel.

ade_editlocked Object Editing Functions

Gets the objects in the save set that have been modified or are new.

```
(ade_editlocked)
```

Returns a selection set or nil.

These are objects that were modified in the project drawing or added to it and are now queued for save back to source drawings.

(setq ss_modified (ade_editlocked))

ade_editlockobjs Object Editing Functions

Locks a set of objects and adds them to the save set.

(ade_editlockobjs sel_set)

Returns the number of objects locked (real) or nil.

sel_set Selection set name.

The function locks the objects contained in the designated selection set. Locking these objects adds them to the save set.

It is a good idea to compare the number of objects locked with the number of objects in the designated selection set. If the number locked is less than the number in the selection set, an error occurred in the locking process, and you should check the error stack.

The following example creates a selection set, adds its object to the save set, and checks the result.

```
(entmake '(
(0."circle")
(62.1)
(1012.02.00.0)
(40.1.0)))
; Get the new entity.
(setq e (entlast))
; Create a selection set containing e.
```

```
(setq ss (ssadd e))
; Check how many objects in ss.
(setq num_tolock (sslength ss))
; Lock the objects in ss and get the number locked.
(setq num_locked (fix (ade_editlockobjs ss)))
    ; Fix truncates the real return value of
    ; ade_editlockobjs.
(if (equal num_tolock num_locked)
    (progn
        (princ "\nObjects locked "
        (princ "and added to save set: ")
        (princ num_locked))
    (princ "\nUh-oh")
)
```

ade_editnew Object Editing Functions

Gets the objects in the saved set that are new.

```
(ade_editnew)
```

Returns a selection set or nil.

These are objects that were added to the project drawing and are now queued for save back to source drawings.

(setq ss_modified (ade_editnew))

ade_editunlockobjs Object Editing Functions

Unlocks a set of objects and removes them from the save set.

(ade_editunlockobjs sel_set)

Returns the number of objects unlocked (real), or nil.

sel_set Selection set name.

The function unlocks the objects in the specified selection set. If the selection set is **nil**, the function unlocks all erased objects. Unlocking objects removes them from the save set.

ade_entsetlocation

Other Functions

Sets a new entity label point

(ade_entsetlocation ename pt)

Returns T or nil

ename AutoCAD entity name.

pt New label point location, a 2D or 3D point (point)

This function sets a new label point for an object. An object's label point is the starting position for text added during a query property alteration. By default, the centroid of the object is the label point.

This function sets the label point as defined by the point argument.

ade_errclear Error Message Functions

Clears the error stack.

(ade_errclear)

Returns T or nil.

ade_errcode Error Message Functions

Gets the error code for a given error in the stack .

(ade_errcode err_index)

Returns an error code (integer) or nil.

err_index Position of the error in the stack (integer), where 0 = first error

ade_errgetlevel Error Message Functions

Gets the system error level.

(ade_errgetlevel)

Returns an error level (integer) or nil.

The system error level determines which <u>error types</u> are pushed to the stack.

Error Levels

0	All errors are pushed.
1	All errors except warnings and diagnostics are pushed (their types are listed below).
~	

2 No errors are pushed.

The following error types are suppressed if the error level is 1.

Suppressed Error Types

01	kAdeWarning	ADE (AutoCAD Data Extension) execution warning.
03	kAseWarning	ASE (AutoCAD SQL Extension) execution warning.
05	kAcWarning	AutoCAD execution warning.
07	kAsiWarning	ASI (AutoCAD SQL Interface) execution warning.

09	kIRDWarning	Extended object data (Xdata) warning.
16	kDiagMessage	Diagnostic message returned.

Errors that are not pushed to the error stack are not displayed in the error dialog in the user interface, and they are not accessible by any error message function.

The error level managed by ade_errgetlevel and ade_errsetlevel has nothing to do with the error level managed by ade_prefgetval and ade_prefsetval (the work session preference, LogMessageLevel). The latter affects which types of message are written to the log file.

The error level is not saved when a session ends. When a session begins, the error level is always 0.

ade_errmsg Error Message Functions

Gets the error message for a given error in the stack .

(ade_errmsg err_index)

Returns an error message (string) or nil.

err_index Position of the error in the stack (integer), where 0 = first error.

ade_errpush Error Message Functions

Pushes an error to the stack .

(ade_errpush [err_code] [level] message)

Returns T or nil.

err_code	Error code (details below).
level	Error level (string). Values: "warning", "error" (default), or the empty string. If the empty string, the error level is "error".
message	Error message (string).

If your application will use custom error codes, define a range for them that does not conflict with any range used for AutoCAD Map <u>error codes</u>. To specify a general error, let the <u>err_code</u> argument be 1 (kAdeErr).

The valid **level** values, "warning", and "error", correspond respectively to the following error types:

- 1 kAdeWarning
- 2 kAdeError

For a list of all error types, including the two that are valid **level** values, see <u>Error Types</u>.

The following example pushes an error to the stack.

(ade_errpush 1 "error" "message text")

ade_errpushstatement

Error Message Functions

Pushes a faulty SQL statement to the error stack.

(ade_errpushstatement statement position)

Returns T or nil.

- statement Faulty statement that caused the error (string)
- positionStarting position of the error in the faulty statement (integer).Position 1 corresponds to the first character.

This function is designed to add diagnostic information to an error you have just pushed. It is associated with the latest error in the stack only. A call to ade_errpushstatement makes sense only if a call to ade_errpush immediately precedes it.

ade_errqty Error Message Functions

Returns the number of errors in the stack.

(ade_errqty)

Returns an error count (integer) or nil.

ade_errsetlevel Error Message Functions

Sets the system error level.

(ade_errsetlevel level)

Returns T or nil.

level Error level (integer): 0, 1, or 2

See <u>ade_errgetlevel</u> for details.

ade_errshowdlg Error Message Functions

Displays the Map Messages dialog box.

```
(ade_errshowdlg)
```

```
Returns T or nil.
```

If there are no errors or messages in the error stack, the dialog box does not display, and the function returns nil.

ade_errstatement

Error Message Functions

Gets the faulty SQL statement for a given error in the stack .

(ade_errstatement err_index)

Returns the faulty SQL statement with error position or nil.

err_index Position of the error in the stack (integer) 0 = first error

The SQL statement and error position are returned in the following format.

```
(faulty_statement err_pos)
```

The **faulty_statement** string quotes the faulty SQL statement that caused the error. The **err_pos** value identifies the starting position of the error in the faulty statement. Position **1** is the first character of the statement.

The expression (ade_errstatement 2), which references the third error in the stack, could return

("xxx" 6)

where "**xxx**" is the faulty statement and **6** tells you that the trouble begins at the sixth character.

ade_errtype Error Message Functions

Gets the type of a given error in the stack.

(ade_errtype err_index)

Returns an error type (integer) or nil.

err_index Position of the error in the stack (integer), where 0 = first error.

ade_expreval

Evaluates an expression.

(ade_expreval [ename] expr type)

Returns the value of the expression or nil.

ename	Optional drawing object name. Required if the expression uses object properties or data.
expr	Expression to evaluate (string)
type	Expected return type (string): "short", "long", "real", "string", or "point".

If the expression uses object properties or data, you must specify the name of a drawing object. For example, if the the expr argument is "(+ 5 6)", no ename argument is required, but to evaluate "(+ ".COLOR" 6)" you need an object to supply the color.

The following example gets the area of a selected object:

```
(setq myobject (car (entsel "Select an object:")))
(setq value (ade_expreval myobject ".area" "real"))
```

Depending on what you specify for the **type** argument in the preceding example, the result can be an integer or a string, as the following two examples

demonstrate. Suppose the area of **myobject** is 2.7. The first expression returns this area as **2**; the second returns it as "2.7".

```
(ade_expreval myobject ".area" "short")
```

```
(ade_expreval myobject ".area" "string")
```

Note If the **expr** argument is an integer calculation and you supply "**string**" for the **type** argument, the resulting string does not contain an integer, but a **real**. For example, the following expression returns "2.0", not "2".

```
(ade_expreval "(+ 1 1)" "string")
```

If you want the string to contain an integer, include the **fix** function in the **expr** argument.

```
(ade_expreval "(fix (+ 1 1))" "string")
```

The ade_expreval function can return an integer string so long as the return value is not the result of an integer calculation. For example, the following code returns an integer string without using fix.

(setq obj (car (entsel "Select an object:")))
(setq objcolor (ade_expreval obj ".color" "string")))

To get the centroid and the layer name of the same object, add these lines:

```
(setq objcentr (ade_expreval obj ".centroid" "point"))
(setq objlayer (ade_expreval obj ".layer" "string"))
```
ade_keycolumnlist SQL Environment Functions

Returns a list of the key column names for the specified link template.

(ade_keycolumnlist linktemplate)

Returns a list of key column names or nil.

linktemplate Link template (string)

For more information about link templates and using SQL, see the AutoCAD online documentation.

ade_odaddfield Object Data Functions

Adds fields to a table.

(ade_odaddfield tabname fieldlist)

Returns T or nil.

- tabnameTable name (string) can be up to 25 characters long. Must be
unique, contain no spaces, and start with an alphanumeric
character
- fieldlist List of fields to add. A sequence of field definitions.

A sequence of field definitions is introduced by the string "**columns**". Each field definition is a list of a-lists, and each a-list consists of a field property and a value, as follows:

Field property	Value
colname	Field name (string) can be up to 31 characters long. Must be unique, contain no spaces, and start with an alphanumeric character.
coldesc	Field description (string)
coltype	Field data type
defaultval	Default field value

The function adds fields to the table and to each of its records. In each record, the new fields are assigned default values in accord with their field definitions. The function has no effect on existing fields. In other words, the function adds fields to each set of object data defined by the table and attached to an object.

For an example, see <u>Adding Fields to a Table</u>.

ade_odaddrecord

Object Data Functions

Attaches data to an object.

(ade_odaddrecord ename table)

Returns T or nil.

ename AutoCAD object name.

tableTable name (string) can be up to 25 characters long. Must be
unique, contain no spaces, and start with an alphanumeric
character

Attaching data to an object is also called attaching a table to an object. This function attaches a new record in a specific table to a specific object. Typically, a record contains information about whatever it is that the object represents. For example, if a line in a drawing represents a section of pipe in a water system, an attached record could contain information about that section.

When a new record is attached, its fields contain default values that correspond to their field definitions. To get a field value, use <u>ade_odgetfield</u>; to change it, use <u>ade_odsetfield</u>. Field definitions are included in the table definition. See <u>ade_oddefinetab</u> for information about table definitions.

You can attach more than one record to the same object with additional calls to ade_odaddrecord. The additional records can be members of the same or different tables. If an object has only one record from a given table, the number of that record is 0. If you attach a second record from the same table, the number

of that record is 1, and so on. Use <u>ade_odrecordqty</u> to find how many records of a given table are attached.

For example, if a section of water pipe is inspected at intervals, you could attach a number of records of the WATER INSPECTION table to the same line in the WATER drawing, and each record could contain the result of a different inspection.

ade_odattachrecord

Object Data Functions

Attaches a new record to an object.

(ade_odattachrecord ename rec_id)

Returns T or nil.

- ename AutoCAD object name.
- rec_id Record ID returned by ade_odnewrecord

ade_oddefinetab

Creates an object data table.

(ade_oddefinetab tab_defn)

Returns T or nil.

tab_defn List of table elements: the table name, the table description, and a sequence field definitions.

The table name is specified by an a-list:

```
("tablename" . "NEWTABLE")
```

The name must be unique, contain no spaces, and start with a character. The name can be up to 25 characters long.

The table description is specified the same way, except that spaces are allowed:

```
("tabledesc" . "New Sample Table")
```

The field definitions are introduced by the string, "columns". At least one field definition is required. Each field definition is a list of a-lists, and each a-list consists of a field property and a value, as follows:

Field property	Field name
colname	Field name (string) can be up to 31 characters long. Must be unique, contain no spaces, and start with an alphanumeric character
coldesc	Field description (string)
coltype	Field data type
defaultval	Default field value

The following code creates a table.

```
; Define new table
(setq tabldefn
 '(("tablename" . "NEWTABLE")
    ("tabledesc" . "New Sample Table")
    ("columns"
    ; Define a field
        (("colname" . "FIELD1")
        ("coldesc" . "Field1 Description")
        ("coltype" . "character")
        ("defaultval" . "Default Value"))
        ; Define more fields as needed
        .
        .
        ; For an example of defining fields, click .
    ))))
; Create the new table
    (ade_oddefinetab tabldefn)
```

Here is another example of creating a table.

```
(setq pt1 "2,2")
(ade_oddefinetab
  (list
        '("tablename" . "valve_id")
        '("tabledesc" . "Valve Storage")
        (list "columns"
        (list
              '("colname" . "LOCATION")
              '("coldesc" . "Valve_id")
              '("coltype" . "point")
              (cons "defaultval" pt1))))
```

ade_oddeletefield

Object Data Functions

Deletes fields from a table.

(ade_oddeletefield tabname fieldlist)

Returns T or nil.

tabname Table name (string) can be up to 25 characters long. Must be unique, contain no spaces, and start with an alphanumeric character

fieldlist List of field names (string)

The function deletes the fields from the table and from each of its records. The data contained in these fields is also deleted. In other words, it deletes the fields and their data from each set of object data defined by the table and attached to an object.

Note The fieldlist argument for ade_oddeletefield is a list of field names only. In the companion functions, ade_odaddfield and ade_odmodifyfield, it is a list of field definitions.

The **ade_oddeletefield** function affects all active drawings in the drawing set. There should not be any queried objects for this operation.

Note This function will not operate unless your end user has superuser privileges.

The following code deletes three fields from a table.

(ade_oddeletefield "table1" '("field1" "field2" "field3"))

ade_oddeletetab

Object Data Functions

Deletes a table.

(ade_oddeletetab tabname)

Returns T or nil.

tabname Table name (string) can be up to 25 characters long. Must be unique, contain no spaces, and start with an alphanumeric character

The function deletes a table and all of its records. It deletes every set of object data defined by the table and attached to an object, as well as the data contained in the records.

The ade_oddeletetab function affects all active drawings in the drawing set. There should not be any queried objects for this operation.

Note This function will not operate unless your end user has superuser privileges.

ade_oddelrecord

Object Data Functions

Deletes a record.

(ade_oddelrecord ename table recnum)

Returns T or nil.

ename	AutoCAD entity name of the object to which the record is attached.
table	Name of the table to which the record belongs, up to 25 characters long (<mark>string</mark>). Must be unique, contain no spaces, and start with a character
recnum	Record number (integer). The number of the first record is 0

The function deletes the record from the object. It deletes the set of object data defined by the table and attached to the object. This deletes the record from the table as well as the data contained in the record.

The record number is necessary because more than one record from the same table can be attached to an object. Use <u>ade_odrecordqty</u> to find how many records of a given table are attached.

ade_odfreerec

Frees the memory claimed in defining a new record.

(ade_odfreerec rec_id)

Returns T or nil.

rec_id Record ID returned by ade_odnewrecord

Warning You must release a new record when you are finished with it.

ade_odgetfield Object Data Functions

Gets a field value.

(ade_odgetfield ename table field recnum)

Returns a field value (<u>data type</u> varies) or <u>nil</u>.

ename	AutoCAD object name.
table	Table name (string) can be up to 25 characters long. Must be unique, contain no spaces, and start with an alphanumeric character
field	Field name (string) can be up to 31 characters long. Must be unique, contain no spaces, and start with an alphanumeric character
recnum	Record number (integer). The number of the first record is 0

To identify a unique record, you need to specify the table to which it belongs, the object to which it is attached, and its record number. The record number is necessary because more than one record from the same table can be attached to an object. For more information about records and record numbers, see <u>ade_odaddrecord</u>.

The field value returned can be one of four <u>data types</u>: integer, character, point, or real.

ade_odgetrecfield Object Data Functions

Gets a field value using a record ID.

(ade_odgetrecfield recID field)

Returns a field value (<u>data type</u> varies) or <u>nil</u>.

recID Record ID (real) returned by ade_odgetrecord

field Field name (string)

This function uses the record ID assigned by ade_odgetrecord to get the value of a particular field. This means of getting an object data field value is generally faster than any other.

ade_odgetrecord

Gets a record ID.

(ade_odgetrecord ename table recnum)

Returns a record ID (real) or nil.

ename	AutoCAD object name.
table	Table name (string)
recnum	Record number (integer); the first record number is 0

The function assigns an ID to the record uniquely determined by the three arguments. Later you can use this record ID with ade_odgetrecfield to return the value of a particular field of this record. This means of getting an object data field value is generally faster than any other.

Three arguments are necessary because an AutoCAD object can be associated with more than one record in a table, in which case the records are distinguished by their record numbers. If there is only one record, its number is **0**. For more information about records and record numbers, see <u>ade_odaddrecord</u>.

ade_odgettables Object Data Functions

Lists the tables attached to an object.

(ade_odgettables ename)

Returns a list of table names (string) or nil.

ename AutoCAD object name.

An object can have records of more than one table attached. This function lists all the tables that have records attached to the object. See <u>ade_odaddrecord</u> for information about records attached to objects.

An object can have more than one record from the same table attached. To find how many records of a given table are attached, use <u>ade_odrecordqty</u>.

ade_odmodifyfield

Object Data Functions

Modifies field properties in a table.

(ade_odmodifyfield tableName fieldList)

Returns T or nil.

tableNameTable name (string) can be up to 25 characters long. Must
be unique, contain no spaces, and start with an alphanumeric
characterfieldListFields to modify. A sequence of field definitions. See Field
Definitions below.

The ade_odmodifyfield function affects all active drawings in the drawing set. There should not be any queried objects for this operation.

Note This function will not operate unless your end user has superuser privileges.

Field Definitions

The field definitions are introduced by the "**Columns**" string. Each field definition is a list of a-lists, and each a-list consists of a field property and a value, as follows:

colname	Field name (string) can be up to 31 characters long. Must be unique, contain no spaces, and start with an alphanumeric character
coldesc	Field description (string)
coltype	Field data type
defaultval	Default field value

In the **fieldlist** argument, specify the fields to modify and their new field definitions. The function acts on these fields only and has no effect on any others. For each field you specify, the function replaces the existing field definition with the new field definition in the table and in each of its records. In each record, the modified fields are assigned default values that correspond to their new field definitions. In other words, the function replaces field definitions in each set of object data defined by the table and attached to an object.

The fieldlist argument has the same format as the fieldlist argument in <u>ade_odaddfield</u>. The entry for this function has source code examples.

If a new field definition changes the field type, field values in existing records are converted to the new type if possible. This conversion may alter the values. For example, if you change the field type from real to integer, existing field values are converted by truncating their decimal parts.

ade_odmodifytab

Object Data Functions

Redefines a table.

(ade_odmodifytab tab_defn)

Returns T or nil.

tab_defn List of table elements: the name of the table you will redefine, a new table description, and a sequence of new field definitions.

The tab_defn argument has the same format as the tab_defn argument in ade_oddefinetab. The entry for this function has source code examples.

For the table you specify in the tab_defn argument, the function replaces the existing table definition with the new one. For every object to which the table is attached, the corresponding fields of each record of the table are replaced. The old fields are deleted, and the new fields are assigned default values in accord with their field definitions.

The ade_odmodifyfield function affects all active drawings in the drawing set. There should not be any queried objects for this operation.

Note This function will not operate unless your end user has superuser privileges.

ade_odnewrecord Object Data Functions

Defines a new object data record.

(ade_odnewrecord table)

Returns a new record ID or nil.

table Table name (string), the existing table to which the new record will belong.

The function creates a new record, populates its fields with default values according to the table definition, and returns the new record ID.

ade_odpresetfield

Object Data Functions

Assigns a value to a field in a new record.

(ade_odpresetfield rec_id field value)

Returns T or nil.

- rec_id Record ID returned by ade_odnewrecord
- field Field name (string)
- value Field value.

Sets the value of a field in an Object Data record defined through ade_odnewrecord.

ade_odrecordqty Object Data Functions

Counts the records attached to an object.

(ade_odrecordqty ename table)

Returns a record count (integer) or nil.

ename AutoCAD object name.

table Table name (string) can be up to 25 characters long. Must be unique, contain no spaces, and start with an alphanumeric character.

The function counts how many records of the same table are attached to the object. See <u>ade_odaddrecord</u> for more information about attaching records to objects.

ade_odsetfield Object Data Functions

Sets a field value.

(ade_odsetfield ename table field recnum value)

Returns T or nil.

ename	AutoCAD object name.
table	Table name (string) can be up to 25 characters long. Must be unique, contain no spaces, and start with an alphanumeric character.
field	Field name (string) can be up to 31 characters long. Must be unique, contain no spaces, and start with an alphanumeric character.
recnum	Record number (integer); the first record number is 0.
value	New field value.

To identify a unique record, you need to specify the table to which it belongs, the object to which it is attached, and its record number. The record number is necessary because more than one record from the same table can be attached to an object. For more information about records and record numbers, see <u>ade_odaddrecord</u>.

ade_odtabledefn Object Data Functions

Gets a table definition.

(ade_odtabledefn table)

Returns a table definition or nil.

tableTable name (string) can be up to 25 characters long. Must be
unique, contain no spaces, and start with an alphanumeric character

The table definition returned by ade_odtabledefn has the same format as the tab_defn argument in <u>ade_oddefinetab</u>. The entry for this function has source code examples.

ade_odtablelist Object Data Functions

Lists the tables in the project.

(ade_odtablelist)

Returns a list of table names (string) or nil.

The list includes all object data tables in the project drawing and in all active source drawings.

ade_osfexpand

Searches a directory and returns a list of file names.

(ade_osfexpand path extension pattern)

Returns a list of file names or nil.

path	Directory in which to search (string) or nil . If nil , the function searches the working directory
extension	File name extension (string) or nil . If nil , the function uses "dwg".
pattern	Wild card pattern (string) or nil. If nil, the function uses "*" (search for all file names with the given extension and path).

For information about wild card patterns, look up "wild-card characters" on the Index tab of AutoCAD Map Help.

ade_projgetctgyname Coordinate Transformation Functions

Identifies the category that a coordinate system belongs to.

(ade_projgetctgyname cscode)

Returns a coordinate system category (string) or nil.

cscode Coordinate system code (string), eight characters

ade_projgetinfo <u>Coordinate Transformation Functions</u>

Gets information about a projection system.

(ade_projgetinfo cscode info_type)

Returns a piece of projection system information or nil.

- cscode Coordinate system code (string), eight characters
- info_type Information type (string); see Information Types below

Information Types

description	Description (string). For example, "World Geodetic System of 1984 Latitude/Longitude in Degrees".
projection	Projection (string). For example, "Unity Conversion, produce/accept lat/longs".
datum	Datum (string). For example, "North American Datum of 1927, Mean Values".

ade_projgetwscode <u>Coordinate Transformation Functions</u>

Gets the project drawing's coordinate system code.

(ade_projgetwscode)

Returns a coordinate system code (string) or the empty string.

ade_projlistcrdsysts Coordinate Transformation Functions

Lists available coordinate systems in a given category.

(ade_projlistcrdsysts categoryname)

Returns a list of available coordinate systems or nil

categoryname Coordinate system code (string), eight characters.

ade_projlistctgy Coordinate Transformation Functions

Lists available coordinate system categories.

(ade_projlistctgy)

Returns a list of coordinate system categories or nil.

ade_projptbackward Coordinate Transformation Functions

Converts point coordinates from destination coordinate system to source.

(ade_projptbackward pt)

Returns corresponding source values or nil.

pt Destination point to convert, a set of 2D or 3D coordinate values (real). If 3D, the Z value is ignored.

Before you can use ade_projptbackward to convert points, you must first identify the coordinate systems that you are converting between. Use ade_projsetsrc to set the source system and ade_projsetdest to set the destination system. The ade_projptbackward function assumes that the coordinate values you pass to it belong to the destination system, and it returns corresponding source values. The ade_projptforward function does the inverse.

For more information, see <u>Converting Coordinates</u>.

ade_projptforward Coordinate Transformation Functions

Converts point coordinates from source coordinate system to destination.

(ade_projptforward pt)

Returns corresponding destination values or nil.

pt Source point to convert, a set of 2D or 3D coordinate values (real). If 3D, the Z value is ignored.

Before you can use ade_projptforward to convert points, you must first identify the coordinate systems that you are converting between. Use ade_projsetsrc to set the source system and ade_projsetdest to set the destination system. The ade_projptforward function assumes that the coordinate values you pass to it belong to the source system, and it returns corresponding destination values. The ade_projptbackward function does the inverse.

For more information, see <u>Converting Coordinates</u>.

ade_projsetdest Coordinate Transformation Functions

Sets the destination coordinate system for converting points.

(ade_projsetdest cscode)

Returns T or nil.

cscode Coordinate system code (string), eight characters.

Before you can use either <u>ade_projptforward</u> or <u>ade_projptbackward</u> to convert points, you must first identify the coordinate systems that you are converting between. Use <u>ade_projsetdest</u> to set the destination system and <u>ade_projsetsrc</u> to set the source system. The <u>ade_projptforward</u> function assumes that the coordinate values you pass to it belong to the source system, and it returns corresponding destination values. The <u>ade_projptbackward</u> function does the inverse.

For more information, see <u>Converting Coordinates</u>.
ade_projsetsrc Coordinate Transformation Functions

Sets the source coordinate system for converting points.

(ade_projsetsrc cscode)

Returns T or nil.

cscode Coordinate system code (string), eight characters.

Before you can use either <u>ade_projptforward</u> or <u>ade_projptbackward</u> to convert points, you must first identify the coordinate systems that you are converting between. Use <u>ade_projsetsrc</u> to set the source system and <u>ade_projsetdest</u> to set the destination system. The <u>ade_projptforward</u> function assumes that the coordinate values you pass to it belong to the source system, and it returns corresponding destination values. The <u>ade_projptbackward</u> function does the inverse.

For more information, see <u>Converting Coordinates</u>.

ade_projsetwscode Coordinate Transformation Functions

Sets the coordinate system for the project drawing.

(ade_projsetwscode cscode)

Returns T or nil.

cscode Coordinate system code (string), eight characters

ade_qldelctgy Query Library Functions

Deletes a query library category.

(ade_qldelctgy ctgy_id)

Returns T or nil.

ctgy_id Category ID (real)

ade_qldelquery Query Library Functions

Deletes a query from the query library.

(ade_qldelquery qry_id)

Returns T or nil.

qry_id Query ID (real)

ade_qlgetctgyinfo Query Library Functions

Gets information about a query category.

(ade_qlgetctgyinfo ctgy_id info)

Returns the requested information or nil.

- ctgy_id Category ID (real)
- info Type of category information to get (string): "name" to get the category name, or "qrylist" to get a list of query IDs of the queries in the category.

The information returned depends on the info argument you use, but it is always in list format. For example:

- An expression such as (ade_qlgetctgyinfo some_id "name") returns a category name, such as ("SomeCategory").
- An expression such as (ade_qlgetctgyinfo some_id "qrylist") returns a list of query IDs, such as (1.23456 2.34567 3.45678).

ade_qlgetqryinfo Query Library Functions

Gets information about a query.

(ade_qlgetqryinfo qry_id info)

Returns the requested information or nil.

qry_id Query ID (real).

info Information type (string). See the Information Types table below.

Information Types

name	Query name.
description	Query description.
category	Category name.
qtype	Query type: $1 = internal$, $2 = external$.
filename	For an externally saved query, full path name (string).

A query gets a name and an ID when it is saved to a query category of the query library. A new query that you have not yet saved does not have a name or an ID.

ade_qllistctgy Query Library Functions

Lists the query category IDs .

(ade_qllistctgy)

Returns a list of category IDs, or if there are no categories, nil.

To find the ID of a category if its name is known, use ade_qllistctgy to get a list of category IDs, and then use ade_qlgetctgyinfo on each ID in turn until you find the ID associated with the name.

ade_qlloadqry Query Library Functions

Makes a saved query current.

(ade_qlloadqry qry_id)

Returns T or nil.

qry_id Query ID (real)

Once loaded, the query becomes the current query. If there is already a current query, this query replaces it.

ade_qlqrygetid Query Library Functions

Gets a query ID.

(ade_qlqrygetid query_name)

Returns a query ID or nil.

query_name Query name (string)

A query gets a name and an ID when it is saved to a query category of the query library. A new query that you have not yet saved does not have a name or an ID.

The query name is enough to identify a query uniquely. The category name is not required. Within a project, no two queries can have the same name, even if they are saved in different categories.

ade_qlsetctgyname Query Library Functions

Changes a query category name .

(ade_qlsetctgyname ctgy_id name)

Returns T or nil.

- ctgy_id Query category ID (real)
- name Name of new category (string), up to 31 characters long. Must be unique, contain no spaces, and start with an alphanumeric character

The category name cannot contain spaces.

ade_qlsetquery Query Library Functions

Changes a query name, description, or the category it belongs to.

(ade_qlsetquery qry_id info value)

Returns T or nil.

qry_id	Query ID (real)
info	Type of information to modify (string). See the Information Types table below.
value	New value (type varies). See the Information Types table below.

Information Types

name	Name of query (string), up to 31 characters long. Must be unique, contain no spaces, and start with an alphanumeric character.
description	Description of query (string), up to 132 characters long. Can contain spaces. Must be unique and start with an alphanumeric character.
category	Category ID.

This function does not change file name or storage type.

A query gets a name and an ID when it is saved to a query category of the query library. A new query that you have not yet saved does not have a name or an ID.

ade_qryclear Query Functions

Clears the current query.

(ade_qryclear)

Returns T or an error code.

This function clears the current query, including any topology parameters, such as topology name and result.

ade_qryexecute

Executes the current query.

```
(ade_qryexecute)
```

Returns the number of queried objects (real). If none, it returns 0.0.

Executing a query makes a new selection set of the queried objects if the "MkSelSetWithQryObj" option is turned on, as follows:

(ade_prefsetval "MkSelSetWithQryObj" T)

The query runs slower in this case because of the extra work involved.

The following code captures the ID of a selection set created by executing the current query.

```
(ade_prefsetval "MkSelSetWithQryObj" T)
(if (> (ade_qryexecute) 0.0)
  (setq queried_objects (ssget "P"))
  (princ "\nNo objects found.")
)
(ade_prefsetval "MkSelSetWithQryObj" nil)
```

The "P" argument in the ssget call identifies the "previous" selection set (the objects currently or most recently selected).

Note Whenever you create a selection set, you replace the previous selection set. Make sure you know which objects you are getting.

ade_qrygetdwgandhandle

Gets the source drawing ID and original handle of a queried object.

(ade_qrygetdwgandhandle ename)

Returns the drawing ID and handle of the queried object or nil.

ename AutoCAD entity name.

This function returns the ID of the source drawing from which the object was queried and the handle by which the object is known in that drawing.

ADSRX equivalent

Returns the drawing ID and handle for the queried object or NULL.

You must release the resbuf.

ade_qrygetentlist

Returns the list of entity handles for all objects that satisfy the current query in a specific drawing.

(ade_qrygetentlist dwg_id)

Returns a list of the handles of selected objects or nil.

dwg_id Drawing ID of the drawing to query (real)

This function executes the current query and finds all objects that satisfy it in the drawing specified by the dwg_id argument. The entity handles of the objects are returned to the calling function.

Once you have the handle to an object, you can get the entity name with the <u>ade_qryhandent</u> function and use it to perform other functions. For example, you could use <u>entget</u> (and <u>ads_entget</u>) to retrieve the entity and its definition data.

ADSRX equivalent

struct
resbuf* ade_qrygetentlist
 ade_id dwg_id);

Returns a list of the handles of selected objects or NULL.

You must release the **resbuf**.

ade_qrygetreptransform

Checks whether transformation is enabled for the current report query.

(ade_qrygetreptransform)

Returns **T** if transformation is enabled or **nil** if transformation is disabled or the report query template is not defined.

ADSRX equivalent

int
ade_qrygetreptransform();

Returns TRUE or FALSE.

ade_qrygroup

Groups a sequence of two or more query conditions.

(ade_qrygroup condition_id1 condition_id2)

Returns T or nil.

condition_id1	ID of first condition of the group (real)
condition_id2	ID of last condition of the group (real)

This function affects the current query.

A query definition consists of a sequence of query conditions. Within such a sequence, you can define subsequences of two or more conditions by grouping them (by enclosing them in parentheses). You can group conditions when you first define the query. See the **bggroups** and **endgroups** parameters of <u>ade_qrydefine</u>. Or you can do it later using <u>ade_qrygroup</u>.

When you call ade_qrygroup, the condition you specify as the first condition of the group (condition_id1) must be a predecessor to the one you specify as the last (condition_id2). The function groups the first and the last and any conditions in between. For example, consider the following query definition, which is composed of conditions A, B, C, and D:

A AND B OR C OR D

The function call (ade_qrygroup IDofB IDofD) changes the definition to

A AND (B OR C OR D)

To ungroup queries, use <u>ade_qryungroup</u>.

ade_qryhandent

Gets the entity name for the specified handle .

(ade_qryhandent dwg_id handle)

Returns the entity name for the specified drawing ID and handle or nil.

dwg_id ID of the drawing in which the object resides (real)

handle Original handle of the object in the specified drawing.

This function provides access to the entity name of an object in a source database.

You must use the retrieved entity name immediately before you call any other function (except ade_expreval) or return control to AutoCAD.

Once you have the entity name of an object, you can use it with other functions. For example, you could use **entget** (or **ads_entget**) to retrieve the entity and its definition data.

To get the original handle of the object in the source drawing, use the <u>ade_qrygetentlist</u> function.

To obtain a drawing ID, use ade_dslist.

To get the ID of a drawing given a drawing file path, use ade_dwggetid.

The following code sample shows how you can combine ade_qrygetentlist and ade_qryhandent to count the number of objects in the source drawing that

are of type line.

```
; clear out old query...
(ade_qryclear)
; define a new query
(ade_qrydefine '("" "" "Location" ("All") ""))
; initialize the count...
(setq total_count 0)
; for each drawing in the drawing set...
(foreach dwg_id (ade_dslist)
  ; if the drawing is active
 (if (ade_dwgisactive dwg_id)
    (progn
      ; get the objects which satisfy the query...
      (setq handle_list (ade_qrygetentlist dwg_id))
      (foreach handle handle_list
        (setq ename (ade_qryhandent dwg_id handle))
        ; if it's a line, increment the counter
        (if (= (cdr (assoc 0 (entget ename))) "LINE")
          (setq total_count (1+ total_count))
        )
      )
    ); progn
  ); if
); foreach
```

ADSRX equivalent

int
ade_qryhandent(

ade_id dwg_id, char* handle, ads_name result);

Returns **RTNORM** or an error code.

result Output the entity name for the specified drawing ID and handle.

ade_qrylist Query Functions

Lists the IDs of the current query conditions.

(ade_qrylist)

Returns a list of the IDs of the current query conditions, or, if there is no current query, nil.

ade_qrysave

Saves the current query.

(ade_qrysave catname qryparams)

Returns a query ID or nil.

catname	Category name (string). The category is created if it does
	not exist.

qryparams List of a-lists, each composed of an information type and a value. See the Information Types table below.

Information Types

name	Query name (string)
description	Query description (string)
qtype	How the query is saved (integer): $1 =$ internal (default), $2 =$ external.
filename	For an external query, full path name (string).
saveoption	Bit code for the save options you are choosing (integer). See the Save Options table below.

The function saves the current query to the project's query library or to a file.

- A query saved to the query library is called an internal query.
- A query saved to a file is called an external query.

You must specify a category name and a query name. In a project, no two queries can have the same name, even if they are saved in different categories. The default value for a description is the same as the query name. The default value for the storage type is internal. If you want to save the query externally, you must specify a file name for it.

Save Options

1	Keep reference in query library.
2	Save list of active drawings.
4	Save location coordinates.
8	Save current property alteration definition.
16	Execute automatically.

A query gets a name and an ID only if it is referenced the query library. A new query that you have not yet saved does not have a name or an ID, and neither does an external query unless you keep a reference to it in the query library.

The following example saves the current query to the query library without saving it to a file.

```
(ade_qrysave "CATEGORY1"
 '( ("name" . "QUERY1")
    ("saveoption" . 2)
 )
)
```

The expression saves the query in CATEGORY1 and names it QUERY1. Because there is no "**qtype**" list element, it saves the query internally by default,

which eliminates the need for a "filename" element.

The following example saves the current query to the query library and also to a file.

```
(ade_qrysave "CATEGORY1"
 '( ("name" . "QUERY1")
   ("description" . "Query1 description")
   ("qtype" . 2)
   ("filename" . "c:\\qryfiles\\Query1.qry")
   ("saveoption" . 3)
 )
)
```

ade_qrysetaltprop Query Functions

Turns property alteration on or off.

```
(ade_qrysetaltprop flag)
```

Returns T or nil.

flag Specifies whether property alteration is on or off: T = on, nil = off.

This function affects the current query.

If there is no current property alteration definition, this function has no effect. To create a property alteration definition, use <u>ade_altpdefine</u>.

ade_qrysetreptransform

Enables or disables transformation for the current report query.

(ade_qrysetreptransform flag)

Returns **T** if successful, otherwise **nil**.

flag T or nil, where T = transformation enabled, and nil = transformation disabled.

The function returns **nil** if there is no report query template.

ade_qrysettype

Sets the query mode: Preview, Draw, or Report.

(ade_qrysettype qrytype [multiline templ filename])

Returns T or nil.

qrytype	Query mode (string): "preview", "draw", or "report", where "preview" = Display queried objects without retrieving them, similar to Quick View, "draw" = Get queried objects from source drawings or external databases and copy to the project drawing, and "report" = Direct queried information to an output file
multiline	Whether to write report rows for sub-objects: T or nil , where T = Write report rows for sub-objects (objects such as vertices of polylines and attributes of blocks), and nil = Write lines for top-level objects only. Relevant only if qrytype is "report ".
templ>	Ordered list of object properties to report (string). Each list element defines a report column. For example, ".type,.layer". Relevant only if qrytype is " report ".
filename	Path and file name of the output file (string). Relevant only if qrytype is "report" .

This function affects the current query.

The three optional parameters, **multiline**, **templ**, and **filename**, are relevant only if the query mode is "**report**". If the query mode is "**preview**" or "**draw**", omit them.

The following example sets the query mode to Report.

```
(ade_qrysettype "report" T ".type,.layer" "output.txt")
```

ade_qryungroup

Ungroups a sequence of two or more query conditions.

(ade_qryungroup condition_id1 condition_id2)

Returns T or nil.

condition_id1	Condition ID of the first grouped condition (real).
condition_id2	Condition ID of the last grouped condition (real).

This function affects the current query.

A query definition consists of a sequence of query conditions. Within such a sequence, there can be subsequences that have been grouped by enclosing them in parentheses. Such groups may have been established when the query was first defined. See the **bggroups** and **endgroups** parameters of <u>ade_qrydefine</u>. Or they may have been established afterward by <u>ade_qrygroup</u>. However established, you can use <u>ade_qryungroup</u> to undo a group (remove its enclosing parentheses).

When you call ade_qryungroup, the condition you specify as the first of the group (condition_id1) must be a predecessor to the one you specify as the last (condition_id2). The function ungroups the first and the last and any conditions in between. For example, consider the following query definition, which is composed of conditions A, B, C, and D:

A AND (B OR C OR D)

The function call (ade_qryungroup IDofB IDofD) changes the query definition to

A AND B OR C OR D

ade_rtdefrange Range Table Functions

Defines a range table.

(ade_rtdefrange tabname description range_defn)

Returns a range table ID or nil.

tabname	Range table name (string); can be up to 31 characters long. Must be unique, contain no spaces, and start with an alphanumeric character
description	Range table description (string)
range_defn	Range table definition (string)

A range table allows you to alter properties of queried entities conditionally. It contains a set of property alteration values from which a single value is selected depending on conditions obtaining in the queried entity to be altered.

The range_defn argument is a range table definition, a list of range expressions. Each range expression includes (1) a condition and (2) a property alteration value to return if the condition is true. This information is expressed as a list of three elements: a <u>range table operator</u> and a comparison value (which together make up the condition), and the return value. You must state each value explicitly. You cannot substitute an expression.

See <u>Using a Range Table</u> for more information.

ade_rtdeltable <u>Range Table Functions</u>

Deletes a range table.

(ade_rtdeltable tablename)

Returns T or nil.

tablenameRange table name (string); can be up to 31 characters long.Must be unique, contain no spaces, and start with an
alphanumeric character
ade_rtgetid Range Table Functions

Gets a range table ID.

(ade_rtgetid tablename)

Returns a range table ID or nil.

tablenameRange table name (string); can be up to 31 characters long.Must be unique, contain no spaces, and start with an
alphanumeric character

ade_rtgetprop Range Table Functions

Gets the value of a range table property.

(ade_rtgetprop rt_id property)

Returns a property value, or list, or nil.

- rt_id Range table ID (real)
- property Property to get the value of (string). See the Range Table Properties table below.

Range Table Properties

name	Range table name (string)	
description	Range table description (string)	
expr	Range table definition (list of range expressions)	

See <u>ade_rtdefrange</u> for information about setting range table properties.

ade_rtlist
<u>Range Table Functions</u>

Lists the IDs of all range tables defined in the project.

(ade_rtlist)

Returns a list of range table IDs or nil.

ade_saveobjs Object Saving Functions

Saves objects queued for saving back to the source drawings.

(ade_saveobjs priorities)

Returns T or nil.

priorities List containing one to four save back operation codes, depending on the number of save back operations you are specifying (integer). List the codes in order of their relative priority. See the Save-Back Operation Codes table below.

Save-Back Operation Codes

Code	Operation	Description
1	Came From	Saves objects to their source drawings
2	Selective	Saves objects you select to the drawings you specify
3	Layer	Saves objects to layers in the source drawings that use the same names as those in which the objects lie
4	Area	Saves objects to the source drawing within whose extents they lie, even if only partially within

During the save back operation, the options are executed in the order specified in the **priorities** list. The list must contain at least one option. For example:

(ade_saveobjs 2 1 4 3)

To save a selection set to a specific drawing, use <u>ade_savetodwg</u>.

ade_savetodwg Object Saving Functions

Saves a selection set to a specific drawing.

(ade_savetodwg sel_set dwg_id)

Returns T or nil.

sel_set Selection set name.

dwg_id Drawing ID of the destination drawing (real)

To save objects queued for saving back to the source drawings, use <u>ade_saveobjs</u>.

ade_sqlgetenvstring SQL Environment Functions

Gets a string describing the SQL environment.

(ade_sqlgetenvstring linktemplate)

Returns a string describing the SQL environment or nil.

linktemplate Link template (string)

This function returns a string with the following information:

"DBname.Catalog.Schema.Table"

For more information about link templates using SQL, see the AutoCAD online documentation.

ade_ssfree

Releases a selection set.

(ade_ssfree ss)

Returns T or nil.

ss Selection set to release.

Selection sets are returned by a number of Visual LISP functions. See <u>Functions</u> <u>That Return Selection Sets</u>. It is important to release selection sets as you finish with them, because the number allowed is limited. If the number runs out, your application will fail.

The following example allocates a selection set and then releases it.

```
(setq ss
	(map_dwgBreakObj
		sscut boundary skiptopo keepod)
)
....
(setq status (ade_ssfree ss))
```

ade_userget <u>User Security Functions</u>

Gets the login name or entity lock name of the local user.

```
(ade_userget [for_entity_locks])
```

Returns a user name (string) or nil.

for_entity_locksWhat to do if the local user is not logged into the
application (optional): T or nil, where T = Get the
user name used to identify the owner of object locks
set locally, and nil = Return nil. Omitting this
argument is the same as supplying nil.

A user name can have as many as 32 characters.

If the local user is logged into the application, the user's application login name is used to identify the owner of object locks set locally. If the local user is not logged into the application, the user's operating system login name is used.

This function helps you determine if the owner of a particular object lock is the local user.

ade_usergetrights User Security Functions

Gets the access rights of a user.

(ade_usergetrights [username])

Returns a bit code for the rights allowed. See the User Rights Codes table below.

username Login name (string), at most 32 characters.

If the **username** argument is omitted or **nil**, the function returns the rights of the current user.

User Rights Codes

Code	User Rights
1	Superuser (in which case the other bits don't matter).
2	Permission to alter the drawing set.
4	Permission to edit objects.
8	Permission to execute a draw query.
16	Permission to edit Feature Class definition.

If the **username** argument is omitted or **nil**, and there is no current user, the function returns a bit code with all bits set, because the no-current-user condition

is possible only if the system option "ForceUserLogin" is set to nil, in which case all users have all rights except those reserved for a superuser.

Only a superuser can specify a login name other than their own. If the **username** argument is not the login name of the current user, and the current user does not have superuser rights, the function returns **nil**, and the message "Access is denied" is added to the error stack.

ade_userlist <u>User Security Functions</u>

Lists the current users.

(ade_userlist)

Returns a list of user login names or nil.

ade_userset <u>User Security Functions</u>

Logs in a user.

(ade_userset [username [password]])

Returns **T** if the user is logged in successfully, otherwise **nil**.

username	Login name (<mark>string</mark>).
password	Password (string).

If either argument is omitted, the User Login dialog box displays. If a login name was specified, it appears in the dialog's Login Name field.

If both arguments are specified, but the user cannot be logged in, one of the following messages is added to the error stack:

Invalid user name. Invalid password.

If the drawing set includes active drawings containing locked entities, the current user cannot be changed. If you attempt to log in a different user under those conditions, the following error message is added to the error message stack:

Cannot login again when drawings are locked/active.

ade_usersetrights User Security Functions

Sets the access rights for a user.

(ade_usersetrights username userrights)

Returns **T** on success, otherwise **nil**.

username Login name (string)userrights A bit code for the rights to allow. See the User Rights Codes table below.

User Rights Codes

Code	User Rights
1	Superuser (in which case the other bits don't matter).
2	Permission to alter the drawing set.
4	Permission to edit objects.
8	Permission to execute a draw query.
16	Permission to edit Feature Class definition.

This function cannot execute unless the current user has superuser rights, and it cannot change the rights of the current user in any case. If an ordinary user is logged in when this function is called, or a superuser is logged in and the

function call would change the rights of the current user, the function returns nil, and the following message is added to the error message stack:

Can't change rights of the current user.

ade_version

Gets the version number of the Data Extension programming interface.

(ade_version)

Returns a version number (string) or nil.

At runtime, this function returns the version number of the Data Extension (ADE) programming interface with which your application is communicating. For example("2.024").

Coordinate Transformation Functions

The coordinate transformation functions begin with ade_proj.

ade_projgetctgyname

Identifies the category that a coordinate system belongs to.

ade_projgetinfo

Gets information about a projection system.

ade_projgetwscode

Gets the project drawing's coordinate system code.

ade_projlistctgy

Lists available coordinate system categories.

ade_projlistcrdsysts

Lists available coordinate systems in a given category.

ade_projptbackward

Computes new coordinates for a source point.

ade_projptforward

Computes new coordinates for a destination point.

ade_projsetdest

Sets the destination coordinate system.

ade_projsetsrc

Sets the source coordinate system.

ade_projsetwscode

Sets the project drawing's coordinate system.

Drawing Functions

The functions for drawing management begin with ade_dwg.

ade_dwgactivate

Activates a drawing.

ade_dwgactualpath

Returns the full path of a drawing.

ade_dwgaliaspath

Returns the alias path of a drawing.

ade_dwgattriblist

Returns a list of attribute tags for the specified block name.

ade_dwgdeactivate

Deactivates a drawing.

ade_dwggetid

Gets the drawing ID of a drawing.

ade_dwggetsetting

Gets a drawing setting value.

ade_dwghaslocks

Checks if a drawing has locked objects.

ade_dwgindex

Applies specified index operations to a drawing.

ade_dwgindexdef

Specifies which indexes are to be created or removed.

ade_dwgisactive

Checks if a drawing is active.

ade_dwgistoplevel

Checks if a drawing is directly attached to the project drawing.

ade_dwgproplist

Lists all values found in a drawing for a given drawing property.

ade_dwgquickview

Displays a quick view of a drawing.

ade_dwgselectdlg

Displays the Select Drawings dialog box.

ade_dwgsetof

Identifies the drawings to which a given drawing is attached.

ade_dwgsetsetting

Sets a drawing setting value.

ade_dwgunlock

Removes all object locks from a drawing.

ade_dwgzoomextents

Zooms to the extents of the active drawings.

Drawing Set Functions

The functions for drawing set management begin with ade_ds.

ade_dsattach

Attaches a drawing to the project.

ade_dsdetach

Detaches a drawing from the project.

ade_dsisnested

Checks if a drawing has nested drawings.

ade_dslist

Lists the drawings attached to a given drawing.

ade_dsproplist

Lists all values found in the drawing set for a given drawing property.

Drive Alias Functions

The drive alias functions begin with ade_alias.

ade_aliasadd

Creates a drive alias.

ade_aliasdelete

Deletes a drive alias.

ade_aliasgetlist

Lists all drive aliases in the project.

ade_aliasupdate

Assigns a new drive and path to a drive alias.

Error Message Functions

The functions for handling error messages begin with ade_err.

ade_errclear

Clears the error stack.

ade_errcode

Gets the error code for a specific error on the error stack.

ade_errgetlevel

Gets the system error level.

ade_errmsg

Gets the error message for a specific error on the error stack.

ade_errpush

Pushes an error message to the stack.

ade_errpushstatement

Pushes a statement to the stack.

ade_errqty

Returns the number of error messages on the stack.

ade_errsetlevel

Sets the system error level.

ade_errshowdlg

Displays the Map Messages dialog box, which shows a list of error messages on the stack.

ade_errstatement

Gets the erroneous statement for a specific error on the stack.

ade_errtype

Gets the type of a specific error in the stack.

Expression Evaluation Function

The expression evaluation function begins with ade_exp.

ade_expreval

Evaluates an AutoCAD Map expression.

Object Data Functions

The functions for object data management begin with ade_od.

ade_odaddfield

Adds fields to a table.

ade_odaddrecord

Attaches data to an object.

ade_odattachrecord

Attaches a new record to an object.

ade_oddefinetab

Creates an object data table.

ade_oddeletefield

Deletes fields from a table.

ade_oddeletetab

Deletes a table.

ade_oddelrecord

Deletes a record.

ade_odfreerec

Frees the memory claimed in defining a new record.

ade_odgetfield

Gets a field value.

ade_odgetrecfield

Gets a field value using a record ID.

ade_odgetrecord

Gets a record ID.

ade_odgettables

Lists the tables attached to an object.

ade_odmodifyfield

Modifies field properties in a table.

ade_odmodifytab

Redefines a table.

ade_odnewrecord

Defines a new object data record.

ade_odpresetfield

Assigns a value to a field in a new record.

ade_odrecordqty

Counts the records attached to an object.

ade_odsetfield

Sets a field value.

ade_odtabledefn

Gets a table definition.

ade_odtablelist

Lists the tables in the project.

Object Editing Functions

The functions for object editing begin with ade_edit.

ade_editdefcen

Defines a new label point for an object.

ade_editlockederased

Gets the objects in the save set that have been erased.

ade_editislocked

Gets lock information about an object if it is locked.

ade_editlocked

Gets the objects in the save set that have been modified.

ade_editlockobjs

Locks a set of objects and adds them to the save set.

ade_editnew

Gets the objects in the save set that are new.

<u>editunlockobjs</u>

Unlocks a set of objects and removes them from the save set.

Object Saving Functions

The object saving functions begin with ade_save.

ade_saveobjs

Saves objects queued for saving back to the source drawings.

ade_savetodwg

Saves a selection set to a specific drawing.

Option Functions

The option functions begin with ade_pref.

ade_prefgetval

Gets an option setting.

ade_prefsetval

Sets an option.

Property Alteration Functions

The functions for specifying how objects retrieved in a query should be altered begin with ade_altp.

ade_altpclear

Clears the current property alteration definition.

ade_altpdefine

Creates a property alteration expression.

ade_altpdelprop

Deletes a property alteration expression.

ade_altpgetprop

Gets a property alteration expression.

ade_altplist

Lists the IDs of the current property alteration expressions.

ade_altpsetprop

Modifies a property alteration expression.

Query Functions

The functions for query management begin with ade_qry.

ade_qryclear

Clears the current query.

ade_qrydefine

Defines a query.

ade_qryexecute

Executes the current query.

ade_qrygetcond

Gets a query condition.

ade_qrygetdwgandhandle

Gets the source drawing ID and original handle of a queried object.

ade_qrygetentlist
Returns entity handles for objects that satisfy the current query.

ade_qrygetreptransform

Checks whether transformation is enabled for the current report query.

ade_qrygroup

Groups query conditions.

ade_qryhandent

Gets the entity name for the specified handle.

ade_qrylist

Lists the IDs of the current query conditions.

ade_qrysave

Saves the current query.

ade_qrysetaltprop

Turns property alteration on or off.

ade_qrysetcond

Modifies a query condition.

ade_qrysetreptransform

Enables or disables transformation for the current report query.

ade_qrysettype

Sets the query mode: Preview, Draw, or Report.

ade_qryungroup

Ungroups query conditions.

Query Library Functions

The functions for query library management begin with ade_ql.

ade_qldelctgy

Deletes a query library category.

ade_qldelquery

Deletes a query from the query library.

ade_qlgetctgyinfo

Gets information about a query category.

ade_qlgetqryinfo

Gets information about a query.

ade_qllistctgy

Lists the query category IDs.

ade_qlloadqry

Makes a saved query current.

ade_qlqrygetid

Gets a query ID.

ade_qlsetctgyname

Changes a query category name.

ade_qlsetquery

Changes a query name, description, or the category it belongs to.

Range Table Functions

The functions for range table management begin with ade_rt.

ade_rtdefrange

Defines a range table.

ade_rtdeltable

Deletes a range table.

ade_rtgetid

Gets a range table ID.

ade_rtgetprop

Gets the value of a range table property.

ade_rtlist

Lists the IDs of all range tables defined in the project.

SQL Environment Functions

The SQL environment functions begins with ade_sql or ade_key.

ade_keycolumnlist

Returns a list of the key column names for the specified link path name.

ade_sqlgetenvstring

Gets a string describing the SQL environment.

Text Label Function

The text label function begins with ade_ent.

ade_entsetlocation

Sets a new entity label point.

User Security Functions

The user security functions begin with ade_user.

ade_userget

Gets the login name or entity lock name of the local user.

ade_usergetrights

Gets the access rights of the specified user.

ade_userlist

Lists the current users.

ade_userset

Logs in a user.

ade_usersetrights

Sets the access rights for the specified user.

Other Functions

Miscellaneous ade_xx functions.

ade_entsetlocation

Sets a new entity label point.

ade_expreval

Evaluates an AutoCAD Map expression.

ade_ssfree

Releases a selection set.

ade_osfexpand

Searches a directory and returns a list of file names.

ade_version

Gets the version number of the Data Extension programming interface.

map_dwgbreakobj Map Tool Functions

Breaks linear objects where they cross boundaries.

(map_dwgbreakobj sscut boundary skiptopo keepod)

Returns a selection set of objects cut by the operation or nil.

sscut	Selection set of objects to cut.
boundary	Object name of a single object or a selection set of multiple objects. Valid objects: line, polyline, circle, arc
skiptopo	Skip flag:1 Skip objects referenced by a topology0 Trim objects referenced by a topology
keepod	 Keep flag that sets whether to keep object data of clipped objects in result object: 1 Retain all object data on any clipped object 0 Drop object data on any clipped object

This function cuts linear objects, such as lines, polylines, circles, and arcs, that cross the selected boundary. Unlike the <u>map_dwgtrimobj</u> function, this function does not delete the parts of the object on either side of the boundary. For example, you could mark a boundary and divide one map into two section maps along this boundary.

The following example prompts you to select an object to break and make

choices about the break operation. It includes error reporting.

```
(prompt "\nSelect object to break :")
(setq sscut (ssget))
(if sscut (progn
   (setq boundary
       (car (entsel "\nSelect boundary object ")))
   (if boundary (progn
       (initget "Yes No")
       (setq kword
               (getkword "\nSkip objects referenced
               by a topology Yes/No <Yes> : "))
       (if (or (null kword) (= kword "Yes"))
           (setq skiptopo 1)
           (setq skiptopo 0)
       )
       (initget "Yes No")
       (setq kword
               (getkword "\nRetain object data
               Yes/No <Yes>: "))
       (if (or (null kword) (= kword "Yes"))
           (setq keepod 1)
           (setq keepod 0)
       )
       (setq result (map_dwgbreakobj
                     sscut
                    boundary
                    skiptopo
                    keepod))
       (if result
           (prompt "\nObject(s) break successfully.")
           (progn
               (setq nberr (ade_errqty) i 0)
               (repeat nberr
                   (prompt (strcat
                       "\nError " (rtos i 2 0) " of "
```



map_dwgtrimobj <u>Map Tool Functions</u>

Trims linear objects inside or outside of a specified boundary.

(map_dwgtrimobj ssclip boundary inorout skiptopo keepod bitflag)

Returns a selection set of objects trimmed by the operation or nil.

ssclip	Selection set of objects to trim.
boundary	Entity name of a boundary object. Valid objects: a single circle or a single closed 2D polyline
inorout	Trim flag: 1 Trim outside boundary 0 Trim inside boundary
skiptopo	Skip flag: 1 Skip objects referenced by a topology 0 Trim objects referenced by a topology
keepod	 Keep flag that sets whether to keep object data of trimmed objects in result object: 1 Drop object data from all trimmed objects 0 Retain object data on all trimmed objects
bitflag	Bit flag that sets the way to handle objects that cannot be trimmed: 0 Delete these objects within or on trim boundary 1 Ignore these objects within or on trim boundary

2 Reference the insertion point of any of these objects within or on trim boundary

map_pltblkatts Plotting Functions

Gets a list of block attributes.

(map_pltblkatts name)

Returns a list of attributes or nil.

name Layout block name (string)

map_pltblklist

Gets a list of valid plot layouts for the current work session.

(map_pltblklist)

Returns the names of the blocks in a list or **nil**.

This function returns the plot layout (block) names that are usable as plot layouts. To qualify, a block must have at least one unique viewport on one of its layers. That is, if the block has more than one viewport, it must have one layer that contains only one viewport.

The unique viewport can share its layer with objects of other types, such as lines, polylines, blocks, and text.

map_pltblkvps <u>Plotting Functions</u>

Returns a list of valid viewport layers in a specified layout blocks.

(map_pltblkvps name)

Returns a list of the valid viewport layers or **nil** if the block name or layout block is invalid.

name Layout block name (string)

map_pltcleanup Plotting Functions

Restores settings altered by map_pltinit.

```
(map_pltcleanup)
```

Returns T or nil.

This function restores certain settings to the state they were in before map_pltinit was called. See map_pltInit for a list of affected settings. Before you can use other plot functions, you must call map_pltinit again.

map_pltcurrdef Plotting Functions

Selects or creates a plot set.

(map_pltcurrdef name)

Returns T or nil.

name Name of the plot set (string)

If a plot set called **name** does not exist in the current work session or if the function <u>map_pltdefread</u> was not called, this function creates a new plot set called **name**.

Use <u>map_pltcurrset</u> to define the attributes of the plot set.

If the plot set name exists and the function map_pltdefread was called, this function loads a copy of name into memory. Use map_pltcurrGet and map_pltcurrSet to examine or change its attributes.

Note Only one plot set can be current.

map_pltcurrdel Plotting Functions

Resets a plot set attribute to its default value.

(map_pltcurrdel attr)

Returns T or nil.

attr Name of the attribute to reset (string)

map_pltcurrget Plotting Functions

Gets an attribute value for the current plot set.

(map_pltcurrget attr)

Returns the value of the specified attribute or nil.

attr Name of the <u>plot set attribute</u> to retrieve (string)

This function retrieves the value of a specific attribute for the current plot set. The data type of the return value depends upon the attribute. To set plot set attributes, use <u>map_pltdefget</u>.

```
map_pltcurrsave
```

Appends the current plot set definition to the plot set list .

```
(map_pltcurrsave)
```

Returns T or nil.

This function appends the current plot set definition to the plot set list of the current work session. This list is not saved in the work session until the user executes a save with a call to <u>map_pltdefsave</u>.

To edit and save a plot set definition

- 1. Get the plot set definition with <u>map_pltdefread</u> and <u>map_pltcurrdef</u>.
- 2. Make the necessary changes.
- 3. Save the definition in the list with map_pltcurrsave.
- 4. Save the list in the current work session with map_pltdefsave.
- 5. Save the work session.

If you do not save before the end of the processing, another application can overwrite your changes with a call to map_pltcurrdef.

map_pltcurrset Plotting Functions

Sets the value of an attribute for the current plot set.

(map_pltcurrset attr value)

Returns T or nil.

attr Name of the plot attribute to set (string)

value Value for the <u>plot set attribute</u> (type varies)

This function sets the value of the specified attribute. The **value** data type depends upon the attribute. To set plot attributes, use <u>map_pltdefget</u>.

map_pltdefdelete Plotting Functions

Deletes a plot set definition.

(map_pltdefdelete name)

Returns T or nil.

name Plot set name (string)

The function updates the plot definition dictionary in the current work session.

map_pltdefget Plotting Functions

Gets the value of an attribute of the plot set definition.

(map_pltdefget name attr)

Returns the value of the specified attribute or nil.

name Plot set name (string)

attr Name of the <u>plot set attribute</u> to retrieve (string).

To get the value of an attribute for the current plot set, use <u>map_pltCurrGet</u>. The data type of the return value depends on the attribute.

map_pltdeflist <u>Plotting Functions</u>

Gets the names of available plot set definitions.

(map_pltdeflist)

Returns a list of plot set names or **nil** if no plot sets are available.

This function returns the available plot set definitions (strings) in the plot definition dictionary in the current work session.

Before using this function, you must call <u>map_pltdefread</u>.

map_pltdefread <u>Plotting Functions</u>

Reads in a plot set definition.

(map_pltdefread)

Returns T if plot set definitions are available to read or nil.

This function provides access to plot sets in the plot definition dictionary for the current work session.

Warning If you do not call <u>map_pltdefsave</u>, a new call to <u>map_pltdefread</u> will erase your new plot set definition or your changes to an existing plot set.

map_pltdefsave

Writes the current plot set definition to the plot definition dictionary .

(map_pltdefsave)

Returns **T** if plot set definitions are available to save or **nil**.

This function stores the plot set definition list in the current work session.

If you do not call this function, a new call to <u>map_pltdefread</u> will erase your new plot set definition or your changes to an existing plot set.

map_pltdefvalid

Plotting Functions

Performs a cursory check of the validity of a plot set .

(map_pltdefvalid name)

Returns T or nil.

name Name of the plot set (string)

The function performs a cursory check of the given plot set to see if all required attributes have been set. It does not attach and query the boundary drawing. When used within a dialog box, this call can quickly check on a plot set's usability.

If the check fails, you can use data extension error message functions (<u>ade_err[xx]</u>) to retrieve errors from the error stack.

To perform an extensive check, use the <u>map_pltdefverify</u> function.

map_pltdefverify <u>Plotting Functions</u>

Performs an extensive check of the validity of a plot set .

(map_pltdefverify name)

Returns **T** if the plot set is valid or **nil**.

name Name of the plot set (string)

This function makes sure that no errors occur when plotting takes place. Since the check includes querying for all boundary objects, it could take some time.

To perform a cursory check, use the <u>map_pltdefvalid</u> function.

If the check fails, you can use data extension error message functions (<u>ade_err[xx]</u>) to retrieve errors from the error stack.

map_pltdisplay <u>Plotting Functions</u>

Generates the plot display for the specified boundary.

(map_pltdisplay bndryname)

Returns T or nil.

bndryname Name for a boundary object (string)

This function prepares the display to plot for the given plot set definition and boundary object name. Generating the plot display for the specified boundary includes

- Switching to paper mode (if necessary)
- Inserting the layout block
- Mapping boundary object data to layout block attributes (if applicable)
- Executing the query(ies) to collect the objects to plot
- Trimming the objects to the boundary (if applicable)
- Displaying the objects in the view port

You must call map_pltdisplay before you call map_pltplot.

map_pltexecute <u>Plotting Functions</u>

Plots the plot set for the specified plot set name.

(map_pltexecute name)

Returns T or nil.

name Name of plot set to execute (string)

This function generates and issues plots for each defined boundary.

To get a list of available plot set names, use <u>map_pltdeflist</u>.

map_pltinit <u>Plotting Functions</u>

Initializes environment for plotting.

(map_pltinit)

Returns T always.

This function must be called before any other plotting functions. The map_pltinit function modifies the following settings and checks that the plot set description file exists:

AutoCAD Variables

Variable	Setting
CMDECHO	0
EXPERT	1

AutoCAD Map Options

Option	Setting
DontAddObjectsToSaveSet	Т
ActivateDwgsOnAttach	Т
MkSelSetWithQryObj	Τ

To restore these settings to their original values, use <u>map_pltcleanup</u>. Before you can use other plot functions, you must call <u>map_pltinit</u> again.

See <u>ade_prefgetval</u> for a complete list of AutoCAD Map options.

map_pltplot <u>Plotting Functions</u>

Executes the current plot script.

(map_pltplot)

Returns T or nil.

The function plots the current screen display. Like the AutoCAD PLOT command, it executes the plot script of the screen display's plot set definition.

You must call <u>map_pltdisplay</u> before you call this function.

map_pltrestore <u>Plotting Functions</u>

Restores display altered by map_pltdisplay.

(map_pltrestore)

Returns T or nil.
map_topoaudit Map Topology Functions

Checks whether a topology is correct.

(map_topoaudit tpm_id)

Returns T or nil.

tpm_id Unique ID for a topology (real). Topology must be open for Read

The function audits the geometry of a topology to determine whether the geometrical relationships defined by the topology object data are correct. It shows the location of errors.

The following example checks a topology named "parcels" and provides error reporting.

```
(setq tpm_id (tpm_acopen "parcels"))
(if tpm_id (progn
    (setq result (map_topoAudit tpm_id))
    (tpm_acclose tpm_id)
))
(if (null result) (progn
    (setq i 0 nberr (ade_errqty))
    (repeat nberr
        (prompt
```

```
(strcat "\nError " (rtos i 2 0) " of " (rtos nberr 2 0) " : " (ade_errmsg i))
)
(setq i (1+ i))
)
```

Converts all polygons in a topology to closed polylines.

(map_topoclose toponame layer group odata aselink)

Returns a selection set of closed polylines created by the function.

toponame	Unique name for a topology (string). Topology must be closed.
layer	Target layer for closed polylines (string). If layer name is incorrect (for example "@@"), polylines are created in the current layer
group	Group flag for complex polygons: 1 or 0 1 group complex polygons 0 do not group complex polygons
odata	Object data flag that sets whether to copy object data on the polygon centroid to the resulting polyline: 1 or 0 1 copy object data on the centroid 0 ignore object data on the centroid
aselink	ASE link flag that sets whether to copy ASE link data on the centroid to the resulting polyline: 1 or 0 1 copy ASE link data on the centroid 0 ignore ASE link data on the centroid

map_topocomplete <u>Map Topology Functions</u>

Completes all objects in a loaded partial topology.

(map_topocomplete toponame)

Returns a selection set of all objects retrieved to complete the topology.

toponame Unique name for a topology (string). Topology must be closed.

This function performs a query to retrieve objects into the work session. The <u>tpm_infocomplete</u> function determines whether the topology is completely represented in the work session.

This function can only complete objects imported from an existing source drawing. For a polygon topology, this function imports links, nodes, and a centroid, if it is missing, from the source drawing. For a network topology, this function imports links and nodes. For a network topology, it imports only nodes.

map_topostat
<u>Map Topology Functions</u>

Gets statistics for a topology.

(map_topostat tpm_id)

Returns a list containing the statistics for the specified topology, or nil.

tpm_id Unique ID for a topology (real). Topology must be open for read

This function returns a list a list of dotted pairs or nil.

The following dotted pairs apply to all topology types:

(node_count . #nodes)
(link_count . #links)
(polygon_count . #polygons)
((min_x . #n) (min_y . #n))
((max_x . #n) (max_y . #n))

Note This function is not designed to count polygons in a partial topology. If the topology in question is partial, the **polygon_count** statistic may be overstated. This is because **map_topostat** counts not only the polygons in the partial topology, but also any polygons that share common edges with them in the complete topology, even if the adjacent polygons are not actually present in the current drawing.

The following dotted pairs apply to network topologies:

(length_total . #n)
(length_average . #n)
(length_min . #n)
(length_max . #n)
(length_variance . #n)
(length_deviation . #n)

The following dotted pairs apply to polygon topologies:

(area_total . #n) (area_average . #n) (area_min . #n) (area_max . #n) (area_variance . #n) (area_deviation . #n) (perimeter_total . #n) (perimeter_average . #n) (perimeter_min . #n) (perimeter_max . #n) (perimeter_variance . #n) (perimeter_deviation . #n)

Boundary Functions

The map boundary functions begin with map_dwg.

map_dwgBreakObj

Breaks objects where they cross boundary edges.

map_dwgTrimObj

Trims linear objects inside or outside of a specified boundary.

Plotting Functions

The map plotting functions begin with map_plt.

map_pltBlkAtts

Gets a list of block attributes.

map_pltBlkList

Returns a sorted list of block names that are usable as plot layouts.

map_pltBlkVps

Returns a list of valid viewport layers in layout blocks.

map_pltCleanup

Restores settings altered by map_pltInit.

map_pltCurrDef

Selects or creates a plot set.

map_pltCurrDel

Resets a plot set attribute to its default value.

map_pltCurrGet

Retrieves the value for a specific attribute for the current plot set.

map_pltCurrSave

Appends the current plot set definition to the plot set list.

map_pltCurrSet

Sets the value of an attribute for the current plot set.

map_pltDefDelete

Deletes a plot set definition.

map_pltDefGet

Gets the value of an attribute of the plot set definition.

map_pltDefList

Returns a list of available plot set definitions in the project.

map_pltDefRead

Reads in a plot set definition from the plot definition dictionary for the project.

map_pltDefSave

Writes the current plot set definition to the plot definition dictionary.

map_pltDefValid

Tests the plot set definition for validity.

map_pltDefVerify

Validates the given plot set to prevent plotting errors.

map_pltDisplay

Generates the plot display for the specified boundary.

map_pltExecute

Executes a plot, given a specified plot set.

<u>map_pltInit</u>

Initializes environment for plotting.

map_pltPlot

Executes the plot script of the current plot set definition.

map_pltRestore

Restores display altered by map_pltDisplay.

Topology Functions

The map topology functions begin with map_topo.

map_topoAudit

Checks the geometrical relationships defined by the topology object data.

map_topoClose

Converts all polygons in a topology to closed polylines.

map_topoComplete

Completes all objects in a loaded partial topology.

map_topoStat

Gets the statistics for a topology.

tpm_acclose

Closes a topology.

(tpm_acclose tpm_id)

Returns T or nil.

tpm_id Topology ID (real) returned by tpm_acopen

When you close a topology, it remains in memory until you unload it.

tpm_acexist Access Functions

Checks whether a topology exists.

(tpm_acexist toponame [source [loaded]])

Returns **T** if the topology exists or **nil**.

toponame	Topology name (string)
source	Source flag T or nil: T Check topologies in the current and source drawings nil Check the current drawing only (default)
loaded	Loaded in memory flag T or nil : T Check only topologies in memory nil Check all topologies (default)

tpm_acload Access Functions

Loads a topology into memory.

(tpm_acload toponame [source])

Returns T or nil.

toponame	Topology name (<mark>string</mark>)
source	Source flag T or nil :
	T Read from source drawings only
	nil Read from current drawing only (default)

Before loading a topology, you must first test for its existence with <u>tpm_acexist</u>.

Topology information is stored in the drawing, but the topology is not automatically loaded when you open the drawing. If you want, for example, to query, edit, or overlay a topology, you must load it into memory. Once loaded, the topology remains in memory until you unload it.

tpm_acopen Access Functions

Opens a topology.

(tpm_acopen toponame [write_access])

Returns the topology ID or nil.

toponame	Topology name (string)
write_access	Access status; T or nil:
	1 Write
	nil Read only (default)

The tpm_acopen function opens a topology and creates a new tpm_id that provides access to it.

If your application opens a topology with write access, your user is the only one who can edit this topology. No other user can even open it. If your application opens topology with read access, other users can open it also, but with read access only. To find out the access status of an already-open topology, use tpm_infostatus.

If you use tpm_acopen to test the status of a topology, always close the topology with tpm_acclose to ensure that you do not leave the topology open with multiple IDs pointing to it.

A topology loaded from a source drawing cannot be opened for write access. To find out if a topology was loaded from a current drawing, use

tpm_infocurrent.

Important You must store the topology ID when it is returned by tpm_acopen, and make sure that you do not lose it. If you do not have the ID of an open topology, you have no way to get it, and you cannot close the topology. Your alternatives are to quit AutoCAD Map or start a new drawing.

tpm_acqty <u>Access Functions</u>

Counts topologies.

(tpm_acqty [source [loaded]])

Returns a topology count (integer) or nil.

wings
(default)

tpm_acunload

Access Functions

Unloads a topology from memory.

(tpm_acunload toponame)

Returns T or nil.

toponame Topology name (string)

Before a topology can be unloaded, all topology IDs that reference it must be closed with tpm_acclose.

tpm_acupgradeopen Access Functions

Allows write access for a topology that is open without it.

(tpm_acupgradeopen tpm_id)

Returns T or nil.

tpm_id Topology ID (real) returned by tpm_acopen

If a topology is already open with write access, you cannot upgrade access with this function. You cannot upgrade access to write for a source topology.

tpm_anabuffer Analyzing Functions

Creates a buffer space around a topology.

(tpm_anabuffer source_id offset var_id [result_name] [result_desc])

Returns T or nil.

source_id	Source topology ID (real)
offset	Buffer offset (string), a standard Data Extension expression
var_id	Topology variables ID (real)
result_name	Resulting topology name (string) or nil
result_desc	Resulting topology description (string) or nil

The topology variables ID references a set of <u>topology variables</u>.

This function draws one or more buffer perimeters. If the result argument is omitted or nil, the buffered topologies are AutoCAD objects only. Otherwise the function creates a new topology that is loaded but not open.

The source topology must be loaded and open. The Read/Write access of the source topology does not affect this function.

You can buffer any of the three topology types, node, network, or polygon.

tpm_anadissolve <u>Analyzing Functions</u>

Merges connected topology elements that have the same value in the specified field.

(tpm_anadissolve source_id field var_id [result_name] [result_desc]
[objtable] [objcolumn])

Returns T or nil.

source_id	Source topology ID (real). Polygon or network topology only.
field	Dissolve field name (string), a standard Data Extension expression. For example, :Value@Tab1 , .Layer .
var_id	Topology variables ID (real).
result_name	Name of new topology (string); topology is loaded but not open. If argument is omitted or nil , the dissolve results in a collection of AutoCAD objects.
result_desc	Resulting topology description (string) or nil.
objTable	Object data table name (string) or nil.
objColumn	Object data field name (string) or nil.

The topology variables ID references a set of <u>topology variables</u>.

The function works for polygon or network topologies. If it finds two or more attached objects that have the same value in the specified dissolve field, it dissolves them.

- Dissolving polygons deletes their centroids and shared links and creates a new polygon with a new centroid.
- Dissolving links deletes shared nodes. The links are merged into one polyline.

The function writes the shared dissolve field value to the object data table and field that you specify and attaches the table to the result object. If this table does not exist, the function creates it. If the table you specify is already attached to one of the source objects, a table with default values is attached to the result object.

tpm_anaoverlay Analyzing Functions

Overlays two topologies.

(tpm_anaoverlay_overlay_id overlay_data source_id source_data oper var_id [obj_table] [obj_tabledesc] [result_name] [result_desc])

Returns T or nil.

overlay_id	Overlay topology ID (real) of topology 1. Must be a polygon topology.
overlay_data	Overlay data expression composed of AutoCAD Map expressions and output object data columns or nil. See the Overlay Data Expressions section below.
source_id	Source topology ID (real) of topology 2. Must be a polygon topology.
source_data	Overlay data expression composed of AutoCAD Map expressions and output object data columns or nil. See the Overlay Data Expressions section below.
oper	Overlay operation (integer). See the Overlay Operations table below.
var_id	ID of topology variables for building the result topology (real).
obj_table	Name of the new object data table that will contain the final data (created by the function). Cannot be an

obj_tabledesc	Object data table description.
result_name	Result topology name (string). Omit if function results produces AutoCAD objects instead of a topology.
result_desc	Result topology description (string).

The source topology must be of polygon type for union and paste operations. For all other operations, it can be of any topology type, such as node, network, and polygon.

The topology variables ID references a set of <u>topology variables</u>.

```
Overlay Data Expressions
```

The **overlay_data** and **source_data** arguments are overlay data expressions. They have the following format:

```
(list (list expr1 colname1 coldesc1 coltype1)
(list expr2 . . .)
. . .)
```

Overlay Data Expression Arguments

expr1	AutoCAD Map expression (string).
col_name1	Object data column name (string).
col_desc1	Object data column description (string).
col_type1	Object data column type (integer): 1 through 4, where $1 =$ integer, $2 =$ real, $3 =$ character, and $4 =$ point.

Overlay Operations

1	Intersect
2	Union
3	Identity
4	Erase
5	Clip
6	Paste

Result Topology

If the **result_name** argument is omitted or **nil**, the result is a collection of AutoCAD objects. Otherwise, it is a new topology that is loaded but not open.

You must specify the list of data values to come from each input topology and the specific data for each.

Additionally, you must specify the name of the result object data table in the **objTable** parameter to contain the final data. If you do not specify a result table, no data is attached to the resulting topology elements. The function creates this table. If the result table you specify already exists, the function returns an error and cancels the overlay process.

Data derived by the overlay process is also attached to the result object data table. This data is written for each topology element in the resulting topology. It is written for each polygon if the source is a polygon topology, for each link if it is a network topology, or for each node if it is a node topology. The table always includes the following fields:

TOPO_ID	Element ID of the new element in the result topology
T1_ID	Element ID of the parent polygon in the overlay

Field Names in Result Object Data Table

	topology
T1_PERCENTAREA	Area of the new polygon in the result topology compared to the area of the parent polygon in the overlay topology. Written only if both overlay and result are polygon topologies
T2_ID	Element ID of parent element in the source topology. The parent element can be a node, link, or polygon.
T2_PERCENTAREA	Area of the new polygon in the resulting topology compared to the area of the parent polygon in the source topology. Written only if both source and result are polygon topologies

The result table includes these fields along with the fields that you specify in the arguments you supply for the <code>overlay_data</code> and <code>source_data</code> parameters. Each field name is prefixed with T1_ or T2_ to indicate which topology its data comes from.

For example, if you specify FIELD1, FIELD2 and FIELD3 from table SOIL for the first topology and FIELD1, FIELD4 and FIELD5 from table WATER for the second, the result table has the following fields:

T1_SOIL_FIELD1 T1_SOIL_FIELD2 T1_SOIL_FIELD3 T1_WATER_FIELD1 T1_WATER_FIELD4 T1_WATER_FIELD5

The following code overlays two topologies. The names t1 and list1 refer to the overlay topology.

```
(setq t1 (tpm_acopen "top1")); a polygon topology
(setq t2 (tpm_acopen "top2"))
(setq v (tpm_varalloc))
(setq list1
 (list (list ".LAYER" "OVERLAY_LAYER" "" 3)))
(setq list2
 (list (list ".LAYER" "SOURCE_LAYER" "" 3)))
(tpm_anaoverlay t1
 list1
 t2
 list2
 1
 V
 "OVERLAY_TABLE"
 nil
 "OVERLAY_TOPO")
 ; Result_desc argument omitted
```

tpm_cleanactionlistdel

Cleanup Functions

Deletes a cleanup action from the action list.

(tpm_cleanactionlistdel clean_var_id index)

Returns T or nil.

clean_var_id	Cleanup variables ID (real) returned by tpm_varalloc
index	Position in the list of the action to delete (integer)

The **clean_var_id** argument references properties for the cleanup operation that you are preparing to initiate (see <u>Cleanup Variables</u>). These properties include the action list.

The **index** argument is a zero-based position in the action list. A value greater than or equal to the list size or less than **0** returns an error.

```
ADSRX Equivalent
```

int
tpm_cleanactionlistdel
 ade_id clean_var_id,
 long index);

Returns **RTNORM** or an error code.

tpm_cleanactionlistqty <u>Cleanup Functions</u>

Gets the number of cleanup actions in the action list.

(tpm_cleanactionlistqty clean_var_id)

Returns the number of cleanup actions (real) or nil.

clean_var_id Cleanup variables ID (real) returned by tpm_varalloc

The **clean_var_id** argument references properties for the cleanup operation that you are preparing to initiate (see <u>Cleanup Variables</u>). These properties include the action list.

ADSRX Equivalent

int
tpm_cleanactionlistqty
 ade_id clean_var_id,
 long *qty);

Returns **RTNORM** or an error code.

qty Outputs the number of cleanup actions

The ADSRX function passes the number of cleanup actions through a parameter

(as a **long**) instead of returning it (as a **real**) as the AutoLISP function does.

tpm_cleanalloc

Allocates the cleanup model.

(tpm_cleanalloc)

Returns the cleanup model ID (real) or nil.

To clean the objects before they become the elements of a topology, you must construct a model of these objects and their relationships. You can use this model to discover and repair drawing errors that would prevent topology creation.

tpm_cleananchorss <u>Cleanup Functions</u>

Gets a cleanup model's anchored entities.

(tpm_cleananchorss clean_var_id)

Returns a selection set of anchored entities or nil.

clean_var_id Cleanup variables ID (real) returned by tpm_varalloc

Call this function after calling tpm_cleaninit.

ADSRX Equivalent

int
tpm_cleananchorss
 ade_id clean_var_id,
 ads_name ss);

Returns **RTNORM** or an error code.

ss Anchored entities, a selection set

tpm_cleancancel

Cancels the cleanup process .

(tpm_cleancancel clean_id)

Returns T or nil.

clean_id Cleanup model ID returned by <u>tpm_cleanalloc</u>

The function clears the cleanup model without updating the drawing. Do one of the following:

- Because the model is still allocated, you can call <u>tpm_cleaninit</u> using the same cleanup model ID. You can then use <u>tpm_cleanstart</u> to start the cleanup process.
- Because tpm_cleancancel does not cancel initialized values, you can clear the cleanup model and continue without calling tpm_cleaninit.

tpm_cleancomplete <u>Cleanup Functions</u>

Tests for groups to clean.

(tpm_cleancomplete clean_id)

Returns T if no more groups need cleaning or nil.

clean_id Model ID returned by <u>tpm_cleanalloc</u>.

You must call this function after <u>tpm_cleangroupnext</u> to determine if another group needs cleaning, because <u>tpm_cleangroupnext</u> does not provide this information.

You must usually execute several cleanup loops, because fixing one error sometimes causes others. After you process all the groups in the selection set, call tpm_cleanstart to return to the beginning, and then call tpm_cleancomplete to test if cleaning is complete. Repeat the loop until cleaning is complete.

tpm_cleanend

Concludes the cleanup process.

(tpm_cleanend clean_id)

Returns T or nil.

clean_id Model ID (real) returned by tpm_cleanalloc

The function resets the cleanup model and updates the drawing. It fixes errors marked with the <u>tpm_cleanerrorfix</u> function. The model is still allocated. It is possible to call <u>tpm_cleaninit</u> using the same cleanup model ID.

Each error has some default method, which tpm_cleanend uses during error fixing. To change this method, use tpm_cleanerrorset.

tpm_cleanerrorcur

Sets the next error to clean in the current group.

(tpm_cleanerrorcur clean_id index)

Returns T or nil.

clean_id Model ID (real) returned by tpm_cleanalloc

index Index of the error to clean. The index of the first error is 0

The next error to clean is also called the current error.
tpm_cleanerrordraw <u>Cleanup Functions</u>

Creates a temporary marker for the current error.

(tpm_cleanerrordraw clean_id)

Returns T or nil.

clean_id Model ID (real) returned by tpm_cleanalloc

Unlike the persistent markers drawn by <u>tpm_cleanerrormark</u>, these markers are deleted when the drawing redraws.

tpm_cleanerrorfix <u>Cleanup Functions</u>

Fixes the current error.

(tpm_cleanerrorfix clean_id)

Returns T or nil.

clean_id Model ID (real) returned by tpm_cleanalloc

This function marks the current error to fix. It fixes the errors in the cleanup model, but does not fix the drawing. The objects in the drawing are not fixed until you call tpm_cleanend.

Each error has a default method that tpm_cleanend uses during error fixing. You can use tpm_cleanerrorset to change this method. For example, the default method for the dangling node error is to erase the link. If you call tpm_cleanerrorset for this error, the dangling node is moved to a new position, but is not erased.

tpm_cleanerrorget <u>Cleanup Functions</u>

Gets the coordinates of the current error point.

(tpm_cleanerrorget clean_id)

Returns a two-dimensional point or nil.

clean_id Model ID (real) returned by tpm_cleanalloc

The error point is a misplaced node. For example, for a line undershoot, the end of the unattached line is the error point.

To specify the error to get, use <u>tpm_cleanerrorcur</u>.

tpm_cleanerrormark

Creates a persistent marker for the current error.

(tpm_cleanerrormark clean_id)

Returns T or nil.

clean_id Model ID (real) returned by tpm_cleanalloc

Unlike the temporary marker drawn by <u>tpm_cleanerrordraw</u>, these markers are AutoCAD objects, and they become part of the drawing until you perform another cleanup. AutoCAD Map automatically erases persistent markers from any previous cleanup process.

tpm_cleanerrorset

Sets the coordinates of an error fix point.

(tpm_cleanerrorset clean_id coords)

Returns T or nil.

clean_id Model ID (real) returned by tpm_cleanalloc

coords 2D point.

You can use this function to change the default method used by tpm_cleanend during error fixing. For example, the default method for the dangling node error is to erase the link. If you call tpm_cleanerrorset for this error, the dangling node is moved to the new position, but is not erased.

You can also use this function to merge a cluster of points to the point you specify.

To tell the clean engine which error to set, use <u>tpm_cleanerrorcur</u>.

tpm_cleanfree <u>Cleanup Functions</u>

Frees the cleanup model.

(tpm_cleanfree clean_id)

Returns T or nil.

clean_id Model ID (real) returned by tpm_cleanalloc

tpm_cleangroupdraw Cleanup Functions

Creates temporary markers for all errors of the current group.

(tpm_cleangroupdraw clean_id)

Returns T or nil.

clean_id Model ID (real) returned by tpm_cleanalloc

tpm_cleangroupfix <u>Cleanup Functions</u>

Fixes all errors of the current group.

(tpm_cleangroupfix clean_id)

Returns T or nil.

clean_id Model ID (real) returned by tpm_cleanalloc

This function marks errors in the entire current group for fixing. It fixes the errors in the cleanup model, but does not fix the drawing. The objects in the drawing are not fixed until you call tpm_cleanend.

To mark only the current error, use tpm_cleanerrorfix.

tpm_cleangroupmark <u>Cleanup Functions</u>

Creates persistent markers for all errors of the current group.

(tpm_cleangroupmark clean_id)

Returns T or nil.

clean_id Model ID (real) returned by tpm_cleanalloc

tpm_cleangroupnext <u>Cleanup Functions</u>

Goes to the next error group.

(tpm_cleangroupnext clean_id)

Returns T or nil.

clean_id Model ID (real) returned by tpm_cleanalloc

The function always returns **T**, and fails only when the **clean_id** parameter is invalid. It cannot inform you when there are no more groups to clean. To check for this condition, use <u>tpm_cleancomplete</u>.

tpm_cleangroupqty <u>Cleanup Functions</u>

Counts the errors in the current group.

(tpm_cleanqroupqty clean_id)

Returns the error count (real) or nil.

clean_id Model ID (real) returned by tpm_cleanalloc

In the ADSRX equivalent function, the error count type is **long**. Because this 32-bit data type is not available in Visual LISP, the Visual LISP function uses the 32-bit type **real**.

tpm_cleangroupsubtype <u>Cleanup Functions</u>

Determines the subtype of the current group.

(tpm_cleangroupsubtype clean_id)

Returns a subtype code or nil.

clean_id Model ID (real) returned by tpm_cleanalloc

Subtype Codes

1	Degenerate entities (subtype of Erase Short Objects)
2	Short entities (subtype of Erase Short Objects)
3	Short segments (subtype of Erase Short Objects)
4	Vertex is near to segment (subtype of Extend Undershoots)
5	Vertex is near to vertex (subtype of Extend Undershoots)

Codes 1 through 3 are subtypes of the short type. Codes 4 and 5 are subtypes of the undershoot type. This function is useful for short and undershoot errors only.

Degenerate Entities A degenerate polyline has only one vertex. This invalid vertex type sometimes results from the drawing cleanup process, and can be removed by further drawing cleanup.

tpm_cleanincludess <u>Cleanup Functions</u>

Gets a cleanup model's Include set.

(tpm_cleanincludess clean_var_id)

Returns the Include set, the selection set of entities to be cleaned.

clean_var_id Cleanup variables ID (real) returned by tpm_varalloc

Call this function after calling tpm_cleaninit.

ADSRX Equivalent

int
tpm_cleanincludess
 ade_id clean_var_id,
 ads_name ss);

Returns **RTNORM** or an error code.

SS Entities to be cleaned, a selection set

tpm_cleaninit <u>Cleanup Functions</u>

Initializes the cleanup model.

(tpm_cleaninit clean_id var_id ss)

Returns T or nil.

clean_id	Cleanup model ID (real) returned by tpm_cleanalloc
var_id	Cleanup variables ID (real) returned by tpm_varalloc
SS	Selection set or nil (see note below about INCLUDEOBJS_AUTOSELECT)

You can free the selection set after the cleanup model has been initialized.

The function reads the cleanup settings and the selected entities into the cleanup model.

If the cleanup variable **INCLUDEOBJS_AUTOSELECT** is set to 1 (select all), all entities in the drawing will be included, regardless of the **SS** argument, in which case this argument can be **nil** instead of a selection set, as the following code sample illustrates.

```
(setq clean_id (tpm_cleanalloc))
(setq var_id (tpm_varalloc))
(tpm_varset var_id "INCLUDEOBJS_AUTOSELECT" 1)
(setq ssInclude nil)
```

(tpm_cleaninit clean_id var_id ssInclude)

If the cleanup variable **INCLUDEOBJS_LAYERS** is set to "*" (all layers), all **ss** entities will be included. If it contains a list of layers, **ss** entities will be included only if they reside on those layers.

The cleanup variables ID references a set of <u>cleanup variables</u>.

tpm_cleaninitanchorset <u>Cleanup Functions</u>

Specifies anchored entities for the cleanup model.

(tpm_cleaninitanchorset clean_id clean_var_id ssanchor)

Returns t or nil.

clean_id	Cleanup model ID (real) returned by tpm_cleanalloc
clean_var_id	Cleanup variables ID (real) returned by tpm_varalloc
ssanchor	Entities to be anchored, a selection set

Anchored entities remain fixed in position during the cleanup process. You can free the selection set after anchored entities have been specified.

Call tpm_cleaninitanchorset before calling tpm_cleaninit.

The **clean_var_id** argument references properties for the cleanup operation that you are preparing to initiate (see <u>Cleanup Variables</u>).

If the cleanup variable **ANCHOROBJS_LAYERS** is set to "*" (all layers), all **ssAnchor** entities will be anchored. If it contains a list of layers, **ssAnchor** entities will be anchored only if they reside on those layers.

ADSRX Equivalent

tpm_cleaninitanchorset
 ade_id clean_id,
 ade_id clean_var_id,
 ads_name ssAnchor);

Returns **RTNORM** or an error code.

tpm_cleanprofileload <u>Cleanup Functions</u>

Loads a drawing cleanup profile.

(tpm_cleanprofileload clean_var_id filename)

Returns T or nil.

clean_var_id	Cleanup variables ID (real) returned by tpm_varalloc
filename	Full path and .dpf file name (string)

Note Loading a profile will reset all properties referenced by the <u>clean_var_id</u> argument.

ADSRX Equivalent

int
tpm_cleanprofileload
 ade_id clean_var_id,
 char *filename);

Returns **RTNORM** or an error code.

tpm_cleanprofilesave <u>Cleanup Functions</u>

Saves a drawing cleanup profile.

(tpm_cleanprofilesave clean_var_id filename)

Returns T or nil.

clean_var_id	Cleanup variables ID (real) returned by tpm_varalloc
filename	Full path and .dpf file name (string)

ADSRX Equivalent

int
tpm_cleanprofilesave
 ade_id clean_var_id,
 char *filename);

Returns **RTNORM** or an error code.

tpm_cleanstart <u>Cleanup Functions</u>

Starts the cleanup process.

(tpm_cleanstart clean_id)

Returns T or nil.

clean_id Cleanup model ID (real) returned by tpm_cleanalloc

The function finds the first group type or subtype that contains errors. See <u>tpm_cleangrouptype</u> and <u>tpm_cleangroupsubtype</u> for lists of types.

Cleanup is an iterative process. Depending on cleanup options and processing order, you may need to run the cleanup loop several times to achieve the desired results. After selecting the groups to process with **tpm_cleangrouptype** and processing all the groups in the selection set, call <u>tpm_cleancomplete</u> to test if cleaning is complete. If complete, call <u>tpm_cleanstart</u> and repeat the cleanup loop.

tpm_editaddelem

Editing Functions

Adds an element to a topology.

(tpm_editaddelem tpm_id type elem)

Returns T or nil.

tpm_id	Topology ID (real).
type	Element type code (integer): 1, 2, or 3, where $1 = Node$, $2 = Link$, and $3 = Polygon$.
elem	Element to add. Depending on the type argument, specify one of the following: If 1 , specify a point or the entity name of a point object. If 2 , specify the entity name of a line object. If 3 , specify a selection set.

The topology must be open with Write access. If you add a node to a link, the link is split.

tpm_editdelelem

Editing Functions

Deletes an element from a topology.

(tpm_editdelelem tpm_id elem_id [delobjs])

Returns T or nil.

tpm_id	Topology ID (real)
elem_id	Element ID (real)
delobjs	T or nil: T Delete object from the drawing also nil Delete object from the topology only

This function deletes elements from a topology. It does not erase corresponding entities in the drawing unless the **delobjs** argument is **T**.

Warning Deleting an element can cause other deletions.

- If you delete a node, you delete any link or polygon that contains it.
- If you delete a link, you delete only the nodes belonging to that link. If the link belongs to one polygon only, you delete the polygon. If the link is shared by two polygons, you merge the polygons.
- If you delete a polygon, you delete any node or link belonging to that polygon only.

tpm_editmodelem

Editing Functions

Modifies a topology element.

(tpm_editmodelem tpm_id elem_id new_val)

Returns T or nil.

tpm_id	Topology ID (real)
elem_id	Element ID (real)
new_val	List consisting of a code for the property to modify and a new value for the property. See Properties and Values below.

Properties and Values

(10 . point)	New coordinates of node or centroid (point)
(40.f_res)	Resistance of node (real), or forward resistance of link
(41.r_res)	Reverse resistance of link (real)
(70 . dir)	Link direction (integer): -1 Reverse 0 Bidirectional 1 Forward

The following example assigns new coordinates to the first node in a node

topology.

```
(setq node_ss
 (ssget))
(setq var_id (tpm_varalloc))
(setq result
 (tpm_mntbuild
   var_id
   "test"
   "test descr"
   1
   node_ss))
(setq tpm_id
 (tpm_acopen "test" T)) ; open for write
(setq elem_id
 (tpm_elemid tpm_id 1 0)) ; gets first node
(setq result
 (tpm_editmodelem
   tpm_id
   elem_id
   (cons 10 (list 1.0 1.0 0.0)))))
```

tpm_editupdelem Editing Functions

Updates a topology element.

(tpm_editupdelem tpm_id elem_id)

Returns **T** if the object was succesfully updated or **nil** if it was not.

tpm_id Topology ID (real) elem_id Element ID (real)

This function updates a topology element so that it reflects the current state of the corresponding entity in AutoCAD. This function is normally used in conjunction with AutoCAD Notification so that changes made in AutoCAD can be reflected in the topology model.

tpm_elemadj Element Information Functions

Compiles a list of adjacent elements for the specified element.

(tpm_elemadj tpm_id elem_id adj_type)

Returns a list of element IDs or nil.

tpm_id	Topology ID (real)
elem_id	Element ID, received from <u>tpm_elemid</u>
adj_type	Type of adjacent elements (int). Values: 1 Node 2 Pink 3 Polygon

tpm_elemfind

Element Information Functions

Finds an element within a topology.

(tpm_elemfind tpm_id type pattern)

Returns an element ID (real) or nil.

- tpm_id Topology ID (real)
 type Type of element to find (int). Values:
- 1 Node
 - 2 Link
 - 3 Polygon
- pattern Point or entity name. If pattern is a point, in which case type must be 1, the function returns the nearest point or link, or the enclosing polygon. If pattern is an entity name, in which case type can have any value, the function returns the corresponding object.

The following example gets a point from the user and finds the nearest node in the topology named Parcel.

(setq pt (getpoint))
(setq topo_id (tpm_acopen "Parcel"))
(setq node_id (tpm_elemfind topo_id 1 pt))

The result is a node ID, such as 4.71389e+007.

tpm_elemget Element Information Functions

Lists information about an element of a topology.

(tpm_elemget tpm_id elem_id)

Returns a list of dotted pairs or nil.

tpm_id	Topology ID (real)
elem_id	Element ID (tpm_id) returned by tpm_elemid

For each dotted pair in the list that is returned, the first value is an integer code for the information type, and the second value is the information. The list format depends on the element type: node, link, or polygon.

Information List Format for Nodes

Туре	Information
0	Persistent topology ID (real).
-1	Element type code (integer). With node lists, always 1, meaning <i>node element</i> .
-2	Entity name of the node object (real).
10	Coordinates of the node object (point).
40	Node resistance (real). Relevant only for nodes belonging to network

Information List Format for Links

Туре	Information
0	Persistent topology ID (real).
-1	Element type (integer). With link lists, always 2, meaning <i>link element</i> .
-2	Entity name of the link object (real).
1	Topology ID of start node (real).
2	Topology ID of end node (real).
3	Topology ID of left polygon (real). Relevant only if the link belongs to a polygon topology. Links in a polygon topology can belong to two adjacent polygons, one on the left, and one one the right.
4	Topology ID of right polygon that shares this link (real). Relevant only if the link belongs to a polygon topology.
40	Forward resistance of the link (real).
41	Reverse resistance of the link (real).
70	Link direction (integer): -1, 0, or 1, where $-1 =$ Reverse, $0 =$ Bidirectional, and $1 =$ Forward.

Information List Format for Polygons

Туре	Information
0	Persistent topology ID (real).
-1	Element type (integer). With polygon lists, always 3 , meaning <i>polygon element</i> .

-2	Entity name of the polygon centroid (real).
10	Coordinates of the polygon centroid (point).
50	Perimeter of the polygon (real).
51	Area of the polygon (real).

When a topology is built, it is given a set of <u>object data fields</u>. Their purpose is to contain the information listed in the preceding tables.

tpm_elemid Element Information Functions

Gets the ID of an element.

(tpm_elemid tpm_id type index)

Returns an element ID (real) or nil.

tpm_id	Topology ID (real)
type	Element type (<mark>integer</mark>). Values: 1 Node 2 Pink 3 Polygon
index	Element index (<mark>real</mark>). The index of the first element is <mark>0</mark>

tpm_elemqty Element Information Functions

Counts topology elements.

(tpm_elemqty tpm_id type)

Returns an element count (real) or nil.

tpm_id	Topology ID (real)
type	Element type (int). Values: 1 Node 2 Link 3 Polygon

In the ADSRX equivalent to this function, the element count type is **long**. Because this 32-bit data type is not available in Visual LISP, the Visual LISP function uses **real**, another 32-bit type.

tpm_elemss

Element Information Functions

Creates a selection set of elements of a given type.

(tpm_elemss tpm_id type)

Returns a selection set or nil.

tpm_id Topology ID.

type Element type (int). Values:

1 Node

2 Pink

3 Polygon

tpm_infobuildvar

Topology Information Functions

Gets the configuration values of a topology.

(tpm_infobuildvar tpm_id var_id)

Returns T or nil.

tpm_id	Topology ID (real)
var_id	Topology variables ID (real)

The topology variables ID references a set of <u>topology variables</u> in which to store the values that this function gets.

If no topology variables are allocated, call <u>tpm_varalloc</u> to allocate a set of them and return their ID.

To read the values that this function gets, use <u>tpm_varget</u> or <u>tpm_varlist</u>. To build a new topology using these variables, use <u>tpm_mntbuild</u>.

The following code opens a topology, reads its configuration values, and gets its node layer.

```
(setq tpm_id (tpm_acopen "SampleTopo"))
(setq var_id (tpm_varalloc))
(tpm_infobuildvar tpm_id var_id)
(tpm_acclose tpm_id)
(setq nlayer (tpm_varget var_id "NODE_LAYER"))
```

tpm_infocomplete <u>Topology Information Functions</u>

Tests if a polygon topology is complete .

(tpm_infocomplete tpm_id)

Returns **T** if the topology is complete, **nil**.

tpm_id Topology ID of a polygon topology.

Use this function to verify that a polygon topology is complete before performing an element trace, overlay, or other topology operation. This function applies only to polygon topologies.

A topology is considered complete if all necessary entities exist in the current drawing. It is considered incomplete if at least one polygon is incomplete.

A query can bring an incomplete topology into the current drawing. Because this part has the same name as the complete topology that it came from, the part could be mistaken for the whole. Although you can execute any topology operation on an incomplete topology, the result may not be what you intend.
tpm_infocorrect <u>Topology Information Functions</u>

Tests whether a topology is correct and performs a geometrical audit .

(tpm_infocorrect tpm_id)

Returns T or nil.

tpm_id Topology ID.

If topology entities are changed using drawing tools, the topology information is modified according to the geometrical changes. If this is impossible, the topology becomes incorrect. Use this function to verify that a topology is correct before performing an element trace, overlay, or other topology operation. See also tpm_infomodified.

To fix an incorrect topology, try unloading and reloading it. If it does not reload, you must use topology edit to fix the topology. If this does not work, rebuild the topology.

Note This function indicates when an error occurs, but does not show its location.

tpm_infocurrent
Topology Information Functions

Checks the source from which the topology was loaded.

(tpm_infocurrent tpm_id)

Returns **T** if the topology is loaded from the current drawing, or **nil** if it comes from source drawings.

tpm_id Topology ID.

tpm_infodesc Topology Information Functions

Gets a topology description.

(tpm_infodesc tpm_id)

Returns a topology description (string) or nil.

tpm_id Topology ID (real)

tpm_infomodified

Topology Information Functions

Checks if topology elements have been modified using drawing tools.

(tpm_infomodified tpm_id)

Returns **T** if any elements have been modified, otherwise **nil**.

tpm_id Topology ID.

If topology elements have been modified using drawing tools, then possibly they are no longer correct topologies. See also <u>tpm_infocorrect</u>.

tpm_infoname Topology Information Functions

Gets a topology name.

(tpm_infoname tpm_id)

Returns a topology name (string) or nil.

tpm_id Topology ID (real)

tpm_infostatus Topology Information Functions

Checks whether the topology is open for Read or Write access.

(tpm_infostatus tpm_id)

Returns **T** if the topology is open for Write or **nil** if open for Read.

tpm_id Topology ID (real)

tpm_infotype <u>Topology Information Functions</u>

Gets a topology type.

(tpm_infotype tpm_id)

Returns a topology type code (int) or nil.

tpm_id Topology ID (real)

Topology type codes: 1 = node, 2 = network, 3 = polygon.

tpm_infoversion <u>Topology Information Functions</u>

Gets a topology version.

(tpm_infoversion tpm_id)

Returns a topology version (string) or nil.

tpm_id Topology ID (real)

A topology's version is the Data Extension version (ADE) in which the topology was created. For example, "2.026". This function gets the same result as <u>tpm_iterversion</u>. The difference is that <u>tpm_infoversion</u> requires a topology ID, and it works only on topologies that are open.

tpm_iterdesc Topology Iterating Functions

Gets a topology description.

(tpm_iterdesc iter_id)

Returns a topology description (string) or nil.

iter_id Iterator ID.

The function gets a description of the topology that the iterator points to.

For an example that shows how you can use Topology Iterating functions to find all the topologies the system knows about, see <u>tpm_iterstart</u>.

tpm_itername Topology Iterating Functions

Gets a topology name.

(tpm_itername iter_id)

Returns a topology name (string) or nil.

iter_id Iterator ID (real)

The function gets the name of the topology that the iterator points to.

For an example that shows how you can use Topology Iterating functions to find all the topologies the system knows about, see <u>tpm_iterstart</u>.

tpm_iternext Topology Iterating Functions

Moves the iterator to the next topology.

(tpm_iternext iter_id)

Returns **T** or **nil** if another topology is not present.

iter_id Iterator ID (real)

The first time this function is called after <u>tpm_iterstart</u>, it sets the iterator on the first topology definition.

For an example that shows how you can use Topology Iterating functions to find all the topologies the system knows about, see tpm_iterstart.

tpm_iterstart Topology Iterating Functions

Allocating a topology iterator.

(tpm_iterstart [source [loaded]])

Returns an iterator ID (real) or nil.

source	Source flag (int):
	${ m T}$ Iterate through the current and source drawings
	nil Iterate through the current drawing only (default)
loaded	Loaded in memory flag (int):
	T Iterate through topologies in memory only
	nil Iterate through all topologies (default)

This function allocates an iterator and positions it before the first topology definition. This behavior has implications to remember when you use the function.

- Because tpm_iterstart always generates an iterator ID, even if the drawing has no topologies to iterate through, the function fails only when it is out of memory.
- Because tpm_iterstart positions the iterator before the first topology definition, the function cannot indicate whether any topologies exist in the drawing. The only way to determine whether the drawing has topologies is to call tpm_internext, which fails if no topology exists beyond the current position of the iterator.

You can have more than one iterator running at the same time.

This example shows how you can use Topology Iterating functions to find all the topologies the system knows about.

```
(setq itr_id (tpm_iterstart))
(if (null itr_id)
  (prompt "\nERROR: Unable to start topology iterator.")
  (while (not done)
    (if (null (tpm_iternext itr_id))
      (setq done T)
      (progn
        (setq lst (list
          (tpm_itername itr_id)
          (tpm_itertype itr_id)
          (tpm_iterdesc itr_id) ) )
        (setq tpmlist (cons lst tpmlist))
      ); progn
    ); if
  ); while
); if
```

tpm_iterstop Topology Iterating Functions

Frees an iterator.

(tpm_iterstop iter_id)

Returns T or nil.

iter_id Iterator ID (real)

tpm_itertype
<u>Topology Iterating Functions</u>

Gets a topology type.

(tpm_itertype iter_id)

Returns a topology type code (int) or nil.

iter_id Iterator ID (real)

The function gets the type of the topology that the iterator is pointing to. The topology type codes are 1 = node, 2 = network, 3 = polygon.

For an example that shows how you can use Topology Iterating functions to find all the topologies the system knows about, see <u>tpm_iterstart</u>.

tpm_iterversion
Topology Iterating Functions

Gets a topology version.

(tpm_iterversion iter_id)

Returns a topology version (string) or nil.

iter_id Iterator ID (real)

The function gets the version of the topology that the iterator is pointing to. The version of a topology is the version of ADE in which it was created, for example, "2.026". The function gets the same result as tpm_infoversion. The difference is that tpm_infoversion requires a topology ID, and so it works only on topologies that are open.

tpm_mntbuild Building and Erasing Functions

Builds a topology.

(tpm_mntbuild var_id name desc type node_ss [link_ss cntr_ss])

Returns T or nil.

var_id	Topology variables ID (real)
name	Topology name (string)
desc	Topology description (string)
type	Topology type code (int): 1 Node 2 Network 3 Polygon
node_ss	Entity selection set of nodes or nil
link_ss	Entity selection set of links or nil . Omit for node topology
cntr_ss	Entity selection set of centroids or nil . Omit for node and network topology

The topology variables ID references a set of <u>topology variables</u>.

The new topology is loaded, but closed. You must open it with <u>tpm_acopen</u>.

The following AutoCAD object types are acceptable for topology elements:

- For links: line, arc, circle, 2D and 3D polyline
- For nodes and centroids: point, insert, and text

A node topology can contain only nodes. A network topology can contain nodes or links, but not centroids. A polygon topology can contain all three.

When the topology is built, all links are assigned a default direction of bidirectional (a value of 0). The forward and reverse resistance values are the length of the link. Nodes are assigned a resistance of 0.

When a topology is built, it is given a set of <u>object data fields</u>. These fields contain information about the elements of the topology.

tpm_mnterase Building and Erasing Functions

Erases a closed topology from the current drawing.

(tpm_mnterase toponame)

Returns T or nil.

toponame Topology name (string)

This function can erase a closed topology whether or not it is loaded.

tpm_mntrebuild Building and Erasing Functions

Rebuilds a topology.

(tpm_mntrebuild toponame)

Returns T or nil.

toponame Topology name (string)

The rebuilt topology is loaded, but closed. You must open it with <u>tpm_acopen</u>.

Rebuilding a topology restores all its object data fields to their default values. Any object data fields modified after the topology was built are lost. See <u>Topology Object Data</u>.

Whether you need to rebuild a topology after you change it depends upon the functions used to make the changes.

• If changes were made using AutoCAD drawing and editing functions, you may need to rebuild the topology.

If the AutoCAD alterations introduce an error, the rebuild could fail. If this happens, you must clean the objects again and use <u>tpm_mntbuild</u>. You can use <u>tpm_infocorrect</u> to check for errors before attempting tpm_mntrebuild.

• If changes were made using Topology functions, you do not have to rebuild the topology. This applies to objects altered with functions such as <u>tpm_editaddelem</u>, <u>tpm_editdelelem</u>, and <u>tpm_editmodelem</u>.

tpm_mntrename

Building and Erasing Functions

Renames a topology.

(tpm_mntrename toponame newname newdesc)

Returns T or nil.

toponameTopology name (string).newnameNew topology name (string).newdescNew topology description (string)

tpm_qrygetresdesc Topology Query Functions

Gets the description of the query result topology.

(tpm_qrygetresdesc)

Returns a topology description (string) or nil if there is no result topology or it has no description.

Gets the name of the query result topology.

(tpm_qrygetrestopo)

Returns a topology name (**string**) if there is a result topology or **nil** if there is not. If no result topology is defined, then no objects are added to the current drawing when the query executes.

If the name of the query result topology starts with *, the result topology is a temporary topology

tpm_qrygettoponame Topology Query Functions

Gets the name of the query source topology .

(tpm_qrygettoponame)

Returns a topology name (string) if there is a query source topology or **nil** if there is not. If there is a query source topology, the current query is a topology query. If there is not, the current query is a standard data extension query.

tpm_qrysetrestopo Topology Query Functions

Defines or or undefines a query result topology.

(tpm_qrysetrestopo result_name result_desc)

Returns T or nil.

result_name	Name of the query result topology (string) or nil
result_desc	Description of the query result topology (string) or nil

A description is optional. Its presence or absence has no effect on the query. Regarding the name:

- If no name is specified, then no query result topology is defined. If a definition already exists, it is canceled.
- If the name begins with an asterisk (*), the next query result is a temporary topology.
- If the name begins with any other acceptable character, the next query result is a standard topology.

tpm_qrysettoponame <u>Topology Query Functions</u>

Defines or undefines a topology query.

(tpm_qrysettoponame toponame)

Returns T or nil.

toponame Name of topology to query (string) or nil

This function determines whether the current query is a topology query or a standard data extension query as follows:

- If the toponame argument is nil, the current query becomes a standard data extension query.
- If the toponame argument is a name, the current query becomes a topology query. But if you specify a topology that does not exist, a subsequent call to ade_qryexecute returns nil.

tpm_tracealloc <u>Network Tracing Functions</u>

Allocates the tracing model.

(tpm_tracealloc tpm_id [node_res link_dir link_forward_res link_reverse_res])

Returns the tracing model ID (real) or nil.

tpm_id	Topology ID (real)
node_res	Expression for node resistance (string): nil = Default resistance
link_dir	Expression for link direction (string): nil = Default direction
link_forward_res	Expression for forward link resistance (string): nil = Default resistance
link_reverse_res	Expression for reverse link resistance (string): nil = Default resistance

This function sets the values for the specified topology to the parameters you enter. The values stored in <u>topology object data</u> when the topology was created are used as defaults if you omit parameters. You can enter any valid expression that evaluates to a numeric result.

The tracing model can be used only with a network or polygon topology.

When you enter a value other than **nil** for any of the optional resistance arguments, this value is used for all objects in the topology of the appropriate type. It overrides the corresponding value attached to the object. For example, if **node_res** is set to **nil**, the tracing model uses the value attached to topology object data when calculating the trace. If **node_res** is set to 10.0, all nodes in the topology are overridden with the value of 10.0 when the trace is calculated.

If you want to omit the optional parameters, you can either enter **nil** to invoke a default value or leave out the parameter altogether, as with other Visual LISP functions. However, before you omit optional parameters, note the dependency relationships indicated by bracketed groups. For example, if you want to use the link_dir parameter, you must enter a **node_res** parameter.

Important! When using the Topology API to perform a network trace, such as a Best Route analysis, the source topology used to create the trace topology should not be unloaded or erased until after all API calls relating to the trace have been made. This is because the trace topology references the nodes and links in the source topology. It does not create its own. So, in order to do anything with the elements of the trace, the source topology must remain loaded.

For example, the following Best Route code will silently fail:

; topo_id is the netTopo topology (tmp_id) (setq topo_id (tpm_acopen topo_name T))

; create the trace model
(setq network_trace_id (tpm_tracealloc topo_id))

```
; create the best route
(setq bestroute (tpm_tracebestroute topo_id network_trace_id 7.0 1.0 4.0 9.0 5.
```

; remove the netTopo topology ; THIS SHOULD NOT BE DONE UNTIL AFTER BESTROUTE TOPO QUE (tpm_acclose topo_id) (tpm_acunload topo_name)

; this next call fails because the element info is on the source topology (setq BestRouteTraceElement (tpm_tracebestroutescan network_trace_id 3))

The simple solution is to defer the calls to tpm_acclose and tpm_acunload until after all tpm_tracebestroutescan calls.

tpm_traceelemedit

Network Tracing Functions

Modifies a tracing model element.

(tpm_traceelemedit trace_id elem_id new_val)

Returns T or nil.

trace_id	Model ID (returned by <u>tpm_tracealloc</u>)
elem_id	Element ID (real)
new_val	List consisting of a code for the property to modify and a new value for the property. See Properties and Values below.

Properties and Values

(40.f_res)	Resistance of node (real), or forward resistance of link
(41.r_res)	Reverse resistance of link (real)
(70 . dir)	Link direction (integer): -1 Reverse 0 Bidirectional 1 Forward

tpm_traceelemget Network Tracing Functions

Lists information about a tracing model element.

(tpm_traceelemget trace_id elem_id)

Returns an information list or nil.

trace_id	Tracing model ID (real) returned by tpm_tracealloc
elem_id	Trace element ID (real)

The list format depends on the element type. For each a-list, the first component is an integer code for the information type, and the second is the information.

List Format for Nodes

(-1 . elem_code)	Element type code (<mark>integer</mark>). With node lists, always 1, meaning node element.
(40 . resistance)	Node resistance (real).

List Format for Links

(-1 . elem_code)	Element type code (<mark>integer</mark>). With link lists, always 2, meaning link element.
(1 . topo_id)	Topology ID of start node (real).

(2 . topo_id)	Topology ID of end node (real).
(40 . fwd_resist)	Forward resistance (real).
(41 . rev_resist)	Reverse resistance (real).
(70 . link_dir)	Link direction (integer): -1, 0, or 1. -1 = Reverse 0 = Bidirectional 1 = Forward

tpm_traceelemid

Network Tracing Functions

Returns the ID of the selected element.

(tpm_traceelemid trace_id type index)

Returns an element ID (real) or nil.

trace_id	Tracing model ID (real) returned by <u>tpm_tracealloc</u>
type	Element type: 1 Node 2 Link
index	Element index. The first element's index is 0

This function returns the ID of the element selected. Use this function to find the trace result in the source topology after performing a trace with <u>tpm_traceshort</u> or <u>tpm_traceflood</u>.

This example performs a short path trace and gets the ID of the first link of the result path.

```
; open topology to trace
(setq tpm_id (tpm_acopen "nettopo" nil))
; create the model
(setq trc_id (tpm_tracealloc tpm_id))
; find shortest path
```

(tpm_traceshort trc_id start_id end_id)
; number of links in path
(setq qty (tpm_traceqty trc_id 2))
; get first link of path
(setq id (tpm_traceelemid trc_id 2 0))

tpm_traceflood <u>Network Tracing Functions</u>

Calculates flood paths.

(tpm_traceflood trace_id start maxres)

Returns T or nil.

trace_idTracing model ID, returned by tpm_tracealloc (real)startElement ID of start node (real)maxresMaximum allowed accumulated path resistance (real)

The accumulated resistance value is the total resistance of the nodes and links that make up the flood trace.

tpm_tracefree
<u>Network Tracing Functions</u>

Frees a tracing model.

(tpm_tracefree trace_id)

Returns T or nil.

trace_id Tracing model ID (real) returned by tpm_tracealloc
tpm_traceqty
<u>Network Tracing Functions</u>

Counts the selected elements after a trace.

(tpm_traceqty trace_id type)

Returns an element count (real) or nil.

trace_id Tracing model ID, returned by tpm_tracealloc. type Element type (integer): 1 Node 2 Link

tpm_tracesetmaxres Network Tracing Functions

Sets maximum resistance for the shortest path algorithm .

(tpm_tracesetmaxres trace_id maxres)

Returns T or nil.

trace_id Tracing model ID (real), returned from tpm_tracealloc

maxres Maximum allowed accumulated path resistance (real)

This function sets an accumulated resistance value for path tracing.

For the shortest path trace to succeed, the total calculated resistance cannot be greater than the value set for the maximum resistance or less than the value set for the minimum resistance. See tpm_tracesetminres. See tpm_traceshort.

tpm_tracesetminres Network Tracing Functions

Sets minimum resistance for the shortest path algorithm.

(tpm_tracesetminres trace_id minres)

Returns T or nil.

trace_id Tracing model ID (real), returned from tpm_tracealloc

minres Minimum accumulated allowed path resistance (real)

This function sets an accumulated resistance value for path tracing. For the shortest path trace to succeed, the total calculated resistance cannot be greater than the value set for the maximum resistance or less than the value set for the minimum resistance. See <u>tpm_tracesetmaxres</u>. See <u>tpm_traceshort</u>.

tpm_traceshort <u>Network Tracing Functions</u>

Calculates the shortest path between two nodes.

(tpm_traceshort trace_id start end)

Returns T or nil.

trace_id	Tracing model ID (real), returned by tpm_tracealloc
start	Element ID of start node (real)
end	Element ID of end node (real)

For the shortest path trace to succeed, the total calculated resistance cannot be greater than the value set for the maximum resistance or less than the value set for the minimum resistance. See <u>tpm_tracesetminres</u> and <u>tpm_tracesetmaxres</u>. The accumulated resistance value is the total resistance of the nodes and links that make up the shortest path.

tpm_traceshortscan

Gets the ID of a link or node in the shortest path.

(tpm_traceshortscan trace_id flag)

Returns a topology ID (real) or nil.

trace_id Tracing model ID (real), returned by tpm_tracealloc flag Path element code (integer): 0 Current element 1 First element 2 Last element 3 Next element 4 Previous element

First use <u>tpm_traceshort</u> to calculate a shortest path.

tpm_traceshortval <u>Network Tracing Functions</u>

Calculates the resistance of the shortest path.

(tpm_traceshortval trace_id)

Returns a resistance (real) or nil.

trace_id Tracing model ID (real) returned by tpm_tracealloc

Before calling this function, use <u>tpm_traceshort</u> to calculate the shortest path.

tpm_varalloc Variables Functions

Allocates a set of configuration variables.

(tpm_varalloc)

Returns a configuration variables ID (real) or nil.

Configuration variables are composed of cleanup variables, cleanup action variables, and topology variables. The variables are initialized to their default values. For a list of these variables and their default values, see <u>Configuration</u> <u>Variables</u>.

You can allocate more than one set of configuration variables.

tpm_varfree <u>Variables Functions</u>

Frees a set of configuration variables.

(tpm_varfree var_id)

Returns T or nil.

var_id Configuration variables ID (real)

Configuration variables are composed of cleanup variables, cleanup action variables, and topology variables. The variables are initialized to their default values. For a list of these variables and their default values, see <u>Configuration</u> <u>Variables</u>.

tpm_varget <u>Variables Functions</u>

Gets the value of a configuration variable.

(tpm_varget var_id var_name)

Returns a variable value or nil.

var_id	Configuration variables	ID	(real)

var_name Variable name (string)

Configuration variables are composed of cleanup variables, cleanup action variables and topology variables. The variables are initialized to their default values. For a list of these variables and their default values, see <u>Configuration</u> <u>Variables</u>.

The following code opens a topology, stores the topology's configuration values in a new set of variables, and gets the value of **NODE_LAYER**.

```
(setq tpm_id (tpm_acopen "SampleTopo"))
(setq var_id (tpm_varalloc))
(tpm_infobuildvar tpm_id var_id)
(tpm_acclose tpm_id)
(setq nlayer (tpm_varget var_id "NODE_LAYER"))
```

tpm_varlist <u>Variables Functions</u>

Gets all the values in aa set of configuration variables.

(tpm_varlist var_id)

Returns a list of name-value pairs or nil.

var_id Configuration variables ID (real)

Each a-list has this format:

(variable name . value)

Configuration variables are composed of cleanup variables, cleanup action variables, and topology variables. The variables are initialized to their default values. For a list of these variables and their default values, see <u>Configuration</u> <u>Variables</u>.

tpm_varset Variables Functions

Sets the value of a configuration variable.

(tpm_varset var_id var_name var_value)

Returns T or nil.

var_id	Configuration variables ID (real)
var_name	Variable name (<mark>cha</mark> r)
var_value	Variable value.

Configuration variables are composed of cleanup variables, cleanup action variables, and topology variables. The variables are initialized to their default values. For a list of these variables and their default values, see <u>Configuration</u> <u>Variables</u>.

For an example that shows how to use tpm_varset and tpm_varget, see tpm_varget.

Access Functions

The functions for accessing topologies begin with tpm_ac.

tpm_acclose

Closes a topology.

tpm_acexist

Checks if a topology exists.

<u>tpm_acload</u>

Loads a topology into memory.

tpm_acopen

Opens a topology.

tpm_acqty

Counts topologies.

tpm_acunload

Unloads a topology from memory.

tpm_acupgradeopen

Changes access from read only to write.

Analyzing Functions

The functions for analyzing topologies begin with tpm_ana.

tpm_anabuffer

Creates a buffer space around a topology.

tpm_anadissolve

Merges topology elements with the same value in the specified field.

tpm_anaoverlay

Overlays two topologies.

Building and Erasing Functions

The functions for topology maintenance begin with tpm_mnt.

tpm_mntbuild

Builds a topology.

tpm_mnterase

Erases a topology from the project drawing.

tpm_mntrebuild

Rebuilds a topology.

tpm_mntrename

Renames a topology.

Cleanup Functions

The functions for cleaning topology objects before building the topology begin with tpm_clean.

tpm_cleanactionlistdel

Deletes a cleanup action from the action list.

tpm_cleanactionlistgetat

Gets the cleanup action at a given list position.

tpm_cleanactionlistins

Inserts a cleanup action in the action list.

tpm_cleanactionlistqty

Gets the number of cleanup actions in the action list.

tpm_cleanalloc

Allocates the cleanup model.

tpm_cleananchorss

Gets a cleanup model's anchored entities.

tpm_cleancancel

Cancels the cleanup process without updating the drawing.

tpm_cleancomplete

Tests for a next group to be cleaned.

tpm_cleancreatedss

Gets created entities following a cleanup process.

tpm_cleanend

Completes the cleanup process and updates the drawing.

tpm_cleanerrorcur

Sets the next error to clean in the current group.

tpm_cleanerrordraw

Creates a temporary marker for the current error.

<u>tpm_cleanerrorfix</u>

Fixes the current error.

tpm_cleanerrorget

Gets the coordinates of the current error.

tpm_cleanerrormark

Creates a persistent marker for the current error.

tpm_cleanerrorset

Sets the coordinates of an error fix point.

tpm_cleanfree

Frees the cleanup model.

tpm_cleangroupdraw

Creates temporary markers for all errors of the current group.

tpm_cleangroupfix

Fixes all errors of the current group.

tpm_cleangroupmark

Creates persistent markers for all errors of the current group..

tpm_cleangroupnext

Goes to the next error group.

tpm_cleangroupqty

Counts the errors in the current group.

tpm_cleangroupsubtype

Determines the subtype of the current group.

tpm_cleangrouptype

Determines the type of the current group.

tpm_cleanincludess

Gets a cleanup model's target entities.

tpm_cleaninit

Initializes the cleanup model.

tpm_cleaninitanchorset

Specifies anchored entities for the cleanup model.

tpm_cleanmodifiedss

Gets changed entities following a cleanup process.

tpm_cleanprofileload

Loads a drawing cleanup profile.

tpm_cleanprofilesave

Saves a drawing cleanup profile.

tpm_cleanstart

Starts the cleanup process.

tpm_cleanunchangedss

Gets unchanged entities following a cleanup process.

Editing Functions

The functions for editing topology elements begin with tpm_edit.

tpm_editaddelem

Adds an element to a topology.

tpm_editdelelem

Deletes an element from a topology.

tpm_editmodelem

Modifies a topology element.

tpm_editupdelem

Updates a topology element.

Element Information Functions

The functions for managing topology elements begin with tpm_elem.

tpm_elemadj

Compiles a list of adjacent elements.

tpm_elemfind

Finds an element.

tpm_elemget

Lists information about an element.

tpm_elemid

Gets the ID of an element.

tpm_elemqty

Counts topology elements.

tpm_elemss

Creates a selection set of elements of a given type.

The functions for iterating through topologies begin with tpm_iter.

Many have counterparts in the topology information functions (tpm_info[xx]). The iterating functions can query any topology, loaded or unloaded, open or closed. The information functions query only topologies that are open.

tpm_iterdesc

Gets a topology description.

tpm_itername

Gets a topology name.

tpm_iternext

Moves the iterator to the next topology.

tpm_iterstart

Creates a topology iterator.

tpm_iterstop

Frees a topology iterator.

<u>tpm_itertype</u>

Gets a topology type.

tpm_iterversion

Gets the version of a topology.

Network Tracing Functions

The functions for network tracing begin with tpm_trace.

tpm_tracealloc

Allocates a tracing model.

tpm_tracebestroute

Calculates the best round-trip route.

tpm_tracebestroutescan

Gets the element ID of a link or node in the best route.

tpm_tracebestrouteval

Calculates the resistance of the best route.

tpm_traceelemedit

Modifies a tracing element.

tpm_traceelemget

Gets information about a tracing element.

tpm_traceelemid

Gets the topology ID of a tracing element.

tpm_traceflood

Traces a flood path from a specified point.

tpm_tracefree

Frees a tracing model.

tpm_traceqty

Counts the selected elements after a trace.

tpm_tracesetmaxres

Sets maximum resistance for the shortest path algorithm.

tpm_tracesetminres

Sets minimum resistance for the shortest path algorithm.

tpm_traceshort

Calculates the shortest path between two nodes.

tpm_traceshortscan

Gets the topology ID of a link or node in the shortest path.

tpm_traceshortval

Calculates the resistance of the shortest path.

Topology Information Functions

The functions for getting information about topologies begin with tpm_info.

Many have counterparts in the topology iterating functions (tpm_iter[xx]). The information functions can query only topologies that are open. The iterating functions can query any topology, loaded or unloaded, open or closed.

tpm_infobuildvar

Stores the configuration values of a topology.

tpm_infocomplete

Tests if a topology is complete.

tpm_infocorrect

Tests if a topology is correct.

tpm_infocurrent

Checks the source from which a topology was loaded.

tpm_infodesc

Gets a topology description.

tpm_infomodified

Checks if topology elements have been modified using drawing tools.

<u>tpm_infoname</u>

Gets a topology name.

tpm_infostatus

Checks whether a topology is open for Read or Write.

tpm_infotype

Gets a topology type.

tpm_infoversion

Gets a topology version.

Topology Query Functions

The functions for querying topologies begin with tpm_qry.

tpm_qrygetresdesc

Gets the description of the query result topology.

tpm_qrygetrestopo

Gets the name of the query result topology.

tpm_qrygettoponame

Gets the name of the query source topology.

tpm_qrysetrestopo

Defines or or undefines a query result topology.

tpm_qrysettoponame

Defines or undefines a topology query.

Topology Variables Functions

The functions for managing configuration variables begin with tpm_var.

tpm_varalloc

Allocates a set of configuration variables.

tpm_varfree

Frees a set of configuration variables.

tpm_varget

Gets the value of a configuration variable.

tpm_varlist

Gets all the values in a set of configuration variables.

tpm_varset

Sets the value of a configuration variable.

Marker Shapes

Drawing cleanup markers are available in the following shapes.

Shape Code	Shape
1	Octagon
2	Triangle
3	Rhombus
4	Square

A topology is given the following set of object data fields when it is built. These fields contain information about the elements of the topology.

Warning Although it is possible to use Data Extension Object Data functions $(ade_od[xx])$ to modify these fields, it is not a good idea to do so. The topology functions assume these fields are managed by them alone.

Element	Field Name	Default
Node	ID (int)	
Node	RESISTANCE (real)	0.0
Link	ID (int)	
Link	START_NODE (int)	
Link	END_NODE (int)	
Link	DIRECTION (int)	0 (Bidirectional)
Link	DIRECT_RESISTANCE (real)	Length
Link	REVERSE_RESISTANCE (real)	Length
Link	LEFT_POLYGON (int)	
Link	RIGHT_POLYGON (int)	
Centroid	ID (int)	

Centroid	AREA (real)	
Centroid	PERIMETER (real)	
Centroid	LINKS_QTY (int)	

The default resistance value Length is interpreted as follows, depending on the AutoCAD entity type:

Entity type	Length
Line	Line length
Pline	Total pline length
Arc	Calculated arc length
Circle	Calculated circumference

The default link direction value is 0 (bi-directional). The other possible direction values are 1 (forward), or -1 (reverse). The forward direction for lines is from start point to end point. For circles and arcs, the forward direction is counter-clockwise.

Location Query

This example defines and executes a location query.

(mapcar 'ade_dwgdeactivate (ade_dslist)) (setq ade_tmpprefval (ade_prefgetval "ActivateDwgsOnAttach")) (ade_prefsetval "ActivateDwgsOnAttach" T) (ade_dsattach "E:\\ADEDWG\\ARCS2.DWG") (ade_dsattach "E:\\ADEDWG\\DONUTS.DWG") (ade_dsattach "E:\\ADEDWG\\DONUTS.DWG") (ade_dsattach "E:\\ADEDWG\\POLYLINE.DWG") (ade_dsattach "E:\\ADEDWG\\POLYLINE.DWG") (ade_prefsetval "ActivateDwgsOnAttach" ade_tmpprefval) (ade_qryclear) (ade_qrysettype 1) (ade_qrydefine '("" "" "Location" (1)"")) (ade_qryexecute)
Property Queries

The following examples define and execute property queries:

```
(mapcar 'ade_dwgdeactivate (ade_dslist))
(setq ade_tmpprefval (ade_prefgetval "ActivateDwgsOnAttach"))
(ade_prefsetval "ActivateDwgsOnAttach" T)
(ade_dsattach "E:\\ADEDWG\\ARCS2.DWG")
(ade_dsattach "E:\\ADEDWG\\DONUTS.DWG")
(ade_dsattach "E:\\ADEDWG\\LINES2.DWG")
(ade_dsattach "E:\\ADEDWG\\MAKE.DWG")
(ade_dsattach "E:\\ADEDWG\\POLYGONS.DWG")
(ade_dsattach "E:\\ADEDWG\\POLYLINE.DWG")
(ade_prefsetval "ActivateDwgsOnAttach" ade_tmpprefval)
(ade_qryclear)
(ade_qrysettype 2)
(ade_qrydefine '("" "" "Property" ("color" "=" "5")""))
(ade_qryexecute)
```

(mapcar 'ade_dwgdeactivate (ade_dslist)) (setq ade_tmpprefval (ade_prefgetval "ActivateDwgsOnAttach")) (ade_prefsetval "ActivateDwgsOnAttach" T) (ade_dsattach "AUTO:\\AQP5.DWG") (ade_dsattach "AUTO:\\AQP5A.DWG") (ade_dsattach "AUTO:\\MAKE.DWG") (ade_prefsetval "ActivateDwgsOnAttach" ade_tmpprefval) (ade_qryclear)

```
(ade_qrysettype "draw")
(ade_qrydefine
    '("" "" "Property" ("objtype" "=" "SPLINE")""))
(ade_qrydefine
    '("OR" "" "" "Property" ("objtype" "=" "ELLIPSE")""))
(ade_qryexecute)
```

Location Query with Property Alteration

This example defines and executes a location query with property alteration.

(mapcar 'ade_dwgdeactivate (ade_dslist)) (setq ade_tmpprefval (ade_prefgetval "ActivateDwgsOnAttach")) (ade_prefsetval "ActivateDwgsOnAttach" T) (ade_dsattach "E:\\ADEDWG\\AQFA3.DWG") (ade_prefsetval "ActivateDwgsOnAttach" ade_tmpprefval) (ade_qryclear) (ade_qrysettype "draw") (ade_qrydefine '("" "" "Location" ("all")"")) (ade_altpclear) (ade_altpdefine "Height" "2") (ade_altpdefine "Style" "Y") (ade_altpdefine "TextValue" "test") (ade_altpdefine "Layer" "newlyr") (ade_altpdefine "Rotation" "0.5") (ade_altpdefine "Color" "blue") (ade_qryexecute)

Location Query with Property Alteration: Before and After

This example shows two versions of a property alteration query. The second example is an edited version of the first. Note how changes to the arguments in the ade_altpdefine calls cause different property alteration effects.

Original Version

(mapcar 'ade dwgdeactivate (ade dslist)) (setq ade tmpprefval (ade prefgetval "ActivateDwgsOnAttach")) (ade_prefsetval "ActivateDwgsOnAttach" T) (ade_dsattach "E:\\ADEDWG\\ARCS2.DWG") (ade_dsattach "E:\\ADEDWG\\CIRCLES2.DWG") (ade_dsattach "E:\\ADEDWG\\DONUTS.DWG") (ade_dsattach "E:\\ADEDWG\\ESPMR.DWG") (ade_dsattach "E:\\ADEDWG\\LINES2.DWG") (ade_dsattach "E:\\ADEDWG\\POLYGONS.DWG") (ade_dsattach "E:\\ADEDWG\\POLYLINE.DWG") (ade_prefsetval "ActivateDwgsOnAttach" ade_tmpprefval) (ade_qryclear) (ade_grysettype "draw") (ade_qrydefine '("" "" "" "Location" ("all")"")) ; ade_altpdefine calls (ade_altpdefine "Color" "5") (ade_altpdefine "Layer" "NEWLYR") (ade_altpdefine "Elevation" "-2") (ade_altpdefine "Thickness" "1.5") (ade_altpdefine "Linetype" "CENTER")

Edited Version

(mapcar 'ade_dwgdeactivate (ade_dslist)) (setq ade_tmpprefval (ade_prefgetval "ActivateDwgsOnAttach")) (ade_prefsetval "ActivateDwgsOnAttach" T) (ade_dsattach "E:\\ADEDWG\\ARCS2.DWG") (ade_dsattach "E:\\ADEDWG\\CIRCLES2.DWG") (ade dsattach "E:\\ADEDWG\\DONUTS.DWG") (ade dsattach "E:\\ADEDWG\\ESPMR.DWG") (ade dsattach "E:\\ADEDWG\\LINES2.DWG") (ade_dsattach "E:\\ADEDWG\\POLYGONS.DWG") (ade dsattach "E:\\ADEDWG\\POLYLINE.DWG") (ade_prefsetval "ActivateDwgsOnAttach" ade_tmpprefval) (ade_qryclear) (ade_qrysettype "draw") (ade grydefine '("" "" "Location" ("all")"")) ; changes in ade_altpdefine calls (ade_altpdefine "Color" "BLUE") (ade_altpdefine "Layer" "XYZ") (ade_altpdefine "Elevation" "7") (ade_altpdefine "Thickness" "1.5") (ade_altpdefine "Linetype" "CENTER") (ade_qryexecute)

The following query definition combines a location buffer fence condition and a property condition with a range table definition, and property alteration is performed on the query result.

(mapcar 'ade_dwgdeactivate (ade_dslist)) (setq ade_tmpprefval (ade_prefgetval "ActivateDwgsOnAttach")) (ade_prefsetval "ActivateDwgsOnAttach" T) (setq dwg_id(ade_dsattach "E:\\ADEDWG\\POPS.DWG")) (ade_prefsetval "ActivateDwgsOnAttach" ade_tmpprefval) (ade_qryclear) (ade_qrysettype "draw") ; location buffer fence (ade grydefine '("" "" "Location" ("bufferfence" "inside" 3.000000 (4.426217 7.991379 0.000000) $(2.385054\ 5.530788\ 0.00000)$ (4.648083 3.912562 0.00000) $(5.912716\ 5.708128\ 0.000000)$ (7.754200 3.823892 0.00000) (8.020439 3.646552 0.000000))"")) (ade grydefine '("OR" "" "" "Property" ("objtype" "=" "ARC,CIRCLE")"")) ; range table (ade_rtdefrange "Rtable" "Range table depending upon object type" '(("=" "CIRCLE" "LAYER-CIRCLE") ("=" "ARC" "LAYER-ARC")

("OTHERWISE" "" "LAYER-OTHERS"))) (ade_altpclear) (ade_altpdefine "Layer" "(Range .TYPE Rtable)") (ade_altpdefine "Color" "YELLOW") (ade_qryexecute)

Topology IDs

Topology function documentation uses a number of different kinds of IDs, as listed in the following table.

Although these IDs are all the same type, ade_id, we give them different names in the documentation according to their purpose.

Туре	Name	Identifies	Used or Returned By
ade_id	clean_id	A cleanup model	tpm_clean[xx]
ade_id	elem_id	A topology element	tpm_editdelelem tpm_editmodelem tpm_elemadj tpm_elemget tpm_traceelemedit tpm_traceelemget
ade_id	iter_id	A topology iterator	tpm_iter[xx]
ade_id	overlay_id	An overlay topology	tpm_anaoverlay
ade_id	source_id	A source topology	tpm_anabuffer tpm_anadissolve tpm_anaoverlay
ade_id	tpm_id	A topology	<u>tpm_acclose</u> <u>tpm_acopen</u>

			tpm_acupgradeopen tpm_editaddelem tpm_editdeleem tpm_editmodelem tpm_elemadj tpm_elemfind tpm_elemget tpm_elemid tpm_elemqty tpm_elemss tpm_info[xx] tpm_info[xx] tpm_mntbuild tpm_mntrebuild tpm_tracealloc tpm_traceelemget map_topoAudit map_topoStat
ade_id	trace_id	A tracing model	tpm_trace[xx]
ade_id	var_id clean_var_id act_var_id	Configuration variables Topology variables Cleanup variables Cleanup action variables	tpm_anabuffer tpm_anadissolve tpm_anaoverlay tpm_clean[xx] tpm_infobuildvar tpm_mntbuild tpm_var[xx]

Location expressions are used as querycond arguments in ade_querydefine calls that define Location conditions.

There are a number of formats to choose from when writing Location expressions. They are listed below. The format to use depends on the Location type, which is identified in each of the following formats by the first argument. The other arguments are described below.

Location-All

("all")

Location-Bufferfence

("bufferfence" searchtype offset pt1 pt2 ... ptN)

Location-Circle

("circle" searchtype centerpt radius)

Location-Fence

("fence" pt1 pt2 ... ptN)

Location-Point

("point" pt)

Location-Polygon

("polygon" searchtype pt1 pt2 ... ptN)

Location-Polyline-Bufferfence

("polyline" "bufferfence" searchtype offset ename)

Location-Polyline-Fence

("polyline" "fence" ename)

Location-Polyline-Polygon

("polyline" "polygon" searchtype ename)

Location-Window

("window" searchtype pt1 pt2)

Location Expression Parameters

searchtype	Search type keyword (string): "inside" or "crossing".
offset	Buffer offset distance (real).

ename	AutoCAD entity name, or a set of points, or "?". If "?", when the query executes, it prompts the user to click a set of points.
pt ptN centerpt	A 2D or 3D point (a list of reals). If a 3D point, the Z coordinate is ignored.
radius	Radius (<mark>real</mark>).

Location Examples

The following examples define Location conditions. The first is a Location-All condition:

The second is a Location-Window condition:

```
(ade_qrydefine
 '("AND" "" "Location"
 ("window" "crossing" (1.0 2.0) (3.0 4.0)) "")
)
```

And the third is a Location-Polyline condition.

```
(ade_qrydefine
  (list "AND" "(" "" "Location"
   (list "polyline" "bufferfence" "inside" 20.0
   (entlast)) "")
)
```

The following three examples of Location-Polyline conditions specify the ename argument in different ways. The first supplies an entity name:

```
(ade_qrydefine
  (list "" "" "Location"
   (list "polyline" "polygon" "crossing"
   (entlast)) "")
)
```

The second supplies a set of points:

```
(ade_qrydefine
  '("" "" "Location"
    ("polyline" "polygon" "crossing" (
               ; 1 = Polyline open, 0 = Polyline closed
        1
        (0 0 1) ; Direction of normal vector:
               ; (0 0 1) identifies the Z axis,
               ; i.e., this polygon is parallel to
               ; the XY plane
                               ; Bulge factor of a vertex
        0.000000
        (4.426217 7.991379 0.000000) ; Coordinates of a vertex
        0.000000
        (2.385054 5.530788 0.00000)
        0.000000
        (4.648083 3.912562 0.000000)
        0.000000
        (5.912716 5.708128 0.000000)
        0.000000
```

```
(7.754200 3.823892 0.000000)
0.000000
(8.020439 3.646552 0.000000)))"")
```

And the third prompts the user to click a set of points:

```
(ade_qrydefine
    '("" "" "Location"
    ("polyline" "polygon" "crossing" "?") "")
)
```

SQL Expressions

SQL expressions are used as querycond arguments in ade_querydefine calls that define SQL conditions.

They have the following format:

(linkpathname sqlcondition)

SQL Expression Parameters

linkpathname	Link template (string).
sqlcondition	SQL condition (string).

The **sqlcondition** argument should contain only the **WHERE** clause of an SQL statement (for example, "last_name = 'Smith'''). To select the entire table, let the **sqlcondition** argument be the empty string ("'').

```
(ade_qrydefine
'("AND" "" "" "Sql"
("EMPLN3" "") "" )
)
```

The data types for object data fields are as follows. The type names in the first column are the coltype arguments for use in field definitions.

Туре	Description
integer	From –2,147,483,648 through 2,147,483,647
character	Any alphanumeric string, including the empty string, ""
point	List of three real numbers, separated by commas, that represent a point with an X, Y, and Z value, enclosed in quotes (a string value)
real	From -1.7E308 through +1.7E308

Adding Fields to a Table

The following sample adds one field of each data type to an existing table.

; Define new fields (setq newfields '("columns" ; Define a character field (("colname" . "NEWFIELD1") ("coldesc" . "New Field 1 Description") ("coltype" . "character") ("defaultval" . "Default Value")) ; Define an integer field (("colname" . "NEWFIELD2") ("coldesc" . "New Field 2 Description") ("coltype" . "integer") ("defaultval" . 1)) ; Define a point field (("colname" . "NEWFIELD3") ("coldesc" . "New Field 3 Description") ("coltype". "point") ("defaultval" . "4.426217, 7.991379, 1.726213")) ; Define a real field (("colname" . "NEWFIELD4") ("coldesc" . "New Field 4 Description") ("coltype". "real") ("defaultval" . 1.2345)))) ; Add new fields to existing table (ade_odaddfield "OLDTABLE" newfields);

You can use the following comparison operators in range table expressions:

< > <= >= /= otherwise

When the first element in the range expression is "otherwise", the second element must be the empty string (""). For example: ("otherwise" "" "red"). Although you do not specify a comparison value when you use the otherwise operator, a second list element is still required.

Note The not-equal operator, represented here by "/=", is represented by "<>" in other Data Extension functions.

Functions that Return Selection Sets

The following functions return selection sets.

tpm_elemss ade_editlocked ade_editnew ade_editlockederased map_dwgbreakobj map_dwgtrimobj map_topoclose map_topocomplete

Plot Set Attributes

This table shows plot set attribute names, descriptions, examples, and default values.

Attribute	Description	Default
name	Query name (string), e.g., "plotset2"	
desc	Description (string), e.g., "Complete Plot Set"	
pnam	Plotter name (string), e.g., "HP LaserJet III on LPT1"	
pscr	Plot script (string), e.g., "e, n, y"	
block	Name of plot layout, a block name (<mark>string</mark>), e.g., "TBLOCK"	
atts	List of attributes for the block (<mark>string</mark> list), e.g., "TITLE"	nil
vlayer	Main viewport layer (string), e.g., "VPORT2"	
vdispl	Layers to disable for the main viewport (string list), e.g., "0 "	nil
kflg	Enables the reference viewport: T or nil T Enabled nil Disabled	Т

	Note that there are two dependent attributes, "klayer" and "kdispl"	
klayer	Key viewport layer (string), e.g., "VPORT1", the layer of the reference viewport, which must be a different layer from the layer used for the main viewport (attribute "vlayer"); if the function map_pltblkvps returns only one viewport layer, it should be used for the main viewport	
kdispl	Key display layers (string list), e.g., " BOUNDARY ", a list of layers to be displayed in the referenced viewport. This should not be nil if " kflg " is enabled. Valid layers are all the layers returned by ade_dsproplist using the option " layer "	nil
kscl	Key viewport scale factor (real)	0.25
dwgs	List of source drawings (string list), e.g., "DRAWING1.DWG" "DRAWING2.DWG"	nil
dact	Force drawing active at plot time: T or nil	Т
qcat	List of query catalogs (string list), paired with qnam list entries, e.g., "Queries"	nil
qnam	List of query names (string list), paired with qcat list entries, e.g., "Query_One"	nil
bdwg	Boundary drawing (string), e.g., "BOUNDARY.DWG"	
bnds	Boundaries (string list), the boundary or boundaries to use by providing the value for the field assigned in "bnamf", e.g., "Boundary #3"	nil
blyr	Boundary layer (string), the name of the layer the boundary is placed on. It can be any layer of the	

	boundary drawing. For example, "BOUNDARY"	
bnamt	Boundary name table (string), the name of the object data table attached to the boundary object, e.g., "BOUNDARIES"	
bnamf	Boundary name field (string), the name of the field to use (from the table assigned in " bnamt ") for the " bnds " attribute, e.g., " name "	
bodfs	Boundary fields (string list), the list of field names to be mapped to attributes in the title block, e.g., " name " (see details and example following this table); the title block is assigned using the attributes from the block assigned in " block "	nil
sflg	Plot to specified scale: T or nil	nil
scale	Plot scale (string), e.g., "1:2000"	1:1
clip	Clip objects against boundary (flag): T or <mark>nil</mark>	nil
pbnd	Plot boundary (flag): T or nil T Plot boundary polylines are plotted nil Plot boundary polylines are erased before plotting	nil
bbuf	Buffer boundary: T or nil	nil
btyp	Buffer type: "true" or "rect" "true" Offsets the minimum bounding rectangle for a map boundary polyline "rect" Specifies an offset of an existing plot boundary line	true
bdist	Buffer distance (real)	0.10

Mapping of object data fields to block attributes is managed by the two plot attributes "**bodfs**" and "**atts**". The mapping is best explained by example. Suppose the following conditions:

- The boundary object data table, defined by the "bnamf" attribute, has the following fields: Field1, Field2, Field3, and Field4.
- The layout block, defined by the "block" attribute, has the following attributes: Attr1, Attr2, Attr3, Attr4, and Attr5.
- You want the value of Field1 to appear in place of Attr2, the value of Field2 in place of Attr3, and value of Field4 in place of Attr1.

To define the desired mapping, set the "bodfs" attribute to ("field1" "field2" "field4") and the "atts" attribute to ("attr2" "attr3" "attr1"). When plotting is done, values of the fields from the object data table attached to the boundary's closed polyline will be assigned to the specified block attributes. The number of elements in each list for "bodfs" and "atts" must be the same.