Getting Started Introduction



The ActiveX Object Model for AutoCAD Land Desktop is an ActiveX and VBA interface that enables developers to create dynamic and powerful applications. This guide describes the concept of exposing objects through an ActiveX interface and programming those objects using various languages including the Visual Basic for Applications programming environment.

The section "Getting Started" covers the example code found in the help reference and the sample applications that are installed with the ActiveX Object Model. Refer to the section "Understanding the ActiveX Object Model" for specific information on features of AutoCAD Land Desktop that can be accessed through the ActiveX Object Model. In the section "Programming Guidelines" there are tips and recommendations for developing applications using various programming languages and development environments.

If you are new to the Visual Basic for Applications programming environment or AutoCAD ActiveX technology, please refer to the <u>AutoCAD ActiveX and VBA Developer's Guide</u> before using this guide.

Topics in this Section:

- <u>Referencing the Automation Type Libraries</u>
- Running the Example Code in this Guide
- <u>Reviewing the Sample Applications</u>

Referencing the Automation Type Libraries

Whenever a new AutoCAD Land Desktop project is created, the supporting type libraries must be referenced. AutoCAD Land Desktop requires reference to three type libraries before any coding begins. The three type libraries are:

- AEC Base 4.0 Application Library (AecXUiBase40.tlb)
- AEC Base 4.0 Object Library (AecXBase40.tlb)
- Autodesk Land 4.0 Type Library

These libraries provide access to all the AutoCAD Land Desktop Objects, Properties, Methods, and Events. The VBAIDE References dialog allows you to set the type libraries for use.

References - ACADProject	<u>></u>
<u>A</u> vailable References:	ОК
Visual Basic For Applications AutoCAD 2004 Type Library	Cancel
 ✓ OLE Automation ✓ AEC Base 4.0 Application Library. ✓ AEC Base 4.0 Object Library 	Browse
Active Soft of the Soft of	<u>H</u> elp
Autodesk Land 4.0 Type Library Location: C:\Program Files\Land Desktop 2004\landauto.t Language: Standard	lb

To open the References dialog box

- 1. From the Tools menu, choose References.
- 2. Scroll through the list and check the required AutoCAD Land Desktop type libraries.
- 3. Click OK.

Running the Example Code



All of the example code in the Reference can be copied from the help files, pasted directly into the AutoCAD VBA environment, and then executed with the following requirements:

- The current active drawing in AutoCAD must be a drawing created with AutoCAD Land Desktop open to model space.
- The active project must contain objects or data that the example is trying to access for use.
- In order to provide clear and simple examples, all the examples provided in the reference assume that the Base Point and North Rotation is the same as the AutoCAD World coordinate system using the default X and Y settings. If you use any of the examples in your own code make sure you test for Base Point and North Rotation and provide appropriate conversion when necessary.
- Some examples contain user input methods such as Utility.GetPoint, which can be called from a macro, but cannot be called from a modal dialog.

To run the examples

- 1. Copy the example from the help file into an empty VBA code module.
- 2. Add a reference to the AEC Base Application, AEC Base Object, and AutoCAD Land Type Libraries.
- 3. Verify that AutoCAD has an AutoCAD Land Desktop drawing open to model space.
- 4. Verify that the active project has data for the example to work with.
- 5. Open the Macros dialog box by entering the command VBARUN.
- 6. Choose the macro and press Run.

NOTE: The examples are provided to simply show how a specific Object, Property, Method, or Event can be used. No provisions for error trapping are provided. Code optimization, format or logic have been kept simple, and to a minimum, for the sake of clarity.

Reviewing the Sample Applications



The AutoCAD Land Desktop ActiveX and VBA ActiveX Object Model comes with several samples that demonstrate how to create your own utilities and extensions to AutoCAD Land Desktop. We have provided samples for each of the most common development environments:

Visual Basic for Applications

- ProjectFind
- PointGroupFncs
- PointGroupReport
- Draw3dSurface
- <u>AlignmentUtilities</u>
- DrapeLine
- PointGroupTable
- <u>ParcelArea</u>
- DrawCrossSection

Visual Basic

• LockWatcher

Visual Lisp

<u>CogoPoints</u>

ObjectARX

- <u>AeccNew</u>
- PointGroup
- PointExport

VisualC++

• EventLogger,

This listing provides the name, description, and location of the main source code file for each sample application. Many sample applications have support files that can be found in the same directory as the main source code file.

Sample Applications:

Sample\Land\Vba\ProjectFind.dvb

Description: VBA dialog for finding AutoCAD Land Desktop projects.

Macros:

ProjectFind This macro launches a modal dialog that you can use search for a project based on a path and filters using wildcards, and optionally load a drawing from that project into AutoCAD Land Desktop.

Sample\Land\Vba\PointGroupFncs.dvb

Description: VBA functions to modify Point Groups.

Macros:

AddPointsToGroup	This macro adds a point list to the selected Point Group. The point list can be retrieved from an AutoCAD drawing or typed in.
CreateGroupByDesc	This macro displays a list of all unique point descriptions. You can then create a new Point Group based on that unique description. Then the macro adds all points with that description to that new Point Group.
DisplayGroupsByPoint	Given a Point number, the macro displays all point groups that contain that Point number. The Point number can be typed or selected from an

AutoCAD drawing.

Sample\Land\Vba\PointGroupReport.dot

Description: VBA macro that creates a custom Point Group report.

Macros:

This macro will accept customer data and print out: PointGroupReport customer data, project statistics, and multiple Point Groups.

NOTE: To use the word template, you must have the AutoCAD Land Desktop running and a drawing open with a point database created in its project. Locate the PointGroupReport.dot with Explorer and double click on the file name. This will open Word and begin a new document using PointGroupReport.dot as the template. From the Tools menu open the Macros dialog and select the macro named PointGroupReport. Select the Run button to display the Point Group Display Dialog. This dialog creates a client data section and a point table listing for each selected point group.

To edit this macro use the Visual Basic Editor from the Macro selection of the Tools menu in Microsoft Word.

Sample\Land\Vba\Draw3dSurface.dvb

Description: VBA dialog for drawing a Surface as 3d Faces.

Macros:

This macro launches a modal dialog that you can use to select a Surface and then draw all visible Surface Faces DrawSurface as AutoCAD 3dFaces. All drawing is done on the current drawing surface.

Sample\Land\Vba\AlignmentUtilities.dvb

Description: VBA dialog for making a copy of an alignment in the alignment database.

Macros:

CopyAlignment and then make a copy of that alignment. The alignment is not drawn in the current AutoCAD drawing.

Sample\Land\Vba\DrapeLine.dvb

Description: VBA dialog for projecting an alignment to a surface, and drawing the result as a set of lines.

Macros:

This macro launches a modal dialog that lets the userDrapeLineselect an alignment and a surface, and drape a projected
series of line entities onto the surface on the current layer.

Sample\Land\Vba\PointGroupTable.dvb

Description: VBA dialog for adding a point table to the drawing.

Macros:

PointGroupTable This macro launches a modal dialog that allows the user to select a point group, and inserts a table on the current layer that identifies point number, eastings, northings and elevations.

Sample\Land\Vba\ParcelArea.dvb

Description: VBA dialog for displaying the total area of a set of parcels.

Macros:

ParcelArea This macro launches a modal dialog that allows the user to select one or more parcels, and displays the total area in the proper units.

Sample\Land\Vba\DrawCrossSection.dvb

Description: VBA utility for drawing a cross section of the first station of the current alignment.

Macros:

DrawCrossSection DrawCrossSection of the CrossSection block, and then drafts each cross section surfaces as a lightweight polyline. The code demonstrates how to process the right and left sides of the surfaces properly.

Sample\Land\ActiveX\LockWatcher

Description: A VB application for monitoring multi-user locks for a project.

NOTE: The VB sample LockWatcher was developed using Visual Basic version 5.0. It is also compatible with VB4-32 and VB6.

Sample\Land\VLisp\CogoPoints.lsp

Description: Various Visual Lisp functions to access the Project and Cogo interfaces. This lisp file is useful for learning the syntax of ActiveX and VLisp.

Functions:

loadAecProj	Loads the AutoCAD, Aecc.Application, and the Active Project interfaces. This routine must be run before all other functions.
cleanup	Releases all global interfaces.
ProjectStats	Displays the active project's properties.
PointGroups	Lists the point group names using the Item() method.
PointGroups2	Lists the point group names using Enumeration.
addCgPoint	Adds a new Cogo Point.

Sample\Land\Arx\AeccNew

Description: This ARX provides the command line equivalents of the

New and Open dialogs. It demonstrates the use of the ACRX_CMD_SESSION and ACRX_CMD_NOINTERNALLOCK flags for registered commands that use the Document.NewProjectBased and Document.OpenProjectBased ActiveX methods.

NOTE: You must modify project settings for the included directories and the library to match your system. The locations of the following type libraries must also be modified in stdafx.h: acad.tlb, aecxbase.tlb, aecxuibase.tlb and landauto.tlb.

Functions:

aeccnew	This prompts you for arguments and creates a new project/drawing.
aeccopen	This prompts you for arguments and opens an existing project/drawing.

Sample\Land Arx\PointGroup

Description: This application displays the currently defined point groups and displays the points for an existing point group.

NOTE: You must modify project settings for the included directories and the library paths to match your system. The locations of the following type libraries must also be modified in stdafx.h: acad.tlb, aecxbase.tlb, aecxuibase.tlb and landauto.tlb.

Functions:

c:pointgroup Lists routines for PointGroup.arx.

c:listpointgroups Lists all point groups for the current project.

listgrouppoints Gives a point group name, will list all of the points.

Sample\Land\ Arx\PointExport

Description: Application will export all Cogo points into an ASCII file.

NOTE: You must modify project settings for the included directories and the library paths to match your system. The locations of the following type libraries must also be modified in stdafx.h: acad.tlb, aecxbase.tlb, aecxuibase.tlb and landauto.tlb.

Functions:

pointexport Exports all Cogo points to an ASCII file.

Sample\Land\ActiveX\EventLogger

Description: A C++ application for monitoring alignment events.

This sample application displays a dialog with a list box that records all alignment events (when an alignment or station equation is written to the database). The sample demonstrates how to implement an event sink in a COM object using MFC and ATL.

Understanding The ActiveX Object Model Introduction



The ActiveX Object Model for AutoCAD Land Desktop exposes features and provides access to project data through an object hierarchy. This hierarchy provides a convenient and consistent programming interface to your projects.

The object hierarchy can be grouped into the following categories:

Alignments

Plan alignments can be defined from tangent, curve, and spiral objects. You can also manage Station Equations.

Project Management

This includes extensions to the AutoCAD object model so that you can manage Land Desktop Projects. It also provides access to Preferences and Project Prototypes.

COM Wrappers for ARX Objects

These are provided for the Point, Contour, and CurveText ARX objects that reside in your drawings. You can modify existing objects, or add and delete objects from the drawing. In addition, programmatic access to the ContourStyle dictionary object is provided so that you can manage the appearance of Contour objects.

Coordinate Geometry

These objects allow you to manage your Cogo Point database, Point Groups, and Description Keys.

Surfaces

Terrain management is exposed through a collection of surfaces that have inputs and generated outputs. You can define a surface from Point Files, Point Groups, Boundaries, Breaklines, Contour data, or DEM Files. You can build a surface and a watershed model. The generated Tin Points, Edges, Faces and Elevation Contours are provided as outputs.

Parcels

Parcel objects can be defined from line and curve objects. Properties such as area, perimeter, centroid, and name can also be accessed.

Profiles, Cross Sections, and Superelevations

The ActiveX Object model provides access to profiles, cross sections, and superelevations. You can also access the profiles and cross section blocks in the drawing for custom drafting.

Topics in this Section:

- The Big Picture
- Project Management
- <u>Alignments</u>
- <u>Surfaces</u>
- <u>CogoPoints</u>
- Profiles
- Cross Sections and Superelevations
- Parcels
- <u>COM Wrappers</u>

The Big Picture



ActiveX Object Model

The ActiveX Object Model is an interface into the AutoCAD Land Desktop product. It is not a replacement for AutoCAD's ActiveX interface but is an extension of that interface.

Through the ActiveX Object Model you can create and edit your project data. Through AutoCAD ActiveX, you can create and modify drawing entities. There is very little overlap between the two. The ActiveX Object Model provides very little drawing interaction. The ActiveX Object Model has no user interface. AutoCAD ActiveX provides these (and others) features. Using AutoCAD ActiveX you can draw and edit AutoCAD entities, as well as prompt the user for entity selection or creation.

By using both the ActiveX Object Model and the AutoCAD ActiveX interface, you can easily customize AutoCAD Land Desktop for your business.

ActiveX Object Model Versions

When designing your applications, you can detect what version of the ActiveX Object Model is installed, and behave appropriately according to the available features. If you are using ActiveX Automation (using IDispatch methods such as Invoke and GetIdsOfNames), you can try to access a new feature with the appropriate error handling (on error handler, try-catch block, etc.). However, if you are creating compiled applications (vtable binding), you must check the program IDs that are registered for the application object:

Sub version()

Dim app As AeccApplication Dim version2 As Boolean Dim version4 As Boolean version2 = False version4 = False

```
Check1:
```

On Error GoTo Check2 Set app = ThisDrawing.Application.GetInterfaceObject("Aecc.Application.2") version2 = True

Check2:

On Error GoTo Done

Set app =

ThisDrawing.Application.GetInterfaceObject("Aecc.Application.4") version4 = True

Done:

If version4 Then

Debug.Print vbCrLf & "Land Desktop 2007 functionality is available." ElseIf version2 Then

Debug.Print vbCrLf & "Land Desktop 2i functionality is available." & _ "Do not try to invoke 2007 features on this desktop."

Else

Debug.Print vbCrLf & "No ActiveX functionality for Land Desktop is available."

End If

End Sub

Loading Civil Engineering Features

To access civl engineering features, such as Alignment Profiles, CrossSections and Superelevations, you must have AutoCAD Civil 3D Land Desktop Companion 2009 running. The "AutoCAD Civil 3D Land Desktop Companion 2009" version of Land Desktop has the civil engineering and survey features included. The version of Land Desktop named "AutoCAD Land Desktop 2009", sometimes referred to as the stand-alone version of Land Desktop, does not have the civil engineering and survey features installed. You may get an error code AECC_E_CIVIL_NOT_LOADED (or message "Civil Desktop\civil engineering features has\have not been loaded.") when you try to access profile objects on the version of Land Desktop that does not contain civil engineering features ("AutoCAD Land Desktop 2009").

If you have AutoCAD Civil 3D Land Desktop Companion 2009 installed, the civil engineering features are automatically loaded.

Data Validation

The AutoCAD Land Desktop product gets input through menus, dialogs, command line input, and the pointer device. In most cases the program validates this input before sending the data to the functions. The ActiveX Object Model exposes the programs functions, bypassing the AutoCAD Land Desktop interface. When using The ActiveX Object Model, you are responsible for the interface. The method used for collecting and validating the data is critical to the design of any program written for use with the ActiveX Object Model. Providing incorrect data to the Methods or Properties provided in the ActiveX Object Model can result in invalid results and incorrect data in the projects databases.

Projects and Drawings

AutoCAD Land Desktop is project-based. The majority of data for a project is stored externally from an AutoCAD drawing. This includes COGO points, surfaces, and alignments. A Drawing is considered part of that project data. A drawing is associated with one project. However, a project can have multiple drawings associated with it. A drawing is a view into the project data. Because you can have multiple drawings associated with a project. In one drawing, you might have all of the COGO points. In another drawing, the alignments might be displayed. The number of project-based drawings and their content is up to the end user. However, all data ends up as project data.

This idea of being project-based is central to AutoCAD Land Desktop, and the ActiveX Object Model is based on it.

Drawings and Documents

A Project can have multiple AutoCAD drawings associated with it. The Drawings collection is this list of AutoCAD drawings. This list is retrieved from the [Project]\dwgs directory.

A Document is a drawing file loaded into AutoCAD Land Desktop. Currently, the ActiveX Object Model only supports one Document being loaded at a time. This Document is derived from the AutoCAD ActiveX Document and all methods are supported.

Retrieving Project Information

The ActiveX Object Model can only retrieve data for the ActiveProject. The ActiveProject is the Project associated with the Document that is currently loaded. You can get a list of all Projects, the statistics for those Projects, and the drawings associated with those Projects. You can not get any other information from those Projects. For example, you can not get a list of alignments or surfaces for a non-active Project. In addition, you can not have an active project if AutoCAD Land Desktop is not running. If AutoCAD Land Desktop is not running, the ActiveX Object Model will not be running and you can not get any information about projects or drawings.

Keeping this in mind, usually the first thing you do is to get the ActiveProject. In VBA, to retrieve the ActiveProject, do:

Dim activeProj As AeccProject Set activeProj = AeccApplication.ActiveProject

Project Navigation

From the Project, you can get most of the Land project information that has been exposed through the ActiveX Object Model. The Object Model diagram shows that from the Project you have direct access to CogoPoints, PointGroups, Description Keys, Surfaces, and Alignments. In addition, you have access to all of the Project specific settings and FileLocks.

CogoPoints, PointGroups, Description Keys, Surfaces, and Alignments are collections. A collection will contain zero or more objects. For example, the Surfaces collection will contain zero or more individual surfaces. To get access to an individual member of a collection, use either the Item or Enumeration methods on the collection.

Note: All collections support the Item and Enumeration methods, and the Count property. Note that all collections begin with an index of 0. This is a

C++ standard, and is used consistently across AutoCAD-based ActiveX implementations.

You can query or modify the object after you retrieve it. Delete or add operations always exist at the collection level. If you think about this, it makes sense. When adding or deleting, you are changing the collection. All additions and deletions to a collection will return the new object.

What's Next

Ok, so now you have the ActiveProject, what is the next step? First, you have to figure out what you want to accomplish with your program. Let's take a simple problem: print out the PointGroup names. The easiest way to use the ActiveX Object Model is to look at the Object Model and work your way backward. The problem is to print out some information on a PointGroup. If we click on the PointGroup in the Developers Guide, it will display all of the methods and properties for the PointGroup. In the Developers Guide, we see that PointGroup has a property called GroupName. This is what we want to print out.

After we have figured out what we want to print, then we work our way up the Object Model until we get to the Application. By doing this, we see that we have to use the PointGroups collection, the Project object, the Projects collection, and finally the AeccApplication. These are the objects needed to print out the PointGroup.GroupName. If we reverse the order and start at the AeccApplication, we can now work our way back down to the PointGroup. This is how to get PointGroup information.

You can traverse the Object Model in the same fashion for any of the objects or collections. First, find what information and object you need, then follow the Object Model down to that object.

Overview of Modules Projects

From Projects, you can get to CogoPoints, PointGroups, DescriptionKeyFiles, Surfaces, Alignments, Parcels, CrossSections and Superelevations. In addition, the PreferencesProject, FileLocks, and Drawings are exposed. The PreferencesProject contains the Cogo, Surface, Alignment, Parcel, Profile, and CrossSection properties. These properties return objects that expose settings for those areas. All settings are for the project.

FileLocks is a collection of all multi-user locks that exist for the project.

Drawing collection is the list of drawings for the project.

Cogo

CogoPoints, PointGroups, and DescriptionKeys are exposed through the ActiveX Object Model. The CogoPoint is a COGO point that exists in the Project database. It may or may not exist in the current drawing. When a COGO point is inserted into a drawing, it can be inserted as an AeccPoint or as a block. The AeccPoint is a COM wrapper around an ARX object.

PointGroups are used to work with a number of points at a time. A PointGroup does not actually hold the CogoPoint. The PointGroup has a range of point numbers. The point number in a point group may contain points that are not contained in the Project point database. A PointGroup can also be used to override the elevation, description, and name for the points in the PointGroup.

A DescriptionKey overrides the raw description for the COGO point. By using DescriptionKeys, you can substitute a short description with a longer description. By doing it with a DescriptionKey, you only have to change the value in one spot.

Surfaces

The Terrain Model is exposed through the Surfaces collection. Using the Surface object, you can define the inputs from PointFiles, PointGroupNames, BreakLines, ContourItems and DEM Files. You can also add Boundaries to your Surface.

You can also get all output information for the Surface. TinPoints, Edges, Faces, ElevationContours, and Watersheds can be retrieved for a Surface. You can query the Surface by point to get the elevation using the Surface.FindPoint method. You can query between two points and retrieve the coordinates where that line crosses a Surface Face by using the Surface.SampleElevations method. For a given elevation, you can retrieve all of the contours on that Surface by using the ElevationContours.Elevation method. This gives you full control over creation and querying of a Surface.

Volumes are also supported as Surface objects that result from DifferenceGrid and Composite calculation methods.

Alignments

The Alignments collection has all of the Alignments for the project. For each Alignment, you can retrieve, add, and delete the entities and station equations. Queries can also be made on the Alignment: Alignment.StationOffset, Alignment.PointLocation.

Parcels

The Parcels collection provides access to all of the parcels in the project. Each Parcel can be queried to obtain property information such as area, perimeter, centroid and name. New parcels can be created or exisiting parcel modified by editing thier individual curve and line objects.

Profiles, Cross Sections, and Superelevations

If you are using AutoCAD Civil 3D Land Desktop Companion, which includes civil engineering and survey features, extensive read access is available for existing ground profiles, finished ground profiles, cross sections, and superelevations properties. If you are using the stand-alone version of Land Desktop (titled "AutoCAD Land Desktop"), the civil engineering and survey features are not available.

You can import profiles and cross sections to the current drawing, and then access the resulting ProfileBlocks and CrossSectionBlocks for custom annotation.

Overview of Events

Events are a mechanism that allows your application to receive notification when a change occurs to the state of some object. To receive notification, you must "enable" events by providing a "handler" or "event sink". Refer to the sample applications for examples of how to implement event handlers in VB and C++.

When coding in VBA, you must provide an event handler for all objects enabled for the Modified event. If you do not provide a handler, VBA may terminate unexpectedly.

Note that no events will be fired while a modal dialog is being displayed.

AcadObject

All of the objects outside of the Projects and Prototypes hierarchy are derived from AcadObject. This means that objects such as AeccApplication and AeccDocument derive all of the properties and methods from the parent ActiveX objects. However, since no additional events were implemented, you will use the base objects (AcadApplication and AcadDocument, for example) to get notifications in your application.

AcadEntity

The COM wrappers for the AeccDbPoint, AeccDbContour and AecDbCurveText objects support the Modified event. This event will be triggered whenever the object is modified. Modification includes whenever the value of a property is set, even if the new value is equal to the current value.

AeccObject

The AeccObject supports the Modified event, and all objects in the Projects and Prototypes hierarchy are dervied from AeccObject. While every object can support event notification, refer to the following table to determine when an event is fired.

Event notification is dispatched by a queueing mechanism that responds when AutoCAD enters a quiescent state. No events will be fired while a modal dialog is being displayed or while a command is in progress. In addition, the event queue will not accept more that one event for a given instance of a COM object at a time. If two or more event conditions occur on an object while a pending event is in the queue, there will be only one event sent when AutoCAD becomes quiescent.

	Event	Objects
Project Management	New Drawing	Project, Drawings
	Open Drawing	Project
Coordinate Geometry	Save/Delete point	Points, Point
	Save?Delete point group	PointGroups, PointGroup
	Save/Delete desc key	DescKeyFiles, DescKeyFile, DescKey
Plan Alignments	Save/Delete	Alignments, Alignment, AlignEntities, StationEquations
Surfaces	Lock/Unlock surface	Surfaces
	Create	Surfaces
	Set current surface	Surfaces
	Save/Save As	Surfaces, Surface

Delete/Rename	Surfaces, Surface
Set description	Surface
Add point file	Surface, SurfaceInputs, Pointfiles
Add point groups	Surface, SurfaceInputs, PointGroupNames
Add breakline	Surface, SurfaceInputs, BreakLines
Add boundary	Surface, SurfaceInputs, Boundaries
Add contour	Surface, SurfaceInputs, ContourItems
Set outer boundary	Surface, SurfaceInputs, Boundaries
Flip face	Surface, SurfaceOutputs, Edges, Faces, WaterSheds
Insert ND breakline	Surface, SurfaceOutputs, TinPoints, Edges, WaterSheds
	Surface, SurfaceOutputs,

Insert line	TinPoints, Edges, WaterSheds
Delete line	Surface, SurfaceOutputs, TinPoints, Edges, Faces, WaterSheds
Minimize flat triangles	Surface, SurfaceOutputs, Edges, Faces, WaterSheds
Add boundary	Surface, SurfaceOutputs, TinPoints, Edges, WaterSheds
Add to all elevations	Surface, SurfaceOutputs, TinPoints, Edges, Faces, WaterSheds
Set elevation	Surface, SurfaceOutputs, TinPoints, Edges, Faces, WaterSheds
Paste surface	Surface, SurfaceOutputs, TinPoints, Edges, Faces, WaterSheds
Build	Surface, SurfaceOutputs, TinPoints, Edges, Faces
Build hydrology model	Surface, SurfaceOutputs, TinPoints, Edges, Faces, WaterSheds

	Compute extended stats	Surface
	Set/Calc bounding box	Surface
Alignments	Set current alignment	Alignments
Parcels	Create	Parcels
	Delete, Import, Merge	Parcel, Parcels
Profiles	Alignment Editor Save	EGProfiles, FGProfiles
	Creat, Edit, Delete	EGProfiles, FGProfiles
CrossSections &	Sample Existing Ground	CrossSections, Superelevations
Superelevations	Process Sections	CrossSections, Superelevations

Running Multiple Instances of AutoCAD

All AecBase objects that are derived from Acad objects support the Init method. This includes AecApplication, AecDocuments, AecDocuments, AecDatabasePreferences, AecPreferences, AecPreferencesFiles, AecPreferencesUser, etc. The help string describes the Init method as an "Initialization method which must be called before accessing the AEC application object" The ActiveX Object Model for AutoCAD Land Desktop performs the call to Init automatically for Aecc objects.

It is not restricted (hidden) because it comes in handy when you are running multiple instances of AutoCAD, Architectural Desktop (AutoCAD Architecture) or AutoCAD Land Desktop. This allows you to pass in an instance of the AutoCAD application object that you are interested in, instead of relying on the active instance handed back from the Running Object Table. If you pass this into an instance of the application object, it will automatically be passed on to any new instance of child objects.

Understanding Project Management



Land Desktop projects are used to organize drawing files, support data files, and settings that are associated with each job you work on. Just as the ActiveDrawing is central to the AutoCAD application object, the Land Desktop application object exposes the ActiveProject. Through the project object, you can access Coordinate Geometry, Plan Alignment, Digital Terrain, Parcels, Profile, Cross Section and Superelevation data and preferences.

To create a new project, use the Projects.Add method, supplying a prototype name from the Prototypes collection and a project name for the new project. To create the first project-based drawing for the new project, use the Document.NewProjectBased method, specifying the drawing template name, the project name, and the new drawing name. This will switch the desktop to the newly created project and drawing.

The Projects collection is updated whenever the ProjectPath property is changed. This allows you search for projects at various paths. To access folders on a network, you must first map the drive. Note that you are limited to the FileLocks and Drawings collections for any Project that is not the ActiveProject . To access the CogoPoints, PointGroups, DescriptionKeyFiles, Alignments, Surfaces, Parcels, Profiles, CrossSections and Superelevations you must be in the project currently loaded into AutoCAD Land Desktop.

Extensions to AutoCAD objects

The Land Desktop object model extends many AutoCAD ActiveX objects by adding properties and methods. This includes the Application, Documents, Document, DatabasePreferences, PreferencesUser, PreferencesFile, and the Utility objects. This enhancement is automatically available when you navigate from the AeccApplication object to other objects in the model.

The Documents collection supports the management of multiple drawings (MDI) in the AutoCAD environment. Since AutoCAD Land Desktop works on a single drawing at a time (SDI), the Documents object is ancillary. You can get the first (and only) document from the Document collection,

but it is generally more convenient to use the AeccApplication.ActiveDrawing method. Also, note that the Documents.Add and Documents.New methods are not available in SDI mode.

The Document object adds two important methods - OpenProjectBased and NewProjectBased. Use these methods to open and create projectbased drawings, respectively. Both methods accept an optional Project name to allow you to switch from one project to another. Note that the Document.Open and Document.New methods do not support projects and will return an error if invoked.

The DatabasePreferences object represents settings that are saved on a pre-drawing (or document) basis. This includes base settings for Architecture and Civil Engineering, as well as AutoCAD Land Desktop specific settings. These settings can be loaded and saved to an external setup profile.

The PreferencesUser and PreferencesFiles objects store user and application level settings, respectively. The PreferencesUser settings control overrides to AutoCAD behavior and the most recently used project and drawings. It also allows a default drawing setup profile to be applied to new drawings, so that the user is not prompted for settings. The PreferencesFiles object exposes the paths used by AutoCAD Land Desktop.

The Utility object adds methods that convert an Easting and Northing to an AutoCAD X and Y and back using the geodetic base point and north rotation for the drawing.

Project-owned objects

In addition to providing access to Coordinate Geometry, Plan Alignments, Parcels, Terrain Modeling, Profiles, Cross Sections, and Superelevations, the Project object exposes the Drawings and FileLocks collection.

The Drawings collection is similar to the Documents collection, except that it represents AutoCAD drawing files in a local or remote folder (directory). These files may or may not be loaded into the application. You can use the Projects and Drawings collections to browse any Land Desktop project on a local or mapped remote drive. For convenience, you can use the Drawings.Open method to switch to the parent project and open the drawing as the ActiveDocument (similar to the Document.OpenProjectBased method).

The FileLocks collection exposes multi-user project locks.

Using Collections

The object model contains simple objects and collections of objects. For some collections, if you change the value of a property, the collection will be rebuilt. You need to refresh your objects to see the effect of this change. For example:

Dim projects As AeccProjects Set projects = AeccApplication.Projects

```
' Get the first project in the collection
Dim project As AeccProject
Set project = projects(0)
```

' Change the project path
projects.Path = "c:\projects2"

```
' Project is a valid object, but no longer represents
' the first item in the projects collection
' refresh project
Set project = projects(0)
```

Changing the Active Project

When a drawing is opened from a different project, the AeccApplication.ActiveProject object and AeccApplication.Projects collection are released. For this reason, it is a good idea to get these objects as needed instead of holding on to them for the duration of your application.

Notifications for changes to drawing settings

The settings exposed by the DatabasePreferences object are stored in the AcDbDictionary named "AEC_VARS". The

"AEC_VARS_DWG_SETUP" key returns an AecDbVarsDwgSetup object that stores the base settings for Architecture and Civil Engineering. The

"AEC_VARS_CIVILSETUP" key returns an AcDbVarsCivilSetup object stores the AutoCAD Land Desktop specific settings.

To receive events when drawing settings are changed, implement an event handler in a class module, declare an object of type AcadObject with events, and assign the desired dictionary object to the handler in your code module or form.

Understanding Alignments



The ActiveX Object Model gives access to AutoCAD Land Desktop horizontal alignments. An Alignment can contain tangents (AlignTangent), curves (AlignCurve) and spirals (AlignSpiral). An Alignment can be set to current. It supports the locking mechanism from AutoCAD Land Desktop. In addition, station equations are supported for each Alignment.

This section describes the fundamentals you need to know when developing an application with Alignments. It provides information on creating, using, and saving Alignments.

Creating an Alignment

If an Alignment is added successfully to the Alignments collection, there are two results: a) the new Alignment is made Current and b) the new Alignment will have a write lock on it. The new Alignment is also saved immediately. AutoSave is ignored when an Alignment is created.

"Current" Alignment and Locks

AutoCAD Land Desktop has the notion of a "current" alignment. Historically, this has been used in the application to relieve the user from having to select an alignment for every command. Also, when an alignment was set to current, locks were created on that alignment. If there was a previous alignment, the locks were freed.

The ActiveX Object Model supports the "Current" Alignment concept. An Alignment can be made Current and locks will be generated at that time. Also, that Current Alignment in ActiveX Object Model will also be the current alignment in AutoCAD Land Desktop. Through the ActiveX Object Model, you can stay in sync with the menus.

When the Alignments collection is created, all of the Alignment names are loaded into memory. At that time, each Alignment is given a kReadLock. No physical lock is created on disk. This lock exists only in the ActiveX Object Model. If the Alignment in the collection is the Current Alignment, then the ActiveX Object Model will retrieve whatever lock currently exists on disk. It could be either a kReadLock or a kWriteLock. When you make an Alignment current, several things will happen. If there was a previous Current Alignment, that will be unlocked. Next, the new Current Alignment will be set in both the ActiveX Object Model and AutoCAD Land Desktop. Finally, a lock will be generated on that Alignment in the ActiveX Object Model and on disk. This lock can be seen by any users currently in that project. The lock can be either a kReadLock or a kWriteLock. The type of lock depends on weather that Alignment was locked by another user.

As a rule, both the ActiveX Object Model and AutoCAD Land Desktop will always try to retrieve a kWriteLock when accessing an object. This applies to Alignments and Surfaces. If someone else has a write lock on that object, then a read lock will be added to that object.

Before any edits are made on an Alignment, the Alignment must be made Current. Also, that Alignment must have a kWriteLock for any changes to be made.

You can add a new alignment to the collection and later define the underlying geometry. Be advised that if you save an alignment without adding any tangents, curves or spirals, you will not be able to edit the geometry using the alignment editor.

AutoSave

Any changes made to an Alignment are automatically saved. When the Alignment is saved all entities and the general Alignment data (starting station, description) are written out. This can impact performance if many changes are made to the Alignment. Remember that every change made results in writing the entire Alignment.

To help with the performance issues, AutoSave was created. If Alignments.AutoSave is True, then every change made to an alignment results in a write operation. If Alignments.AutoSave is False, then no write operations are performed. It is the responsibility of the programmer to call Alignment.Save for the write operation.

AutoSave is at the Alignments collection level. When the AutoSave value is set, all alignments in the collection are updated.

Understanding Surfaces



Through Terrain Model Explorer, a three dimensional surface model can be generated from data. In the ActiveX Object Model, you can create and query this three dimensional model through the Surfaces collection. You can use the Surface collection to iterate through all of the existing Surfaces, create an new Surface, or delete an existing Surface. These are just a few of the Surface operations at the collection level.

A Surface is broken down into inputs (SurfaceInputs) and outputs (SurfaceOutputs). You can modify the SurfaceInputs by changing the ContourItems, Breaklines, Boundaries, PointFiles, PointGroups or DEMFiles. From the SurfaceOutputs, you can get the TinPoints, Edges, Faces, ElevationContours, and Watersheds.

Volume information can be created by using the Surfaces.DifferenceGrid or the Surfaces.CompositeGrid methods. Both methods create a new Surface. This new Surface can then be queried for volume information.

Surface Boundary Points

The implementation of IAeccBoundaries::Add() will close a boundary, if the last point and first point are not identical. In addition, if the last and first points are not identical and cause lines that intersect each other, the surface will not behave as expected. If two points on the boundary are co-located, the boundary will be rejected. The best way to create a boundary is to allow the implementation to close the boundary automatically.

Visible Areas

Using Breaklines and Boundaries, areas of a Surface can be made invisible. Watersheds and Contours will not be returned for the invisible areas. Faces, Edges, and TinPoints will be returned for the invisible areas.

Build Settings

Surface.Build will use the settings defined in Terrain Model Explorer Build dialog. These settings include the "Apply Edit History" and "Log Errors To

File" options.

Edit History

Edit History is not exposed in this release of the ActiveX Object Model. However, if the "Apply Edit History" is turned on in Terrain Model Explorer, then Surface.Build will use this setting.

Surface Locking

Initially, all Surfaces have no locks applied to them. When a method or property is called, then the Surface is opened. At that time, a lock will be created on that Surface. The ActiveX Object Model will always try to create a write lock (kWriteLock). If someone else has control of that Surface, you will get a read lock (kReadLock). If the Surface is selected as the Current Surface, the locks will also be generated. When the object has gone out of scope, the Surface will go back to its original state. If it was opened and locked before the ActiveX Object Model object was created, then it will stay opened and locked. If the Surface was not opened (and therefore, not locked), then when the ActiveX Object Model object goes out of scope, the Surface will be closed and unlocked.

Elevation Contours

After a surface has been build, you can generate contour data by setting the ElevationContours.Elevation property. Zero or more ElevationContour objects will be added to the collection, depending on the elevation you set. You can use the ElevationContour.Coordinates property to set the vertices of an AeccContour object.

Understanding CogoPoints



For each Land Desktop Project, there is a COGO points database. The ActiveX Object Model gives access to this database through the CogoPoints collection. Through this collection, you can add, delete, and modify a CogoPoint.

A CogoPoint has Northing, Easting, and Elevation fields. If the GroupName is set for the CogoPoint, then you will also see the overrides from that PointGroup.

PointGroups and DescriptionKey are also exposed through the ActiveX Object Model. These collections and objects allow you to work with groups of COGO points.

CogoPoint and AeccPoint

In a Land Development document, a COGO point can exist in the drawing. This point can be represented as either an Arx object or a Softdesk block. If the point is an Arx object, then you can access it by using the COM wrapper AeccPoint. This point does not necessarily exist in the point database. If it does exist in the database, then you could access it by using a CogoPoint object.

To state it differently, a CogoPoint object is a point database object and an AeccPoint is a COM wrapper for the Arx object.

A CogoPoint and AeccPoint can represent the same geometry. If you change the coordinates for a CogoPoint, the Arx object in the drawing will be updated.

Point Number verses Item Number

When the collection class is built for CogoPoints, several things happen. First, a list of the currently used points is generated. These points are then added to the collection. For example, the point database has points with point numbers 1, 3, 5, and 7. When the collection is built, then the collection will have 4 items. Item(0) will be point 1, Item(1) point 3, Item(2) point 5, and Item(3) point 7. There is no correspondence between the Item number and the point number. If you want to work with points by point number, use CogoPoints.PointByNumber.

Understanding Profiles



The vertical alignment data is represented by two collections: the EGProfiles and the FGProfiles. The EGProfile, or existing ground profiles, contains the sampled surface data along the alignment. The FGProfile, or finished ground vertical alignments, contain the PVIs which represent the basis for the final roadway design. The EGProfile can be imported into the drawing, so that the finished vertical design data can be drafted onto the ProfileBlock.

Existing Ground Profiles

While the ActiveX interface does not require you to define any EGProfiles, the vertical alignment editor expects the Existing Ground Center to be defined.

Finished Ground Profiles

When you create a new FGProfile object, a default PVI is added to the PVIs collection as a place holder. The properties of this PVI are all set to 0.0. This behavior is intentional since at least one datum is required to access the vertical alignment editor.

It is up to the user to make sure that the geometry for the Finished Ground Profile is correct. When defining a profile, be sure to check for the following:

- A Finished Ground Profile cannot support a vertical curve on the first or last PVI.
- The Finished Ground Profile object does not check for overlapping vertical curves.
- The Finished Ground Profile object does not check to ensure the start and end points coincide with the paln alignment.

Locking

Existing Ground Profiles and Finished Ground Profiles objects use the same locking as the Alignment. If the Alignment has a read lock, then Existing Ground Profiles and Finished Ground Profiles have a read lock.

The same is true for write locks. Remember, the only way to generate a write lock is to set the Alignments.CurrentAlignment to the one you want to work with.

ProfileBlocks

ProfileBlocks do not support xrefs. If an drawing is XRefed into another drawing and it has a profile, the ProfileBlocks collection will not put it into its list.

In a drawing, an alignment can have more than one ProfileBlock drawn. If this happens, there will be no unique identifier for the ProfileBlock. The ProfileBlocks collection will have both ProfileBlock objects but it is up the user to determine which profile will be used.

The scale factor used for the vertical exaggeration of the existing ground and finished ground profiles is the ProfileBlock.VerticalScale divided by the DatabasePreferences.DatabaseScale.

The Coordinates of the ProfileBlock are one inch below the DatumElevation baseline.

Understanding Cross Sections and Superelevations



The CrossSections and Superelevations collections represent a slice of vertical alignment data at station intervals along the alignment. For a given CrossSection, the data is presented as a collection of point codes and surfaces, such as existing ground, top surface, match surface, templates and subassemblies. Superelevations are presented as a collection of stations at which the superelevation changes. CrossSections can be imported into the drawing as a CrossSectionBlock for custom annotation.

Read-only CrossSection and Superelevation Objects

The process of designing and applying roadway templates, subassemblies, transitions, and superelevations is a complex procedure. For this reason, the CrossSection and Superelevation objects are readonly. You must use the appropriate commands to create and edit the data. The ActiveX objects provide access to the data for analysis and custom drafting.

Managing the Current Alignment

When you access the CrossSection and Superelevation objects, the "current alignment" is set to the Alignment you are accessing. You can use the Alignments.CurrentAlignment property to save and restore the current alignment if required.

CrossSectionBlocks

CrossSectionBlocks do not support xrefs. If a drawing is XRefed into another drawing and it has a cross section, the CrossSectionBlocks collection will not put it into its list.

In a drawing, an alignment can have more than one CrossSectionBlock drawn for a given station. If this happens, there will be no unique identifier for the CrossSectionBlock. The CrossSectionBlocks collection will have both CrossSectionBlock objects but it is up the user to determine which profile will be used. The scale factor used for the vertical exaggeration of the cross section is the CrossSectionBlock.VerticalScale divided by the DatabasePreferences.DatabaseScale.

Understanding Parcels



The Parcels collection provides access to the Parcel database and allows a defined Parcel to be imported in the drawing. Similar to an Alignment, a Parcel is made up of geometric entities that can be enumerated using the ParcelEntites collection.

Closing Parcel Perimeters

When defining a Parcel, the perimeter is not automatically closed for you. A Parcel that has not been closed will return an area of 0.0 square units.

Locking

The Parcel database supports multi-user access with locks. Unlike Alignments, the locking occurs automatically on a command by command basis. A Parcel cannot be locked for the duration of a design session.

Understanding Com Wrappers



Overview

A COM wrapper provides an instance of an ActiveX object that represents a drawing entity. Your application uses the properties, methods, and events from this object to access and manage the corresponding entity in the drawing. COM wrappers are implemented for AeccDbContour, AeccDbPoint, and AecDbCurveText entities.

AutoCAD Land Desktop objects that are created in model space, use the AcadModelSpace collection to iterate through objects. For example:

```
Dim pnt As AeccPoint
Dim obj As AcadObject
For Each obj In ThisDrawing.ModelSpace
If obj.ObjectName = "AeccDbPoint" Then
Set pnt = obj
End If
Next
```

Similarly, you can iterate existing entities in a selection set to access your custom ARX objects through their COM wrappers. When using the FilterData parameter for an AcadSelectionSet, specify the object name that appears in the Properties window (or when you list an object): AECC_CONTOUR, AECC_POINT, or AEC_CURVETEXT.

Object Creation

You can use the AcadModelSpace.AddCustomObject method to (1) create an instance of your ARX object and (2) return an instance of your COM wrapper so you can use its properties to initialize the object. Note that you cannot create AutoCAD Land Desktop objects in PaperSpace or Blocks. For example:

```
Dim pnt As AeccPoint
Set pnt = ThisDrawing.ModelSpace.AddCustomObject("AeccDbPoint")
pnt.Easting = 1000
pnt.Northing = 1000
pnt.Elevation = 100
```

```
pnt.Number = 1
pnt.Description = "test point"
```

Events

The AutoCAD Land Desktop objects support notification through the Modified event.

Contour Styles

Contour styles are saved on a per-drawing drawing basis in the dictionary. To facilitate the management of styles for contour objects, a COM wrapper for the contour style is provided. You can access any style loaded in the current drawing through the "AECC_CONTOUR_STYLE"; dictionary:

```
Dim cs As AeccContourStyle
Dim obj As AcadObject
For Each obj In
ThisDrawing.Dictionaries("AECC_CONTOUR_STYLES")
Set cs = obj
If cs.Name = "Standard" Then
Exit For
End If
Next
```

```
Dim Contour As AeccContour
Set Contour =
ThisDrawing.ModelSpace.AddCustomObject("AeccDbContour")
Contour.SetContourStyle cs
```

Programming Guidelines Introduction



This section provides useful information on how to use the ActiveX Object Model with various programming languages and development environments.

Topics in this Section

- <u>VBA</u>
- <u>VB</u>
- <u>C++</u>
- <u>J++</u>
- Other Languages

VBA Programming Guidelines



Option Explicit

A good programming practice is to add the Option Explicit declaration at the top of your module. This will require you to declare all variables using the Dim, Private, Public, ReDim, or Static statements. If you attempt to use an undeclared variable name, an error occurs at compile time. This prevents Visual Basic from automatically creating a variant for the undeclared variable on your behalf, which leads to program errors.

For Next Loops

When using a For Next loop to iterate through an object collection use "EXIT FOR" if you need to exit the loop befor conclusion. If a "GOTO" is used to exit the loop the reference count to the object is not cleaned and will create an error condition.

IntelliSense

If an AutoCAD Land Desktop object does not display the complete list of properties and methods in the IntelliSense pick list, try assigning the object to a typed variable first. For example:

Dim prefuser As AeccPreferencesUser Set prefuser = AeccApplication.Preferences.User ' The prefuser object will now correctly display ' all properties and methods with IntelliSense

AutoCAD Land Desktop Enumerated Value

AutoCAD Land Desktop enumerated values are named constants used in the AutoCAD Land Desktop ActiveX Automation methods and properties.

In VBA, you cannot create an instance of an object of the enumerated type. For example:

' Works in VB, but not VBA Dim borderstyle As eAeccBorderStyle

Globals

In VBA, global variables are not released until the VBA project is unloaded. For example, in your project the following variables are declared as global variables:

Option Explicit

Public acad As AcadApplication Public app As AeccApplication Public projs As AeccProjects

The objects acad, app, and proj will not be released until the VBA project is unloaded. This is correct behavior but you must refresh the variables when you come back into the VBA code. If you do not refresh the variables, you might not get the latest information. For example, additional projects might be added to the AeccProjects collection.

VB Programming Guidelines



In general, all of the guidelines from the VBA section also apply to VB. This section includes some additional information that is specific to developing stand-alone (out-of-process) applications using Visual Basic.

Connecting to AutoCAD Land Desktop

Visual Basic provides a function called GetObject to get a reference to a currently active instance of an ActiveX object of the specified type. Since a compiled VB application is in a different process than AutoCAD Land Desktop, you must "connect" to the application object:

Dim acad As AcadApplication

Set acad = GetObject(, "AutoCAD.Application")

Next, you use the AcadApplication.GetInterfaceObject method to get an instance of the AutoCAD Land Desktop application object:

Dim app As AeccApplication

Set app = acad.GetInterfaceObject("Aecc.Application")

You might want to declare these variables as Public at the module level so that they are available to all procedures in all modules.

For Next Loops

When using a For Next loop to iterate through an object collection use "EXIT FOR" if you need to exit the loop befor conclusion. If a "GOTO" is used to exit the loop the reference count to the object is not cleaned and will create an error condition.

Quiescent State and ActiveX calls

When AutoCAD is at the command line prompt, it is in a quiescent state. When AutoCAD has completed processing the last command and there are no pending commands, it is in a state that AutoCAD can process an ActiveX call.

Be aware that if AutoCAD is busy when your application tries to invoke an ActiveX property or method, AutoCAD may display a modal dialog indicating that the ActiveX request was not handled. You are then prompted to retry the request.

To prevent this situation, your application should use the AcadApplication.GetAcadState method to determine if AutoCAD is busy. This method returns an AcadState object, which is a transient object used to check for AutoCAD quiescence from out-of-process applications.

C++ Programming Guidelines



Importing Type Libraries

COM replaces the notion of header files with Type Libraries. In the Microsoft Visual C++ environment, you use the preprocessor directive #import in the StdAfx.h file to make all interfaces, properties, methods, events, enumerations, etc. available in your code. Use the no_namespace option to disable the requirement of qualifying each object with the type library name.

For example:

// create C++ wrapper classes for the COM interfaces

// new TLH and TLI files will be generated each time

#import "acax16enu.tlb" /*no_implementation*/ raw_interfaces_only
no_namespace named_guids rename("GetObject", "acaxGetObject")

#import "AecXUIBase40.tlb" no_implementation raw_interfaces_only
named_guids no_namespace

#import "AecXBase40.tlb" no_implementation raw_interfaces_only
named_guids no_namespace

#import "LandAuto46.tlb" no_implementation raw_interfaces_only
named_guids no_namespace

Note that the AecXBase type library references the AutoCAD AxDb16 type library. While you do not have to explicitly import AxDb16enu.tlb, it must be in the path or properly registered on your workstation, otherwise you will encounter errors when importing AecXBase40.tlb.

Initializing the COM libraries

When you create your project, you may safely omit the options "Automation" and "ActiveX Controls" to reduce the code size of your project. However, you will need to make 2 manual changes to ensure that the COM libraries are initialized: In StdAfx.h:

#include <afxdisp.h> // MFC OLE automation classes

In your application's InitInstance method:

```
// Initialize OLE libraries
if (!AfxOleInit())
{
    AfxMessageBox("OLE initialization failed.");
    return FALSE;
}
```

Connecting to the Land Desktop Application Object

In general, these are the steps to get to the running instance of the Land Desktop Application Object:

- Call GetActiveObject to get the running instance of the AutoCAD application object. This method returns a pointer to IUnknown.
- Call QueryInterface to get the interface IAcadApplication.
- Call GetInterfaceObject to get the running instance of the Land Desktop application object. This method returns a pointer to IDispatch.
- Call QueryInterface to get the interface IAeccApplication.

If you have several instances of AutoCAD applications running at the same time (AutoCAD, Land Desktop, Architectural Desktop, Raster Design), and you are developing an in-process application (e.g. ARX DLL), you can use these steps to get the correct instance of AutoCAD:

- Call AfxGetApp to get a pointer to the CWinApp derived object.
- Call GetIDispatch to get the IDispatch interface for the AutoCAD application object.
- Call QueryInterface to get the interface IAcadApplication interface.

- Call GetInterfaceObject to get the running instance of the Land Desktop application object. This method returns a pointer to IDispatch.
- Call QueryInterface to get the interface IAeccApplication.
- Call Init, passing in the IAcadApplication pointer to initialize the object model with the proper instance of AutoCAD.

When developing a stand alone (out of process) application that needs to handle multiple running instances of AutoCAD applications, you must enumerate the Running Object Table for the desired instance of AutoCAD in place of the call to AfxGetApp. See MSDN Article 190985 for details and sample code.

Restricted Properties and Methods

When you import a Type Library into your project, you may seem to have access to read-only properties and be able to invoke methods that are not documented in the AutoCAD Land Desktop ActiveX and VBA Reference. These "restricted" members of the interface are hidden in VBA and other macro languages, but are not filtered out by the import directive.

Be aware that these members cannot be called arbitrarily. Most are used for initializing the COM object, and will not update your project database. Refer to the AutoCAD Land Desktop ActiveX and VBA Reference for the list of supported interface members, or use the OLE Viewer supplied with the Microsoft Visual Studio Tools to determine which members are restricted.

Caching Interfaces

It is recommended that you cache the application object only (i.e. IAeccApplicationPtr). This will allow access to any object throughout the duration of your application.

For all other objects, it is generally safer to query for interfaces upon each use instead of caching and re-using pointers as class members or other non-local variables. This will avoid any confusion about the state of the interface object at the time of use.

Exception Handling

When you import a Type Library into your Microsoft Visual C++ project, the preprocessor generates both "raw" methods and "wrapper" methods in the *.tlh file. The raw methods return HRESULTS, which should be checked by the programmer. The wrapper methods hide the HRESULTS, providing error handling for you, and will issue a COM exception when an error occurs. While the wrapper methods provide a convenience to the developer, unhandled exceptions will appear in a dialog before your users.

In either case, you should handle these errors in your application. This can be accomplished in one of two ways:

- Wrap all calls to wrapper methods in try-catch blocks
- Check the HRESULT from the raw methods and implement error checking yourself

Smart Pointers

When you import a Type Library, the generated wrapper methods will use "smart pointers" whenever the method returns an interface. You must use a smart pointer in your code to ensure that the reference count on the object does not decrement to 0 before the call to the wrapper method returns. If you use a standard pointer, you will be left with the address of a released COM object, and an exception will occur when you try to use it.

COM Support classes

If an argument or return value of a wrapper class expects a _variant_t (or _bstr_t), do not use a VARIANT (or BSTR). This will compile, but will ultimately cause problems.

For example, the raw method for AeccCogoPoints.PointStringToArray looks like this:

```
virtual HRESULT __stdcall raw_PointStringToArray (BSTR PointString,
VARIANT * Points ) = 0;
```

The wrapper method looks like this:

;_variant_t PointStringToArray (_bstr_t PointString);

The COM support class _variant_t automatically initializes its internal VARIANT using the correct constructor (the one that takes a VARIANT*, in this case). Note that while in general, the client is responsible for deallocating the memory allocated for the variant, the _variant_t (or _bstr_t) classes do it automatically for you.

MFC class Wizard Support

Through Microsoft's MFC classWizard, you can generate MFC classes from the Type Library. This gives you easy access to wrapped methods and properties. However, some of the Land Desktop objects are derived from AutoCAD or Aec objects. For example: AeccApplication, AeccDocument. When a MFC class is generated for these classes through classWizard, then only the additional methods and properties that Land Desktop added will be implemented in that new class. classWizard does not implement the methods and properties from the parent class.

ARX development

When registering a new command that will invoke ActiveX methods, use the ACRX_CMD_MODAL command flag. The command must be modal (and not transparent) so that no other commands or programs are currently active.

If you plan to use the Document.OpenProjectBased or Document.NewProjectBased methods in your registered command, be sure to add the ACRX_CMD_SESSION and ACRX_CMD_NOINTERNALLOCK flags. The former flag runs the command handler in the session fiber, while the later avoids locking the AutoCAD state out of quiescence (normally used with ActiveX calls). These flags are required to ensure that AutoCAD is in the proper context when switching documents.

Return values

In general, all properties and methods return S_OK when no errors are

encountered. The return value E_FAIL is returned when an internal error condition prevents successful completion of the operation. An example is a memory allocation or other Win32 service failure. In certain cases, a return value of S_FALSE is returned to indicate that the operation did not complete due to user error. Some examples include trying to append an alignment entity that has no matching endpoint with the previous entity, or trying to access the coordinates of an elevation contour using an invalid elevation for the surface.

In your application, check for a return value of S_OK to indicate that no internal or user errors have occured.

Using "IntelliSense"

If you typically use IntelliSense to determine what properties and functions are available for a given interface, please be advised that restricted functions will show up in IntelliSense. These restricted functions are not supported, and in most cases will cause undesirable effects. The safe way to determine the supported functions is to consult the Help for AutoCAD Land Desktop ActiveX Objects.

J++ Programming Guidelines



Importing Type Libraries

After creating a Windows Application project in Visual J++, you can import COM objects through the Type Libraries. Visual J++ creates class wrappers that provide the interface for accessing the COM objects. These class wrappers are added as packages in your project directory.

Note: To access a specific COM object from another project, you can avoid wrapping the objects for each project by placing the COM wrapper classes in the CLASSPATH.

The COM wrappers can be created from the Visual J++ IDE or at the command line using the jactivex.exe utility.

To create the COM wrappers from the command line: Invoke the jactivex.exe utility. For example: jactivex /wfc /w /xi /X:rkc /d "d:\VJ++\COM_Wrappers" landauto46.tlb Build the class files for each COM Wrapper java file using the Java compiler. The *.java files can be removed from the CLASSPATH after the class files have been created.

To generate the COM Wrappers from the Visual J++ IDE: On the COM Wrapper dialog, select the Land Type Library 1.1 check box and click OK. Select the Build menu item to build the solution and generate the class files from the java files The COM wrapper folders may now be removed from the project to prevent the class files from being regenerated during each build. This will reduce the size of the solution and reduce the build time

Getting the Application object

This example shows how to access the AutoCAD Land Desktop Application object through the running object table.

// Import the library packages so objects can be referred by their short
names:
// Required Visual J++ imports:
import com.ms.win32.Ole32;

```
import com.ms.wfc.ui.*;
import com.ms.com.*;
```

```
// Required Landauto imports:
import acax16enu.*;
import AecXBase40.*;
import AecXUIBase40.*;
import AxDb16enu.*;
import LandAuto46.*;
```

```
public class LandAutoSample extends Object
{
    private IAeccApplication m_iaecApp;
    private IAeccProjects m_iaecProjects;
    private IAeccProject m_iaecProject;
```

```
// Constructor:
public LandAutoSample()
{
    m_iaecApp = GetAeccApplication();
    m_iaecProjects = m_iaecApp.getProjects();
    m_iaecProject = m_iaecApp.getActiveProject();
}
```

```
// Sample class method to get the existing IAeccApplication object:
private IAeccApplication GetAeccApplication()
```

```
// *** Get the existing AutoCAD session:
```

{

```
Ole32 ole32 = new Ole32();

String sAcadApp = "AutoCAD.Application";

_Guid _guid = ole32.CLSIDFromProgID(sAcadApp);

String sGuidAcad = _guid.toString();

String sMonikerDispName = "!" + sGuidAcad;

IRunningObjectTable irot = ole32.GetRunningObjectTable(0);

IMoniker imoniker = ole32.CreateItemMoniker(sMonikerDispName,"");

IAcadApplication iacApp = (IAcadApplication)
```

irot.GetObject(imoniker);

// *** If you are going to use civil enginering feature interfaces, Cross Sections, Profiles, etc.

// initialize civil engineering features interface in case it has not been
initialized.:

```
IAcadDocument iacDoc = (IAcadDocument)
iacApp.getActiveDocument();
iacDoc.SendCommand("(cd_mnl) ");
```

// *** Get the AeccApplication interface object:

```
String sAeccApp = "Aecc.Application";
IAeccApplication iaecApp = (IAeccApplication)
iacApp.GetInterfaceObject(sAeccApp);
return iaecApp;
```

```
}
```

```
// Sample method that returns a String array containing Project Names
 // that could be displayed in a ListBox control:
 public String[] GetProjectNames()
 ł
  IAeccProject proj = m_iaecApp.getActiveProject();
  int iCount = m_iaecProjects.getCount();
  String[] sProjects = new String[iCount];
  String sProjName = "";
  Variant v = new Variant();
  for (int i=0;i<iCount;i++)
  ł
   v.putInt(i);
   proj = m_iaecProjects.Item(v);
   sProjects[i] = proj.getName();
  }
  return sProjects;
 }
}
```

Restricted Properties and Methods

When you import a Type Library into your project, you may seem to have access to read-only properties and be able to invoke methods that are not documented in the AutoCAD Land Desktop ActiveX and VBA Reference. These "restricted" members of the interface are hidden in VBA and other macro languages, but are not filtered out by the import directive.

Be aware that these members cannot be called arbitrarily. Most are used for initializing the COM object, and will not update your project database. Refer to the AutoCAD Land Desktop ActiveX and VBA Reference for the list of supported interface members, or use the OLE Viewer supplied with the Microsoft Visual Studio Tools to determine which members are restricted.

Other Programming Guidelines

The ActiveX Object Model opens up programmatic access to AutoCAD Land Desktop Projects to many programming environments other than VB, VBA, C++, J++ and VisualLISP. These include:

- Any Microsoft Office 97 product, including Word, Access, Excel, Project and PowerPoint.
- Any VBScript enabled product, including Outlook and Internet Explorer.
- Any product that supports Automation (ActiveX Dispatch interfaces), including Delphi.

Please refer to the documentation provided with each product.