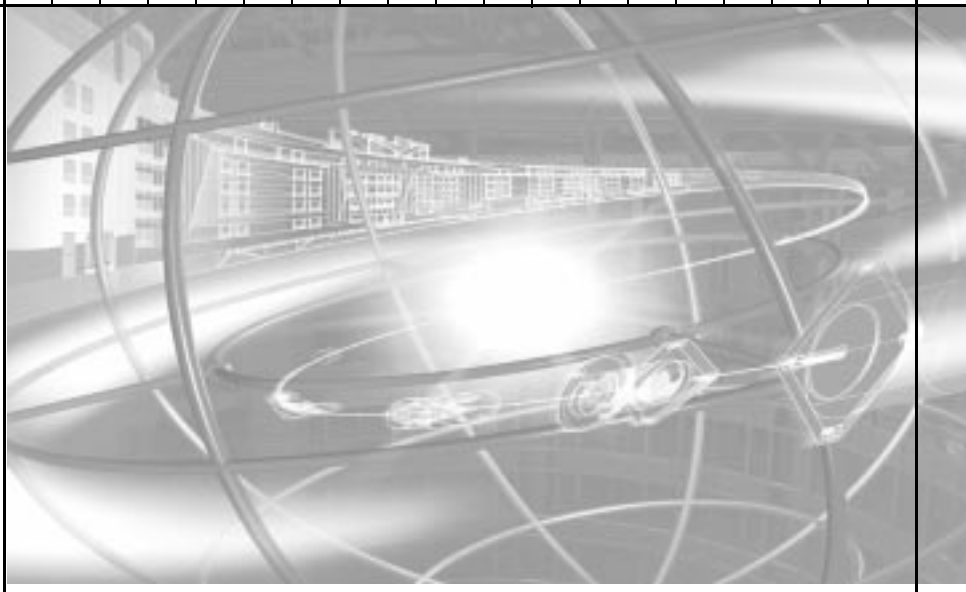
Autodesk.

AutoCAD® 2000



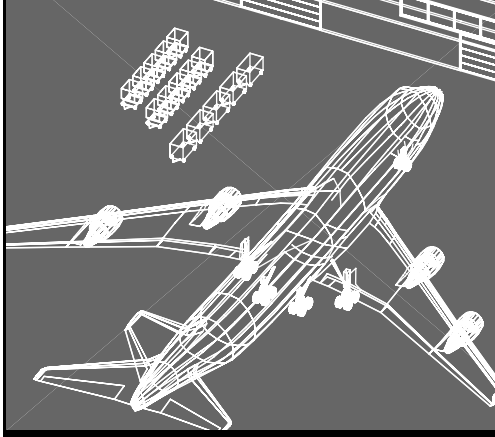
MIGRATION GUIDE FOR APPLICATIONS

00120-010000-5070 January 21, 1999

1 2 3 4 5 6 7 8 9 10

Contents

Introduction



In This Chapter

- Documentation Overview
- Typographical Conventions
- Using This Guide
- Supported Environments

The *Migration Guide for Applications* covers the changes made in the three AutoCAD® programming environments: ObjectARX™, Visual Basic® for Applications (VBA), and Visual LISP®. Each of these environments has a Software Development Kit (SDK), and each SDK's documentation set contains both printed and online guides. This chapter gives a brief overview of these guides, and discusses the organization and conventions of the *Migration Guide for Applications*.

Documentation Overview

There are three groups of SDK documentation, one for each SDK. Their descriptions and organization follow:

ObjectARX documentation set			
Title	**Printed**	**Online**	**Description**
ObjectARX Developer's Guide	X	X	Explains the concepts of developing an ObjectARX application, with example code and step-by-step procedures.
Migration Guide for Applications (this document)	X	X	Provides an overview of the new features and changes in the latest versions of the AutoCAD programming environments, including ObjectARX, VBA, and Visual LISP.
ObjectARX Reference		X	A programmer's reference that provides detailed information on each class and function in the ObjectARX API.

The online help files are in the *\ObjectARX\docs* directory. For last-minute changes or additions to ObjectARX, see the *ReadARX* file located in the *\ObjectARX\docs* directory.

VBA documentation set			
Title	**Printed**	**Online**	**Description**
ActiveX and VBA Developer's Guide	X	X	Explains how to use ActiveX® and VBA to develop an application, with example code and step-by-step procedures.
ActiveX and VBA Reference		X	A programmer's reference that provides detailed information on each object, method, and property available in VBA.

The online help files are in the *\acad\help* directory. For last-minute changes or additions to ActiveX and VBA, see the ActiveX and VBA sections in the AutoCAD *Readme* file located in the *\acad\help* directory.

Visual LISP documentation set			
Title	**Printed**	**Online**	**Description**
Visual LISP Developer's Guide	X	X	An overview of the new Visual LISP environment, with samples and AutoLISP migration information.
AutoLISP Reference		X	The AutoLISP language reference.
Visual LISP Tutorial	X	X	A step-by-step guide to using Visual LISP.

The online documentation is in the *\acad\VLISP\help* directory, and the readme file for last-minute changes is *\acad\VLISP\ReadMe*.

NOTE The online documentation for every SDK may include updates to the printed material.

Where to Start

New users should start by looking through the *Migration Guide for Applications* to get an overview of what is new in each of the SDKs, and what has changed since Release 14. The *Migration Guide for Applications* does not duplicate material that can be found in the various *Developer's Guides* and *Reference Manuals*. Instead, it provides a high-level overview of all the new features and changes, and refers the reader to other documents for more in-depth information. Experienced users, and those upgrading from previous versions of SDK, should start with chapter 2, "Porting Considerations," and then move on to the more detailed material in the *Developer's Guide* for their SDK.

Typographical Conventions

The following typographical conventions are used in this book:

Typographical conventions

Text Element	Examples
Prompts	<First point>/Object/Add/Subtract:
Text you enter	At the Command prompt, enter **shape**
Keys you press on the keyboard	CTRL, F10, ESC, RETURN
Keys you press simultaneously on the keyboard	CTRL+C
File or folder names, names of directories, and instructions after prompt sequences	*teapot.cpp* *c:\dwgunplugged\samples* Select objects: *Use an object selection method*
AutoCAD-named objects, such as layers, linetypes, styles, and AutoCAD commands and system variables	DISC-FRONT, DASHDOT, STANDARD SAVE, COPY, DWGNAME, BLIPMODE
Sample code and text in ASCII files	The variable `PI` is preset to a value of pi `***POP1`
Function names	`command acedCommand()`
Formal arguments specified in function definitions	The `string` and mode arguments

Using This Guide

The following sections explain the organization of the *Migration Guide* and the conventions it uses, to help you use this book more effectively.

Organization

The *Migration Guide* is organized into three main sections. The first section consists of this introduction and the following overview. The second section is chapter 2, which lists the items that you must consider to migrate your application successfully to AutoCAD 2000. The third section is made up of the remaining chapters, and describes the many new features in AutoCAD 2000. The chapters are organized as follows:

- Chapter 3: Multiple Document Interface
- Chapter 4: Application Usability Features
- Chapter 5: Entity Features
- Chapter 6: Display and Hardcopy Features
- Chapter 7: Collaboration Features
- Chapter 8: Architectural Features
- Chapter 9: Extensibility Features
- Chapter 10: API Modernizations

The final section is made up of the appendices that function as quick references for the SDK. There is one appendix for each SDK language, plus an appendix on migrating from DWG Unplugged™ to ObjectDBX™.

Supported Environments

The following environment is supported:

- Windows®-only, 32-bit platforms
- Pentium® P90/32M is the target platform
- TCP/IP for License Manager
- Microsoft® Visual C++® 6.0 compiler

Overview

In This Chapter

1

- Compatibility Issues
- New Features
- ObjectARX Logo Compliance

This chapter gives an overview of the highlights of this release of the AutoCAD SDKs. A general description of the new features is provided, along with a brief summary of the changes made since the previous release. The following chapters cover these topics in more detail.

Compatibility Issues

Compatibility issues with earlier releases of AutoCAD are described here.

Release 14 Compatibility

This section covers changes made in this release that affect compatibility with the previous release of AutoCAD.

ACIS-based entities: 3DSOLID and BODY

When saving these entities to the Release 14 format, data will be lost if you encounter the ACIS® error 3709: "Unable to save variable radius blends in the requested file format." This kind of data cannot be saved for use by any ACIS version prior to 3.0 and can only be created in AutoCAD by ACISIN or by an application.

Release 13 Compatibility

This section describes compatibility issues between this release and AutoCAD Release 13.

ACIS-based entities: 3DSOLID, REGION, and BODY

When saving these ACIS-based entities to the Release 13 format, AutoCAD Release 13c4 is required to be able to read the resulting files. Otherwise, the drawing open function will result in the ACIS error 28006: "Save file is from a later version of ACIS." This is because Release 13c4 had a drawing format change due to the introduction of ACIS 1.6, while Releases 13 through 13c3 used ACIS 1.5. SAVEAS for Releases 13c4 and 14 will save in ACIS 1.6. Pre-Release 13c4 users will not be able to view ACIS data successfully.

For a list of changes that have been made to existing ObjectARX functionality, please see appendix A, "ObjectARX Changes." The corresponding information for VBA is provided in appendix B, "ActiveX Changes," and in appendix C, "Visual LISP Changes" for Visual LISP.

New Features

AutoCAD 2000 contains many new features for the AutoCAD user. In turn, many of these features have new or changed APIs. These are briefly described below, and are organized by type of feature. This organization is also used for the chapters in this document, making it easy for you to refer to the complete information for any particular feature.

Multiple Document Interface

The AutoCAD 2000 feature with the largest effect on developers is the new Multiple Document Interface (MDI). The MDI changes are wide-ranging and require more than a simple recompile. Please see chapter 2, "Checklist for Changes in Your Application" section for highlights of the changes, as well as the information in chapter 3, "Multiple Document Interface."

Application Usability Features

Other new features that your applications can use to benefit your users include AutoSnaps, short cut, or right-click context menus; multiple UCS; and MFC-style dialogs. Solids editing and 3D viewing have been enhanced, and there is now provision for DWG summary information that can be used in an open file dialog. Greater flexibility in specifying the default format of the SaveAs command is available. The APIs for these features and more are described in chapter 4, "Application Usability Features."

Entity Features

The existing APIs for AutoCAD 2000 multiline text have been redone to implement new capabilities. There is also a new Object Property Manager, with its own API. These are described in chapter 5, "Entity Features."

Display and Hardcopy Features

AutoCAD 2000 has a brand-new plotting interface, plus additions to viewports and paperspace. Lineweights now can be plotted and displayed at their true widths, and OLE objects can be scaled more easily. The APIs for these features are found in chapter 6, "Display and Hardcopy Features."

Collaboration Features

Sharing information between multiple drawings is now easier than ever, using the APIs in chapter 7, "Collaboration Features." For example, AutoCAD 2000 now offers in-place editing for xrefs. The large API for this feature involves selection, filtering, transactions, and locking. Working across the Internet has been enhanced with the new Internet features.

Architectural Features

The material in chapter 8, "Architectural Features," covers the new or changed libraries for ObjectDBX, AcGi, and ACIS. Also described are the API changes due to the objectification of Release 12 and earlier entities. New opcodes in DXF™ and the effect of the new long symbol names are found here too.

Extensibility Features

Along with the changes in the features of AutoCAD 2000, two of the AutoCAD SDKs have seen substantial changes as well. AutoLISP® has been improved with Visual LISP, and ActiveX/VBA now has greater capability than ever before. Accessing information stored in a relational database has been improved with the Database Connectivity feature, a new implementation in VBA of the earlier ASI and ASE C++ and AutoLISP libraries. Meanwhile, the formerly separate ADS interface has been bundled into ObjectARX. For more information on these changes, see chapter 9, "Extensibility Features."

API Modernizations

Several changes to the SDKs were not part of a specific feature. These changes are grouped in chapter 10, "API Modernizations."

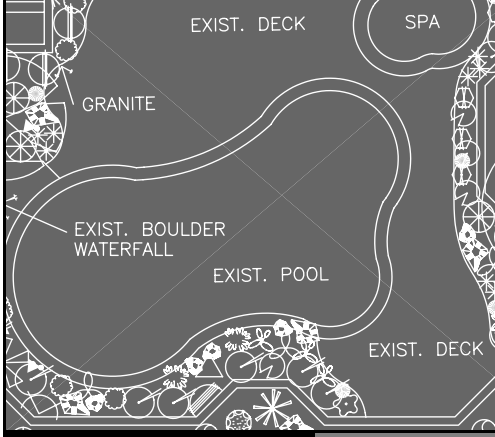
ObjectARX Logo Compliance

One of the biggest new features of ObjectARX doesn't have any code at all. Autodesk® now offers a "Built with ObjectARX" logo program for AutoCAD applications that use ObjectARX. If you are creating AutoCAD 2000-related products based on ObjectARX technology, you should look into this program. Developers with products that meet the "Built with ObjectARX" test criteria will be eligible to license and use ObjectARX branding on product packaging, collateral items, and Web sites, and to participate with Autodesk

in related marketing initiatives. To find out more about this logo program, go online to *http://www.veritest.com/autodesk/main(f).htm* and follow the process listed there.

Porting Considerations

This chapter describes the issues you must address so that your existing applications continue to work correctly. The ObjectARX features that require more than a simple recompile are listed individually, with a description of the actions you need to take.



In This Chapter

2

Checklist for Changes in Your Application

This is a list of changes that you need to address in order to make your applications work with AutoCAD 2000. More than a simple recompile will be required for almost every application.

- Change the library names that your application links with.
- Check for renamed, removed, or deprecated classes, methods, and functions.
- Check for fixed-length buffers being used with extended symbol names.
- Change `ACRX_DXF_DEFINE_MEMBERS` macro calls to include new arguments.
- Make the necessary changes for MDI support.

These points are described in further detail below.

Library Changes

There are new libraries in this release, and some existing libraries have been renamed to reflect their version:

Library file names

Release 14 Library Name	AutoCAD 2000 Library Name
acfirst.dll	*ac1st15.dll*
asertvc4.lib	*aseapi15.lib*
asiatvc4.lib	*asiapi15.lib*
ism.lib	*acISMobj15.lib*
libacge.lib	*acge15.lib*
libacgex.lib	*acgex15.lib*
libacbr.lib	*acbr15.lib*
Not present	*acdb15.lib*
Not present	*achapi15.lib*
Not present	*acrx15.lib*
Not present	*acutil15.lib*

Library file names (*continued*)	
Release 14 Library Name	**AutoCAD 2000 Library Name**
Not present	*axauto15.lib*
Not present	*oleaprot.lib*

Renamed Items

You should rename functions with changed names: for instance, all the ADS functions. This is not required because the old names have been defined in *migrtion.h*, but it is something that you will need to do eventually because the legacy names will not be available in the next release. See chapter 9, "ADS" section, and appendix A, "Changed Items" section, for a list of changes.

Removed Items

The global function `acdbCurDwg()` is no longer an exported API. Instead, your application should use the `workingDatabase()` method of the class `AcDbHostApplicationServices`, found in *dbapserv.h*, and it must link to the *AcUtil15.lib* library.

For a comprehensive list, see appendix A, "Removed Items" section.

Deprecated Items

The `kOleUnloadAppMsg`, formerly used by applications with ActiveX (OLE) Automation, is being deprecated. Applications that use this message are usually also COM servers, and as such should rely on reference counts to determine when to unload. Supporting this message under these circumstances requires that the application hardcode the name of the COM server to be able to identify it, defeating one of the purposes of COM. The message is still being maintained for backward compatibility.

To illustrate, suppose that AutoCAD will be using a custom object. AutoCAD checks the Registry and loads up the custom object's DLL to handle the request for the custom object interface. The DLL loads the corresponding ARX program to service the requests and then passes back an interface to AutoCAD. Later on, AutoCAD sends the ARX application a `kOleUnloadAppMsg`. The ARX application has no idea whether or not it can be unloaded, because the DLL is keeping track of the interfaces and knows if the ARX application is being used by others. So, the ARX application tries to ask the DLL if it's OK to unload, but to do that, the ARX application needs to

know (have hard-coded) the name of the DLL. The "knowing the name" part is not COM-like, and is the main problem in using the message. Instead, AutoCAD should not issue the message, but rather should rely on the DLL to keep track of the ARX application's use, and to release it at the appropriate time.

Extended Symbol Names

The extended symbol names feature increases the previous maximum length of symbol names from 32 characters to 255, allowing you to use more descriptive names throughout your application. You should examine your application with these points in mind:

- Check for fixed-length buffers that could overflow with extended names.
- Avoid any design or coding decisions based on any of the criteria that differentiate extended and legacy symbol names. These criteria are name length, allowable characters, character case, and character set.
- Note that all objects derived from `AcDbSymbolTableRecord` and `AcDbDictionary` may now contain extended symbol names.

See chapter 8, "Extended Symbol Names" section, for more information.

Class Versioning

AutoCAD 2000 adds support for class versioning, and you must change your calls to `ACRX_DXF_DEFINE_MEMBERS` for all custom classes. See chapter 10, "Class Versioning" section, and also the *ObjectARX Developer's Guide* for additional information.

Multiple Document Interface

The list below is the minimum that you must do to have your ObjectARX application successfully operate under the Multiple Document Interface (MDI) in AutoCAD 2000. Though these changes will bring your application to only the lowest level of MDI compliance, this is still a good way to start updating your application. At this level your application still will not run when the MDI mode is active, but it should work without failing under the Single Document Interface (SDI) used by earlier releases of AutoCAD:

- You *must* recompile and relink your application. (This is true of all ObjectARX applications.)
- Check for static and global variables. These will need to be encapsulated in order to work with MDI.

- If you use `AcEditorReactor::sysVarWillChange()` or `AcEditorReactor::sysVarChanged()`, you must add the new `AcApDocument*` parameters supplied in order to recompile successfully.
- Ensure that your application can process an `AcRx::kUnloadAppMsg` immediately following a return from processing the `AcRx::kInitAppMsg`. This is the result if your application doesn't register as being MDI-Aware, and AutoCAD 2000 already has multiple documents running and cannot change to SDI mode.
- Don't register as being MDI-Aware in your `acrxEntryPoint()` function until you have completed your changes.

See chapter 3, "Multiple Document Interface," for more information.

Additional Items

The following items are not required to port your application to AutoCAD 2000. You should still consider making these changes to ensure that your application works even more smoothly with AutoCAD.

Split Your Application

Separating your application into database and user interface components is now possible with the advent of DBX files. This procedure allows you to create files that handle custom objects separately from the user interface portion of your application. The separate database portion can be used by other applications that need access to your object, without duplication of code. Please see the *ObjectARX Developer's Guide* for a complete description of this feature.

Migration Assistance

Some definitions have been provided to help you migrate your application from Release 14 to AutoCAD 2000. The definitions are provided in a new header file, *migrtion.h*. The first definition allows you to use the former name of `acedSetCurrentVPort()`, which was renamed for AutoCAD 2000:

```
#define acdbSetCurrentVPort acedSetCurrentVPort
```

The next definition requires that you include *dbapserv.h* and add *acutil15.lib* to the LINK line in your makefile:

```
#define acdbCurDwg acdbHostApplicationServices()->workingDatabase
```

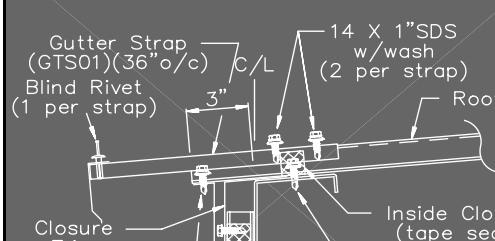
The *migrtion.h* file will not be available for releases subsequent to AutoCAD 2000. At some point, you will need to make global substitutions in your code.

File Compatibility

Changes in the ACIS solids modeling library and in AutoCAD 2000 itself have caused corresponding changes in the DXF and DWG formats. See chapter 8, “DXF” section, for more information.

Multiple Document Interface

AutoCAD now supports a Multiple Document Interface (MDI) that allows you to have more than one drawing loaded at once in a single AutoCAD session. The MDI has caused changes throughout the ObjectARX APIs. This section gives an overview of what you must do to achieve MDI compatibility.



In This Chapter

3

Overview

You should modify your application to take advantage of the Multiple Document Interface (MDI). However, doing so will require you to make many changes in your application. Your application will not be able to keep any data pointing to document resources in global or static defined variables. This includes `AcDbDatabase*`, `AcDbObjectId`, `AcDbObject*`, header variable values, document variable values, selection sets, or other document-specific information. Explicit document locking is only required when executing from the application context. Only legacy applications using modeless dialogs or responding to Windows messages or callbacks should need to implement explicit locking. You may find it desirable to use a lock mode for a registered command other than the default of Shared Write. For complete details, see the MDI chapter in the *ObjectARX Developer's Guide*.

Terminology

Some commonly used MDI terms need to be defined:

Active Document	The document that has window focus, and receives the next user input event, unless the user is switching to (activating) another document. The active document is always the current document when processing user input, but programs can temporarily change the current document for some operations.
Application	The term for the overall running program and associated objects that are independent of, or common to, all open documents, such as the MFC class `CwinApp`. There is one and only one application per invocation of an executable Windows program.
Application Context	This is a shorthand term for "application execution context." See the "Execution Context, Application" definition.
Command	Throughout this chapter, instead of your basic AutoCAD command, "command" is usually meant to represent the abstract concept of a program sequence performed as a logical unit of work that can be requested by the user or one of the AutoCAD scripting engines. No matter what construct is used, a command can be undone independently of other actions performed during system

operation. In exact terms regarding the MDI API, a command is a sequence of code that begins by locking a document and ends by unlocking a document. In common cases, this locking and unlocking will be performed by the system, but during other times the application must do the locking and unlocking directly. The logical concept of *command* is shown by any one of the following constructs in AutoCAD:

- An AutoCAD built-in command.
- Built-in commands executed directly from the command processor, such as F2 for change screen. This includes function and control keys, with a Windows or AutoCAD Classic preferences switch determining which keys do what.
- A Visual LISP function invocation, which can be defined either in Visual LISP or in an ObjectARX application using `acedDefun()`.
- External program commands defined in *acad.pgp*. These commands only interact with AutoCAD (and thus require document locking/unlocking) if the return code is specified as 1, 2, or 3, causing an input operation to occur after execution.
- An AcEd-registered command registered from AutoCAD.
- An action taken from a modeless dialog window or some other external process, typically hosted by an ObjectARX application.
- A set of actions taken from an ActiveX application in an external process.
- An action taken from VBA through the ActiveX interface. It is distinguished from ActiveX in an internal process in that VBA is built into AutoCAD, and thus can be monitored at a level above individual ActiveX Automation requests.
- A right-click context menu invocation.

Command, Multi-Document

A set of commands that appears as one to the user, where the user is allowed to change the current document, and a continued logical flow of user prompting is maintained. In this way, an active command can be prompting for user input in the current document, but when (or if) the user switches the document, the application invokes `AcApDocManager::SendStringToExecute()` to cancel the

	command in the old current document, and queue up a command to commence execution in the new current document.
Command, Non-reentrant	A command that cannot be executed in more than one document at a time. Non-reentrancy can be done for commands that for whatever reason should not be available for more than one document at a time, or where the demands of supporting multiple instantiation are too great to be worth the overhead.
Command Processor	The standard input message polling mechanism in AutoCAD that facilitates combined keyboard and digitizer interaction. A separate command processor exists for each open document. The state of the command processor is maintained as an execution context. **NOTE** Commands that execute outside the context of a single document, such as modeless dialogs and toolbars posted by AutoCAD, or ObjectARX applications, execute from within the application context.
Current Document	Programmatic requests can be made to cause a document's execution context to become active without the user actually perceiving the document as "activated." This is a transient state only, used primarily by ActiveX and VBA applications.
Database	An actual AutoCAD drawing, specifically an instance of `AcDbDatabase`. While part of a document, it is not synonymous with a document.
Document	A document consists of an MDI document window, an execution context, an associated editor state, and a single current database, plus any number of side databases that are opened in association with it. The current database is the one being displayed and edited via commands. The side databases will be further distinguished by being either used by xref or for general use. The document also includes system variables that are associated with a given drawing such as current viewport, and so on. Documents are uniquely identified by their address, such as the `AcApDocument*` pointer.
Drawing	Synonymous with Database.

Edit Session	Usually synonymous with Document, but sometimes includes its entire history from open to the present, as well as the current state of the session.
Execution Context, Application	The command state that is active when new Windows messages are pending. It is independent from all document execution contexts. The following types of commands execute from this context: external ActiveX Automation requests (such as Visual Basic), VBA, and modeless dialog boxes. These types of commands will typically work on the active document, although they will not necessarily be bound to do so. The intent is to handle document locking and unlocking reasonably transparently for external ActiveX applications and VBA. However, ObjectARX applications posting modeless dialogs will be required to lock and unlock documents explicitly in order to interact with their databases.
MDI-Aware	ObjectARX applications (and ActiveX and COM applications, but not necessarily Visual LISP applications) that meet all criteria needed to be executed successfully in an MDI AutoCAD. These criteria are listed in the "MDI-Aware" section of this chapter. ARX applications can register themselves as MDI-Aware by calling `acrxDynamicLinker->registerAppMDIAware(pkt);` when receiving the `kInitAppMsg` within their `acrxEntryPoint()` function.
Per-Application	A data structure that needs to exist only once per application.
Per-Context	A data structure that needs to be instantiated and maintained for each execution context, which includes document execution contexts and the application execution context. The AutoCAD command processor is an example of a per-context instantiation. Per-document storage usually relates to extended model state, while per-context storage tends to relate to extended control state.
Per-Document	Any data structure, value, or other item that needs to be instantiated and maintained for each document.
Quiescent	When the command processor in a given edit session has no active AutoCAD command, no active ObjectARX commands, Visual LISP evaluations, ActiveX requests being serviced, AutoCAD menu macros being processed,

	or VBA macros being run. At this point, the Command prompt is being displayed in the command window. Notice that modeless dialogs and toolbars can be posted, as they do not operate through the command processor.
Session	Synonymous with Application.
Undo Stack	A repository of state recorded during an edit session, which is used to undo the effects of the edit session, command by command, when requested. Databases are often (in AutoCAD's case, usually) associated with an undo stack, which is supplied by the host application. In the case of AutoCAD, databases are typically opened under only one document at a time, because undo stacks correspond to documents.

Changes for Editing with MDI

Three principal changes have been made to AutoCAD to support editing multiple documents. These changes involve

- Execution contexts

  The application execution context is distinct from the document execution contexts. It is the application execution context that processes the Windows message loop.

- Data instances

  There is now a separate instance of all data elements that have to do with the database and current command processing state for each document. This includes the command processor, input processor, Visual LISP environment, databases, selection sets, and (most, but not all) system variables.

- Document locking

  All modifying interactions with documents and their contents, including databases, must be conducted while the documents are appropriately locked.

You will also need to consider changes in several other areas, including

- Database undo and transaction management facilities
- Document-independent databases

- Non-reentrant commands
- Multiple document commands
- Selection sets

For more information, refer to the MDI chapter in the *ObjectARX Developer's Guide*.

SDI System Variable

A compatibility mode for the single drawing interface (SDI) of previous releases is provided, and is controlled by the SDI system variable. Possible values for SDI are shown in the following table:

SDI system variable values

Value	Meaning
0	MDI is enabled
1	SDI mode set by user
2	SDI mode implied by loaded non–MDI-Aware applications
3	SDI mode implied both by user and by loaded non–MDI-Aware applications

Be aware of the following restrictions on changing the value of SDI:

- SDI can only be set to values of 2 or 3 by AutoCAD, as applications are loaded and unloaded.
- Always check the current value of SDI before making changes. If the current value is 2 or 3, do not change it.
- Changing SDI to 1 from 0 will fail when AutoCAD has more than one document open.

All document lock checking is disabled when AutoCAD is running in any of the SDI modes.

NOTE This compatibility mode and the SDI variable will be removed in the next full release of AutoCAD.

Levels of Compatibility

Your application may have one of four levels of compatibility with MDI:

- SDI-Only
- MDI-Aware
- MDI-Capable
- MDI-Enhanced

SDI-Only is the minimum requirement, but MDI-Capable is what most applications should have as their compatibility goal.

The requirements for getting existing applications to work reliably vary according to the level of MDI compatibility desired, and fulfilling the requirements won't be easy in many cases. As a developer, your main concern should be any code written under the assumption that only one current database and associated editor state exist at a given moment, and that programs always run in a single thread and single execution context. Those assumptions are no longer valid under MDI.

MDI will support an execution context per document and provides a facility for allowing a single execution context to be active when switching documents. While the requisite code changes will be simple, they will also be pervasive throughout most applications.

SDI-Only

This is the basic level of compatibility and is not sufficient for most robust applications. This level will not allow your application to run under MDI, but it should work without failing under SDI:

- You *must* recompile and relink your application. (This is true of all ObjectARX applications.)
- If you use `AcEditorReactor::sysVarWillChange()` or `AcEditorReactor::sysVarChanged()`, you must add the new `AcApDocument*` parameters supplied in order to recompile successfully.
- Register your application as SDI-Only by calling `AcRxDynamicLinker::registerAppNotMDIAware()` in the `kInitAppMsg` handler.
- Ensure that your application can take an `AcRx::kUnloadAppMsg` immediately following a return from processing the `AcRx::kInitAppMsg`. This will occur if your application doesn't register as being MDI-Aware, and AutoCAD already has multiple documents open.

MDI-Aware

This level is the first step in making your application MDI-compatible, and involves much more work than the subsequent MDI-Capable and MDI-Enhanced levels. These are the things you must do before legitimately being able to set the SDI system variable to 0 and not have your application (or others) fail:

- Examine every use of your static and global memory state. Any occurrence of such state that might be corrupted if your application is utilized in multiple concurrent edit sessions should be recoded to reside in heap-resident structures that can be allocated and freed as documents are initiated and closed. Some specific examples are given below.

  All such state used by code sections of your application that can be invoked in the context of any ObjectARX callback other than AcEd-registered commands should not rely on serialized edit sessions. This includes all functions defined with `acedDefun()` as well. The unit of execution for all operations is all processing done between prompts for live user input. Between such prompts, the code for a single session runs completely between polling for user input, so static data whose persistence never spans user input prompts should be OK, though the use of such static data is not recommended.

  All function codes defined with `acedDefun()` should be upgraded to run with multiple Visual LISP states active all at once. Each document has its own complete Visual LISP environment. Your functions must maintain per-document state explicitly, like any other ObjectARX application.

  You have a choice in the case of AcEd-registered command-specific cases, and only such cases. You can use a static variable to keep track of when a specific command is currently in progress, and refuse to perform it in other edit sessions, preferably with some appropriate user interface. See the next section, "MDI-Capable," for the ideal level of support (full command reentry).

- At the beginning of every AcEd-registered command, functions defined with `acedDefun()`, `AcEditorReactor` callbacks, `AcRxDynamicLinkerReactor` callbacks, or Windows dialog event handler callbacks that need to work with the current drawing or document must be queried using `AcApDocManager->curDocument()->database()`. The current document cannot be changed for your execution context afterwards, although every prompt for user input from ObjectARX functions and members can result in a document switch. This should exclude all callbacks related to AcDb, which are object-specific or database-specific and thus document-specific.

- Menus are not specific to documents, and changing menus as part of a command is more strongly discouraged now even than before. Doing so violates Windows logo compliance, and now changing menus affects the entire application, not just the document being edited. Still, it is possible to do, and menu macro expansions already in progress will not be affected.
- The `kLoadDwgMsg` and `kUnloadDwgMsg` `acrxEntryPoint()` messages are your "per-document" setup and take-down callbacks, but now they can occur in "mid-edit session" contexts for other open documents when AutoCAD is running in MDI mode. This behavior is contrary to the "between all edit sessions" condition they imply when AutoCAD is running in SDI mode. This behavior may or may not substantially affect existing code that is executed at the beginning and end of "all existing" drawing edit sessions. For example, if you code one of these message responses to delete and reallocate something, you may want to separate the delete code to the `kUnloadDwgMsg` message and allocation code to `kLoadDwgMsg`. Any code in your application currently invoked when the `kLoadDwgMsg` and `kUnloadDwgMsg` messages are sent to your `acrxEntryPoint()` function demands careful consideration as to whether it works with overlapping and multiple edit sessions, and whether the code is per-document or global for the application. For example, AcEd-registered commands are global, so therefore you don't want to disable them in a `kUnloadDwgMsg`. Instead, you should wait for `kUnloadAppMsg` to unregister AcEd-registered commands and other global constructs as described in this document. However, `acedDefun()` will apply on a per-document basis, so you *do* want to register such commands in all of the `kLoadDwgMsg` callbacks made to your application.
- Regarding `acrxEntryPoint()` messages, `AcRx::kQuitMsg` will now be sent after all documents and their databases are closed when the SDI system variable is non-zero; however, they will continue to be sent before the document and current database is destroyed when SDI is 0, to preserve application compatibility. `AcRx::kSaveMsg` will be sent out for each document saved during a QUIT operation, while the document and database are still viable. In the case of the END command, `AcRx::kEndMsg` will be sent out once for each drawing open, in lieu of `AcRx::kSaveMsg`.
- All code that can be invoked, directly or indirectly, by means listed in the "Application Execution Context" section must abide by the rules and limitations described in that section. This includes all code that can be invoked via direct document and database manipulation, except code that can only be invoked through an AutoCAD command or Visual LISP.
- The Visual LISP memory model is per-document, while ActiveX Automation and ObjectARX memory models are per-application. Usually it does not matter which context an application is running from, as long as the application knows which documents' contents it is operating on.

Here are a few implications for ObjectARX and ActiveX automation applications pertaining to Visual LISP:

All calls to `acedDefun()` should still be made from every `kLoadDwgMsg` call you get, because functions defined that way are only made known to the context of the current document. This also applies to `acedUndef()`.

Visual LISP can be invoked from a document application context but not the application execution context. It needs a command processor to operate.

The current document should not be changed in an ObjectARX or ActiveX invocation from Visual LISP, that is, while Visual LISP is active.

- `AcEditorReactor` callbacks demand explicit definition with regard to multiple documents. Methods that are tracking per-drawing or per-document information will need to query the affected document. If a database is passed into the method, it can be mapped to a document. Other methods should only be able to be invoked from the current document, which can be queried from the reactor, and the `sysVarWillChange()` and `sysVarChanged()` methods have had new parameters added to indicate the affected document.
- If your application polls for input to detect user cancellation in long or computation-intensive operations, you must do one of two things: either redo the code involved in the long operation to make it reentrant or disable document switching.
- Any code you want to have work with ObjectDBX should not use any of the MDI, specifically, any constructs that begin with ACAP (case insensitive). These are not part of the ObjectDBX repertoire.
- Using `acedCommand()` (formerly `ads_command()`) to call either _NEW or _OPEN will only work in SDI mode. Applications ported to MDI must use the new `AcApDocumentManager` methods `openDocument()` and `newDocument()` for open and new operations, respectively.
- Register your application as MDI-Aware, or else it will only be usable when AutoCAD is running in SDI mode, no matter what work you have done. However, do *not* register your application as MDI-Aware until you have inspected and modified it according to the guidelines given in this section.
- Remember that you also still want to support SDI mode. SDI mode can be established by the user and by the presence of non–MDI-Aware applications, even those other than your own. You must continue to support SDI mode. In SDI mode, document locking still performs UNDO and notification support, but lock conflicts are ignored.

MDI-Capable

Reaching this level of compliance involves making your code work as efficiently and naturally in MDI mode as it does (or did) in SDI mode. Only a couple of items are explicitly described in this section, but they can both be time consuming:

- Code sections comprising AcEd-registered commands that are invoked directly from those constructs will likely pause for user input, and are more likely susceptible to corruption when multiple sessions are being done. The most common case is to enter a command in one document, prompt for user input, then switch documents and enter the same command in a different document. As long as you are maintaining vital information on a per-open-document basis, your application should function properly. Otherwise, your code should disable document switching.
- When it's time to look at performance, heap-resident pointer dereferencing to what were formerly static addresses can bog down program performance. The alternative here is to maintain a static memory buffer of elements of the current document, which would be scanned in from and written out to the document-specific heap elements.

MDI-Enhanced

These additional steps will make your application integrate seamlessly with the MDI:

- Consider migrating your `AcEditorReactor::commandXxx()` and `AcEditorReactor::LispXxx()` callbacks to work instead from `AcApDocManagerReactor::documentLockModeWillChange()` and `AcApDocManagerReactor::documentLockModeChanged()` notifications. Then they will account for application execution context operations, which have been previously difficult to detect by ObjectARX applications.
- Implement all the features described in Document-Independent Databases, Multi-Document Commands, and Application Execution Context Usage sections.

Application Usability Features

In This Chapter

4

- Input Point Processing
- 3D Graphic System
- Solids Editing
- DWG Summary Information
- MFC Dialogs
- SaveAs
- AutoCAD DesignCenter
- Multiple UCS per Viewport
- Shortcut Menus

This release of the AutoCAD SDKs provides many new features to enhance the usability of your applications. This chapter gives an overview of these features.

Input Point Processing

Custom input and snap points and alignment settings are accessible through Autodesk APIs. In Release 14 developers could create their own object snap settings, but could not access them through the AutoSnap feature. This release allows developers to use the Input Point facility and pass a point back to AutoCAD along with a glyph and ToolTip that AutoCAD displays like other Osnap points. The Alignment behavior applies to custom object snaps in addition to the built-in Osnap settings.

See the *ObjectARX Developer's Guide* for a complete discussion of the new input point processing facility, in the "Input Point Processing" section.

3D Graphic System

AutoCAD's 3D Graphic System is a retained-mode API through which 3D and 2D models can be displayed. It creates and manages a high-performance 3D display cache and a live connection to the database. The Graphic System comprises a set of C++ classes that are provided in a dynamic link library. While the intended primary client for this component is AutoCAD, and although the Graphic System makes use of a few abstract interface classes defined by AutoCAD, it is a separate and fully independent component that could very well be used by other applications.

The Graphic System maintains bi-directional links to the databases to which it is connected. This allows it to manage complex tasks such as synchronizing a graphics cache with model edits. Since it is based on a polygonal renderer, it also uses these links to manage a tessellation cache for nonpolygonal objects such as curves, smooth surfaces, and solids. Other important capabilities the Graphic System transparently provides, or offers with minimal involvement of the client application are

- Database connectivity
- Output device connectivity
- Views (cameras and viewports)
- A full set of geometric primitives well adapted to modern graphics hardware
- Specification of attributes, transformations, selection IDs, and clipping regions
- Support for transient, nonentity graphics such as icons, grids, and cursors
- Support for editing, highlighting, dragging, and screen repair
- Support for selection (entity, subentity, face, edge, vertex)

- Facilitation of graphics caching
- Maintenance of a specified frame rate during interactive manipulations
- 3D hardware support
- Multithreading in order to take full advantage of multiple CPUs
- Sorting for attribute coherence
- Dynamic tessellation level of detail management

The interfaces the Graphic System provides are split into several portions, which are discussed below.

Geometry and Attributes

The AutoCAD Graphics Interface (AcGi) is a set of classes through which geometric primitives, attributes, transformations, and clipping regions can be specified. It is an interface familiar to thousands of AutoCAD third-party developers, and it comprises the interface by which AcDb and custom objects describe themselves to AutoCAD. This API is the mechanism by which a database describes its geometry to the Graphic System. For details, please consult the "AcGi Library" section in chapter 8 in this book and the *ObjectARX Reference*.

AcGiDrawable Class

All displayable objects are derived from `AcGiDrawable` (which is referred to throughout as a "drawable"). This class implements the protocol by which objects can be queried by the Graphic System for their geometry, attributes, cache pointers, and database IDs.

A drawable need not have displayable output. Some drawables only affect attributes, transformations, or clipping regions; others contain other drawables and have no geometry or attributes of their own. AutoCAD's `AcDbLayerTableRecord` and `AcDbBlockTableRecord` classes are examples of such drawables.

A drawable need not be an AcDb database object. The UCS Icon and the Grid are two examples of nonpersistent drawables. These are visual elements of the AutoCAD editor that do not belong to an AcDb database.

The AcGiDrawable class definition can be found in *drawable.h*. The class methods are described in the *ObjectARX Reference*.

Graphic System Meta-API

Several classes make up a meta-API for the 3D Graphic System. These are described in the following sections.

AcGsDevice Class

`AcGsDevice` is an abstraction of a drawing surface and encapsulates the underlying device to which graphics are rendered. At present, an `AcGsDevice` corresponds to a GUI display window. Eventually, other classes of devices will be supported (such as hardcopy, file (*.bmp*, *.wmf*, *.dwf*), and clipboard).

An `AcGsDevice` owns one or more `AcGsView` objects and must manage damaged areas and delegate updates to these views. In addition, it responds to operating system notifications such as a change in window size forwarded from the underlying device by the Graphic System client. The class definition can be found in *gs.h* and its description is in the *ObjectARX Reference*.

AcGsView Class

`AcGsView` corresponds to a portion of a GUI window along with a set of camera parameters that define a view. More specifically, an `AcGsView` can be used to represent an AutoCAD viewport.

As described above, an `AcGsView` is added to an `AcGsDevice`. A view may belong to only a single device at a time. Views need to be told what they are viewing. The client adds and removes drawable, model pairs corresponding to the viewed piece of the specified model.

Each `AcGsView` has a rendering mode that corresponds to the algorithm employed when drawing geometry in that view, and a projection mode for the type of projection employed. The `AcGsView` class defines an enumeration for each of these modes. `RenderMode` specifies the rendering algorithm to use, and `Projection` specifies the type of projection used in the view. The class definition can be found in *gs.h*.

Viewport size and position in normalized device coordinates and screen coordinates are specified with `setViewport()` calls. You must specify valid input (for instance, the lower-left corner should not be above or to the right of the upper-right corner). Viewports must also have positive width and height.

In addition to views, the `AcGsView` class also provides methods for dealing with the following types of functionality:

- Clip planes
- Matrix use
- Render mode
- Drawables
- Validation
- Updates
- Visibility
- Viewport visibility of layers

- Selection
- Highlighting

Refer to the *ObjectARX Reference* for a detailed description of the class.

AcGsModel Class

The `AcGsModel` is an abstraction of a database. This database may be an `AcDbDatabase`, it may be a container for transient graphics (such as the Icon and Grid), or it may represent another persistent database format such as DWF. Most significantly, the `AcGsModel` provides a mechanism for the database to inform the Graphic System of changes (additions, deletions, and modifications).

Each `AcGsModel` has a `RenderType` enumeration, which provides a hint to the Graphic System about how this model's geometry should be rendered:

RenderType

Name	Value	Usage
kMain	0	Use main Z-buffer
kSprite	1	Use alternate Z-buffer, for sprites
kDirect	2	Render on device directly
kCount	3	Count of RenderTypes

The class description is in the *ObjectARX Reference* and its definition can be found in *gs.h*.

AcGsConfig Class

This class contains one enumerated type, `Handedness`, and methods for specifying information about the graphics driver.

Graphic System Initialization and Shutdown

The Graphic System may be dynamically loaded through the `CreateClassFactory` entry point. This method is of the type `AcGsCreateClassFactoryFunc`. When called, it creates an `AcGsClassFactory` that can be used to create the rest of the Graphic System Meta-API objects.

When the client is finished with the Graphic System, the class factory should be destroyed by the dynamically bound `DeleteClassFactory` entry point, which is of type `AcGsDeleteClassFactoryFunc`.

The Graphic System Meta-API objects hide behind a pure virtual interface; they are constructed via the AcGsClassFactory.

AcGsManager Class

The AcGsManager interface provides services that help to hide the details of connecting AutoCAD and ObjectARX applications to the Graphic System.

Each AutoCAD AcDisplay object has an AcGsManager. (See GetGSManager() below for access.) Thus, AcGsManager objects are 1-to-1 with MDI Client Windows and 1-to-1 with AcDbDatabase objects. The AcGsManager class provides methods for creating and destroying views, and for retrieving the Graphic System class factory object, the correct AcGsModel, and the current AcGsDevice. See the *ObjectARX Reference* for descriptions of these methods. The class definition is in *AcGsManager.h*.

Utility Methods

The following Graphic System–related methods can be found in *acgs.h*:

```
AcGsManager*
acgsGetGsManager(
    CView* pView = NULL);
```

Pass in NULL to get the AcGsManager associated with the current MDI Client Window; otherwise, pass in an AutoCAD MDI Client CView to retrieve the AcGsManager associated with that MDI Client window.

```
Adesk::Boolean
acgsSetViewParameters(
    int viewportNumber,
    const AcGsView* pView);

Adesk::Boolean
acgsGetViewParameters(
    int viewportNumber,
    AcGsView* pView);
```

Set and get the view parameters for the indicated viewport.

```
Adesk::Boolean
acgsSetLensLength(
    int viewportNumber,
    const double& lensLength);

Adesk::Boolean
acgsGetLensLength(
    int viewportNumber,
    double& lensLength);
```

Set and get the lens length for the indicated viewport.

```
AcGsView*
acgsGetGsView(
    int viewportNumber,
    bool bCreateIfNone);
```

Retrieve the persistent `AcGsView` associated with a given viewport. This method will return `NULL` if there is no such view unless the `bCreateIfNone` flag is set to true; in this case, an `AcGsView` will be created and linked to the specified viewport.

Solids Editing

New editing capabilities for solids are now available in AutoCAD. These new capabilities allow users to apply the following operations efficiently to solids of varying complexity:

- extrude, move, rotate, taper, offset, or delete faces
- copy a face as a REGION or BODY entity
- copy an edge as a LINE, ARC, CIRCLE, ELLIPSE, or SPLINE entity
- change the color of a face or edge
- imprint geometry to create new faces
- apply shelling or hollowing to create thin walls
- separate discrete volumes into independent solids

These capabilities are implemented by the `AcDb3dSolid` and `AcEdSolidSubentitySelector` classes. For a complete description of both classes, see the *ObjectARX Reference*.

AcDb3dSolid Class

`AcDb3dSolid` is an existing `AcDbEntity`-derived class used to represent ACIS solids in AutoCAD, and has been extended to support the solids editing operations added by the Solids Editing feature. These new methods are used by the SOLIDEDIT command. Many of the methods are unavailable from the ObjectDBX SDK; this is noted in the individual method descriptions. The class is defined in the *dbsol3d.h* header file.

AcEdSolidSubentitySelector Class

The `AcEdSolidSubentitySelector` class provides an interface for the selection of face subentities of a 3D solid object in the current space of the current document. It is defined in the header file *acedsel.h*.

DWG Summary Information

The Drawing Property dialog allows AutoCAD users to embed ancillary data (called Summary Information) in their DWG files, and assists in retrieving DWG files based on this data. With this feature, the DWG file format becomes more than just a repository for drawing data, and provides AutoCAD users with base-level file retrieval and management capabilities.

Through Windows Explorer, the properties of a drawing can be viewed outside AutoCAD. This adds additional value to the DWG file format, extending its usability across the enterprise. Used in conjunction with the AutoCAD DesignCenter Advanced Find feature, Summary Information delivers additional value to the customer when searching or cataloging drawings. DesignCenter lets the customer search for drawings by Title, Subject, Author, and Keywords values.

API Changes

The Summary Information feature has created three new classes and three new global functions, which are defined in *suminfo.h*. There is also a new method in the `AcDbDatabase` class. For complete information on these new items, see the *ObjectARX Reference*.

AcDbDatabaseSummaryInfo Class

The `AcDbDatabaseSummaryInfo` class encapsulates a set of character strings that can be used to add additional information to a DWG file. The maximum length of these strings is 511 characters. This information is stored and retrieved in the Summary Information object with specific methods for each information field. The predefined fields are

- Title
- Subject
- Author
- Keywords
- Comments
- Last saved by
- Revision number
- Hyperlink base

You may create your own custom fields in addition to the predefined fields. These custom fields are stored in a list, and you can manipulate custom fields by either their name (or key) or by their position (index) in the list. Custom fields are indexed starting at 1, and there is no limit to the number of fields you may create.

AcDbSummaryInfoReactor Class

This class provides a reactor to let you know if the summary information is changed.

AcDbSummaryInfoManager Class

The `AcDbSummaryInfoManager` class organizes the `summaryInfoHasChanged` reactors.

There are global functions to set and get the summary information object and to get the global instance of the summary information manager. Also, the following method has been added to `AcDbDatabase`:

```
Adesk::Boolean
dwgFileWasSavedByAutodeskSoftware();
```

This method works with Release 14 and later drawings, and returns `Adesk::kTrue` if it determines that the database was last saved by Autodesk software (such as AutoCAD or AutoCAD LT). `Adesk::kTrue` will always be returned for Release 13 and earlier drawings, for databases created from DXF files, and whenever the function is called from an ObjectDBX program.

MFC Dialogs

The Microsoft Foundation Class (MFC) library allows a developer to implement standard user interfaces quickly. The ObjectARX environment provides a set of classes that allows a developer to create MFC-based user interfaces that behave and appear as the built-in Autodesk user interfaces. For a review of how to use the MFC library as part of an ObjectARX application and a discussion of the use of shared and statically linked MFC libraries, as well as issues related to upgrading ObjectARX applications, see chapter 8 in the *ObjectARX Developer's Guide*.

SaveAs

AutoCAD 2000 incorporates a change to the SaveAs functionality, and now provides the ability to specify the default file format for the SAVEAS, SAVE, and QSAVE commands. (The AUTOSAVE command always saves drawing in the AutoCAD 2000 drawing file format.) This feature enables your users to save their drawings to their desired file format repeatedly and easily, by setting the default file format to something other than the AutoCAD 2000 DWG file format. Possible reasons for needing to do this include

- The user's company is gradually migrating to the new version of AutoCAD, so they must save their drawings to a file format that everyone can read.
- Clients or contractors are using an older version of AutoCAD.

New APIs

Two new methods have been added to ObjectARX to set and get the SaveAs feature settings. There are also two new global functions that allow you to save to Release 13 and Release 14 drawing formats.

New Enumeration

The enumerated type `SaveFormat` has been added to `AcApDocument`. Its values are shown in the following table:

SaveFormat		
Name	**Value**	**Usage (file extension)**
kR12_dxf	1	AutoCAD Release 12/LT2 DXF (*.dxf)
kR13_dwg	4	AutoCAD Release 13/LT95 Drawing (*.dwg)
kR13_dxf	5	AutoCAD Release 13/LT95 DXF (*.dxf)
kR14_dwg	8	AutoCAD Release 14/LT97 Drawing (*.dwg)
kR14_dxf	9	AutoCAD Release 14/LT97 DXF (*.dxf)
kR15_dwg	12	AutoCAD 2000 Drawing (*.dwg)
kR15_dxf	13	AutoCAD 2000 DXF (*.dxf)

SaveFormat (*continued*)		
Name	**Value**	**Usage (file extension)**
kR15_Template	14	AutoCAD 2000 Drawing Template File (*.dwt)
kNative	kR15_dwg	Current DWG version is AutoCAD 2000
kUnknown	-1	Invalid format

This enum is used to define the format to be used when saving a drawing to a file. The values are not random but correspond to the following formula:

Value = (Release # - 12) * 4 + (0 for DWG, 1 for DXF, 2 for DWT)

New Methods

The `formatForSave()` method has been added to `AcApDocument`. This method returns the current save format being used by the SAVEAS, SAVE, and QSAVE commands:

```
AcApDocument::SaveFormat
formatForSave();
```

The value returned may be either the session-wide default setting or a different setting that the user has selected for this document. If it is an override for this document, it will not persist across sessions.

The `setDefaultFormatForSave()` method has been added to the `AcApDocManager` class, and uses one of the `SaveFormat` values to set the file format to use when saving a drawing with the SAVEAS, SAVE, and QSAVE commands. This sets the session-wide default, which the user may choose to temporarily override for an individual document:

```
Acad::ErrorStatus
setDefaultFormatForSave(
    AcApDocument::SaveFormat format);
```

These methods only directly report on or set the file format for interactive commands entered by the user. If you want your application to use the current save format, every time you wish to save the database, you will first need to call `formatForSave()`, and then use the returned `SaveFormat` value to determine which function to call. For example, if `formatForSave()` returned `kR14_dxf`, you would call `acdbDxfOutAsR14()` to write the database as a Release 14 DXF file.

Be sure to take the following into account:

- Either you or your user may set a persistent session-wide default format for save that will be honored by all save commands except AUTOSAVE.
- Only the user can temporarily (not persistently between sessions) override this setting for a particular document.
- The `formatForSave()` method returns the format in which the user wishes an individual document to be saved; this will be either the session-wide default or the temporary override, as appropriate.

New Global Functions

Two new global functions are part of the SaveAs feature:

```
Acad::ErrorStatus
acdbSaveAsR13(
    AcDbDatabase* pDb,
    const char* fileName);

Acad::ErrorStatus
acdbSaveAsR14(
    AcDbDatabase* pDb,
    const char* fileName);
```

Both functions accept a database pointer and a filename, and write out the drawing in AutoCAD Release 13 or Release 14 DWG format, respectively.

AutoCAD DesignCenter

The AutoCAD DesignCenter provides an API that can be used to display and manage custom content information. This API consists of two Component Object Model (COM) interfaces:

- The `IContentView` interface that provides a way for applications to manage their content in an AutoCAD DesignCenter ActiveX control.
- The `IContentFinder` interface that provides a way for applications to participate in the Find mechanism for finding content on the disk.

Using the API

To provide the content in the AutoCAD DesignCenter, you must implement at least the `IContentView` interface. If your application desires to participate in the mechanism for finding content, then the application needs to support `IContentFinder` as well. This means that implementing the `IContentFinder` interface is optional.

Your application will need to set up some registry keys under the AutoCAD root key in the registry. The following is a list of the minimum information to be provided:

- Name of content provider
- Content types
- Search key (if the provider participates in Find)
- Path to the content provider server

IContentView Interface

The interface to the `IContentView` object is described in the following two tables. The first table describes methods and the second describes events:

IContentView methods

Method	Explanation
void SetNavigatorImageList(LPDISPATCH ImageList);	Sets the image list on the navigator.
BSTR GetNavigatorSelectedItemText();	Returns the text of currently selected item in the navigator.
LPDISPATCH NavigatorHitTest(float x, float y);	Returns the node item at given coordinates.
LPDISPATCH AddNavigatorNode(VARIANT* Relative, VARIANT* Relationship, VARIANT* Key, VARIANT* Text, VARIANT* Image, VARIANT* SelectedImage);	Adds a node in the navigator.
void SetPaletteIcons(LPDISPATCH ImageList);	Sets the image list for icons view in the palette.
void SetPaletteSmallIcons(LPDISPATCH ImageList);	Sets the image list for small icons view in the palette.

IContentView methods (*continued*)

Method	Explanation
LPDISPATCH AddPaletteItem(VARIANT* Index, VARIANT* Key, VARIANT* Text, VARIANT* Icon, VARIANT* SmallIcon);	Adds a palette item.
LPDISPATCH PaletteHitTest(float x, float y);	Returns the list item at the given coordinates.
LPDISPATCH AddPaletteColumnHeader(VARIANT* Index, VARIANT* Key, VARIANT* Text, VARIANT* Width, VARIANT* Alignment);	Adds a column header in the palette.
void SetPaletteMultiSelected(BOOL bSelect);	Sets the option to whether to allow multi-selections of items in the palette.
BSTR GetPaletteSelectedItemText();	Returns the currently selected item's text.
LPDISPATCH GetPaletteItem(VARIANT* Index);	Returns the palette item at a given index.
void SetPreviewImage(BSTR ImageFileName);	Sets the preview image with the image in the given file name.
void DrawImage(HDC hdc, RECT DirtyRect);	Draws an image in the given device context and the dirty rectangle.
void SetDescriptionText(VARIANT* Text);	Sets the description text for the currently selected item in the palette.

These are the events that the `IContentView` object recognizes:

IContentView events

Events	Explanation
void OnNavigatorNodeClick(LPDISPATCH Node);	Handler for when a node in the navigator is clicked.
void OnNavigatorDblClick();	Handler for when a double-click event happens in the navigator.
void OnNavigatorMouseUp(short Button, short Shift, OLE_XPOS_PIXELS x, OLE_YPOS_PIXELS y);	Handler for when a mouse button is released in the navigator.
void OnPaletteItemClick(LPDISPATCH Item);	Handler for when a palette item is clicked.
void OnPaletteMouseUp(short Button, short Shift, OLE_XPOS_PIXELS x, OLE_XPOS_PIXELS y);	Handler for when a mouse button is released in the palette.
void OnPaletteColumnClick(LPDISPATCH ColumnHeader);	Handler for when a column header is clicked.
void OnPaletteOLECompleteDrag(long* Effect);	Handler for the OLECompleteDrag event in the palette.
void OnPaletteOLEStartDrag(LPDISPATCH Data, long* AllowedEffects);	Handler for the OLEStartDrag event in the palette.
void OnPaletteDblClick();	Handler for when a double-click happens in the palette event.

IContentView events (*continued*)

Events	Explanation
void OnPaletteOLEDragDrop(LPDISPATCH* Data, long* Effect, short* Button, short* Shift, float* x, float* y);	Handler for when an object is dropped in the palette.
void OnPaletteOLEDragOver(LPDISPATCH* Data, long* Effect, short* Button, short* Shift, float* x, float* y, short* State);	Handler for when an object is dragged over in the palette area.
void OnPaletteOLESetData(LPDISPATCH* Data, short* DataFormat);	Handler to set the data for a dropped object in the palette.
void OnPaletteOLEGiveFeedback(long* Effect, BOOL* DefaultCursors);	Handler to set the appropriate mouse cursor to indicate what would happen if an object is dropped in the target under the cursor.

IContentFinder Interface

The methods and events for the `IContentFinder` object, along with their descriptions, follow in separate tables, with the methods appearing first:

IContentFinder methods

Method	Explanation
void SetSearchKey(VARIANT* Key);	Sets the search key string.
void AddSearchField(VARIANT* Field);	Sets the search field string.

IContentFinder methods (*continued*)	
Method	**Explanation**
void EnableBrowsing(BOOL Enable);	Enables/disables the Browse button.
BSTR GetSelectedField();	Gets the user-selected search field.
BSTR GetSearchPath();	Gets the search path.
BSTR GetSearchKey();	Returns the search text typed in by the user.
long GetNoOfFoundItems();	Returns the number of found items.
LPDISPATCH GetFoundItems();	Returns a list of found items.
LPDISPATCH GetFoundItem(VARIANT* Index);	Returns the found item at a given index.
HRESULT Find();	Performs the find.

These are the events for the `IContentFinder` object:

IContentFinder events	
Events	**Explanation**
void Stop();	Stops the find process.
void OnFoundItemClick(VARIANT Index);	Handler for when a found item is clicked.
void OnFoundItemDblClick(VARIANT Index);	Handler for when a found item is double-clicked.

Multiple UCS per Viewport

AutoCAD users may now specify a different UCS (User Coordinate System) for each of their viewports. This feature allows the UCS to be changed transparently for both viewport changes and view changes, and is especially useful for users who work in orthogonal views.

API Additions

New methods to define, set, and query multiple viewports have been added in several classes. There is also a new enumerated type to specify the orthographic viewing directions. These are all described in the following sections.

OrthographicView Enumeration

The following enumerated type represents the orthographic viewing directions:

OrthographicView enumeration

Name	Value	Usage
kNonOrthoView	0	Indicates a view that is not orthogonal.
kTopView	1	Indicates a view from the top.
kBottomView	2	Indicates a view from the bottom.
kFrontView	3	Indicates a front view.
kBackView	4	Indicates a back view.
kLeftView	5	Indicates a view from the left.
kRightView	6	Indicates a view from the right.

System Variable Changes

The following table lists the system variables that have been modified or are new in AutoCAD 2000 for this feature:

Multiple UCS per viewport system variables

Name	Description	Data Type	Values	Initial Value	Saved in	Accessed in
UCSBASE	Contains name of UCS from which orthographic UCSs are defined.	string	Valid named UCS or *World*	*World*	Drawing	CLI, UCS, View dialogs
UCSVP	Controls whether the UCS in active viewports remains fixed or "floats"– assumes the UCS of the current viewport.	Integer	0 - Unlocked (floats) 1 - Locked (fixed)	1	Drawing (viewport specific)	CLI, UCS dialog
UCSORTHO	Controls automatic restoration of orthographic UCSs when an orthographic view is restored.	Integer	0 - Don't automatically restore orthographic UCSs 1 - Restore orthographic UCS when an orthographic view is restored	1	Registry	CLI, View dialog
UCSVIEW	Controls whether the current UCS is saved with a named view.	Integer	0 - Do not save 1 - Save	1	Registry	CLI, -View, View dialog

AcDbDatabase Class

The following methods have been added to the AcDbDatabase class:

```
AcDbObjectId
ucsBase() const;

AcDbObjectId
pucsBase() const;
```

The ucsbase() method returns the object ID of UCSBASE for model space on which all orthographic UCSs and views are based. If a NULL object ID is returned, then orthographic views and UCSs will be based on the WCS (World Coordinate System). The pucsBase() method is similar but works on paper space.

```
Acad::ErrorStatus
setUcsBase(
    const AcDbObjectId& ucsId);

Acad::ErrorStatus
setPucsBase(
    const AcDbObjectId& ucsId);
```

Sets UCSBASE for model and paper space, respectively. The value of ucsId must either be NULL or point to a valid AcDbUCSTableRecord object. If NULL, then UCSBASE will be defined as the WCS.

```
Adesk::Boolean
isUcsOrthographic(
    OrthographicView& view) const;

Adesk::Boolean
isPucsOrthographic(
    OrthographicView& view) const;
```

These methods return whether the UCS is one of the six orthographic views, relative to UCSBASE, for model and paper space, respectively.

```
Acad::ErrorStatus
setWorldUcsBaseOrigin(
    const AcGePoint3d& origin,
    OrthographicView view);

Acad::ErrorStatus
setWorldPucsBaseOrigin(
    const AcGePoint3d& origin,
    OrthographicView view);
```

These methods define the base origin point for the six orthographic views when UCSBASE is NULL (that is, when it is WCS). setWorldUcsBaseOrigin() applies to model space, and setWorldPucsBaseOrigin() applies to paper space.

AcDbViewport and AcDbAbstractViewTableRecord Classes

The following methods have been added to the `AcDbViewport` and `AcDbAbstractViewTableRecord` classes.

```
Acad::ErrorStatus
getUcs(
    AcGePoint3d& origin,
    AcGeVector3d& xAxis,
    AcGeVector3d& yAxis) const;
```

This methods gets the UCS definition for the view or viewport.

```
AcDbObjectId
ucsname() const;
```

The `ucsname()` method returns the object ID of the `AcDbUcsTableRecord` object if the UCS for the view or viewport is named.

```
Adesk::Boolean
isUcsOrthographic(
    OrthographicView& view) const;
```

This method returns whether the UCS is one of the six orthographic views relative to UCSBASE in model or paper space.

```
double
elevation() const;
```

Returns the elevation for the view or viewport.

```
Acad::ErrorStatus
setUcs(
    AcGePoint3d& origin,
    AcGeVector3d& xAxis,
    AcGeVector3d& yAxis);
```

This `setUcs()` method defines a new, unnamed UCS for the view or viewport.

```
Acad::ErrorStatus
setUcs(
    const AcDbObjectId& ucsId);
```

This method defines a named UCS for the view or viewport. The `ucsId` argument must reference a valid `AcDbUcsTableRecord` object.

```
Acad::ErrorStatus
setUcs(
    OrthographicView orthoView);
```

This version of `setUcs()` defines a new unnamed UCS for the view or viewport for one of the six orthographic views, relative to the value of UCSBASE.

```
Acad::ErrorStatus
setUcsToWorld();
```

Sets the UCS to the WCS.

```
Acad::ErrorStatus
setElevation(
    double elev);
```

Sets the elevation for the view or viewport.

```
Acad::ErrorStatus
setViewDirection(
    OrthographicView& View);
```

Sets the viewing direction to one of the six orthographic views relative to UCSBASE.

```
Adesk::Boolean
isViewOrthographic(
    OrthographicView& view) const;
```

The `isOrthographicView()` method returns whether view is one of the six orthographic views relative to the value of UCSBASE.

AcDbViewTableRecord Class

The following methods have been defined in the `AcDbViewTableRecord` class:

```
Adesk::Boolean
isUcsAssociatedToView() const;
```

Returns whether the view has an associated UCS (from calling `setUcs()`).

```
Acad::ErrorStatus
disassociateUcsFromView();
```

Disassociates the UCS from the view. After this call, the UCS will not change when the view is restored.

AcDbUCSTableRecord Class

The following methods have been defined in the `AcDbUCSTableRecord` class:

```
AcGePoint3d
ucsBaseOrigin(
    OrthographicView view) const;
```

This method returns the UCS base origin for one of the six orthographic views relative to this UCS.

```
Acad::ErrorStatus
setUcsBaseOrigin(
    const AcGePoint3d& origin,
    OrthographicView view) const;
```

Sets the UCS base origin for one of the six orthographic views relative to this UCS.

Global Functions

The following function sets the current drawing in AutoCAD to use the previous UCS for that drawing. The function is declared in the *aced.h* header file:

```
Acad::ErrorStatus
acedRestorePreviousUCS();
```

If there is no previous UCS to restore, `Acad::eNotApplicable` is returned. `Acad::eOk` is returned if the function was successful.

The `acedGetCurrentUCS()` function sets `mat` to the values that describe the current UCS in the current drawing in AutoCAD:

```
Acad::ErrorStatus
acedGetCurrentUCS(
    AcGeMatrix3d& mat);
```

mat	Output set to the matrix that defines the current UCS in the current drawing

The first row of the matrix is the UCS *X*-axis (in WCS coordinates), the second row is the UCS *Y*-axis, and the third row is the UCS *Z*-axis.

The function is declared in the *aced.h* header file and returns `Acad::eOk` if successful.

DXF Changes

This feature adds several new codes to DXF. These are described in the following tables by class:

AcDbViewportTableRecord DXF codes

Group Code	Description
65	UCSVP system variable

AcDbViewport DXF codes

Group Code	Description
71	UCSVP system variable

AcDbViewportTableRecord, AcDbViewTableRecord, and AcDbViewport DXF codes

Group Codes	Description
79	Orthographic view type
110,120,130	UCS origin
111,121,131	UCS X-axis
112,122,132	UCS Y-axis
146	Elevation
340	UCS name
341	UCS base reference if orthographic view type is not kNonOrthoView

AcDbLayout DXF codes

Group Codes	Description
13,23,33	UCS origin
16,26,36	UCS X-axis
17,27,37	UCS Y-axis
76	Orthographic view type
146	Elevation
340	UCS name
341	UCS base reference if orthographic view type is not kNonOrthoView

AcDbUCSTableRecord DXF Codes

Group Codes	Description
13,23,33	UCS base origin (71 and 13,23,33 always occur in pairs)
71	Orthographic view type for UCS base origin
79	Orthographic view type
146	Elevation

Shortcut Menus

Shortcut menus (sometimes called right-click or context menus) are now implemented throughout AutoCAD, and are invoked by clicking the right mouse button. Release 14 right-click behavior is also maintained and made available to users as AutoCAD Classic behavior (selectable in the Preferences dialog). The Release 14 command line interface context menu is also fully supported in this release of AutoCAD.

Four basic types of Shortcut menus are available in the AutoCAD drawing area:

- The Default menu, which appears when there is no selection set, no command in progress, and users right-click in the drawing
- The Edit-mode menu, which appears when users right-click when there is a selection set but no command in progress
- Command-mode menus, which appear when users right-click during a command in progress
- Dialog-mode menus, which appear when users right-click on modal dialogs

Context Class

The following class is used by ObjectARX applications to add items to a Shortcut menu:

```
class AcEdUIContext : public AcRxObject
{
public:
    virtual void*
    getMenuContext(
        const AcRxClass* pClass,
        AcDbObjectIdArray& objIds) = 0;

    virtual void
    onCommand(
        Adesk::UInt32 cmdId) = 0;
};
```

Shortcut menu items provided by ObjectARX applications should contain no keyboard accelerators. No status line menu-item help is displayed for these added items.

Additional APIs

These two APIs take an AcEdUIContext instance and add it to or remove it from a customized object:

```
Adesk::Boolean
acedAddObjectContextMenu(
    const AcRxClass* pClass,
    AcEdUIContext* pContext,
    const void* appId);
```

The appId argument is the void* passed in as the second parameter to the application's acrxEntryPoint() function during the AcRx::kInitAppMsg call that occurs when the application is initially loaded.

Only the void* passed in to acrxEntryPoint() during the AcRx::kInitAppMsg call is valid to be used by this function. So, if this function will be called at any other time, the void* passed in during the AcRx::kInitAppMsg call must be stored for later use.

```
Adesk::Boolean
acedRemoveObjectContextMenu(
    const AcRxClass* pClass,
    AcEdUIContext* pContext);
```

These two APIs add or remove default AcEdUIContext instances:

```
Adesk::Boolean
acedAddDefaultContextMenu(
    AcEdUIContext* pContext,
    const void* appId,
    const char* appName = NULL);

Adesk::Boolean
acedRemoveDefaultContextMenu(
    AcEdUIContext* pContext);
```

These four APIs all return Adesk::kTrue if successful.

Default Mode

When no command is running, there is no pick-first selection set, and the user right-clicks for a Shortcut menu, the menu will contain some standard items read from the CMDEFAULT POP menu in the MNU file. If there are loaded ObjectARX applications that wish to append items to this Shortcut menu, they can do so. The ObjectARX applications loaded can supply HMENUS and callback functions for items that will appear in the Shortcut menu as a submenu. The top-level item (submenu name) may be supplied by you, and will default to the ObjectARX application's file name.

Edit Mode

When a pick-first selection set exists and the user right-clicks for a Shortcut menu, the menu will contain some standard items read from the CMEDIT POP menu in the MNU file. In addition, the selected object(s) can append items to the menu. Each `AcRxClass` in the hierarchy of an object can contain an `AcEdUIContext` object. The Shortcut menu code determines a common parent for all objects in the selection set. Starting with that class and going up the class hierarchy, it queries each class for an `AcEDUIContext` object and then appends items to the Shortcut menu from the supplied `HMENU`. In this way, each class in an object's class hierarchy can contribute items to a Shortcut menu. If the user picks one of these appended items, AutoCAD calls the `onCommand()` member function of the `AcEDUIContext` object for the object that provided the appended menu item, using the ID associated with the appended menu item.

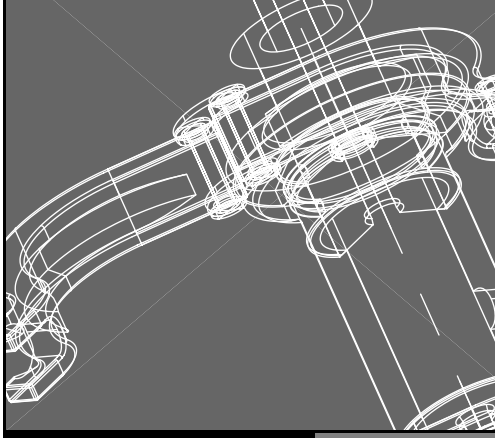
Command Mode

When an ObjectARX command is running and the user right-clicks for a Shortcut menu, the menu will contain some standard items read from the CMCOMMAND POP menu in the MNU file. In addition, if the command prompt issued by the ObjectARX application has command options in it and the command prompt uses the standard syntax, these options will appear as individual items in the Shortcut menu. When the user picks a command option from the Shortcut menu, the option keyword will be sent to the current command processor. These command options are always added to the Shortcut menu and the ObjectARX developer need only provide a command prompt in standard form to get this functionality.

If the ObjectARX application wishes to append additional items to the Shortcut menu for a particular command, it must supply an `HMENU` and a callback function to AutoCAD. This is done through the `AcEdUIContext` object in the ObjectARX command object, and the `AcEdCommandStack::addCommand()`.

Entity Features

This chapter describes changes to multiline text and dimensioning, and the new Object Property Manager.



In This Chapter

5

- Multiline Text
- Dimension Features
- Object Property Manager

Multiline Text

Multiline text (mtext) has been significantly enhanced in AutoCAD 2000. Multiline text is now faster in use and has improvements in the areas of overrides, fractions, case conversion, font management, line spacing, and spell checking. Most of these improvements are only seen from the user interface, but line spacing has had a new enumeration and APIs added. These are described in this section.

New Enumeration

The `LineSpacingStyle` enumeration is used to specify whether the line spacing factor is an exact value or a minimum value:

LineSpacingStyle

Name	Value	Usage
kAtLeast	1	Minimum spacing - individual lines may occupy more space, never less. This is the default spacing.
kExactly	2	Exact spacing - regardless of line's contents, spacing is forced to an exact value.

New Functions

Global functions are provided to set and get both the line spacing attributes and attachment point type:

```
Acad::ErrorStatus
setLineSpacingFactor(
    double dFactor);
```

dFactor — The line spacing value. `dFactor` is always relative. 1.0 is "normal," 2.0 is double, and so on. The default, which is Release 14–compatible, is 1.0.

```
double
lineSpacingFactor() const;
```

```
Acad::ErrorStatus
setLineSpacingStyle(
    LineSpacingStyle eStyle);
```

eStyle — The spacing style. The default is `kAtLeast`.

```
LineSpacingStyle
lineSpacingStyle() const;

Acad::ErrorStatus
setAttachmentMovingLocation(
    AttachmentPoint type);
```

The `setAttachmentMovingLocation()` API supplements the `setAttachment()` function:

```
Acad::ErrorStatus
setAttachment(
    AttachmentPoint type);
```

By setting the attachment attribute, it also updates the location such that the body of text does not move around significantly. This is similar but not identical to the way the mtext editor behaves, because the editor applies some additional logic depending on prevailing conditions.

Dimension Features

The ObjectARX APIs for dimensioning have been modified to increase significantly ease of use in drawing and editing dimensions, and creating DimStyles. The changes were made with these goals in mind:

- Continue to support the Release 14 API.
- Support the full `AcDbEntity` protocol (this was not possible in Release 14 because `AcDbDimension` was not yet objectified).
- Rationalize the APIs of `AcDbLeader` and `AcDbFcf` with that of `AcDbDimension`.
- Provide a full dimension variable (dimvar) API for `AcDbDimension`, and provide a limited dimvar API for `AcDbLeader` and `AcDbFcf`.
- Deprecate unwanted or inconsistent Release 14 API methods that will be removed in the next release. See "Obsolete Methods" at the end of this section for a list of these APIs.

Object and Style Overview

The dimension object class hierarchy is unchanged from Release 14, but the header information for these classes has been moved from *dbents.h* to *dbdim.h*.

The dimension variables used in Release 14 continue as before, though a few have been added or removed (see the following table) for AutoCAD 2000. A summary of changes follows:

- New methods that use `const char*` for returning string values have been added and the old methods have been deprecated. This was done to help remove the issue of memory management across DLL boundaries. The deprecated methods are not being provided for the new `AcDbDimension` API.
- A new method for the leader arrowhead `dimldrblk()` is being added, and the existing arrowhead controls will no longer apply to `AcDbLeader`.
- New lineweight control methods, `dimlwd()` and `dimlwe()`, have been added for dimension line and extension line.
- The `dimfit()` method is being deprecated in favor of two new variables (and get and set APIs), `dimfit2()` and `dimtmov()`.
- The `dimunit()` method is being made obsolete in favor of two new variables (and get and set APIs), `dimlunit()` and `dimfrac()`.
- Other new variables (with their corresponding get and set APIs) are `dimdsep()`, `dimadec()`, and `dimaltrnd()`.

Obsolete Methods

The following table lists the dimensioning methods that are now obsolete, and will not be available in the next release of AutoCAD. The table also includes the methods to use instead, and the reasons for their use:

Obsolete dimensioning methods

Do Not Use	Use Instead	For This Purpose
dimfit()	dimfit2(), dimtmove()	Separate control of dimline/arrow/text behavior and text movement
dimunit()	dimlunit(), dimfrac()	Separate control of linear units and fractional style
setDimVars()	setDimensionStyle(), setDimstyleData()	Separate control of setting the dimstyle and setting overrides
getDimblk()	dimblk()	Getting the object ID of a dimblk
setDimblk(char*&)	setDimblk(AcDbObjectId)	Setting the object ID of a dimblk

Obsolete dimensioning methods (*continued*)

Do Not Use	Use Instead	For This Purpose
getDim[a]post()	const char* dim[a]post()	Getting the dim[a]post string
setDim[a]post(char*&)	setDim[a]post(const char*)	Setting the dim[a]post string

Style Query Methods

The table below contains a complete list of the dimension style query methods. The status for each method is either

- Unchanged from Release 14
- Changed from Release 14
- New to AutoCAD 2000
- Obsolete

Notes on the changed methods follow the table. Also, the rightmost five columns in the table show which methods are available in which classes. For information about any dimension variable, see appendix B, "System Variables," in the AutoCAD *Command Reference*:

Dimension variable get methods in alphabetic order

Method	Status	AcDbDatabase	AcDbDimStyleTableRecord	AcDbDimension	AcDbFcf	AcDbLeader
int dimadec() const;	New	X	X	X		
Adesk::Boolean dimalt() const;	Unchanged	X	X	X		
int dimaltd() const;	Unchanged	X	X	X		

Dimension variable get methods in alphabetic order (*continued*)

Method	Status	AcDbDatabase	AcDbDimStyleTableRecord	AcDbDimension	AcDbFcf	AcDbLeader
double dimaltf() const;	Unchanged	X	X	X		
double dimaltrnd() const;	New	X	X	X		
int dimalttd() const;	Unchanged	X	X	X		
int dimalttz() const;	Unchanged	X	X	X		
int dimaltu() const;	Unchanged	X	X	X		
int dimaltz() const;	Unchanged	X	X	X		
const char* dimapost() const;	Unchanged	X	X	X		
double dimasz() const;	Unchanged	X	X	X		X
int dimaunit() const;	Unchanged	X	X	X		
int dimazin() const;	New	X	X	X		
AcDbObjectId dimblk() const;	Changed	X	X	X		
AcDbObjectId dimblk1() const;	Changed	X	X	X		
AcDbObjectId dimblk2() const;	Changed	X	X	X		

Dimension variable get methods in alphabetic order (*continued*)

Method	Status	AcDbDatabase	AcDbDimStyleTableRecord	AcDbDimension	AcDbFcf	AcDbLeader
double dimcen() const;	Unchanged	X	X	X		
AcCmColor dimclrd() const;	Unchanged	X	X	X	X	X
AcCmColor dimclre() const;	Unchanged	X	X	X		
AcCmColor dimclrt() const;	Unchanged	X	X	X	X	
int dimdec() const;	Unchanged	X	X	X		
double dimdle() const;	Unchanged	X	X	X		
double dimdli() const;	Unchanged	X	X	X		
char dimdsep() const;	New	X	X	X		
double dimexe() const;	Unchanged	X	X	X		
double dimexo() const;	Unchanged	X	X	X		
int dimfit() const;	Obsolete	X	X			
int dimfit2() const;	New	X	X	X		
int dimfrac() const;	New	X	X	X		

Dimension variable get methods in alphabetic order (*continued*)

Method	Status	AcDbDatabase	AcDbDimStyleTableRecord	AcDbDimension	AcDbFcf	AcDbLeader
double dimgap() const;	Unchanged	X	X	X	X	X
int dimjust() const;	Unchanged	X	X	X		
AcDbObjectId dimldrblk() const;	New	X	X	X		X
double dimlfac() const;	Unchanged	X	X	X		
Adesk::Boolean dimlim() const;	Unchanged	X	X	X		
int dimlunit() const;	New	X	X	X		
AcDb::LineWeight dimlwd() const;	New	X	X	X		X
AcDb::LineWeight dimlwe() const;	New	X	X	X		
const char* dimpost() const;	Unchanged	X	X	X		
double dimrnd() const;	Unchanged	X	X	X		
Adesk::Boolean dimsah() const;	Unchanged	X	X	X		
double dimscale() const;	Unchanged	X	X	X	X	X
Adesk::Boolean dimsd1() const;	Unchanged	X	X	X		

Dimension variable get methods in alphabetic order (*continued*)

Method	Status	AcDbDatabase	AcDbDimStyleTableRecord	AcDbDimension	AcDbFcf	AcDbLeader
Adesk::Boolean dimsd2() const;	Unchanged	X	X	X		
Adesk::Boolean dimse1() const;	Unchanged	X	X	X		
Adesk::Boolean dimse2() const;	Unchanged	X	X	X		
Adesk::Boolean dimsoxd() const;	Unchanged	X	X	X		
int dimtad() const;	Unchanged	X	X	X		X
int dimtdec() const;	Unchanged	X	X	X		
double dimtfac() const;	Unchanged	X	X	X		
Adesk::Boolean dimtih() const;	Unchanged	X	X	X		
Adesk::Boolean dimtix() const;	Unchanged	X	X	X		
double dimtm() const;	Unchanged	X	X	X		
int dimtmov() const;	New	X	X	X		
Adesk::Boolean dimtofl() const;	Unchanged	X	X	X		
Adesk::Boolean dimtoh() const;	Unchanged	X	X	X		

Dimension variable get methods in alphabetic order (*continued*)

Method	Status	AcDbDatabase	AcDbDimStyleTableRecord	AcDbDimension	AcDbFcf	AcDbLeader
Adesk::Boolean dimtol() const;	Unchanged	X	X	X		
Adesk::Int8 dimtolj() const;	Unchanged	X	X	X		
double dimtp() const;	Unchanged	X	X	X		
double dimtsz() const;	Unchanged	X	X	X		
double dimtvp() const;	Unchanged	X	X	X		
AcDbObjectId dimtxsty() const;	Unchanged	X	X	X	X	X
double dimtxt() const;	Unchanged	X	X	X	X	X
int dimtzin() const;	Unchanged	X	X	X		
int dimunit() const;	Obsolete	X	X			
Adesk::Boolean dimupt() const;	Unchanged	X	X	X		
int dimzin() const;	Unchanged	X	X	X		

The methods marked “Changed“ are the dimension block get and set methods. The old obsolete methods are

```
Acad::ErrorStatus
getDimblk(
    char*& pOutput) const;

Acad::ErrorStatus
setDimblk(
    const char* val);
```

The new methods are:

```
AcDbObjectId
dimblk() const;

Acad::ErrorStatus
setDimblk(
    AcDbObjectId val);
```

The new get methods have different names, and the new set methods are overloaded versions of the Release 14 names.

Notes on the AcDbDimension Class

`AcDbDimension` includes query and edit (get and set) methods for all the variables listed in the table above, except the obsolete methods, because `AcDbDimension` is a new API. See the `AcDbDimension` class section in the *ObjectARX Reference* for more information.

Notes on the AcDbDatabase Class

`AcDbDatabase` includes query and edit (get and set) methods for all the variables listed in the table above. There are also new methods for retrieving the child and parent dimension styles:

```
AcDbObjectId
getDimstyleChildId(
    const AcRxClass* pDimClass,
    AcDbObjectId& parentStyle) const;
```

pDimClass	A dimension class descriptor
parentStyle	A parent dimension style ID

Given a parent dimension style ID and a dimension class descriptor, this method returns the child dimension style ID, if one is available for the given dimension type. The given parent dimension style ID will be returned if no child dimension style is available that corresponds to the given dimension class.

```
AcDbObjectId
getDimstyleParentId(
    AcDbObjectId& childStyle) const;
```

childStyle A child dimension style ID

Given a child dimension style ID, this method returns the parent dimension style ID. The parent dimension style ID for the given child dimension style is returned. If the parent is unavailable, the given child dimension style ID is returned.

See the `AcDbDatabase` class section in the *ObjectARX Reference* for more information.

Notes on the AcDbDiametricDimension and AcDbRadialDimension Classes

Be aware that while the `setLeaderLength()` method works correctly, the `leaderLength()` method will usually return zero.

Calling `setLeaderLength()` will cause AutoCAD to compute a text position that results in a dimension line with the desired leader length. Once that leader length data is used (that is, after the next call to `subClose()` occurs), the leader length data is set to zero. Calling `leaderLength()` will return 0.0 unless the call is made after the `setLeaderLength()` call and before the `subClose()` call occurs. Passing a leader length to the constructor is effectively similar to calling `setLeaderLength()`, and the same rules apply.

These methods are being deprecated and may not be available in the next release of AutoCAD. See the *ObjectARX Reference* for additional information.

Notes on the AcDbDimStyleTableRecord Class

`AcDbDimStyleTableRecord` will include query and edit (get and set) methods for all of the variables listed above. See the `AcDbDimStyleTableRecord` class section in the *ObjectARX Reference* for more information.

Notes on the AcDbLeader Class

`AcDbLeader` will include the query and edit (get and set) methods found in the `AcDbLeader` class section in the *ObjectARX Reference*. This includes the addition of methods for `dimldrblk()`, a new variable that eliminates the need to look at `dimsah` to determine whether the arrowhead is in `dimblk` or `dimblk1`.

Two additional methods have been added to AcDbLeader:

```
virtual AcGeVector3d
annotationOffset() const;

virtual Acad::ErrorStatus
setAnnotationOffset(
    const AcGeVector3d& offset);
```

These methods get and set the "annotation offset" for the leader. The values may be set and retrieved for any leader, but they will affect the geometry only if the leader has associated annotation, that is, if AcDbLeader::annotationObjId() returns a valid object ID.

Note that in many circumstances you will have to call AcDbLeader::evaluateLeader() after setAnnotationOffset() in order for the change to take effect.

If the leader does have an associated annotation, then the location of the final leader endpoint is always determined by the location of the annotation to which it associates, not the point you set with AcDbLeader::appendVertex().

For block reference annotation, the leader endpoint is located at

```
AcDbBlockReference::position() + AcDbLeader::annotationOffset()
```

For AcDbMText and AcDbFcf annotation the endpoint is placed at

```
AcDbMText::location() + AcDbLeader::annotationOffset() +/-
    AcGeVector3d(dimgap(), 0, 0);
```

or

```
AcDbFcf::location() + AcDbLeader::annotationOffset() +/-
    AcGeVector3d(dimgap(), 0, 0);
```

The dimgap is added or subtracted according to whether the annotation attaches to the left or right of the leader.

Notes on the AcDbFcf Class

AcDbFcf will include query and edit (get and set) methods, found in the AcDbFcf class section in the *ObjectARX Reference*.

Object Property Manager

The Object Property Manager is a modeless and extensible way for users to view and modify object properties. It replaces the existing commands DDMODIFY and DDCHPROP. Object Property Manager supports user-defined objects, allowing users to view or modify object-specific properties of user-defined objects. Unlike the LISP routines DDMODIFY and DDCHPROP, Object Property Manager can modify the properties of a selection set containing multiple objects of the same type. Object Property Manager can also change the common properties of multiple objects in a selection set of different types.

Static Properties and the Object Property Manager

The Object Property Manager is essentially a control that parses type information from COM objects to determine their properties. When objects in the drawing are selected, the selection set is converted into an array of `IUnknown` pointers representing the COM objects that wrap all native entities in AutoCAD. These COM object wrappers are the basis of the ActiveX Automation interface and are the underlying objects that the Object Property Manager interacts with. You can access these COM wrapper objects in C++ by calling this *acad.lib* exported function:

```
IUnknown*
GetIUnknownOfObject(
    AcDbObjectId& objId,
    CLSID& clsid);
```

These COM object wrappers implement `IDispatch` as well as other interfaces. `IDispatch` is the COM interface the Object Property Manager uses to get and set property data. It is also the native object representation in Visual Basic and VBA. In order to determine what properties are available for an object, Object Property Manager calls `IDispatch` method `GetTypeInfo()`, which is implemented by all the AutoCAD COM wrappers. This method returns the type information for the object (an object that implements `ITypeInfo`). `ITypeInfo` is a standard Microsoft interface that wraps a data structure describing the methods and properties available on that object. Collections of type information are used by VB and VBA to define the ActiveX object model and are called type libraries. A good way to see which methods and properties are available in the object model is to load up VBA in AutoCAD and, from the IDE, view the ObjectBrowser, which uses type info to display itself.

By default all custom ObjectARX objects have the following changeable properties in the Object Property Manager, called the entity common properties:

- Color
- Layer
- Linetype
- LinetypeScale
- PlotstyleName
- LineWeight
- Hyperlink

Based on the type of the property as it is defined in the Interface Description Language (IDL), Object Property Manager constructs a property editor window appropriate to that type of property. For example, if the property type is numeric or textual, Object Property Manager constructs an edit box for it. If the property type is an enum, Object Property Manager creates a combo box with the enumerated value list in it, and so on. For an entity common property, Object Property Manager constructs the standard drop downs that are the same as those for the Object Properties Toolbar (OPT).

The static type info for each COM object is not the only source of property information for the Object Property Manager. The Object Property Manager also queries the object for a few other interfaces to control things such as property categorization, property value names for drop-down lists, and instancing dialogs for per-property editing (such as the ellipsis button dialogs). These are referred to collectively as "flavoring" interfaces.

Static Object Property Manager COM Interfaces

AutoCAD's COM objects support several interfaces, only some of which are used by the Object Property Manager. For completeness they are all listed here, along with a brief description, whether or not the Object Property Manager QIs (calls QueryInterface) for this interface, and whether or not they need to be implemented by the developer or are implemented via the AutoCAD Type Library (ATL) template base classes. All methods imple-

mented by Autodesk via the template base classes can be overridden by you when you implement the object:

COM interfaces

Interface	Description	Used by OPM	Required by OPM	Implemented in the ATL Base Classes
IUnknown	Used to determine object capabilities.	Yes	Yes	Yes
IDispatch	Used to get/set property data and retrieve type info.	Yes	Yes	Yes
IAcadObject	Implements properties all DB resident objects must have, ex. Name, Handle, Get/Set xdata, etc.	No*	No	Yes
IAcadBaseObject	Manages the object ID and CLSID.	No*	No	Yes
IAcadEntity	Manages entity-specific methods—Color, Layer, Linetype, etc.	No*	No	Yes
IRetrieveApplication	Manages back pointer to Application object.	No*	No	Yes
IConnectionPointContainer	Connectable objects implementation, used to notify that objects, properties have changed and need to be updated.	Yes	Yes	Yes
ISupportErrorInfo	Used to get extended error information	Yes	Yes	Yes
ICategorizeProperties	Used to categorize the properties in the Object Property Manager category tab	Yes	No	Yes, all methods return E_NOTIMPL unless using macros
IPerPropertyBrowsing	Used to provide custom strings for property drop-down lists as well bring up Property dialogs (via an ellipsis button)	Yes	No	Yes, all methods return E_NOTIMPL unless using macros

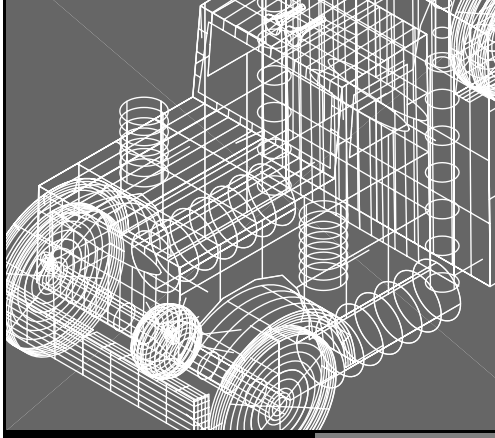
COM interfaces (*continued*)

Interface	Description	Used by OPM	Required by OPM	Implemented in the ATL Base Classes
IOPMPropertyExtension	Used to manage the display of properties in the Object Property Manager	Yes	No	Yes, all methods return E_NOTIMPL unless using macros
IOPMPropertyExpander	Used to expand one property into multiple ones	Yes	No	Yes, all methods return E_NOTIMPL

*Denotes interfaces that are not currently used by Object Property Manager, but may be used in the future.

It is important to note here that if a custom object does not implement a COM object wrapper for itself, `GetIUnknownOfObject()` will generate a default wrapper that implements the methods of `IAcadEntity` or `IAcadObject`. This depends on whether the underlying object can be cast to an `AcDbEntity`. The Object Property Manager then uses this object to display the entity common properties. `ICategorizeProperties`, `IPerPropertyBrowsing`, and `IOPMPropertyExtension` are the flavoring interfaces referred to earlier. Notice that all the flavoring interfaces are implemented (that is, QI returns a valid interface) but the methods return `E_NOTIMPL`, indicating that you must override these virtual methods in their object class if you want to customize the appearance of their properties.

Display and Hardcopy Features



In This Chapter

6

- Hardcopy
- Paper Space Layouts
- Lineweight
- OLE Scaling and Printing

This chapter gives an overview of the features regarding hardcopy output and related viewing items.

Hardcopy

The AutoCAD 2000 Hardcopy feature modernizes and updates AutoCAD's plotting and output functionality. This feature replaces the traditional Autodesk Device Interface™ (ADI) with the new Hardcopy Device Interface (HDI) pipeline. This feature replaces the text-based plotter configuration interaction with a set of modern graphical user interfaces, based on the Windows standard.

This feature also works with the Paper Space Layouts feature, described in "Paper Space Layouts." The two features can be used together to improve the way a drawing is printed. Also, users may now create "electronic plots" by sending the output to a DWF file.

All plotting ActiveX APIs are accessed via the AutoCAD ActiveX `Plot` object. This object has changed from the `Plot` object as presented in Release 14. With the new plotting pipeline, information related to "what-to-plot" (that is, "plot-with-this-scale") has been completely separated from "how-to-plot" information" (such as "plot to this device"). This is enforced in ActiveX—what-to-plot information is accessed in the Paper Space Layouts ActiveX interface, and how-to-plot information is accessed in the new `Plot` object.

The following table lists the current `Plot` object ActiveX APIs:

Plot object ActiveX APIs

ActiveX Method or Property Name	Implementation Signature	Description
Application	get_Application(IAcadApplication*);	Retrieves the current AcadApplication object associated with the given AcadPlot object. This is unchanged from Release 14 behavior.
DisplayPlotPreview	DisplayPlotPreview(AcPreviewMode);	Invokes the new plot preview with whatever what-to-plot, how-to-plot settings are current.

Plot object ActiveX APIs (*continued*)

ActiveX Method or Property Name	Implementation Signature	Description
PlotToFile	PlotToFile(BSTR, BSTR);	Uses the current what-to-plot and how-to-plot settings to plot to file. The first argument is the same argument type as in Release 14: the fully qualified output plot filename. The second argument, which is optional, will specify a PC3 file to use instead of the current configuration. However, the output will still be set up to plot to a file.
PlotToDevice	PlotToDevice(BSTR);	Will use the current what-to-plot and how-to-plot settings to plot to device. The argument can be a null string, which indicates "plot with current settings." The argument can also be the full path and name of a PC3 file, which will be used as an overriding PC3 file during the subsequent plot. Note that a PC3 file may have plot-to-file configuration information already in it, which implies that a PlotToDevice call may end up actually plotting to file.
QuietErrorMode	get_QuietErrorMode(VARIANT_BOOL*); put_QuietErrorMode(VARIANT_BOOL);	Toggles plot-related error messages "on" (normal, message-box alerts, etc.) or "quiet" (errors logged to logfile). This is for automated clients (such as batchplot) that need to have "uninterrupted" plotting.
NumberOfCopies	get_NumberofCopies(long*); set_NumberofCopies(long);	Gets and sets the number of copies plotted for the current plot settings.
SetLayoutsToPlot	SetLayoutsToPlot(VARIANT);	Specifies an array of Layout names to plot. The default, if the array is NULL, or if this method isn't called, is to plot the currently active Layout.

Plot object ActiveX APIs (*continued*)

ActiveX Method or Property Name	Implementation Signature	Description
StartBatchMode	StartBatchMode(long)	Invokes "batch mode." This is intended for ActiveX clients who want to plot batches of drawings but still take advantage of the AutoCAD Plot Progress dialog (such as batchplot). The argument is a count of the total number of entries to plot in this batch. Using this API allows the Plot Progress dialog to show the progress of a batch of plots, and to keep the Progress dialog active until the batch is done or canceled.
BatchPlotProgress	get_BatchPlotProgress(VARIANT_BOOL *); set_BatchPlotProgress(VARIANT_BOOL);	When BatchModeStart is invoked, this can query the current status of the batch plot progress, and optionally cancel the batch mode, if the client application goes away, for example. A value of TRUE indicates "in progress," a value of FALSE indicates "was canceled/aborted."

As mentioned earlier, several what-to-plot methods have been moved from the `Plot` object to the new `Layouts` object. The following table describes these and other changes and possible migration paths:

Changed or deleted plot object ActiveX APIs

Original Release 14 ActiveX Method or Property Name	AutoCAD 2000 Changes	Migration Options
Application	Unchanged	Not applicable.
PlotToFile	Will use new what-to-plot and how-to-plot information.	No change required for this particular API. You just need to be aware of the overall changes in the what-to-plot and how-to-plot information. Note that the second argument (overriding PC3 file) is optional.

Changed or deleted plot object ActiveX APIs (*continued*)

Original Release 14 ActiveX Method or Property Name	AutoCAD 2000 Changes	Migration Options
PlotToDevice	Will allow the use of a PC3 file as an argument, indicating the source of how-to-plot information for this particular plot.	Previously, Release 14 allowed use of a "plot configuration" name in this API. You need to be aware that such configurations no longer exist, and that PC3 files must be used instead.
PlotOrientation	Removed	Use AcadLayout.Rotation.
PlotScale	Removed	Use AcadLayout.Scale, StandardScale, or CustomScale.
Rotation	Removed	Use AcadLayout.Rotation.
HideLines	Removed	Use AcadLayout.Hidden.
Origin	Removed	Use AcadLayout.PlotOffset.
PlotUnits	Removed	Use AcadLayout.PaperUnits.
PaperSize	Removed	In Release 14, this corresponded to the physical dimensions of the media, which are now how-to-plot information, and must be set in a PC3 configuration file (via the PC3 Editor). AcadLayout. PaperSizeDescription can get and set the current media size name.
AdjustAreaFill	Removed	No migration in AutoCAD 2000 ActiveX. This is now how-to-plot information, and must be set in a PC3 configuration file (via the PC3 Editor).
PlotWithConfigFile	Removed	Use PlotToDevice instead, and specify a PC3 file as an argument.
PlotExtents	Removed	Use AcadLayout.AreaToPlot.
PlotLimits	Removed	Use AcadLayout.AreaToPlot.
PlotView	Removed	Use AcadLayout.AreaToPlot and AcadLayout.ViewToPlot.
PlotWindow	Removed	Use AcadLayout.AreaToPlot.

Changed or deleted plot object ActiveX APIs (*continued*)

Original Release 14 ActiveX Method or Property Name	AutoCAD 2000 Changes	Migration Options
LoadPC2	Removed	PC2 files are not supported in AutoCAD 2000. You will need to convert your PC2 files to PC3 files via the Add-a-Plotter Wizard.
SavePC2	Removed	See LoadPC2.

ActiveX developers should be aware of the fact that modification of how-to-plot information is not supported in ActiveX; that is, there is no ActiveX API to modify the new PC3 configuration file settings (such as which media source tray to use, and so forth). All configuration file modification is now accomplished via the PC3 Editor mechanism, which enforces PC3 file structure.

Paper Space Layouts

Layouts introduces a new and improved user interface to assist in laying out drawings for printing. It enhances the powerful Model and Paper Space concepts currently employed within AutoCAD by offering a native Windows interface. A layout incorporates one Paper Space, all the Model Space viewports you add in that Paper Space, and all the plot settings that you set in the PageSetup dialog and that are stored with the layout. Now you may have multiple layouts, each of which has its own Paper Space.

Features

Layouts provide many benefits and features:

- Streamlined print options and a WYSIWYG interface offer better understanding of what will actually be sent to the printed page, thereby reducing time involved with printing a drawing in AutoCAD today. The required depth of knowledge and experience, and the number of check plots required to produce plots and printouts of drawings quickly and accurately, will be reduced.
- Multiple layout capability improves drawing layout organization by allowing all layouts for a drawing to be easily stored, accessed, and printed from within the same drawing. Each layout contains its own print settings, so it is possible to have one layout configured to print an A-size

sheet of paper to a local printer, and another configured to print an E-size sheet to a large pen plotter.
- Ability to create nonrectangular viewports in Paper Space layouts and clip rectangular viewports to entities. The list of entities that can be used to clip a rectangular viewport into a nonrectangular viewport includes: circles, ellipses, splines, 2D polylines, 3D polylines, 3D faces, and regions.
- Enhanced viewport property sheets make it easier to change the scale and display properties of viewports. The ability to lock viewports reduces the chance of accidentally changing the scale of a viewport after it has been placed.

The new Model and Layout tabs replace the TILE toggle on the status bar, so the TILEMODE system variable will no longer be necessary or documented, but it will still be a valid variable. Setting TILEMODE to 0 will activate the last active layout.

API Information

The Paper Space layouts feature adds the capability to create multiple paper spaces (referred to as layouts) in a single DWG file. The existing programmatic interface for switching between Paper Space and Model Space (the previous function switched between TILEMODE=0 and TILEMODE=1) has been extended to allow switching between the new Model tab and multiple Layout tabs. In addition, third-party software may now create, delete, copy, move, and enumerate these layouts with the `AcApLayoutManager` class. Don't manipulate the `AcDbLayout` objects directly; use the Layout Manager. The following example shows how to use the Layout Manager to switch layouts:

```
AcApLayoutManager* pLMgr =
            (AcApLayoutManager*) AcApLayoutManager::getPointer();
    if (pLMgr == NULL)
        return;
    if (pLMgr->setCurrentLayout(newCurrName) != Acad::eOk)
        acutPrintf("setCurrentLayout() failed!");
}
```

The Layout classes are defined in the *acaplmgr.h* header file.

Three new database objects are introduced, along with modifications to other database objects. The `AcDbPlotSettingsValidator` class contains methods to

- get a list of the available devices on the system
- get the available media size descriptions for a particular device
- get the available plot style sheets
- set the plot settings and perform validation checks on the values

Note that the AcDbPlotSettingsValidator class is not available to ObjectDBX applications.

The AcDbPlotSettings class, derived from AcDbPlotSettingsValidator, provides a container for all plot setting values along with their "get" methods. The AcDbLayout class is derived from AcDbPlotSettings, and stores characteristics of each Paper Space layout. These Layout objects are stored in an AcDbDictionary object, with a lookup key of ACAD_PLOTSETTINGS, to allow easy iteration and indexing. The AcDbDatabase class has two new methods to provide access to these new dictionaries. The AcDbHostApplicationServices class contains a method to retrieve the plot settings validator object for the active layout. The AcDbBlockTableRecord object has been modified to accommodate mapping block table records to layouts. The AcDbViewport object now accommodates nonrectangular viewport clipping. Refer to the *ObjectARX Reference* for complete information on these classes and their methods.

Lineweight

Lineweight is the apparent width of linear objects, and is a property of most objects and of Layers. The user can specify a lineweight, in actual units, at which they want the line to plot, and AutoCAD 2000 will determine a display weight in pixels.

Overview

The new Lineweight feature provides these benefits:

- It gives the user relative lineweights in model space.
- Lineweight in plot preview and layout mode now behaves in a WYSIWYG manner.
- It eliminates the need for assigning widths in the plotting pen table, while maintaining that functionality, for any of the supplied line widths.

The Lineweight feature also provides for round-trip data preservation to Release 14.

Lineweight in Earlier Releases

A drawing created with any previous release of AutoCAD will not have lineweights in it. To prevent these drawings from plotting with the output device's thinnest possible line, AutoCAD 2000 will use the lineweight kLnWtByLwDefault as the default lineweight. The actual width is set by using

a new sysvar, LWDEFAULT, or by two new methods in the AcDbAppSystemVariables class:

```
Acad::ErrorStatus
setLwdefault(
    AcDb::Lineweight LW,
    Adesk::Boolean bUndo = Adesk::kFalse);

AcDb::Lineweight
lwdefault() const;
```

LW can be any of the AcDb::LineWeight enumerated values. If bUndo is Adesk::kTrue, the LWDEFAULT sysvar will be operated on by the UNDO command.

API Changes

A new enumerated type, AcDb::LineWeight, provides the values used to distinguish the various available lineweights. See "AcDb::LineWeight" on page 184 for a list of values.

For AutoLISP access via (entmake), (entmod), and (entget) the lineweight will be an integer value that corresponds to the values in the AcDb::LineWeight enum. Lineweights are identified by DXF group code 370.

To access the Lineweight functionality from ObjectARX, new methods are provided in the AcDbDatabase, AcDbEntity, and AcDbLayerTableRecord classes.

AcDbDatabase Class

The AcDbDatabase class has several new methods for working with lineweights.

There are new get and set methods for the current entity's lineweight value. (This value is also stored in the new CELWEIGHT header variable.)

```
AcDb::LineWeight
celweight() const;

Acad::ErrorStatus
setCelWeight(
    AcDb::LineWeight weight);
```

The setLineWeightDisplay() method is used to turn the display of lineweights on and off for the database, while the lineWeightDisplay() method returns whether or not lineweights are to be displayed:

```
Adesk::Boolean
lineWeightDisplay() const;
```

```
Acad::ErrorStatus
setLineWeightDisplay(
    bool showWeight);
```

`isValidLineWeight()` will return `true` for lineweights that match one of the predefined lineweights, and `false` otherwise:

```
static bool
isValidLineWeight(
    int weight);
```

weight	Lineweight in hundredths of a millimeter

The `getNearestLineWeight()` method takes a lineweight and returns the nearest `AcDb::LineWeight` enum value. For example, if you pass in 4, then the value `AcDb::kLnWt005` is returned:

```
static AcDb::LineWeight
getNearestLineWeight(
    int weight);
```

weight	Lineweight in hundredths of a millimeter

AcDbEntity Class

The `AcDbEntity` class has new get and set methods for the entity's lineweight value:

```
AcDb::LineWeight
lineWeight() const;
```

```
virtual Acad::ErrorStatus
setLineWeight(
    AcDb::LineWeight weight,
    Adesk::Boolean doSubents = Adesk::kTrue);
```

AcDbLayerTableRecord Class

The `AcDbLayerTableRecord` class has new get and set methods for the layer's lineweight value (used by entities with lineweight BYLAYER):

```
AcDb::LineWeight
lineWeight() const;
```

```
Acad::ErrorStatus
setLineWeight(
    AcDb::LineWeight weight);
```

File Formats

Lineweight information is preserved in the following file types:

- DWG – AutoCAD drawing file
- DXF – AutoCAD 2000 drawing interchange file
- DWF – AutoCAD drawing Web format file

- BMP – Device-independent bitmap file
- EPS – Encapsulated PostScript file

For Encapsulated PostScript, it is recommended that you configure for a PostScript® plotter and then plot to file.

AutoCAD 2000 will not export lineweight to the following file formats:

- DXF – AutoCAD Release 14 and AutoCAD LT®98 (and earlier) drawing interchange file
- 3DS – 3D Studio® file
- SAT – ACIS solid object file
- STL – Solid object stereo-lithography file
- WMF – Windows Metafile

DXF

CELWEIGHT is a per-document header variable and is stored in the drawing as hundredths of a millimeter, and written out in the DXF file. Every object will have an entity code with its lineweight written out in the DXF file. It is the 370 code listed below:

```
LINE
  5
4D
100
AcDbEntity
  8
0
370
    9
```

The 9 means 9/100 of a millimeter.

OLE Scaling and Printing

AutoCAD 2000 provides new features for working with Object Linking and Embedding (OLE). These features deal with scaling and printing or plotting.

Scaling

In AutoCAD Release 14, OLE objects could only be scaled dynamically by selecting the object after it had been pasted into the drawing and then dynamically dragging one of eight handles (grips) on the object. The OLE

object didn't report its size in AutoCAD units and it didn't provide a user interface to enter the height and width of the object in AutoCAD units.

In AutoCAD 2000, OLE objects can be sized in several different ways. By default, an OLE object is assigned a default size when it is copied in the clipboard. When initially pasted into AutoCAD, that size is not translated to AutoCAD units. Instead, the object scales itself to display at "full size" in the drawing editor, based on the display resolution. The object can then be assigned a different size by any of the following:

- By entering a width and height in AutoCAD units
- By specifying a width and height scale factor (percentage of the current height and width)
- By assigning a height to a point size of one of the fonts contained in the OLE object

Thereafter it can be resized at any time by selecting the object and moving any of its eight handles, or by bringing up the Scale OLE Object dialog, which is accessed through its Shortcut menu.

Printing or Plotting

AutoCAD 2000 adds new APIs to set and get the value of the OLESTARTUP environment variable:

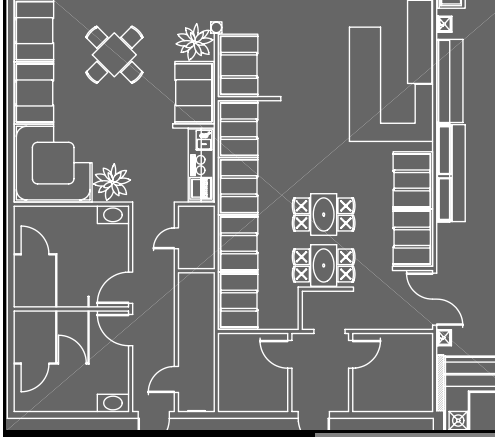
```
Adesk::Boolean
AcDbDatabase::oleStartUp() const;
```

This new method returns the current OLESTARTUP value for the database. If OLESTARTUP is `kTrue` when AutoCAD prints or plots a drawing, AutoCAD will launch the application corresponding to any OLE object in the drawing. This is done to get higher-quality output for the OLE object, at the cost of speed.

```
Acad::ErrorStatus
AcDbDatabase::setOleStartUp(
    Adesk::Boolean val);
```

This method sets `val` to be the current OLESTARTUP value for the database. See appendix B of the *AutoCAD Command Reference* manual for information on OLESTARTUP. The method returns `Acad::eOk` if successful.

Collaboration Features



In This Chapter

7

- Reference Editing
- Internet

This chapter describes the new ways that AutoCAD allows you to share information between drawings and between databases by using the new reference editing capabilities. New features to access the Internet are also explained here.

Reference Editing

To increase AutoCAD's extensibility and aid collaboration in the design process, AutoCAD 2000 further enhances the reference capabilities in AutoCAD. Users may select a block reference or external reference (xref) and specify a portion of the reference to edit in-place. The selected geometry will be instantiated in the current drawing, edited, then saved back to the block definition or reference file.

In addition, AutoCAD 2000 provides APIs to

- Allow a nested selection to be instantiated in the current drawing for editing while maintaining links to the original block definition.
- Provide mechanisms to lock reference files when the drawing is checked out, and release the lock after it is checked back in. As in Release 14, only the model space geometry in an xref can be referenced in the current drawing.
- Maintain a working set of the objects instantiated in the current drawing and accommodate changes to the data, allow additions or deletions to the working set, and update the reference (block) definition.
- Allow changes to the updated block definition to be saved back to the externally referenced file.

With these APIs, you can create your own commands to edit block definitions directly and save back changes. You can use the checkout and instantiation methods, and the checkin and save back mechanisms to alter block definitions and external reference files. We also provide access to the Index and Filter classes used to implement xref clipping in Release 14, and provide API facilities so you can add your own classes.

The result is that users can perform minor edits on references by editing reference instances in-place, through a very light and streamlined UI.

API Overview

The API aspects are divided into the following areas of functionality:

- Index and Filter Classes
- Concrete Default Classes
- Long Transactions
- Xref Pre- and Post-Processing
- File Locking and Consistency Checks

These functional areas are described in the following sections. Note that only an overview is given for each of the classes. For detailed information about a class's methods, see the *ObjectARX Reference*.

Index and Filter Classes

The classes and functions described in this section provide a scheme for applications to define custom indexes and custom filtering of block data. An application can define its custom implementations of `AcDbFilter`, `AcDbIndex`, and `AcDbFilteredBlockIterator`. It can also register the `AcDbFilter` with a block reference with the `AcIndexFilterManager::addFilter()` method, and register an `AcDbIndex` with the corresponding block table record through the `AcIndexFilterManager::addIndex()` method. From that point on, regens of xrefs and blocks will respect the query defined by the `AcDbFilter`, and use the `AcDbFilteredBlockIterator` to decide what object IDs will be processed during regen.

The indexes will be kept up-to-date through one of two methods. The application may explicitly call `AcIndexFilterManager::updateIndexes()`, or the application can rely on the AutoCAD Save operation calling `AcIndexFilterManager::updateIndexes()` on the `AcDbDatabase` being saved.

The `AcDbIndex::rebuildFull()` or the `AcDbIndex::rebuildModified()` method gets invoked during the `AcIndexFilterManager::updateIndexes()` call.

A current use of the indexing scheme in AutoCAD is fast demand loading of clipped xrefs. A spatial index (an `AcDbSpatialIndex`) is stored in the xrefed drawing. An `AcDbSpatialFilter` defines the clip volume of the block reference to the xref in the host drawing. When demand loading is turned on for the xref, the spatial filter volume is used to traverse the xref data through the spatial index, to page in from the DWG file only those entities whose graphics intersect the clip volume.

The main classes and namespaces involved are

- `AcDbIndexFilterManager`
- `AcDbIndex`
- `AcDbFilter`
- `AcDbFilteredBlockIterator`
- `AcDbCompositeFilteredBlockIterator`

These methods and functions provide an interface for

- Updating indexes
- Adding and removing indexes to block table records
- Adding and removing filters to block references
- Querying for indexes from block table records
- Querying for filters from block references
- Iterating through block table records and visiting only a subset of entities

AcIndexFilterManager Namespace

This collection of functions provides index and filter access and maintenance functionality.

AcDbIndex Class

This is the base class for Index objects. `AcDbSpatialIndex` and `AcDbLayerIndex` derive from this class. `AcDbIndex` is an abstract base class.

Keeping the index up-to-date is achieved through the `AcIndexFilterManager::updateIndexes()` calls being explicitly invoked (either by an application or AutoCAD).

The `AcDbFilteredBlockIterator` serves as the means to visit all the `AcDbObjectId` items that are "hits" from the query, which was defined by the `AcDbFilter`, passed to its constructor. For example, in the spatial index case, the `AcDbSpatialFilter` object instance passed to the `newIterator()` method will define a query region. The `AcDbSpatialIndex` object, through its `newIterator()` method, will provide an `AcDbSpatialIndexIterator` that will return object IDs that correspond to entities that fit within the query volume.

AcDbIndexUpdateData Class

This class is used to serve as a mapping between an `AcDbObjectId` and any index-specific data or flags. For example, during spatial index update, one could associate an entity's extents with the `AcDbObjectId` within this instance.

AcDbIndexUpdateDataIterator Class

This class is used for iteration through the `AcDbIndexUpdateData` class.

AcDbBlockChangeIterator Class

This class is used for iterating through modified entities. This has to be efficient, and is tied to the mechanism used for incremental saves. Access to the internal modified list data for custom Index objects is read-only.

AcDbFilter Class

This class defines a query and provides the key to the `AcDbCompositeFilteredBlockIterator` for which the corresponding index is obtained through the `indexClass()` method. This means that the `AcDbSpatialFilter::indexClass()` method will return `AcDbSpatialIndex::desc()`.

Applications that want to provide their own indexing scheme need to provide their versions of three classes:

- `AcDbIndex`
- `AcDbFilter`
- `AcDbFilteredBlockIterator`

See also the `AcDbCompositeFilteredBlockIterator` class.

AcDbFilteredBlockIterator Class

This class provides methods to process a query on an index, and is used by the `AcDbCompositeFilteredBlockIterator` class. This is an abstract base class. Typically, concrete `AcDbFilteredBlockIterator` objects will be constructed through the concrete implementation of the `AcDbIndex::newIterator()` or `AcDbFilter::newIterator()` methods, which would use more specific constructors of the `AcDbFilteredBlockIterator`-derived custom iterator.

AcDbCompositeFilteredBlockIterator Class

This is the class that provides the alternative to normal block iteration. By providing the filter list in the `init()` method, the `AcDbCompositeFilteredBlockIterator` object looks for corresponding `AcDbIndex`-derived objects through the `AcDbFilter::indexClass()` method, and creates `AcDbFilteredBlockIterator` objects. If the matching up-to-date `indexClass()` objects are not available, it creates an `AcDbFilteredBlockIterator` through the `AcDbFilter::newIterator()` method. It then orders the composition of the `AcDbFilteredBlockIterator` objects based on the `AcDbFilteredBlockIterator::estimatedHits()` and `AcDbFilteredBlockIterator::buffersForComposition()` methods. The collection of filters is a conjunction of conditions. That is, an ID is output from the iterator only if the `accepts()` method of each filter would accept the ID.

This class is an abstract base class. The method to construct an instance of `AcDbCompositeFilteredBlockIterator` is through the static method `newIterator()`.

Concrete Default Classes

In AutoCAD, layer index and spatial index objects are optionally saved in the drawing for purposes of xref demand loading efficiency. This section describes those classes.

AcDbLayerFilter Class

This is the list of layers that need to have their corresponding entity lists traversed during filtered block traversal. The `newIterator()` method does return a pointer to a valid `AcDbFilteredBlockIterator` object.

The DXF definition is just a sequence of layer names whose entities are to be traversed. For example

```
100
AcDbFilter
100
AcDbLayerFilter
8
Layer1
8
Layer2
```

AcDbLayerIndex Class

The `AcDbLayerIndex` class is derived from the `AcDbIndex` class and provides no additional methods. See the `AcDbIndex` description in the *ObjectARX Reference* for more information.

AcDbLayerIndexIterator Class

The `AcDbLayerIndexIterator` class is derived from the `AcDbFilteredBlockIterator` class and does not add any new functionality.

AcDbSpatialFilter Class

The `AcDbSpatialFilter` class defines a clip volume. It is used in AutoCAD to define the xclip boundary and volume.

AcDbSpatialIndex Class

The DXF data for this class is simply the hex of the binary compressed definition in the DWG file. In some respects, it is similar to that of ACIS entities. The DXF format is

```
0
SPATIAL_INDEX
  5
57
102
{ACAD_REACTORS
330
```

```
56
102
}
330
56
100
AcDbIndex
 40
2451108.397619861
100
AcDbSpatialIndex
 40
780755846.246974
 40
8.276642
 40
625199846.949926
 40
-0.079514
 40
2165225144377.788
 40
-0.001755
 90
        0
 90
      126
310
01FF0000FFFF0000FFFF0000FFFF010000045200CDCD000020000000108000002
0109000002010A000002010B000002020109000001080000020109000002010A00
0002010B00000202010A000001080000020109000002010A000002010B00000202
010B000001080000020109000002010A000002010B000002020200
  1
END ACDBSPATIALINDEX BINARY DATA
  0
ENDSEC
  0
EOF
```

The first six 40 group codes are used to convert from floating point to an internal integer-based compression scheme. The next 90 group code is the number of entities with unknown extents, followed by an optional list of 330 group code entity handles. The remainder is the compressed binary data in chunks, followed by the string "END ACDBSPATIALINDEX BINARY DATA".

AcDbSpatialIndexIterator Class

The `AcDbSpatialIndexIterator` class is derived from the `AcDbFilteredBlockIterator` class and does not add any new functionality. See the *ObjectARX Reference* for a description of this class.

Long Transactions

These classes and methods provide a scheme for applications to check out entities for editing and check them back into the location they came from, replacing the original objects with the edited ones. There are three types of checkouts:

- From a normal block within the same drawing
- From an xref of the drawing
- From an unrelated, temporary database

In all cases, the checked-out entities are added to the model space of the destination database. Also included here is a utility function that deep clones objects between databases. It is a complement to the current `deepCloneObjects()` function. The `queueBlockReferencesForRegen()` method is also added here; it provides a way to avoid full regeneration each time a long transaction is checked back in.

The main classes/methods are

- `AcDbLongTransaction`
- `AcDbLongTransWorkSetIterator`
- `AcLongTransactionReactor`
- `AcLongTransactionManager`
- `AcDbDatabase::wblockCloneObjects()`
- `AcDbBlockTableRecord::queueBlockReferencesForRegen()`

AcDbLongTransaction Class

This class contains the information needed to track a long transaction. Currently, the `AcLongTransactionManager` class takes the responsibility for creating and appending `AcDbLongTransaction` objects to the database. It then returns the `AcDbObjectId` of the object. Like all other database-resident objects, its destruction is handled by the database. This means that construction is handled by the `AcLongTransactionManager` class constructor, and destruction is handled by the `AcDbDatabase` in which the object has been appended.

Note that while `AcDbLongTransaction` objects are added to a database while they are active, they are erased once the transaction has completed, and are not stored in DWG or DXF files. They are not persistent.

The `AcDbDatabase::wblock()` method will not be blocked during a checkout.

AcDbLongTransWorkSetIterator Class

This class provides read-only access to the objects in the work set. In its construction in `AcDbLongTransaction::newWorkSetIterator()`, it can be set up to include only the active work set, or also include objects that were added to the work set because they are referenced by objects in the work set (secondary objects), and/or objects that have been removed from the work set, either by `AcDbLongTransaction::removeFromWorkSet()`, or by being erased.

AcLongTransactionReactor Class

The `AcLongTransactionReactor` class provides notification specific to Long Transaction operations. It is designed to be used in conjunction with the `deepClone` notifications that will also be sent, but will vary depending on which type of checkout or checkin is being executed. To connect these notifications with the `deepClone` notifications, the `AcDbIdMapping` object used for cloning can be retrieved by calling the `activeIdMap()` method of the `AcDbLongTransaction` class.

AcLongTransactionManager Class

This class is the manager for starting and controlling long transactions. There is only one per AutoCAD session, and it is accessed via a pointer returned by the `acLongTransactionManager()` function.

AcDbDatabase::wblockCloneObjects() Method

This method deep clones objects from one database to another. It follows hard references so that all dependent objects are also cloned. A new enumeration has been created to indicate how duplicate records will be handled:

AcDb::DuplicateRecordCloning

Name	Value
kDrcNotApplicable	0
KDrcIgnore	1
KDrcReplace	2
kDrcXrefMangleName	3
kDrcMangleName	4
kDrcUnMangleName	5

```
Acad::ErrorStatus
wblockCloneObjects(
    AcDbObjectIdArray& objectIds,
    AcDbObjectId& owner,
    AcDbIdMapping& idMap,
    AcDb::DuplicateRecordCloning type,
    Adesk::Boolean deferXlation = Adesk::kFalse);
```

objectIds — List of object IDs of the objects to be cloned into the owner object.

owner — Object ID of the new owner. This must be a valid container, an `AcDbDictionary` for objects, or an `AcDbBlockTableRecord` for entities.

idMap — The `AcDbIdMapping` object to use for the cloning.

type — The `DuplicateRecordCloning` value to use for the cloning. There are four acceptable values:

- `AcDb::kDrcIgnore` – Duplicate symbols are not cloned. This is similar to how the INSERT command behaves.
- `AcDb::kDrcReplace` – Duplicate symbols are cloned and replace the existing symbol in the destination database.
- `AcDb::kDrcMangleName` – Duplicate symbol names are mangled with 0 prepended.
- `AcDb::kDrcUnMangleName` – Similar to `kDrcIgnore` except that any mangled names found will first have their mangling removed. This is the mode used during reference checkin.

deferXlation — Just like `deepCloneObjects()`, if `Adesk::kTrue`, the `deepCloneXlation` is not performed before returning. This makes it possible to call the function multiple times with different owner objects. The final call must have `deferXlation` set to `Adesk::kFalse`.

The behavior of symbol table records, when duplicates are found, is determined by the `type` parameter. The `type` parameter has been added to the `AcDbIdMapping` class as a separate flag. The following chart shows the rela-

tionship of the `type` parameter to the existing DeepCloneTypes and commands:

Duplicate record handling		
Commands	**DeepCloneType**	**SymbolCloneType**
COPY	kDcCopy	kDrcNotApplicable
EXPLODE	kDcExplode	kDrcNotApplicable
BLOCK	kDcBlock	kDrcNotApplicable
BIND	kDcXrefBind	kDrcXrefMangleName
INSERT/BIND	kDcXrefInsert	kDrcIgnore
XRESOLVE	kDcSymTableMerge	kDrcXrefMangleName
INSERT	kDcInsert	kDrcIgnore
insert()	kDcInsertCopy	kDrcIngore
WBLOCK	kDcWblock	kDrcNotApplicable
deepCloneObjects()	kDcObjects	kDrcNotApplicable
wblockObjects()	kDcObjects	kDrcIgnore
wblockObjects()	kDcObjects	kDrcReplace
wblockObjects()	kDcObjects	kDrcMangleName
wblockObjects()	kDcObjects	kDrcUnmangleName

The method has several possible return values, described in the following table:

Return values	
Error Code	**Explanation**
Various Open errors	If the owner cannot be opened for Write, or the objects to be cloned cannot be opened for Read.
eInvalidOwnerObject	If the owner is not an AcDbDictionary or AcDbBlockTableRecord.
eWrongDatabase	If the function is called multiple times with different owners, and they are not all from the same database.
eInvalidInput	If the SymbolCloneType is not set to one of the three acceptable types, AcDb::kScIgnorDups, AcDb::kScReplaceDups, or AcDb::kScMangle.

AcDbBlockTableRecord::queueBlockReferencesForRegen() Method

From the block table record, this function follows up all of its parent trees to find all root level block references and queues them for graphics update. Modification to a block table record, or any of the entities it contains, will not queue the references for update. If an error is returned, because a parent object along the tree could not be opened, either the object must be closed and the function called again, or a regen all must be initiated in order to have the modifications regenerated.

Xref Pre- and Post-Processing

Xref pre- and post-processing makes it possible to restore an attached xref's in-memory `AcDbDatabase` so that it can be saved back to a file. During xref resolution many symbol table records are mangled, and some erased. Historically, this was done to simplify the resolve process, and was acceptable because the databases were read-only. This processing makes it possible to temporarily reverse the resolution changes so that the xref database can be modified and written back to its file.

This functionality has been added to `AcDbDatabase`. The methods include a utility function to find the associated block table record from an xref database, as well as the ability to restore the resolved xref, and to reset it back to the proper resolved condition after restoration.

The customary usage for these methods is to do the restore to Original Symbols, make the modifications to the database, save the database, and then restore the Forwarded Symbols. These steps must be written into a single block of code, such that there are no attempts to regenerate the host drawing, execute any xref commands, or provide user prompts while the xref database is in its restored condition.

The methods are

- `AcDbDatabase::xrefBlockId()`
- `AcDbDatabase::restoreOriginalXrefSymbols()`
- `AcDbDatabase::restoreForwardingXrefSymbols()`

See the *ObjectARX Reference* for the calling sequence and complete description of these methods.

File Locking and Consistency Checks

Reference editing now includes support for file locking and xref reloading.

AcEdXrefFileLock Class

This base class handles the management of xref file locking. Its main purpose is to prepare the xref block in a drawing for in-place editing, though it can be used for other purposes. It is assumed that these xref file methods will be operating on the current database drawing. Refer to the *ObjectARX Reference* for a complete description of the class and its methods.

acedXrefReload() Function

This function processes the list of xref block table record IDs for xref reload. It is assumed that each xref block table record ID references an xref drawing file that can be reloaded to the current drawing. It has the same functionality as the ACAD XREF subcommand for Reload. See the *ObjectARX Reference* for additional information on `acedXrefReload()`.

Internet

The Internet capabilities of AutoCAD 2000 have been enhanced with three new features:

- Electronic plotting (ePlot)
- URLs for file I/O
- Hyperlinks

Electronic plotting uses the Hardcopy API; URLs (Uniform Resource Locators) for file I/O are described as part of the host application services. (See "Internet Methods" on page 124 in chapter 8, "Architectural Features.") Hyperlinks (URLs) are also available for general use in third-party software, and are described in this section.

Hyperlinks

The Hyperlink feature has been enhanced in AutoCAD 2000. The concept of URLs has been expanded to hyperlink; the scope has been expanded to include non-Web hyperlinks such as local files. It is also easier now to attach, view, edit, and list hyperlinks within your application. The Hyperlink APIs are defined in the *achapi.h* header file. To use these new APIs, you must include *achapi.h*, link to *achapi.lib*, and ensure that *achapi.dbx* is loaded.

An overview of the hyperlink classes follows, but for complete information on the classes and their methods, see the *ObjectARX Reference*.

AcDbEntityHyperlinkPE Class

The methods of the `AcDbEntityHyperlinkPE` class allow you to set, get, and count the hyperlinks associated with an entity. The information associated with hyperlinks consists of the name of the link, a description of the link, and a sublocation within the link. For AutoCAD, a sublocation is a named view, while in a spreadsheet application, for example, a sublocation might be a cell or group of cells.

AcDbHyperlink Class

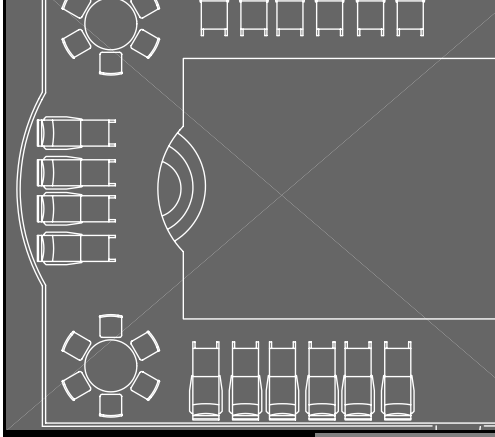
An `AcDbHyperlink` object contains the hyperlink name (for example, *http://www.autodesk.com*), a sublocation within that link, the hyperlink description or friendly name ("Click here for Autodesk"), and a display string for the hyperlink. The display string is usually the same as the hyperlink's description. If the description is null, the hyperlink's name and sublocation are used instead, in "name – sublocation" format.

Hyperlinks may also have nesting levels. Nesting level is only of interest when dealing with hyperlink collections associated with an entity within a block, or with collections associated with an INSERT entity.

AcDbHyperlinkCollection Class

This class is a collection of `AcDbHyperlink` objects, and has a variety of methods for adding and removing those objects. The `AcDbHyperlinkCollection` deletes its contents when they are removed, and when the collection object itself is deleted. Hyperlinks in the collection are numbered from 0.

Architectural Features



In This Chapter

8

This chapter contains descriptions of the features that involve architectural changes to AutoCAD.

- Release 12 Objectification
- Extended Symbol Names
- Partial Loading
- ObjectDBX
- DXF
- AcGi Library
- ACIS

Release 12 Objectification

Application developers for AutoCAD now have the ability to define their own entity types by deriving from the `AcDbEntity` class and from the classes in the class hierarchy underneath `AcDbEntity`. Before this release of AutoCAD, developers could not safely derive new entity classes from what is called the "R12 entity set." The Release 12 entity set consists of entity types that were present in AutoCAD Release 12 and earlier. The classes for these entity types now have all of their methods implemented, even key methods such as `worldDraw()`.

In implementing the methods in the Release 12 entity classes, three areas were addressed:

- The `AcDbObject` base class methods
- The `AcDbEntity` base class methods
- The `AcDbCurve` base class methods for those classes derived from `AcDbCurve`

The `dxfInFields` methods (`AcDbObject` base class) and the `worldDraw()`/`viewportDraw()` methods (`AcDbEntity` base class) have been implemented. Also, the `getOsnapPoints()`, `getGripPoints()`, `getStretchPoints()`, `moveGripPointsAt()`, `moveStretchPointsAt()`, `explode()`, `list()`, `audit()`, and `getTransformedCopy()` methods have been implemented for all the Release 12 entity classes.

The required base class methods have also been implemented for the following Release 12 entity classes derived from `AcDbCurve`:

- `AcDbArc`
- `AcDbCircle`
- `AcDbLine`
- `AcDb2dPolyline`
- `AcDb3dPolyline`
- `AcDbPolyMesh`
- `AcDbPolyFaceMesh`

You are now allowed to derive from the `AcDbArc`, `AcDbCircle`, and `AcDbLine` classes. You still cannot derive from the `AcDb2dPolyline`, `AcDb3dPolyline`, `AcDbPolyMesh`, or `AcDbPolyFaceMesh` classes.

Changed Items

Some existing components of ObjectARX were changed during Objectification. These are described in this section.

DXF File Format Changes

As part of the `AcDbViewport` objectification, the MVIEW xdata, stored in viewports since their inception in Release 11, is being moved over to be true class member data. Now that the MVIEW data for viewports is stored as true member data, it will appear in AutoCAD 2000 DXF files as normal entity data rather than xdata as in previous AutoCAD DXF file formats. It will still appear as xdata when written to a pre-AutoCAD 2000 DXF format. The new DXF codes for this data are listed in the following table:

New DXF group codes for MVIEW xdata

DXF Code	Data Represented
12,22	View center point (2d)
13,23	Snap base point (2d)
14,24	Snap spacing (14 == *X* spacing, 24 == *Y* spacing)
15,25	Grid spacing (15 == *X* spacing, 25 == *Y* spacing)
16,26,36	View direction vector (3d)
17,27,37	View target point (3d)
42	View lens length
43	Front clip plane *Z* value
44	Back clip plane *Z* value
45	View height (in model space units)
50	Snap angle
51	View twist angle
72	Circle zoom percent
341	(Repeating) viewpoint frozen layer object IDs
90	Flag bits

Some of the bits stored with the DXF 90 group code are for MVIEW data, while others are for non-rectangular viewport data. The MVIEW data bits include:

MVIEW xdata flag bits

Position	Value If Set
Bit 8	Perspective mode is on
Bit 9	Front clipping is on
Bit 10	Back clipping is on
Bit 11	UCS follow is on
Bit 12	Front clip not at eye
Bit 13	UCS icon is visible
Bit 14	UCS icon is at origin
Bit 15	Fast zoom is on
Bit 16	Snap is on
Bit 17	Grid is on
Bit 18	Snap type isometric unset is standard style
Bit 19	Hideplot is on
Bits 20 and 21	Isopair. If bit 20 is set and bit 21 is not set, then it is isopair top. If bit 20 is not set and bit 21 is set, then it is isopair right. If both bits 20 and 21 are set, then it is isopair left.

These data values were chosen to match, wherever possible, the same data items in the `AcDbViewportTableRecord` class. There are other new codes for non-rectangular viewport data. See chapter 6, "Paper Space Layouts."

(entget) and acdbEntGetX() Data List Changes

All of the new viewport MVIEW data members will be written out as normal entity data in the data list returned by `(entget)` and `acdbEntGetX()`. In addition, to maintain compatibility with old LISP and ADS code, `(entget)` and `acdbEntGetX()` calls that request the xdata for the ACAD regapp will get back the MVIEW data in xdata format as well. This provides upward compatibility of old code, while also allowing new code to make use of the easier data access provided by the new normal entity data fields.

copyFrom() Method

`AcDbPolyFaceMesh::copyFrom()` and `AcDb2dPolyline::copyFrom()` are now implemented so that they will correctly and safely work on both database-resident and non-database-resident polylines and meshes including copying between database-resident and non-database-resident versions. Xdata, extension dictionaries, and persistent reactors will not be copied, but copies of all vertices will be made, and if the destination polyline has more vertices than the source, the extra vertices will be removed if the destination polyline is not database-resident, or erased if it is database-resident. The `copyFrom()` methods of `AcDbBlockReference` and `AcDbMInsertBlock` also work and will copy any `AcDbAttributes` owned by the block reference or Minsert as done for polyline vertices.

The signature of `AcRxObject::copyFrom()` has been changed from

```
virtual void
copyFrom(
    const AcRxObject* other);
```

to

```
virtual Acad::ErrorStatus
copyFrom(
    const AcRxObject* other);
```

This change was made because there are situations where `copyFrom()` may fail; in such situations, it is useful to be able to let the caller know that something went wrong. This signature change will cause compilation errors in ObjectARX applications that override this method in any subclasses. To fix this, simply change your function to return `Acad::eOk`, or take advantage of the ability to return a meaningful `ErrorStatus` value and modify your methods accordingly.

Removed Items

These methods have been removed:

```
AcDbPolyFaceMeshVertex::vertexType()
AcDbPolyFaceMeshVertex::setVertexType()
```

These methods were no longer necessary because this vertex class can only have one type of vertex: `AcDb::k3dSimpleVertex`. If your application uses these methods, you will see compilation errors. Just remove the calls from your code to fix the errors. Note that the `setVertexType()` method was available only in the ObjectDBX (formerly DWG Unplugged) product.

New Items

```
virtual Adesk::Boolean
AcDbDxfFiler::isModifyingExistingObject() const;
```

This function returns `Adesk::kTrue` if the filer is being used for an entmod operation or `Adesk::kFalse` if not. (An entmod operation occurs when an object modification is made via the AutoLISP `(entmod)` function or the `acdbEntMod()` function in ObjectARX.)

```
void
AcDbObjectId::convertToRedirectedId();
```

When an object is in a drawing that is an xref in a host drawing, the object is either copied into the host drawing (in which case it is said to be "redirected" and its object ID is said to be a "redirected ID") or the actual object from the xref is added to the host drawing (that is, given a handle and object ID in the new drawing), in which case it's said to be "commandeered."

If the `AcDbObjectId` object that this method is called for is the object ID of an object in an xref database and that object has been redirected to the host drawing, then this method will change the value in the `AcDbObjectId` object to the value of the object ID at the end of the redirected chain (the object ID in the host drawing). If the `AcDbObjectId` object on which this is called is not redirected, or if it has a value of `AcDbObjectId::kNull`, then this function will do nothing.

```
AcDbDatabase*
AcDbObjectId::database();
```

Returns a pointer to the database associated with this object ID.

```
Acad::ErrorStatus
AcDb2dPolyline::straighten();
```

Removes all curve and spline fit vertices, sets all spline control vertices to be simple vertices, and sets the polyline type to be `AcDb::k2dSimplePoly`. This is the same as the PEDIT command's DECURVE option.

```
Acad::ErrorStatus
AcDb2dPolyline::curveFit();
```

Curve fits the polyline. This is the same as the PEDIT command's FIT option.

```
Acad::ErrorStatus
AcDb2dPolyline::splineFit();
```

Spline fits the polyline using the polyline's database (or the working database if the polyline is not yet in a database) `splineType` and `splineSegs` values. If `splineSegs` is negative, then its absolute value is used. This is the same as the PEDIT command's SPLINE option.

```
Acad::ErrorStatus
AcDb3dPolyline::straighten();
```

Removes all spline fit vertices, sets all spline control vertices to be simple vertices, and sets the polyline type to be `AcDb::k3dSimplePoly`. This is the same as the PEDIT command's DECURVE option.

```
Acad::ErrorStatus
AcDb3dPolyline::splineFit()
```

Spline fits the polyline using the polyline's database (or the working database if the polyline is not yet in a database) `splineType` and `splineSegs` values. If `splineSegs` is negative, then the polyline is curve fit after being spline fit. This is the same as the PEDIT command's SPLINE option.

```
Acad::ErrorStatus
AcDbPolygonMesh::straighten();
```

Removes all surface fit vertices, sets all control vertices to be simple vertices, and sets the polyMesh type to be `AcDb::kSimpleMesh`. This is the same as the PEDIT command's DECURVE option.

```
Acad::ErrorStatus
AcDbPolygonMesh::surfaceFit();
```

Surface fits the polygonMesh using the mesh's database (or the working database if the mesh is not yet in a database) `surfType`, `surfu`, and `surfv` values. This is the same as the PEDIT command's SMOOTH SURFACE option.

Extended Symbol Names

The allowable length of symbol names has been increased in AutoCAD 2000 from 31 characters to 255. This increase has an effect in several areas, such as symbol table records, xrefs, dictionary entries, some system variables, DXF usage, and user interface design. In addition, the set of characters permitted in these names has been expanded, and letter case is now preserved.

AutoCAD 2000 extends symbol table record names to make them more descriptive. Symbol table record names, which we'll refer to throughout this document as symbol names, are the identifiers that name objects such as layers, blocks, and linetypes. In prior releases, symbol names conformed to a strict set of rules governing their length and content. In AutoCAD 2000, these restrictions have been relaxed to allow longer and more descriptive names. These new, longer, and more descriptive symbol names are referred to as Extended Symbol Names (ESNs). When referring to the symbol names from the earlier releases, the term *legacy symbol names* will be used. Symbol

names will refer to both the extended symbol names and the legacy symbol names.

Extended and Legacy Symbol Names

The legacy and extended symbol names differ in three respects:

- The maximum number of characters allowed in the symbol name.

  The maximum length takes into account that the symbol name may contain code-page independent format (CIF) and multibyte interchange format (MIF) string sequences. Thus, when AutoCAD checks the length, it effectively counts each of these sequences as one character. AutoCAD also considers the two bytes of a double-byte character to be one character. Therefore, it is possible for a symbol name to consist of a larger number of bytes than characters.
- The set of characters you can use in the symbol name.

  Legacy symbol names can only contain characters from a small set of allowable characters. In AutoCAD 2000, the set of allowable characters has been expanded.
- How AutoCAD treats lowercase and uppercase alphabetic characters.

  Legacy symbol names were always forced to uppercase. In AutoCAD 2000, letter case is preserved.

The following table gives a summary of the rules for symbol names:

Rules for extended and legacy symbol names

Characteristic	Extended Symbol Names	Legacy Symbol Names
Maximum length	No internal limit, although AutoCAD commands that prompt for a new symbol name will enforce a 255-character limit.	At most 31 characters.
Allowed characters	Characters allowed in Windows 95 and Windows NT filenames except comma (‘,’), semicolon (‘;’), accent grave (‘‘’), and equal sign (‘=’).	Alphanumeric characters (‘0’ ... ‘9’, ‘A’ ... ‘Z’), the dollar sign (‘$’), the underscore (‘_’), the hyphen (‘-’), and the vertical bar (‘|’).
Case handling	Preserve the case of letters but ignore it during comparisons. Therefore, names are still case-insensitive (“Floor” is the same symbol name as “FLOOR”), but AutoCAD will not convert the letters to uppercase.	Lowercase letters converted to uppercase letters. All symbol name comparisons occur with uppercase symbol names and are stored in uppercase in the drawing file.

Other Changed Strings

The new rules for symbol names also apply to other kinds of strings used in AutoCAD, such as dictionary entries and system variables. This section describes these additional types of strings.

Dictionary Entry Names

In this document, any discussion of symbol names (extended and legacy) also implicitly includes dictionary entry names.

Symbol Names Not in Symbol Table Records

Extended symbol names may exist outside of symbol table records. The best example of this existence is the CLAYER system variable. This variable contains the name of the current layer. Because layers in AutoCAD 2000 are extended symbol names, this variable may now have a value that conforms to the rules of ESNs. If you had source code that assumed that the value of CLAYER was at most 31 characters, then the code may not work properly when it encounters a layer name that's longer than 31 characters. The other predefined system variables that hold symbol names include

- CELTYPE
- CMLSTYLE
- CPLOTSTYLENAME
- DIMSTYLE
- DIMTXSTY
- INSNAME
- TEXTSTYLE
- UCSNAME

Any place that may contain a symbol name may now contain an extended symbol name.

If your source code reads in AutoCAD-generated files that contain symbol names, the symbol names in these files may be extended symbol names. For example, when you DXFOUT a file in AutoCAD 2000, the symbol names in the resulting file may contain extended symbol names. Another example is the *acad.lin* file, which contains the name of linetypes. Because linetypes are symbol table records, the linetype name in the *acad.lin* file may now be longer than the legacy symbol names and may contain a wider range of characters. Another example is the ATTEXT command, which allows you to extract specific information about blocks and their attributes. The information in the output may contain symbol names, such as blocks and layers, so you will need to anticipate extended symbol names.

New Symbol Name Validation Functions

Currently, to validate symbol names, you would use either `acdbSNValid()` from ObjectARX or `snvalid` from AutoLISP. These functions have been enhanced to provide a pass or fail test for the extended-symbol-name syntax. For AutoCAD 2000, there are new ObjectARX functions allowing you to perform more specific tests on symbol names.

User Interface Guidelines

For user interfaces that display and prompt for symbol names, you mainly need to be concerned with the three changed aspects of extended symbol names: the length, the content, and the case of alphabetic letters. You should follow the appropriate user interface standards; however, keep in mind that symbol names can be extended symbol names. Most of these guidelines are intended to provide consistent user interfaces within AutoCAD and between AutoCAD and third-party applications.

Assume that you will need to work with long symbol names. This assumption means you will need to design and code dialogs, MFC controls, and other graphical user interfaces to accommodate long symbol names. For example, when you need to place a very long symbol name in a field for editing, you should allow the user to access the entire symbol name, scrolling the name horizontally to the left and right as appropriate. In list boxes, you may need to consider how to best collapse a long symbol name so that it fits within the window of the list box. If you're using symbol names as static text in dialogs, consider how you would handle a long symbol name. Extended symbol names can "bleed" out of the dialog box. For line-oriented output, such as that generated by commands entered in the command window, you need to accommodate displaying long symbol names.

You should not assume the content of the symbol names. For example, you should not design edit fields that only allow the characters valid for legacy symbol names. Instead, you should allow all characters and then call the appropriate symbol name validation functions to determine if the symbol name is valid. To assist users, when you display symbol names, you should enclose them in quotation marks in case the symbol name contains spaces or other punctuation that may be confusing when viewed in the context of the surrounding text. AutoCAD allows users to use quotation marks in several places for strings that contain spaces, such as filenames, so you can safely assume that symbol names will not contain quotation marks.

The vertical bar ('|') has a special meaning in symbol names. AutoCAD uses it to construct symbol names from xref-attached drawings. For example, suppose AutoCAD brings in a layer named "FLOOR" from an attached drawing

named “B” during an xref attach. The resulting layer in the host drawing has the name “B|FLOOR.” This new layer is now xref-dependent. AutoCAD places some restrictions on vertical bars in symbol names due to its convention of using vertical bars to indicate an xref dependency in symbol names. Symbol names may not begin or end with a vertical bar. Some AutoCAD commands will not allow users to enter symbol names with vertical bars because the commands will create a new symbol or rename an existing symbol. In these situations, users are creating new symbols, so it doesn’t make sense for their names to look like xref-dependent names.

You should only use the vertical bar in names of xref-dependent symbols. If you use the vertical bar in a symbol name that is not xref-dependent, you might confuse users. AutoCAD only uses, at most, one vertical bar in a symbol name. You should not use more than one vertical bar for xref-dependent symbol names to be consistent with AutoCAD.

The last item to consider is how to deal with letter case. AutoCAD does not differentiate between symbol names that only differ in the case of the letters, such as “House” and “HOUSE”. To AutoCAD, these two symbol names are the same. In AutoCAD 2000, AutoCAD does preserve the case of the symbol names, so that when users type in “House”, AutoCAD will store the name “House”, as opposed to converting the name to uppercase as it did in earlier releases. You should follow this convention to remain consistent with AutoCAD. Lists containing sorted symbol names should perform case-insensitive sorting so that the symbol names will display in the same order as if they were all converted to uppercase letters.

General Programming Guidelines

With the relaxed restrictions on symbol names, you will need to ensure that your source code can properly access and manipulate the extended symbol names. If you ensure that your source code can accommodate the extended symbol names, then your code will also support the legacy symbol names.

Guideline Overview

In general, you should avoid any design or coding decisions based on any of the criteria that differentiate extended and legacy symbol names:

- Length
- Content
- Case
- Character set

Such decisions will usually result in hardcoded and inflexible limitations.

Assume Very Long Symbol Names
Do not assume that symbol names have a maximum length. For example, don't use a buffer of 32 bytes to store a symbol name. Such a buffer could overflow when your code encounters extended symbol names in AutoCAD 2000. This buffer could also overflow if you encounter double-byte characters in the symbol name. For C++ code, use a dynamically growing string class like MFC's `CString` or the standard C++ library's `string` class. As a final fallback, you should use the standard C library's string routines such as `strlen()` and `strdup()` to determine the length of symbol names and dynamically allocate string buffers. Although AutoCAD will not allow users to enter new symbol names longer than 255 characters, symbol names longer than 255 characters can still be created through a combination of xref attaching and binding.

Assume Any Character Is Valid
Avoid assumptions about the content of symbol names. Defer as many decisions regarding symbol name content to AutoCAD and any validation functions available through the published programming APIs, such as `acdbSNValid()`. For example, if you previously assumed that symbol names could not have spaces, you may have written code to write symbol names to a data file delimited with spaces. With extended symbol names, it's now possible for you to write out a symbol name containing an embedded space (or even multiple embedded spaces). If you had additional source code reading this data file, it may not properly detect the difference between the space within the symbol name and the delimiter.

Although AutoCAD 2000 will still not allow certain characters to appear in symbol names, these restrictions may be lifted in future releases. By assuming any character may be valid, you avoid needing to re-examine your code if AutoCAD lifts its few remaining restrictions on symbol names. The safest assumption to make is that all characters in symbol names will be printable characters.

Assume Mixed-Case Symbol Names
In AutoCAD 2000, AutoCAD no longer forces letters in symbol names to be uppercase characters. You should assume that symbol names contain mixed-case characters. When comparing symbol names, use a case-insensitive comparison routine. Suppose you had code that required an input string consisting of only uppercase characters. Previously, you could retrieve the symbol name directly from AutoCAD and use the name in this code. Now, you must convert the retrieved symbol name to uppercase.

Anticipate Single- and Double-Byte Character Sets
Symbol names may be entered by users from different countries using different code pages and character sets. In particular, symbol names may contain double-byte characters in addition to CIF and MIF sequences. If your source

code scans symbol names for specific characters, then you should write your code to support double-byte characters. Several references exist for working with double-byte characters, such as the online references for Visual C++ (search for "double-byte character set") or Nadine Kano's book, *Developing International Software for Windows 95 and Windows NT*. Microsoft's C run-time library provides some standard routines for working with multi-byte character sets. They are declared in header file *mbstring.h*. The Microsoft MFC Library's `CString` class provides some support for double-byte characters.

ObjectARX

All objects derived from `AcDbSymbolTableRecord` and `AcDbDictionary` may now contain extended symbol names. The following table lists the ObjectARX classes directly affected by extended symbol names:

ObjectARX classes containing extended symbol names

Class Name	Description
AcDbSymbolTableRecord	Base class for all symbol table records
AcDbBlockTableRecord	Class containing a block definition
AcDbAbstractViewTableRecord	Base class for AcDbViewTableRecord and AcDbViewportTableRecord
AcDbViewTableRecord	Class containing a named view
AcDbViewportTableRecord	Class containing a tiled viewport
AcDbLinetypeTableRecord	Class containing a linetype
AcDbLayerTableRecord	Class containing a layer
AcDbTextStyleTableRecord	Class containing a textstyle
AcDbUCSTableRecord	Class containing a UCS
AcDbRegAppTableRecord	Contains the name of a registered application. You use the names to access Xdata
AcDbDimStyleTableRecord	Class containing a dimension style
AcDbDictionary	General-purpose container class storing AcDbObjects and keyed on a string

AutoLISP

AutoLISP code does not need to be written to consider fixed-sized buffer problems because the language is mainly built around the notion of atoms and lists of arbitrary size. You do need to worry about whether your AutoLISP code is making assumptions about the characters in the symbol name. To validate strings as symbol names, you can use the `snvalid` function.

Chapter 9 of the *Customization Guide*, "Selection Set, Object, and Symbol Table Functions," and the section "Symbol-Handling Functions" in chapter 12, "AutoLISP Function Synopsis," list the functions that operate on symbol tables and symbol table records. Some of the functions that use symbol names include `entmake`, `entmod`, `tblsearch`, `tblobjname`, and `tblnext`. You will need to be aware of symbol names in system variables (the `getvar` and `setvar` functions) and other areas that may not be directly connected to symbol table records, such as the layer name in the list returned by `entget`.

Partial Loading

AutoCAD 2000 provides substantial improvements in drawing load times when working with very large drawings (10 MB and up) by allowing the user to "partially open" a drawing. It is possible to specify a portion of the drawing to edit, and only the area or layers specified are read into the drawing editor. This section describes the partial load feature.

Only AutoCAD 2000 drawings can be partially loaded, and they can only be saved in the AutoCAD 2000 format. Partial loading is disabled when recovering a drawing, and a drawing file is always locked when it is partially loaded, even when it is loaded in Read-only mode. Also, while any number of users may simultaneously have a single file partially opened in Read-only mode, if any user has a file opened partially for reading and writing, then no one else may have the drawing opened in *any* mode.

API Information

New methods have been added to the `AcDbDatabase` and `AcEditorReactor` classes.

AcDbDatabase Class Changes

```
Adesk::Boolean
AcDbDatabase::isPartiallyOpened() const;
```

Returns `Adesk::kTrue` if the database is in a partially opened state, and `Adesk::kFalse` otherwise.

```
Acad::ErrorStatus
AcDbDatabase::applyPartialOpenFilters(
    const AcDbSpatialFilter* pSpatialFilter,
    const AcDbLayerFilter* pLayerFilter);
```

pSpatialFilter — Specifies the model space volume to be used for spatial filtering. The coordinates are in WCS. A null pointer will result in no spatial filtering, meaning that objects at *all* locations in model space that pass the layer filtering will be brought in.

pLayerFilter — Specifies the layers to be used for filtering. A null pointer will result in objects on *all* layers that pass spatial filtering being brought in. If the array is empty, then no layers will be used for filtering and no entities in model space except those that have a hard pointer dependence for the rest of the drawing will be brought in.

This function can be called on a database to filter its model space entities to only bring part of the drawing into memory. The filter applied is the intersection of entities in the view and entities on the specified layers. Thus, only entities both within the view and on the specified layers will be brought into memory. To be effective, this method must be called after an `AcDbDatabase::readDwgFile()` call and before calling `closeInput()` on the database. This function can also be called on a partially opened database to read additional entities off the disk into the drawing.

Returns `Acad::eOk` if successful. If `applyPartialOpenFilters()` is unsuccessful, then the model space of the database will not be filtered.

```
void
AcDbDatabase::disablePartialOpen();
```

When called during partial open notification, this function vetoes the partial open operation.

AcEditorReactor Class Changes

```
virtual void
AcEditorReactor::partialOpenNotice(
    AcDbDatabase* pDatabase);
```

pDatabase — Specifies the `AcDbDatabase` involved in the operation

All loaded ARX applications that add an `AcEditorReactor` will get a new notification before a partial open of an `AcDbDatabase` just before an attempt to partially open it. At that time any of the currently loaded applications can veto the operation. To do this, an application should override the `partialOpenNotice()` method of `AcEditorReactor` in its derived class and

call `enablePartialOpen(Adesk::kFalse)` on the database pointer passed in, as shown here:

```
class AppClassDerivedFromAcEditorReactor : public AcEditorReactor
{
    virtual void partialOpenNotice(AcDbDatabase* pDb);
};

void
AppClassDerivedFromAcEditorReactor::partialOpenNotice(
    AcDbDatabase* pDb)
{
    pDb->enablePartialOpen(Adesk::kFalse);
};
```

ObjectDBX

The ObjectDBX SDK is the interface among host applications, drawing (*.dwg*) files, custom application (*.arx*) files, and custom object (*.dbx*) files.

Overview

ObjectDBX comprises a set of DLLs that can be used to implement custom objects contained in an AutoCAD 2000 drawing file, and to implement graphics applications that work without the presence of AutoCAD. Part of this capability was formerly presented in the DWG Unplugged product, but the ObjectDBX SDK replaces and goes beyond the DWG Unplugged technology by providing the support necessary for intelligent object systems. The ObjectDBX SDK allows you to create non-AutoCAD host applications that can read and write DWG files.

Host Applications

A "host application" is one that contains a `main()`, `WinMain()`, or `dllMain()` function in its code, and provides the host services that an ObjectDBX or ObjectARX program needs.

Two types of host applications can take advantage of the interface ObjectDBX provides: AutoCAD 2000, with or without associated ObjectARX applications, and a non-AutoCAD host application, which cannot load an ObjectARX application can only take advantage of the specific interfaces provided by the DLLs contained in ObjectDBX.

WARNING! Any host application (that is, an executable file with an *.exe*, *.dll*, or *.ocx* extension) must link with *acdb.lib* first, followed by *acutil.lib*, *acrx.lib*, and any other libraries. If *acdb.lib* is not the first library linked in, the load will fail due to dependencies between the libraries. The order after *acdb.lib* is not important.

DBX Libraries

DBX libraries contain the intelligence allowing custom objects (geometry, components, non-graphic objects, and so on) to operate as object extensions inside AutoCAD. These files are given the extension *.dbx* (DataBase eXtension). A DBX file is an ObjectARX application, written to work with the ObjectDBX libraries instead of with the ObjectARX (AutoCAD) libraries.

User Interface and Database Access

ObjectDBX allows you to write separate executables for the user-interface (UI) and database (DB) portions of your application. The file containing the UI component has the extension *.arx* and contains the code that issues prompts, displays dialogs, modifies menus, and so forth. The DB component file has the extension *.dbx* and contains the code that implements your custom objects by creating them, displaying them, transforming them, and so on. If your application is separated into user-interface and database portions, your custom objects will still be handled properly without the presence of the ObjectARX application that provided the user interface.

For example, suppose you implement a custom object called "Sink," and that the code to display and modify Sink is in *sinkDB.dbx*, while the code to prompt the user for Sink creation values is in *sinkUI.arx*. Your user can load *sinkUI.arx* from AutoCAD and use it (with *sinkDB.dbx*, which can be loaded automatically) to create a custom sink in a drawing. Later, that drawing can be loaded by any other host application (including AutoCAD), and the Sinks will display properly, instead of as proxies, if the user has a copy of the *sinkDB.dbx* file available.

NOTE *sinkDB.dbx* will be loaded automatically if *sinkUI.arx* is linked to *sinkDB.lib*, or if the code internal to *sinkUI.arx* makes a call to `acrxLoadModule(sinkDB.dbx)`. The load is automatic to the user, but you still have to put one of these mechanisms in place to make sure that it happens.

Both of the sink components must be linked to *rxapi.lib*, and both components will have *.def* files that export `acrxGetApiVersion()`. In most cases your UI component, *sinkUI.arx*, will also be implicitly linked to your DB component by linking to *sinkDB.lib*. You will have created *sinkDB.lib* as part of building *sinkDB.dbx*.

It is critical in such a case that you link to *rxapi.lib before* linking to *sinkDB.lib*. If you link to *sinkDB.lib* first, `acrxGetApiVersion()` in *sinkUI.arx* will actually be forwarding into *sinkDB.dbx*, rather than pulling in a static definition directly from *rxapi.lib*. If for some reason the user's AutoCAD environment is corrupted so that when he or she loads *sinkUI.arx*, *sinkDB.dbx* cannot be found, this linkage situation can lead to an immediate unexpected shutdown.

Localization and XMX Files

Since *acdb.xmx* is translated for all localized AutoCADs, pretranslated XMX files are now shipped as part of the ObjectDBX SDK, so you can create a fully language-localized end product. This will allow you to create several language versions of your application, or your application can query the user with a choice of several languages.

Choosing the language will be the responsibility of your application. To that end, the task of loading of the XMX file has been moved out of the `DllMain()` function in *AcDb15.dll*, and into the function `acdbValidateSetup()`. This function now takes a localization ID (LCID) parameter to specify the application's choice of language. The function will attempt to load that XMX file first by using the `AcDbHostApplicationServices::findFile()` method and, if that fails, by then looking in the directory that contains *AcDb15.dll*.

The new signature for `acdbValidateSetup()` is

```
Acad::ErrorStatus
acdbValidateSetup(
    long lcid);
```

The *acdb.xmx* file is now named *acdbLLL.xmx*, where *LLL* is the three-letter language-localization abbreviation that can be derived from the LCID.

Autodesk supports, and will eventually ship or otherwise provide, *acdbLLL.xmx* in the following languages:

XMX file types

Language	Language Abbreviation	Language ID from LCID
English (USA)	ENU	0409
Chinese (Taiwan)	CHT	0404
Chinese (Simplified)	CHS	0804
Czech	CSY	0405

XMX file types (*continued*)

Language	Language Abbreviation	Language ID from LCID
Danish	DAN	0406
Dutch (Belgian)	NLB	0813
Dutch (Default)	NLD	0413
Finnish	FIN	040b
French (Canadian)	FRC	0c0c
French (Default)	FRA	040c
German (Default)	DEU	0407
Greek	ELL	0408
Hungarian	HUN	040e
Italian	ITA	0410
Japanese	JPN	0411
Korean	KOR	0412
Norwegian (Bokmal)	NOR	0414
Polish	PLK	0415
Portuguese (Brazilian)	PTB	0416
Portuguese (Default)	PTG	0816
Russian (Default)	RUS	0419
Slovak	SKY	041b
Spanish (Default)	ESP	040a
Spanish (Mexican)	ESM	080a
Swedish	SVE	041d
Turkish	TRK	041f

As an ObjectDBX developer, you must do the following to create a language-localized end-product:

- You must ship the appropriate *acdbLLL.xmx* files along with your product.
- You must inform ObjectDBX which *acdbLLL.xmx* file to load, by passing the appropriate LCID to `acdbValidateSetup()`.

If the LCID does not correspond to one of the three-letter abbreviations above, or if the appropriate XMX file was not shipped, your ObjectDBX application will fail to load properly.

If it is unable to find the desired *acdb.xmx* file, `acdbValidateSetup()` will attempt to load English as a default. Again, it will first use `findFile()` and next assume the same path as *acdb15.dll*. If it finds English, but English was not the requested language, `Acad::eFileNotFound` is returned. If the function is unable to find any *acdb.xmx* file, it will halt with `fatalError()`, and your application will not load.

AutoCAD has been modified to call `acdbValidateSetup()` with the current language-translated version, which it determines by looking at the registry. If the registry has not been initialized (as in a developer build) it assumes English.

Transaction Management

Transaction handling is now part of ObjectDBX instead of AutoCAD, and the corresponding library is *acdb.dll* instead of the AutoCAD executable. There is one new class, `AcDbTransactionManager`, as part of this change.

AcTransaction and AcTransactionReactor Classes

These classes have been moved from AutoCAD to the ObjectDBX DLL. Their header file is now *dbtrans.h*, though you may continue to use the *actrans.h* header since *actrans.h* includes *dbtrans.h*. For information about the classes, please see the sections on `AcTransaction` and `AcTransactionReactor` in the *ObjectARX Reference*.

AcTransactionManager and AcDbTransactionManager Classes

The `AcDbTransactionManager` class is new in this release, and the existing `AcTransactionManager` class is now derived from `AcDbTransactionManager`. All the methods of `AcTransactionManager`, except `enableGraphicsFlush()` and `flushGraphics()`, now belong to `AcDbTransactionManager`. The `enableGraphicsFlush()` and `flushGraphics()` methods are still members of the `AcTransactionManager` class. For a description of the classes and methods, please see the *ObjectARX Reference*. The `AcTransactionManager` class is still part of the *acad.lib* library.

AcDbDatabase Class

AutoCAD 2000 provides a new `AcDbDatabase` method so that you can access the correct transaction manager for a particular database, including transient databases:

```
AcDbTransactionManager*
AcDbDatabase::transactionManager() const;
```

This method returns the `AcDbTransactionManager` associated with the database. Note that the transaction manager returned is not of the type `AcTransactionManager`.

If you need to access `AcTransactionManager`, you must first be working with the current database, associated with current document. In this case, you can use the `actrTransactionManager` macro.

Application Services

When ObjectDBX is used to create an application, that application is known as the "host application." The code in the ObjectDBX library expects the host application to provide it with certain services—for example, a file find mechanism. AutoCAD itself is a host application for ObjectDBX.

When you are writing ObjectARX applications, you are free to query the host application, such as AutoCAD, for certain services or information. There is a distinction between ObjectARX applications that work only in the presence of AutoCAD, using APIs from *acad.lib*, and those that may work with any ObjectDBX host application. This section refers to the latter kind as "independent" ObjectARX applications.

In addition, if you are writing an ObjectDBX host application, you are required to implement certain services, which will be used by both ObjectDBX itself and potentially by AutoCAD-independent ObjectARX applications.

The header file for application services is *dbapserv.h*. The classes and methods in this header file fall into one of three categories:

- Those you *must* override, because no default implementation is provided as the method is assumed to be very application-specific. These are declared to be pure virtual.
- Those you *may* override, but that have a default implementation that will minimally satisfy the database code. These are declared virtual.
- Those you *may not* override, as they are expected to operate identically in all host applications. These are not declared virtual.

The following classes are used to provide services to the host application.

AcDbHostApplicationServices Class

The `AcDbHostApplicationServices` class provides various services to ObjectDBX at runtime. These services are defined by the ObjectDBX client.

In AutoCAD 2000, it is required that any ObjectDBX host application *must* provide a class derived from `AcDbHostApplicationServices`. A default service is not provided. This is different from the last release of ObjectDBX. Each method is described in the *ObjectARX Reference* with its default implementation (if it has one), what you must do to override the method successfully, and how you may call the method. A description of the functions to set and get the service follows the `AcDbHostApplicationServices` class overview.

The `AcDbHostApplicationServices` class provides the following types of methods.

General Support Methods

Most of the methods in this group are ones you may override to find files, handle errors, detect user keyboard breaks, specify temporary paths, check environment variables, audit drawing information, and set the working database.

Output Methods

These methods provide for printing characters and displaying OLE objects.

Product Methods

These methods are called by the database in various places to, for example, display "About" information or provide version numbers.

Progress Meter Methods

This group of methods sets and gets the current progress meter class for the host application. The database will report progress to a progress meter in two ways:

- By creating a new progress meter, using it, and deleting it
- By operating on a global progress meter for file open types of operations

Controller and Transaction Methods

Controller and transaction methods specify the undo controller and transaction manager that your application will use. You are not required to override or call these methods.

Internet Methods

The Internet methods allow your application to use URLs directly in your code or through a browser.

System and Locale Methods

These methods should generally not be overridden in your host application services class. These methods are set during construction of `AcDbHostApplicationServices` and do not need to be changed.

Service Functions

Two global functions are used to set and get the services; they are how you make use of the services class you derive from `AcDbHostApplicationServices`.

```
inline AcDbHostApplicationServices*
acdbHostApplicationServices()
```

Use this function to access the services. `acdbHostApplicationServices()` may be called by database code, by object component applications, and by host applications.

```
void
acdbSetHostApplicationServices(
    AcDbHostApplicationServices* pServices);
```

Use this function to set your instance of services. This should be done as an early step in your application—for example, in the `InitInstance()` method of your application. Your host application should only call this method once.

```
Acad::ErrorStatus
acdbValidateSetup();
```

After your application has created an instance of your host application services class and used `acdbSetHostApplicationServices()` to set it up, your application should next call `acdbValidateSetup()`, which will verify that everything is set up correctly. `acdbValidateSetup()` will return `eNullObjectPtr` if there isn't any `AcDbHostApplicationServices` object registered.

AcDbHostApplicationProgressMeter Class

At various points, the database code will attempt either to create a temporary progress meter or to modify a single global progress meter. If you would like the database code to display its progress in your application, you should implement a progress meter by deriving from the `AcDbHostApplicationProgressMeter` class. This meter can display in any fashion you choose, but it must implement `start()`, `stop()`, `meterProgress()`, and `setLimit()` methods.

This class falls into the "may override" category. If you do not override the class, a default implementation that does nothing will be used by the database code. It is not anticipated that the ObjectARX applications or host applications would ever need to call these methods directly. These methods are called by the database code itself. However, it will not cause any error for either type of application to generate, use, and delete instances of this class for whatever purpose suits them. Refer to the *ObjectARX Reference* for a complete list of the methods in this class and their descriptions.

DXF

ObjectARX developers may read and write DXF (Drawing Exchange Format) files through `AcDbDatabase` APIs. It is once again possible to bring partial DXF files into existing drawings (an ability that was partially taken away in Release 14) and to view preview images of DXF files from the Open and DINSERT dialog boxes. Also, the internal DXF code has been completely redesigned so that developers working with DXF will now get all the advantages and features of the DWG file format. Global functions are provided that allow you to save drawings as Release 12, 13, or 14 DXF files.

File Formats for Solids

The DWG and DXF file formats have changed for ACIS solids. The ACIS solids (such as 3DSOLID, REGION, and BODY) are now saved in ACIS 3.0 format. AutoCAD Release 14.01 used the ACIS 3.0 format, Releases 14 and 13c4 used ACIS 1.6, and Releases 13 through 13c3 used ACIS 1.5.

DXF Functions

New methods and functions are available to write out DXF files from the current database, and in previous release formats.

AcDbDatabase Class Methods

The following methods have been added to `AcDbDatabase` to let you read and write DXF files directly from a drawing database:

```
Acad::ErrorStatus
AcDbDatabase::dxfOut(
    const char* filename,
    const int precision=16,
    const Adesk::Boolean saveThumbnailImage = Adesk::kFalse);
```

filename — Name of the new DXF file to create. The *.dxf* extension for the filename is not added automatically.

precision — The number of bits of accuracy, from 0 to 16. This may also be set to –1 to change the output format from the default ASCII DXF format to the binary DXF format. Note that the precision is 16, and not 6.

saveThumbnailImage — Allows you to save the thumbnail image, if there is one in the database.

```
Acad::ErrorStatus
AcDbDatabase::dxfIn(
    const char* filename,
    const char* logFilename=NULL);
```

filename — The name of the DXF file to be read. The filename must include the full path and extension of the file (*.dxf*).

logFilename — The name of a log file to record all warning and error messages from reading the DXF file. The default argument is to have no output file. When there's no output file, the function will output messages character by character through the `displayChar()` method of `AcDbHostApplicationServices`.

This reads the DXF file specified into the database object and returns `Acad::eOk` if successful.

WARNING! This function should be used only on a newly created `AcDbDatabase` that was created with its constructor's `buildDefaultDrawing` argument set to `Adesk::kFalse`. If this method is used on an `AcDbDatabase` created with `buildDefaultDrawing` set to `Adesk::kTrue` or an `AcDbDatabase` that already has information in it (for any reason, including a previous call to this method), then memory leaks or possibly fatal errors will result.

AcDbDxfFiler Class Method

A method has been added to `AcDbDxfFiler` to get a pointer to the drawing database of the object being filed out:

```
virtual AcDbDatabase*
AcDbDxfFiler::database() const;
```

This method is useful when you need to acquire information about the object, such as line style, from the object's database.

Global Functions

This function retrieves the preview bitmap from the DXF file:

```
Acad::ErrorStatus
acdbGetThumbnailBitmapFromDxfFile(
    const char* filename,
    void*& pBitmap);
```

filename — The name of the DXF file to retrieve the bitmap from.

pBitmap — Pointer to a `BITMAPINFO` structure, or `NULL` if the file does not have a bitmap. The caller is responsible for deleting the `BITMAPINFO` returned.

These functions write out a drawing database in the DXF file format corresponding to AutoCAD Release 12, 13, or 14, respectively:

```
Acad::ErrorStatus
acdbDxfOutAsR12(
    AcDbDatabase* pdb,
    const char* filename,
    const int precision=16);

Acad::ErrorStatus
acdbDxfOutAsR13(
    AcDbDatabase* pdb,
    const char* filename,
    const int precision=16);

Acad::ErrorStatus
acdbDxfOutAsR14(
    AcDbDatabase* pdb,
    const char* filename,
    const int precision=16);
```

pdb	Pointer to the `AcDbDatabase` object to write out as a DXF file.
filename	Name of the new DXF file to create. The *.dxf* extension for the filename is not added automatically.
precision	The number of bits of accuracy, from 0 to 16. This may also be set to –1 to change the output format from the default ASCII DXF format to the binary DXF format.

AcGi Library

The AcGi interfaces have been modified to improve their usability and to allow the complete implementation of the original AutoCAD entity set.

New Concepts

AcGi now supports clip boundaries and state restoration.

Clip Boundaries

Since AutoCAD Release 14 supported clipping for block references and external references, this release of AcGi does also. Clip boundaries can now be defined and used by AcGi clients.

Nested clip boundaries are applied in a stack, one after the other.

To define a clip boundary for your geometry

1. Define an `AcGiClipBoundary`.
2. Push your clip boundary onto the stack using `pushClipBoundary()`.
3. Draw your components.
4. Remove your boundary from the stack using `popClipBoundary()`.

Restore the State

The enrichment of the AcGi interfaces requires that clients now restore any modified state to its original values. This applies specifically to

- Subentity traits
- Clip boundaries
- Transformations

Restoration is currently not required for any state that was included in the previous release of AutoCAD, such as

- Color
- Layer
- Linetype
- Filltype

For the new options, it is imperative that your implementations of `worldDraw()` and `viewportDraw()` restore any state that is modified.

Interface Reorganization

The following classes have had many methods added for this release of AcGi:

- `AcGiContext`
- `AcGiGeometry`
- `AcGiViewport`
- `AcGiSubEntityTraits`

`AcGiContext` abstracts out the commonality between `AcGiViewportDraw` and `AcGiWorldDraw`. This allows you to code graphics routines in terms of `AcGiContext` that will work in either case. To maintain their original signatures, `AcGiWorldDraw` and `AcGiViewportDraw` now derive from `AcGiContext`.

The new method `rawGeometry()` has been introduced to give access to a set of geometry common to both cases.

`AcGiGeometry` abstracts out the commonality between `AcGiViewportGeometry` and `AcGiWorldGeometry`. To maintain their original signatures, `AcGiWorldGeometry` and `AcGiViewportGeometry` now derive from `AcGiGeometry`.

Note that when using the `AcGiSubentity` class, you are responsible for restoring the values of any traits that you modify. These traits are

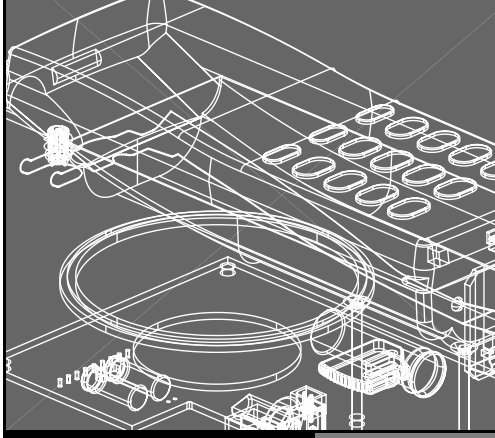
- Dimming
- Lineweight
- End cap style
- Join styles
- Thickness
- ByBlock properties

ACIS

AutoCAD now uses ACIS 4.2, while AutoCAD Release 14.01 used ACIS 3.0, Releases 14 and 13c4 used ACIS 1.6, and Releases 13 through 13c3 used ACIS 1.5. ACIS 4.2 supports variable-radius blend surfaces. The 3.0 version provides support for the 3D objects defined by the `AcDbRegion`, `AcDb3dSolid`, and `AcDbBody` classes. Non-manifold solids are handled by the `AcDb3dSolid` class now, rather than `AcDbBody` as done previously.

The DWG and DXF files formats for solids have changed because of the changes in ACIS. For more information, see the "DXF" section in this chapter. Be sure to check the readme file */docs/ReadARX* for any last-minute changes in ACIS version level.

Extensibility Features



In This Chapter

9

- Visual LISP
- VBA
- ActiveX
- AST
- Database Connectivity
- ADS

These extensibility features allow you to extend the functionality of AutoCAD. This includes changes to the programming environments that are available for AutoCAD, as well as new features.

Visual LISP

AutoLISP has been upgraded to include Visual LISP (VLISP), the new LISP environment for AutoCAD. VLISP is a software tool designed to simplify LISP program development. VLISP's integrated development environment provides features to help ease the tasks of source code creation and modification, program testing, and debugging of AutoLISP programs.

Visual LISP supports existing AutoLISP applications, while providing support for modern programming features. In addition, VLISP provides a vehicle for delivering standalone applications written in AutoLISP.

Visual LISP Features

With VLISP, you can now perform most programming operations inside a single environment that permits text editing, program debugging, and interaction with AutoCAD and other applications.

The following are components of the Visual LISP development environment:

- *Syntax Checker.* Recognizes erroneous AutoLISP constructs and improper arguments in calls to built-in functions.
- *File Compiler.* Improves the execution speed and provides a secure and efficient delivery platform.
- *Source Debugger.* Designed specifically for AutoLISP. This feature supports stepping through AutoLISP source code in one window while simultaneously displaying the code effect in an AutoCAD drawing window.
- *Text File Editing*. Available with AutoLISP and DCL color coding, as well as other AutoLISP syntax support features.
- *AutoLISP Formatter.* Restructures programs into an easily readable format.
- *Comprehensive Inspector and Watch*. Provides convenient access to variable and expression values for data structure browsing and modification. These features may be used to explore AutoLISP data and AutoCAD drawing entities.
- *Context-Sensitive Help*. Provided for AutoLISP functions. Also contains Apropos, a powerful feature for symbol name search.
- *Project Management.* Makes it easy to maintain multiple-file applications.
- *Simplified Packaging*. Packages compiled AutoLISP files into a single module.
- *Desktop Save and Restore*. Preserves and reuses the windowing environment from any VLISP session.
- *Intelligent Console*. Introduces a new level of convenience and efficiency for AutoLISP users. The basic functions of the Console window correspond

to the AutoCAD Text Screen functions and provide a number of interactive features, such as history scrolling and full-input line editing.

New AutoLISP Interpreter

There is a new AutoLISP interpreter in AutoCAD 2000. This new version of AutoLISP corrects some problems that existed in the past and changes the implementation of some functions.

For example, you may encounter the following:

- Protected symbols, such as T, are now truly protected and result in error messages if you attempt to modify them.
- Functions are no longer defined internally as lists. If you have code that depends on the list structure of functions, you can use the DEFUN-Q, DEFUN-Q-LIST-SET, and DEFUN-Q-LIST-REF functions to maintain backward compatibility.

Refer to the *AutoLISP Reference* in Visual LISP Help for information on using these functions.

AutoLISP Integer Conversion

Previous versions of AutoLISP treated integer overflow (positive numbers greater than 2,147,483,647 and negative numbers less than -2,147,483,648) by returning negative numbers or producing seemingly random numbers. The following examples illustrate this behavior:

```
Command: !2147483647
2147483647

Command: !2147483648
-2147483648

Command: !-2147483648
-2147483648

Command: !-2147483649
2147483647

Command: !9999999999
1410065407
```

In AutoCAD 2000, AutoLISP handles integer overflow by returning a valid real number, if the integer was entered at the AutoCAD Command prompt

or the VLISP Console prompt. In addition, the maximum negative number is slightly different than it was in Release 14.

The following examples illustrate how AutoLISP handles large integers in AutoCAD 2000:

```
Command: !2147483647
2147483647          ;Same behavior as Release 14

Command: !2147483648
2.14748e+009        ;Valid real, instead of negative integer

Command: !-2147483647
-2147483647         ;|This is now the largest negative integer; one less than the
                    largest in Release 14 |;

Command: !-2147483648
-2.14748e+009       ;Valid real, instead of positive integer

Command: !9999999999
1.0e+010            ;Valid real, instead of random integer
```

Note that conversion to reals only occurs when you enter a value at the AutoCAD Command prompt or the VLISP Console prompt. If an arithmetic operation on integers results in overflow, AutoLISP returns an invalid integer number. For example, adding 1 to 2147483647 results in a negative number, as shown below:

```
Command: !(+ 2147483647 1)
-2147483648
```

For complete information on the new Visual LISP environment, see the *Visual LISP Developer's Guide and Reference*.

VBA

The VBA (Visual Basic for Applications) environment has been dramatically upgraded, allowing developers to share more easily and distribute VBA macros and dialogs. The VBA environment now supports the following features:

- Embedding a VBA project in an AutoCAD drawing
- Loading multiple VBA projects into the VBA development environment
- Additional AutoCAD interface control of VBA, including new commands, menus, and a VBA toolbar
- New VBA project management tools

Embedding a VBA Project in an AutoCAD Drawing

Developers now have the option of storing their VBA projects in an AutoCAD drawing (embedded projects) or in an external DVB file (global projects).

Embedded projects are automatically loaded whenever the drawing in which they are contained is opened in AutoCAD. Embedded projects cannot open or close AutoCAD drawings because they function only for the document that they reside in.

Loading Multiple VBA Projects into the VBA IDE

You can now load multiple projects into the VBA Interactive Development Environment (IDE). At any given time a user can have both embedded and global projects loaded into their AutoCAD session.

Loading multiple projects allows developers the option of using references between projects. Referencing one VBA project from another allows developers to share code more easily. Developers can create libraries of commonly used macros and then reference the library when needed. This keeps the shared code centrally located and supported, while allowing a large number of developers to utilize the code.

Additional AutoCAD Interface Controls

New AutoCAD interface controls for VBA are provided. There are AutoCAD commands, menu options, and now a VBA toolbar as well.

There are six AutoCAD VBA commands:

VBAIDE	Activates the VBA Interactive Development Environment.
VBALOAD	Activates the Open VBA Project dialog. If FILEDIA=0, this command accepts a project name from the command line.
-VBALOAD	A command line version of VBALOAD. This command requires a project name to be specified.
VBAUNLOAD	This command unloads a project from the command line. Although this command was available in Release 14, it now requires a project name to be specified. To unload a project using dialog control, use the VBAMAN command.
VBARUN	Activates the VBA Macros dialog. This dialog allows you to select a macro to run.

-VBARUN	A command line version of the VBARUN command, this command accepts a macro name at the command line and executes that macro.
VBAMAN	Activates the VBA Manager dialog.
VBASTMT	Executes a VBA statement from the command line.

The VBA menu options are found on the Tools menu, under the Macros submenu. The menu options include: Macros, Load Project, VBA Manager, and Visual Basic Editor. The functionality of these menu options corresponds to the VBARUN, VBALOAD, VBAMAN, and VBAIDE commands above.

The VBA toolbar is located in the ACAD menu group. It contains toolbar buttons for the VBARUN, VBALOAD, VBAMAN, and VBAIDE commands.

New VBA Project Management Tools

There are three new VBA dialogs available to help developers manage their VBA projects. These new dialogs are the VBA Macros dialog, the VBA Manager dialog, and the VBA Options dialog.

The VBA Macros dialog lists all the macros loaded in the current session of AutoCAD. From this dialog a developer can create a new macro, delete, run, or step into a given macro, open the VBA Manager dialog, open the VBA Interactive Development Environment, or open the VBA Options dialog.

The VBA Manager dialog lists all the projects loaded in the current session of AutoCAD. From this dialog a developer can create new projects, embed global projects into a drawing, save projects, and load or unload projects.

The VBA Options dialog allows developers to toggle between the following options:

- Enable auto embedding
- Allow break on errors
- Enable macro virus protection

This dialog is only accessible from the VBA Macros dialog.

ActiveX

The ActiveX Automation interface has undergone significant architectural changes. While VBA users will notice these changes in the form of new objects, methods, properties, and events, COM users will notice the new Active Template Library.

The Active Template Library (ATL) provides a set of `IDispatch`-derived reusable template classes for building new COM components. This feature is most important to those ObjectARX users who are developing custom objects. More information on the ATL architecture and usage can be found in the "COM and ActiveX Automation" chapter of the *ObjectARX Developer's Guide*.

In addition to its new architecture, the AutoCAD ActiveX Automation library contains the following new features:

- Application, document, and object level events
- Toolbar and menu customization
- External reference support
- New entities, and extended control over existing entities
- Multiple document interface support
- New plotting controls
- New Preferences objects
- HWND and SendCommand support
- Programmatic access to the VBA Development Environment
- Enumerated constants
- Exception numbers
- Support for ObjectDBX

In addition to new features, there have been changes and deletions of several existing methods and properties. The list of changed and deleted functions can be found in appendix B, "ActiveX Changes."

Events

There are three new types of events in AutoCAD:

Application Level Events — Application level events respond to changes in the AutoCAD application and its environment. These events respond to the opening, saving, closing, and printing of drawings, creation of new drawings, issuing of AutoCAD commands, loading or unloading of ObjectARX and Visual LISP applications, changes to system variables, and changes to the application window.

Document Level Events	Document level events respond to the changes of a specific document or its contents. These events respond to the addition, deletion, or modification of objects, activation of a Shortcut menu, changes in the pickfirst selection set, changes to the drawing window, and regeneration of the drawing. There are also document level events for the opening, closing, and printing of a drawing, and the loading or unloading of ObjectARX and Visual LISP applications from the drawing.
Object Level Events	Object level events respond to the changes of a specific object. Currently there is only one object level event. It is triggered whenever an object is modified.

You can find information on all new and previously existing events in the *ActiveX and VBA Reference*.

Toolbar and Menu Customization

AutoCAD ActiveX Automation provides extensive control over the customization of menus and toolbars in the current AutoCAD session.

You can edit or augment the existing menu structure, or you can completely replace the current menu structure with a new one. You can also manipulate toolbars and shortcut menus.

Menu customization can improve productivity by exposing application-specific tasks or by condensing tasks with multiple steps into a single menu selection.

AutoCAD ActiveX provides several menu-related objects. The following outline diagrams the hierarchy of the menu and toolbar-related objects. More information on these objects can be obtained in the *ActiveX and VBA Reference*:

```
Application
    MenuBar
        PopupMenu
    MenuGroups
    MenuGroup
        PopupMenu
            PopupMenus
                PopupMenuItem
        Toolbars
            Toolbar
                ToolbarItem
```

The properties and methods of these new objects are listed in appendix B, "ActiveX Changes."

External Reference Support

The external references associated with a block have been exposed. You can now attach or overlay an external reference drawing file as well as bind, detach, reload, and unload external references. You can also get the drawing information associated with an external reference.

The properties and methods of the `ExternalReference` object are listed in appendix B, "ActiveX Changes." Information on the new `ExternalReference` object can also be found in the *ActiveX and VBA Reference*.

New Entities and Extended Control over Existing Entities

All the objects and properties that are available via the Object Property Manager in AutoCAD are available through the ActiveX interface. To support this functionality, the following new objects have been added to ActiveX: `MLine`, `MInsertBlock`, `XRecord`, `Dim3PointAngular`, and `PolyfaceMesh`. In addition to these new objects, all the existing objects have new properties to enhance their customization control via ActiveX.

The new objects, properties, and methods are listed in appendix B, "ActiveX Changes." You can also find information on these objects in the *ActiveX and VBA Reference*.

MDI Support

To support the new multiple document interface for AutoCAD, ActiveX has added a Documents collection. This new collection allows you to create, open, close, and iterate through AutoCAD drawings.

The new Document interface allows you to activate a drawing and change the size and state of drawing windows.

The new methods and properties for the Documents collection and Document object are listed in appendix B, "ActiveX Changes." You can also find information on these objects in the *ActiveX and VBA Reference*.

New Plotting Controls

The approach to plotting via ActiveX has changed dramatically. The `Plot` object now represents the "how-to-plot" settings and the `Layout` object represents the "what-to-plot" information.

For example, to perform a batch plot you would first use the `Layouts` collection to configure the plotting information for each individual layout. Then you would use the methods and properties on the `Plot` object to initiate the actual plot of the configured layouts.

In addition to the new `Layout` object, there is also a new `PlotConfiguration` object and collection. The `PlotConfiguration` object is simply a named plotting configuration that is not immediately associated with any geometry. The `Layout` object represents a specific configuration that has been assigned to a collection of geometry in the drawing.

The new methods and properties for the `Plot`, `PlotConfiguration`, and `Layout` objects are listed in appendix B, "ActiveX Changes." You can also find information on these objects in the *ActiveX and VBA Reference*.

New Preferences Objects

Due to the tremendous number of preferences available, the `Preferences` object has been divided into two main objects.

The two main objects divide the options stored in the drawing (database-resident) from the options stored in the registry (nondatabase-resident.) The database-resident options are contained on the `DatabasePreferences` object, which is located below the `Document` object. The nondatabase-resident options are contained on the `Preferences` object, found below the `Application` object.

Even with this breakdown, the `Preferences` object still had a tremendous number of options on it, making it difficult to find the option you were looking for. Consequently, the `Preferences` object has been further subdivided into eight subobjects. Each subobject corresponds to a tab on the Options dialog, making it easier to find the option you are looking for. The eight subobjects are: `PreferencesDisplay`, `PreferencesFiles`, `PreferencesOpenSave`, `PreferencesOutput`, `PreferencesSystem`, `PreferencesUser`, `PreferencesConstructionTools`, and `PreferencesSelection`.

The properties and methods of the preferences objects are listed in appendix B, "ActiveX Changes." Information on the objects can also be found in the *ActiveX and VBA Reference*.

HWND and SendCommand() Support

The HWND property and `SendCommand()` method on the `Document` object extend the programmatic control of ActiveX Automation over AutoCAD.

The HWND property returns the window handle of the specified document. Since many Windows operating environment functions require the window handle as an argument, this property now opens up these operation environment functions to you.

The `SendCommand()` method sends a text string to the AutoCAD interface to be evaluated in the context of the current document.

More information on the HWND property and the `SendCommand()` method can be found in the *ActiveX and VBA Reference*.

Programmatic Access to the VBA IDE

AutoCAD ActiveX Automation now provides access to the VBA IDE through the VBE property on the `Application` object. Using the VBE property to obtain the Extensibility object model, you can greatly extend the Visual Basic development environment.

More information on the Extensibility object model can be found in the Microsoft Visual Basic or VBA documentation.

Enumerated Constants

In AutoCAD 2000, the Automation constants have been replaced in the type library by enumerated versions of the same values. Enumerations allow the IntelliSense feature of the VBA IDE to display automatically the list of appropriate constants to the user. You can also define variables as an enumeration type.

The new enumerated constants are listed in the *ActiveX and VBA Reference* for all the topics that contain constant values.

Exception Numbers

In AutoCAD Release 14, only the error description property was used to define automation errors. In AutoCAD 2000, exception numbers have been added to the `Error` object. This means that errors can now be identified via their number or their description, thus allowing applications to be localized more easily.

More information on using exception numbers can be found in the *ActiveX and VBA Developer's Guide*.

Support for ObjectDBX

AutoCAD 2000 ActiveX Automation interface is now usable from ObjectDBX applications. To accomplish this, the `Automation` object model disassociated the database functionality from the user interface functionality.

The database functionality is now available via the `IAcadDatabase` interface. All methods and properties below the document level that referred to `IAcadApplication` have been changed to `IDispatch`. Additionally, the Zoom methods have been moved from the `Viewport` object to the `Application` object. Also, the database-resident options have their own object, `DatabasePreferences`, found under the `Document` object.

More information on this feature is available in the "COM and ActiveX Automation" chapter of the *ObjectARX Developer's Guide*.

AST

The AST (AutoCAD SQL Technology) product is being deprecated in favor of the new Database Connectivity feature. Some components of AST from earlier releases are not available in AutoCAD 2000, while most others have had their APIs modified. Autodesk recommends that you use the tools provided with the new "Database Connectivity" feature, described in this chapter. Changes to the remaining components of AST follow.

Changes for AutoLISP

The ASE (AutoCAD SQL Environment) has been dropped for AutoCAD 2000. The ASI (AutoCAD SQL Interface) is still provided, though you will need to make changes to move your Release 14 applications to AutoCAD 2000. These changes are detailed below, but Autodesk recommends that you access the VBA ActiveX interface from Visual LISP and use the Connectivity Automation Object instead of ASE.

General Changes

The single row select statement is not supported in the ASI API anymore, because the OLEDB providers do not support it, and ASI does not interpret SQL statements after switching to OLEDB.

Deprecated Functions

These functions are still present in AutoCAD 2000 but do not operate correctly. Do not use them:

- asi_getcfg()
- asi_setcfg()
- asi_cmdtype()

Changed Functions

The asi_connect() function now requires a session handle, and the asi_alloc() function now accepts an optional "updatability" argument.

Loading the Data Provider

```
(asi_connect <data_link_file>
    [[<user_name]
     <password>])
```

This function loads the OLE DB data provider and connects to the database, accessible through this provider by implementation of a <connect> statement. <Data_link_file> is the name of the Microsoft Data Link (*mdl*) file. If an extension is not specified, *.mdl* is assumed. The file is located using the following rules:

- by full path name if specified
- by looking in the current directory
- by looking in the AutoCAD directory
- by looking in directories specified by PATH environment variable

The (asi_connect) function automatically initializes the driver that runs this SQL connection. Optional parameters <user_name> and <password> are the user name and password, respectively. The possibility of the loaded data provider supporting the SQL features can be checked with the (asi_feature) function after the connection has established. This function returns the SQL session descriptor in the case of success, and nil otherwise.

The following example connects to the data source defined in the Microsoft Data Link file, "MODBC_DB3.UDL", which is saved in the "data links" directory under the AutoCAD directory:

```
(asi_connect "data links/modbc_db3" "" "")
```

Allocating the Cursor

```
(asi_alloc <stm_dsc> <cursor_name>
    [[[<cursor_scrollability>]
      <cursor_updatability>]
      <cursor_insensitivity>])
```

This function allocates the SQL cursor by means of <allocate cursor statement>. <stm_dsc> in the SQL statement descriptor. If it identifies a statement other than a cursor specification, an exception condition "prepared SQL statement not a cursor specification" is raised. <cursor_name> is the SQL cursor name. The cursor name can be used by

<preparable update : positioned> and <preparable delete : positioned> statements to reference the current row position in the cursor to update or delete it. The optional parameter <cursor_scrollability> specifies whether or not the cursor should be scrollable. A scrollable cursor means that its rows can be fetched in any direction. A nonscrollable cursor can be fetched only in the forward direction. The possible values of this parameter are SCROLL or NON_SCROLL. If this parameter is not specified, a nonscrollable cursor is allocated. The second optional parameter, <cursor_updatability>, specifies whether the cursor should be read-only or updatable. The possible values of this parameter are READ_ONLY, UPDATABLE, or nil, where nil means UNKNOWN value. The third optional parameter, <cursor_insensitivity>, specifies whether or not the cursor should be insensitive. An insensitive cursor means that if the current transaction changes data other than through that cursor, the effect of that change will not be visible through that cursor before it is closed. The single possible value is INSENSITIVE. If this parameter is specified, an insensitive cursor should be allocated. If a cursor is declared INSENSITIVE or SCROLL, it is assumed to not be updatable. The state of the cursor that has already been allocated can be retrieved by means of the (asi_statecsr) function. This function returns the SQL Cursor descriptor in the case of success, and nil otherwise.

New Functions

Three new functions have been added to get additional cursors:

```
(asi_databases <provider_name>)
```

Gets the cursor to the table of the databases, which can be accessed through the given OLE DB data provider. <provider_name> is the name of one of the providers, supported by ASI.

```
(asi_infschema <env> <rowset_name>)
```

Returns the cursor to one of the system tables of the Information Schema. <env> is the SQL session descriptor. <rowset_name> is one of the predefined schema rowsets (see the *OLE DB Programmer's Reference*, appendix B, "Schema Rowsets").

```
(asi_providers)
```

Gets the cursor to the system table of the OLE DB data providers, supported by ASI.

Changes for C and C++

There have been changes to header files, libraries, and classes used with C and C++. The header file changes apply to both ASE and ASI, while the library and class changes are described separately.

Header File Changes

The following header files have been removed:

ase.h	Header file for AutoCAD SQL Environment and "C" call interface
asi.h	Header file for AutoCAD SQL Interface and "C" call interface
asiconf.h	Windows initialization file and ASI configuration file classes
aseconst.h	Windows initialization file and ASE configuration file classes

The following header files are new for AutoCAD 2000:

Oledb.h	OLE DB header file for interface definitions
Oledberr.h	OLE DB error header file
msdaguid.h	Microsoft Data Access GUID definitions
csptypes.h	Definitions for the interfaces
avoidtry.h	Catch and throw exception handling

Library Changes

The following libraries have been removed from AutoCAD 2000:

Asertvc4.lib	Library for all ASE classes and function
Asirtvc4.lib	Library for all ASI classes and function

The library files below are new in AutoCAD 2000:

Aseapi50d.lib	Library for all ASE classes and function
Aseipi50d.lib	Library for all ASI classes and function
Tmptbl.lib	Library for all stringArray classes and function

Class Changes

The Database Connection C++ API is mostly intended for compatibility with previous versions of ASE. It has restricted support of features added for AutoCAD 2000. There is no access to Label Templates, DBMS tables attached to the drawing, Link Groups, and SQL Queries saved in the drawing or on session level. Database Connection implicitly supports Label Templates in the `CAseLink` class's `setDAXxx()` and `getDAXxx()` methods. Database Connection does not support the MLPN Release 13 and 14 feature at the C++ API level because it was completely redesigned. Now it is the Link Group feature. Its

format is incompatible with the MLPN format and appropriate methods like `isPrimary()` or `isSecondary()` no longer apply. The Database Connection C++ API does not provide access to the link information stored at the session level.

The method `connectAse()` in the class `CAsePath` now tries to establish connection with the blank user name and password, and does not cause a connect dialog to appear if no parameters are specified.

Working in SDI, AutoCAD ASE used `acdbCurDwg()` as the default value of the parameters specifying `AcDbDatabase`. `acdbCurDwg()` became obsolete in AutoCAD 2000, and the default value of such Database Connection methods was replaced with the define `WORKINGDWG()` having NULL value. This means that Database Connection works with the drawing database currently active in AutoCAD MDI. It is preferable to specify a working database directly in the user application.

Classes No Longer Provided
These classes have been removed:

- `CAsiNativeStm`
- `CAsiInterDBStm`
- `CAsiImpDef`
- `CAsiPoint`

New classes
These classes have been added to the ASI:

New ASI C++ classes

Class Name	Use
CAsiCsrTable	Superclass for table access
CAsiCsrRegTable	Regular table access
CAsiCsrInfSchema	Information Schema access
CAsiEnumerator	Data Source access
CAsiBinding	Input/Output parameters support
CAsiTACond	Represents atomic condition for the table cursor
CAsiTCCond	Represents database Table cursor condition
CAsiBlob	Supports BLOB data type

New or Changed Functions and Methods

This section lists the new or changed methods in ASI, grouping them by class:

`CAsiSQLObject` Class Changes

```
virtual EAsiBoolean
IsRowset(void) const;

ErrMsg(
    int d=0);

SQLSTATE(
    int d = 0);
```

`CAsiSession` Class Changes

```
EAsiBoolean
Connect(
    const CAsiUcStr& Provider,
    const CAsiUcStr& Database,
    const CAsiUcStr& Username,
    const CAsiUcStr& Password);

EAsiBoolean
ConnectDirect(
    const CAsiUcStr& Provider,
    const CAsiUcStr& Database,
    const CAsiUcStr& Username,
    const CAsiUcStr& Password);

virtual EAsiBoolean
GetStatus(
    EAsiStatInfo,
    ulong*);
```

`CAsiCsr` Class Changes

```
Allocate(
    CAsiExecStm* csrspec,
    const CAsiIdent& csrname,
    EAsiCurScr sc = kAsiNonScroll,
    EAsiCurSns sn = kAsiSnsUndef,
    EAsiBoolean upd = kAsiUnknown);

virtual int
ParamQty(void) const;

virtual CAsiParameter*
ParamDsc(
       int) const;

virtual EAsiBoolean
Bind(
    int,
    CAsiData*);
```

```
virtual EAsiBoolean
Bind(
    int,
    void*,
    short*,
    int,
    EAsiHostType);

virtual EAsiBoolean
Bind(
    const CAsiIdent&,
    CAsiData*);

virtual EAsiBoolean
Bind(
    const CAsiIdent&,
    void*,
    short*,
    int,
    EAsiHostType);

virtual EAsiBoolean
Sob(
    int,
    CAsiData*);

virtual EAsiBoolean
Sob(
    int,
    void*,
    short*,
    int,
    EAsiHostType);

EAsiBoolean
Svl(
    int,
    CAsiData*);

virtual EAsiBoolean
CheckCCondition(
    CAsiData**);

virtual int
ColQty(void) const;

virtual CAsiColumn*
ColDsc(
    int) const;

virtual EAsiBoolean
Cvl(
    int,
    CAsiData*);
```

```
virtual EAsiBoolean
Cvl(
    int,
    void*,
    short*,
    int,
    EAsiHostType);

virtual EAsiBoolean
GetStatus(
    EAsiStatInfo,
    ulong*);

virtual EAsiBoolean
IsRowset(void) const;

virtual EAsiBoolean
is_insertable(void) const;
```

`CAsiStm` Class Changes

```
SetSession(
    CAsiSession*);
```

`CAsiExecStm` Class Changes

```
EAsiBoolean
IsRowset(void) const;

EAsiBoolean
Prepare(
    CAsiSession*,
    const CAsiUcStr&,
    EAsiBoolean trans = kAsiTrue);

EAsiBoolean
ImmediateExecute(
    CAsiSession*,
    const CAsiUcStr&,
    EAsiBoolean trans = kAsiTrue);

int
ParamQty(void) const;

CAsiParameter*
ParamDsc(
    int) const;

virtual CAsiColumn*
ColDsc(
    int) const;

virtual int
ColQty(
    void) const;
```

```
virtual EAsiBoolean
GetStatus(
    EAsiStatInfo,
    ulong*);
```

`CAsiCsrTable` Class Changes

```
virtual int
ColQty(void) const;
```

```
virtual CAsiColumn*
ColDsc(
    int) const;
```

```
virtual EAsiBoolean
Open(void);
```

```
virtual EAsiBoolean
Close(void);
```

```
virtual EAsiBoolean
Update(
    const CAsiRow&);
```

```
virtual EAsiBoolean
Cvl(
    int,
    CAsiData*);
```

```
virtual EAsiBoolean
Cvl(
    int,
    void*,
    short*,
    int,
    EAsiHostType);
```

```
virtual EAsiBoolean
SetOrder();
```

```
virtual EAsiBoolean
SetFilter(
    CAsiTCCond*);
```

```
void
DeallocCsr(void);
```

`CAsiCsrRegTable` Class Changes

```
EAsiBoolean
Allocate(
    CAsiSession * ses,
    const CAsiUcStr& catalog,
    const CAsiUcStr& schema,
    const CAsiUcStr& table,
    EAsiCurScr sc = kAsiNonScroll,
    EAsiBoolean upd = kAsiUnknown);
```

`CAsiCsrInfSchema` Class Changes

```
EAsiBoolean
Allocate(
    CAsiSession* ses,
    EAsiInfSchTab ischema,
    EAsiCurScr sc = kAsiNonScroll);
```

`CAsiEnumerator` Class Changes

```
EAsiBoolean
Allocate(
    CAsiAppl*);
```

```
EAsiBoolean
Allocate(
    CAsiAppl*,
    CLSID& clsidEnum);
```

```
void
SetFilter(
    EAsiDataSourceType dst,
    const CAsiUcStr& pName,
    EAsiBoolean isParent = kAsiUnknown);
```

```
virtual EAsiBoolean
Fetch(void);
```

`CAsiTACond` Class Changes

```
CAsiTACond(void);
```

```
CAsiTACond(
    const CAsiTACond&);
```

`CAsiTCCond` Class Changes

```
CAsiTCCond(void);
```

```
CAsiTCCond(
    int);
```

```
CAsiTCCond(
    const CAsiTCCond&);
```

```
int
Count(void) const;
```

```
void
Append(
    const CAsiTACond&);
```

```
CAsiTACond&
operator[](int);
```

`CAsiBlob` Class Changes

```
void
SetType(
    EBlobType);

EBlobType
GetType() const;

EAsiDataType
Type() const;

uint
Length() const;

uint
ReturnLength() const;

EAsiBoolean
storeValue(
    real);

EAsiBoolean
storeValue(
    integer);

EAsiBoolean
storeValue(
    smallint);

EAsiBoolean
storeValue(
    char*,
    uint);

EAsiBoolean
storeValue(
    const CAsiUcStr&);

EAsiBoolean
storeValue(
    const CAsiData&);

EAsiBoolean
set(
    IUnknown*);

EAsiBoolean
getValue(
    real*) const;

EAsiBoolean
getValue(
    integer*) const;

EAsiBoolean
getValue(
    smallint*) const;

EAsiBoolean
getValue(
    char*,
    uint) const;
```

```
EAsiBoolean
getValue(
    CAsiUcStr*) const;

EAsiBoolean
getValue(
    CAsiData&) const;

EAsiBoolean
get(
    IUnknown**);

EAsiBoolean
SQLType(
    CAsiUcStr*) const;

EAsiBoolean
SQLLiteral(
    CAsiUcStr*) const;

unsigned long
packSize(void) const;

unsigned long
packTo(
    CAsiObjPack*) const;

EAsiBoolean
unpackFrom(
    CAsiObjPack*);

unsigned long
packValueSize() const;

unsigned long
packValueTo(
    CAsiObjPack*) const;

EAsiBoolean
unpackValueFrom(
    CAsiObjPack*);

void
Clear();

void
SetFormat(
    EBlobFormat);

void
SetPintLen(
    int);

HRESULT
ReadHead();
```

`CAsiData` Class Changes

```
static CAsiData*
MapColumnInfoToAsi(
    DBCOLUMNINFO* pColumnInfo);

static CAsiData*
MapParamInfoToAsi(
    DBPARAMINFO* pParamInfo);

MapAsiToColumnInfo(
    DBCOLUMNINFO*);

virtual DBPARAMINFO*
MapAsiToParamInfo(
    DBPARAMINFO*);
```

Global Helper Functions

```
EAsiBoolean AsiAllocateDSEnumerator(
       const GUID* guidEnum,  // [in] Parent enumerator CLSID
       const CAsiUcStr& pDS,  // [in] DS name
       CAsiEnumerator** pEnum // [out] returned enumerator
);

EAsiBoolean AsiGetDSEnumerator(
       const GUID* guidEnum, // [in] Parent enumerator CLSID
       const CAsiUcStr& pDS, // [in] DS name
       GUID* guidDsEnum      // [out] enumerator CLSID
);
```

Database Connectivity

Database Connectivity is the AutoCAD 2000 version of the AutoCAD SQL Extension (ASE) presented in AutoCAD Releases 12, 13, and 14. Database Connectivity combines an access to external data stored in a variety of DBMS with linkage processing between drawing entities and DBMS table rows. The original multi-platform DBMS drivers implemented by AutoCAD have been replaced with Microsoft's OLE DB technology, and the database connection user interface has been changed significantly. The existing database connection AutoLISP, C, and C++ interfaces have been extended through use of the new Database Connection Automation Object interface.

The Database Connection Automation Object (DCO) API interacts with AutoCAD Automation Objects in order to operate on AutoCAD drawing properties and with Microsoft's ActiveX Database Objects (ADO) in order to operate on external DBMS data. It duplicates an existing C++ API for old ASE features and introduces interfaces to new Database Connectivity functionality.

The DCO API provides you with access to link information stored in AutoCAD drawing documents and supported by the database connection. DCO receives data from external databases via ADO, which in turn uses OLE DB functionality introduced by data providers such as Microsoft DASQL. The Database Connection Service Provider (CSP) acts as a wrapper over OLE DB data providers and extends its default interfaces with advanced SQL statement processing. The database connection functional module uses CSP to access DBMS data and process it. From the other direction, the database connection functional module interacts with AutoCAD to access drawing information and support data integrity in the drawing. DCO is integrated with the AutoCAD Automation API at the OLE Automation and COM levels. It operates with AutoCAD Automation Objects such as drawing document and layouts and drawing objects.

Database Connectivity introduces new administrative functions for synchronization of drawing link information and DBMS information like `synchronize()` and `reloadLabel()`. Database Connectivity introduces the new Link Template Group. This is an extension of the MLPN feature in previous releases of AutoCAD. The former bi-level hierarchical design of MLPN has been replaced with a single level structure. All link keys used in any Link Template joined into a group can be used to select records in any DBMS table referenced by the Link Template in the Group. Database Connectivity also introduces new formatting facilities for labels. User-defined settings for labels can be stored in the named Link Template and used for any Link Template referencing the same DBMS table. Database Connectivity now supports attaching external DBMS tables and SQL Queries to the drawing and opening and executing them.

ADS

ADS (AutoCAD Development System) was ported to ObjectARX in Release 14. Continuing that process, AutoCAD 2000 renames the ADS functions to follow the standard ObjectARX naming conventions. All the `ads_Xxxx()` functions are now named either `acedXxxx()`, `acdbXxxx()`, or `acutXxxx()`, depending on which library will eventually contain them (*acad.lib* for AcEd, *acdb15.lib* for AcDb, and *acutil15.lib* for AcUt). In addition to renaming the functions, all those with new names of `acutXxxx()` have been moved over to *acutil15.lib* and some of the `acdbXxxx()` functions have been moved over to *acdb15.lib*.

The function declarations that used to be in *ads.h* are now split out into three new header files:

- *acdbads.h* – functions that have been moved to *acdb15.dll*
- *acutads.h* – functions that have been moved to *acutil15.dll*
- *acedads.h* – functions that remain in AutoCAD

The ADS names are still available for use by including the *adsmigr.h* header file (which is included by the *migrtion.h* header file for your convenience). The *adsmigr.h* file contains #define macros for each ADS function. These names are being deprecated, and you should try to use the new names in your future work, as well as update any ADS calls in your existing code. The *ads.h* header file now simply contains #includes for *adsmigr.h* followed by #includes for *acdbads.h*, *acutads.h*, and *acedads.h*. This allows existing applications to build with no code changes because the #defines in *adsmigr.h* will do the necessary name changing automatically.

NOTE Any source code that has its own declarations of the old `ads_Xxxx()` functions will need to be updated to declare the new function names instead.

New Names for ADS Database Functions

The following functions have been grouped together because they all perform some sort of interaction with the drawing database. Because of this, their new prefix is "acdb":

ADS database functions

ADS Name	New Name	ADS Name	New Name
ads_angtof	acdbAngToF	ads_angtos	acdbAngToS
ads_dictadd	acdbDictAdd	ads_dictnext	acdbDictNext
ads_dictremove	acdbDictRemove	ads_dictrename	acdbDictRename
ads_dictsearch	acdbDictSearch	ads_distof	acdbDisToF
ads_entdel	acdbEntDel	ads_entget	acdbEntGet
ads_entgetx	acdbEntGetX	ads_entlast	acdbEntLast
ads_entmake	acdbEntMake	ads_entmakex	acdbEntMakeX
ads_entmod	acdbEntMod	ads_entnext	acdbEntNext

ADS database functions (*continued*)			
ADS Name	**New Name**	**ADS Name**	**New Name**
ads_entupd	acdbEntUpd	ads_fail	acdbFail
ads_handent	acdbHandEnt	ads_inters	acdbInters
ads_name_clear	acdbNameClear	ads_name_equal	acdbNameEqual
ads_name_nil	acdbNameNil	ads_name_set	acdbNameSet
ads_namedobjdict	acdbNamedObjDict	ads_point_set	acdbPointSet
ads_regapp	acdbRegApp	ads_regappx	acdbRegApp
ads_rtos	acdbRToS	ads_snvalid	acdbSNValid
ads_tblnext	acdbTblNext	ads_tblobjname	acdbTblObjName
ads_tblsearch	acdbTblSearch	ads_xdroom	acdbXdRoom
ads_xdsize	acdbXdSize	ads_xstrcase	acdbXStrCase
ads_xstrsave	acdbXStrSave		

Note that many of these functions are not available when using ObjectDBX. The functions available in ObjectDBX are

- `ads_xdroom` (`acdbXdRoom`)
- `ads_xdsize` (`acdbXdSize`)
- `ads_name_set` (`acdbNameSet`)
- `ads_point_set` (`acdbPointSet`)
- `ads_name_clear` (`acdbNameClear`)
- `ads_name_nil` (`acdbNameNil`)
- `ads_name_equal` (`acdbNameEqual`)

New Names for ADS Editor Functions

The following functions have been grouped together because they all have to do with user interaction and editing. Because of this, their new prefix is "aced":

ADS editing functions

ADS Name	New Name	ADS Name	New Name
ads_agetcfg	acedGetCfg	ads_agetenv	acedGetEnv
ads_alert	acedAlert	ads_arxload	acedArxLoad
ads_arxloaded	acedArxLoaded	ads_arxunload	acedArxUnload
ads_asetcfg	acedSetCfg	ads_asetenv	acedSetEnv
ads_cmd	acedCmd	ads_command	acedCommand
ads_defun	acedDefun	ads_draggen	acedDragGen
ads_entsel	acedEntSel	ads_findfile	acedFindFile
ads_fnsplit	acedFNSplit	ads_getangle	acedGetAngle
ads_getappname	acedGetAppName	ads_getargs	acedGetArgs
ads_getcfg	acedGetCfg	ads_getcname	acedGetCName
ads_getcorner	acedGetCorner	ads_getdist	acedGetDist
ads_getenv	acedGetEnv	ads_getfiled	acedGetFileD
ads_getfuncode	acedGetFunCode	ads_getinput	acedGetInput
ads_getint	acedGetInt	ads_getkword	acedGetKword
ads_getorient	acedGetOrient	ads_getpoint	acedGetPoint
ads_getreal	acedGetReal	ads_getstring	acedGetString
ads_getstringb	acedGetStringB	ads_getsym	acedGetSym
ads_getvar	acedGetVar	ads_graphscr	acedGraphScr
ads_grdraw	acedGrDraw	ads_grread	acedGrRead

ADS editing functions (*continued*)			
ADS Name	**New Name**	**ADS Name**	**New Name**
ads_grtext	acedGrText	ads_grvecs	acedGrVecs
ads_help	acedHelp	ads_initget	acedInitGet
ads_invoke	acedInvoke	ads_menucmd	acedMenuCmd
ads_nentsel	acedNEntSel	ads_nentselp	acedNEntSelP
ads_osnap	acedOsnap	ads_prompt	acedPrompt
ads_putsym	acedPutSym	ads_redraw	acedRedraw
ads_regfunc	acedRegFunc	ads_retint	acedRetInt
ads_retlist	acedRetList	ads_retname	acedRetName
ads_retnil	acedRetNil	ads_retpoint	acedRetPoint
ads_retreal	acedRetReal	ads_retstr	acedRetStr
ads_rett	acedRetT	ads_retval	acedRetVal
ads_retvoid	acedRetVoid	ads_setcfg	acedSetCfg
ads_setenv	acedSetEnv	ads_setfunhelp	acedSetFunHelp
ads_setvar	acedSetVar	ads_setview	acedSetView
ads_ssadd	acedSSAdd	ads_ssdel	acedSSDel
ads_ssfree	acedSSFree	ads_ssget	acedSSGet
ads_ssgetfirst	acedSSGetFirst	ads_sslength	acedSSLength
ads_ssmemb	acedSSMemb	ads_ssname	acedSSName
ads_ssnamex	acedSSNameX	ads_sssetfirst	acedSSSetFirst
ads_tablet	acedTablet	ads_textbox	acedTextBox
ads_textpage	acedTextPage	ads_textscr	acedTextScr
ads_trans	acedTrans	ads_undef	acedUndef
ads_update	acedUpdate	ads_usrbrk	acedUsrBrk

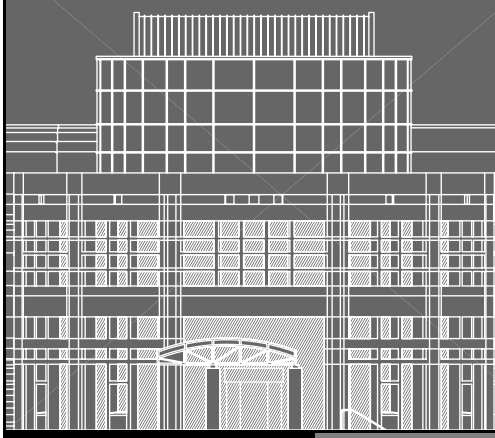
ADS editing functions (*continued*)			
ADS Name	**New Name**	**ADS Name**	**New Name**
ads_vports	acedVports	ads_xformss	acedXformSS

ADS Name	New Name
ads_ssGetKwordCallbackPtr	acedSSGetKwordCallbackPtr
ads_ssGetOtherCallbackPtr	acedSSGetOtherCallbackPtr
ads_ssSetKwordCallbackPtr	acedSSSetKwordCallbackPtr
ads_ssSetOtherCallbackPtr	acedSSSetOtherCallbackPtr

New Names for ADS Utility Functions

The following functions have been grouped together because they all perform a utility function, such as identifying a setting. Because of this, their new prefix is "acut":

ADS utility functions			
ADS Name	**New Name**	**ADS Name**	**New Name**
ads_angle	acutAngle	ads_buildlist	acutBuildList
ads_cvunit	acutCvUnit	ads_distance	acutDistance
ads_isalnum	acutIsAlNum	ads_isalpha	acutIsAlpha
ads_iscntrl	acutIsCntrl	ads_isdigit	acutIsDigit
ads_isgraph	acutIsGraph	ads_islower	acutIsLower
ads_isprint	acutIsPrint	ads_ispunct	acutIsPunct
ads_isspace	acutIsSpace	ads_isupper	acutIsUpper
ads_isxdigit	acutIsXDigit	ads_newrb	acutNewRb
ads_polar	acutPolar	ads_printf	acutPrintf
ads_relrb	acutRelRb	ads_tolower	acutToLower
ads_toupper	acutToUpper	ads_wcmatch	acutWcMatch



10

API Modernizations

This chapter reviews all the SDK changes that were not part of a specific feature for AutoCAD 2000.

In This Chapter

- Memory Management
- Class Versioning
- Embedded Objects
- Drawing Creation and Update Times
- DllEntryPoint@12 Removal
- Universal 8-Byte Alignment
- AutoLISP List Return Values in ObjectARX
- New DocLockMode Setting
- Class Changes
- Global Function Changes

Memory Management

The following memory management functions are no longer available, and have been superseded by new functions:

Removed memory management functions

Removed Function	New Function
newAcDbString()	acutNewString()
newAcDbBuffer()	acutNewBuffer()
updAcDbString()	acutUpdString()
freeAcDbString()	acutDelString()
freeAcDbBuffer()	acutDelString()

These functions are declared in the *acutmem.h* header file. There are also `#define` macros in *acutmem.h* to map the old names to the new names.

In addition, the following memory management functions are being deprecated in AutoCAD 2000. The following table shows the functions along with their suggested replacements:

Deprecated memory management functions

Deprecated Function	New Function
acad_malloc()	acutNewBuffer()
acad_free()	acutDelString()
acad_msize()	acutUpdString()
acad_realloc()	acutUpdString()
acad_calloc()	acutNewBuffer()
acad_strdup()	acutNewString()

AutoCAD no longer uses a third-party memory management system. This means that you should be able to use bounds checking and memory leak tools successfully during software development.

Class Versioning

Beginning with AutoCAD 2000, every custom class must provide a drawing and maintenance version number. The drawing value corresponds to the release of AutoCAD that was current when the class was created. The maintenance value can be set to whatever is appropriate for your class. For ObjectARX classes, the maintenance value will be set to zero every time the drawing version changes due to a new AutoCAD release. The version values are defined in the *acdb.h* header file. The `ACRX_DXF_DEFINE_MEMBERS` macro has been modified in AutoCAD 2000 to take two new arguments, `DWG_VERSION` and `MAINTENANCE_VERSION`:

```
#define ACRX_DXF_DEFINE_MEMBERS(CLASS_NAME,PARENT_CLASS,\
        DWG_VERSION,MAINTENANCE_VERSION,PROXY_FLAGS,DXF_NAME,APP)
```

These two arguments must be provided, and there are no defaults. The new arguments specify the version when the class was introduced. They become data members of the `AcRxClass`, but they are not persistent, that is, they are not stored in the class section of DWG and DXF files. See the *ObjectARX Developer's Guide* for a complete description and examples of class versioning.

Embedded Objects

It is now possible for your application to support embedded objects for display and when reading and writing DWG and DXF files.

Overview

There are essentially two ways to embed an object in another object. Either the encapsulating object has a data member that is an actual embedded object, or the encapsulating object has a pointer to an object (that object is then considered to be embedded). In either case the encapsulating object is responsible for the allocation and deallocation of the embedded object. The encapsulating object must also forward all calls to the embedded object's methods since AutoCAD will know nothing about the embedded object. So, to display the embedded object, the encapsulating object's `worldDraw()` must make a call to the embedded object's `worldDraw()`.

For `getGripPoints()` and `getStretchPoints()` you will need to call the embedded object's method last so that its points will end up with indices higher than those of the encapsulating entity. Then, in `moveGripPointsAt()` and `moveStretchPointsAt()`, if any of the indices passed in are above the range of those of the encapsulating entity, you should subtract off the highest index value of the encapsulating entity and pass the result into the embedded object's `moveGripPointsAt()` or `moveStretchPointsAt()`. For example, here's the pertinent code from a version of the ARXLAB's jblob sample, updated to have an embedded `AcDbCircle` entity:

```
Acad::ErrorStatus
Jblob::getGripPoints(
    AcGePoint3dArray& gripPoints,
    AcDbIntArray& osnapMasks,
    AcDbIntArray& geomIds) const
{
    assertReadEnabled();
    gripPoints.append(mp);
    gripPoints.append(mp + 0.5 * (mpblob - mp));
    gripPoints.append(mpblob);
    AcGeVector3d xoff(mrblob, 0, 0);
    AcGeVector3d yoff(0, mrblob, 0);
    gripPoints.append(mpblob + xoff);
    gripPoints.append(mpblob + yoff);
    gripPoints.append(mpblob - xoff);
    gripPoints.append(mpblob - yoff);
    return circle.getGripPoints(gripPoints, osnapMasks, geomIds);
}

Acad::ErrorStatus
Jblob::moveGripPointsAt(
    const AcDbIntArray& indices,
    const AcGeVector3d& offset)
{
    AcGePoint3d oldquad, newquad;
    assertWriteEnabled();
    AcDbIntArray circleIndices;
    for (int i = 0; i < indices.length(); i++) {
        int idx = indices[i];
        switch(idx) {
        case 0:
            mp += offset;
            continue; // Stretch begin point.
        case 1:
            mp += offset;
            mpblob += offset;
            continue; //Move.
        case 2:
            mpblob += offset;
            continue; // Stretch blob center.
        //Stretch blob radius.
        //
```

```
        case 3:
            oldquad = mpblob + AcGeVector3d(mrblob, 0, 0);
            break;
        case 4:
            oldquad = mpblob + AcGeVector3d(0, mrblob, 0);
            break;
        case 5:
            oldquad = mpblob - AcGeVector3d(mrblob, 0, 0);
            break;
        case 6:
            oldquad = mpblob - AcGeVector3d(0, mrblob, 0);
            break;
        default:
            if (idx > 6)
                circleIndices.append(idx - 7);
            continue;
        }
        newquad = oldquad + offset;
        mrblob = newquad.distanceTo(mpblob);
    }
    if (circleIndices.length() > 0)
        return circle.moveGripPointsAt(circleIndices, offset);
    else
        return Acad::eOk;
}
```

Filing

For filing purposes, the encapsulating object must call the embedded object's `dwgOutFields()`, `dwgInFields()`, `dxfOutFields()`, and `dxfInFields()` methods from within its own methods.

Filing for DWG Files

For DWG filing, the call to the embedded object's `dwgOutFields()` and `dwgInFields()` methods may occur at any point in the encapsulating object's `dwgOutFields()` (after the call to the base class's method, of course). Just make sure that the call occurs in the same place in both `dwgOutFields()` and `dwgInFields()`. For example, here is the pertinent code from the updated jblob sample program:

```
Acad::ErrorStatus
Jblob::dwgInFields(
    AcDbDwgFiler* filer)
{
    assertWriteEnabled();
    AcDbEntity::dwgInFields(filer);
    filer->readItem(&mp);
    filer->readItem(&mpblob);
    filer->readItem(&mrblob);
    filer->readItem(&mnormal);
    return circle.dwgInFields(filer);
}
```

```
Acad::ErrorStatus
Jblob::dwgOutFields(
    AcDbDwgFiler* filer) const
{
    assertReadEnabled();
    AcDbEntity::dwgOutFields(filer);
    filer->writeItem(mp);
    filer->writeItem(mpblob);
    filer->writeItem(mrblob);
    filer->writeItem(mnormal);
    return circle.dwgOutFields(filer);
}
```

Filing for DXF Files

For DXF filing, the embedded object must be filed out/in after all the data of the encapsulating object has been filed out/in, so the call to the embedded object's `dxfOutFields()`/`dxfInFields()` method should come last in the encapsulating object's `dxfOutFields()`/`dxfInFields()` methods' code. There is also a separator between the encapsulating object's data and the subsequent embedded object's data. This separator is similar in function to the groups 0 or 100 in that it causes the filer to stop reading data. The normal DXF group code 0 cannot be used because DXF proxies use the group code 0 to tell when to stop reading data. Therefore, the new DXF group code 101 has been introduced.

The new `AcDb::DxfCode` enum value for DXF group code 101 is

```
AcDb::kDxfEmbeddedObjectStart
```

The data string "Embedded Object" will be automatically written out by the filer for this DXF group code. There are also two new methods in the `AcDbDxfFiler` class: `writeEmbeddedObjectStart()` and `atEmbeddedObjectStart()`.

```
virtual Acad::ErrorStatus
AcDbDxfFiler::writeEmbeddedObjectStart();
```

The implementation of this function should be (and is in the ARX internal filers) to write out the `AcDb::DxfCode` of `AcDb::kDxfEmbeddedObjectStart` (DXF group code 101) and suppress the writing out of the `AcDb::DxfCode` of `AcDb::kDxfStart` (DXF group code 0) for the embedded object that follows. A string data value of "Embedded Object" should always be (and is in the ARX internal filers) written out for the `AcDb::kDxfEmbeddedObjectStart` group code.

This method must be called in the encapsulating object's `dxfOutFields()` method just before calling the `dxfOutFields()` method of the embedded object. If multiple embedded objects are involved, then this method must be called just before each embedded object's `dxfOutFields()` method is called.

This method should only be called in the `dxfOutFields()` of the encapsulating object.

The base class implementation of this method is to abort the program, so it must be overridden in derived classes in which the method is expected to be called. The method returns `Acad::eOk`:

```
virtual Adesk::Boolean
AcDbDxfFiler::atEmbeddedObjectStart();
```

The implementation of this function should be (and is in the ARX internal filers) to test to see if the filer is currently pointing to an item with an `AcDb::DxfCode` of `AcDb::kDxfEmbeddedObjectStart`. If this is the case, then the filer moves the file pointer to the next item and returns `Adesk::kTrue`. Otherwise the file pointer is left alone and `Adesk::kFalse` is returned.

The base class implementation of this method is to abort the program, so it must be overridden in derived classes on which the method is expected to be called.

To demonstrate how these new methods are used, an example of the pertinent code from an updated jblob sample program is provided here:

```
Acad::ErrorStatus
Jblob::dxfInFields(
    AcDbDwgFiler* filer)
{
    assertWriteEnabled();
    struct resbuf rb;
    Acad::ErrorStatus es = AcDbEntity::dxfInFields(filer);
    if (es != Acad::eOk) {
        return es;
    }
    if (!filer->atSubclassData(kClassName)) {
        return Acad::eBadDxfSequence;
    }
    mnormal = AcGeVector3d(0, 0, 1); //Set default value.
    while (es == Acad::eOk) {
        if ((es = filer->readItem(&rb)) == Acad::eOk) {
            switch(rb.restype) {
            case AcDb::kDxfXCoord:
                mp.set(rb.resval.rpoint[X], rb.resval.rpoint[Y],
                    rb.resval.rpoint[Z]);
                break;
            case AcDb::kDxfXCoord+1:
              mpblob.set(rb.resval.rpoint[X], rb.resval.rpoint[Y],
                    rb.resval.rpoint[Z]);
                break;
            case AcDb::kDxfReal:
                mrblob = rb.resval.rreal;
                break;
```

```
            case AcDb::kDxfNormalX:
             mnormal.set(rb.resval.rpoint[X], rb.resval.rpoint[Y],
                    rb.resval.rpoint[Z]);
            }
        }
    }
    if (filer->atEmbeddedObjectStart())
        return circle.dxfInFields(filer);
    else {
        filer->setError(Acad::eMissingDxfField,
            "missing expected embeddedObject marker");
        return filer->filerStatus();
    }
}

Acad::ErrorStatus
Jblob::dxfOutFields(
    AcDbDxfFiler* filer) const
{
    assertReadEnabled();
    AcDbEntity::dxfOutFields(filer);
    filer->writeItem(AcDb::kDxfSubclass, kClassName);
    filer->writeItem(AcDb::kDxfXCoord, mp);
    filer->writeItem(AcDb::kDxfXCoord + 1, mpblob);
    filer->writeItem(AcDb::kDxfReal, mrblob);
    if (filer->includesDefaultValues()
        || mnormal != AcGeVector3d(0,0,1))
    {
        filer->writeItem(AcDb::kDxfNormalX, mnormal);
    }
    filer->writeEmbeddedObjectStart();
    return circle.dxfOutFields(filer);
}
```

The output in a DXF file (the 310 group data strings have been shortened for readability) looks like this:

```
0
JBLOB
  5
52
330
19
100
AcDbEntity
  8
0
 92
      256
310
0001000004000003C0000006000000002000000...
310
0000000000000000000000000000F03F700000...
310
0000
100
```

```
Jblob
 10
4.026791
 20
3.172968
 30
0.0
 11
5.916743
 21
5.299622
 31
0.0
 40
1.458724
101
Embedded Object
100
AcDbEntity
100
AcDbCircle
 10
5.916743
 20
5.299622
 30
0.0
 40
0.729362
```

Drawing Creation and Update Times

Two new sysvars, TDCREATE and TDUPDATE, have been created to contain the create and update times of the current drawing in universal time. The existing sysvars, TDCREATE and TDUPDATE, contain those times in local time.

Also, the TDCREATE and TDUPDATE values are now correct in any time zone. Thus if you create a drawing at 1:00 p.m. in New York and open the drawing in Chicago, the TDCREATE value will be 12:00 p.m. In previous releases of AutoCAD, TDCREATE would have incorrectly been 1:00 p.m.

New AcDbDatabase Methods

```
const AcDbDate
tducreate() const;
```

This method returns the current drawing's creation date and time in universal time, as opposed to `tdcreate()`, which returns it in local time.

```
const AcDbDate
tduupdate() const;
```

`tduupdate()` returns the current drawing's update date and time in universal time, as opposed to `tdupdate()`, which returns it in local time.

New AcDbDate Methods

```
void
getTime(
    SYSTEMTIME& st) const;
```

This method returns the `AcDbDate` object's value in a Win32 SYSTEMTIME structure.

```
void
setTime(
    const SYSTEMTIME& st);
```

`setTime()` sets the `AcDbDate` object's value from a Win32 SYSTEMTIME structure.

```
void
localToUniversal();
```

`localToUniversal()` converts the `AcDbDate` object's value from local to universal time.

```
void
universalToLocal();
```

This method converts the `AcDbDate` object's value from universal to local time.

New AcDbIndex Methods

```
AcDbDate
lastUpdateAtU() const;
```

This method gets the `AcDbIndex` object's last updated date and time in universal time. This is different from the `lastUpdatedAt()` method, which gets the value in local time.

```
void
setLastUpdateAtU(
    const AcDbDate& time) const;
```

This method sets the `AcDbIndex` object's last updated date and time in universal time, as opposed to `setLastUpdatedAt()`, which sets the timestamp in local time.

DllEntryPoint@12 Removal

Prior to AutoCAD 2000, every ObjectARX application had a `DllEntryPoint@12` (defined in *rxapi.lib*) as its entry point, to be called by the operating system. Because of this, you had to explicitly specify "DllEntryPoint@12" as the "Entry-Point Symbol" on the Link tab in the MSDEV C++ Project Settings.

This restriction has been removed. The static library *rxapi.lib* no longer has a `DllEntryPoint@12` implementation, and you will need to remove `DllEntryPoint@12` as your application's entry-point symbol.

To remove the symbol

1. Start the MS Developer's IDE.
2. Open your existing project file.
3. Go to the Project menu and select "Settings..."
4. Select the Link tab.
5. Remove DllEntryPoint@12 from the "Entry-point symbol" edit box.

If you don't make this change, you will get unresolved symbol "DllEntryPoint@12" messages.

_hdllInstance Removal

As a side-effect of removing `DllEntryPoint@12` from *rxapi.lib*, the `_hdllInstance` variable was removed too. In order to get the instance handle to your ObjectARX application, you can now define `DllMain()`. `DllMain()` will be called by the runtime code, and the first argument of the function passed in will be your DLL instance handle.

In order to use `DllMain()` correctly, some existing ObjectARX applications will require editing. Any application that uses MFC, and has `_USRDLL` defined, should be changed by replacing `_USRDLL` with `_AFXEXT`. The reason for this change is that in Release 14, ObjectARX applications using MFC were MFC extension DLLs. `_USRDLL` was used with `_AFXDLL` to indicate that the application was a "Regular DLL using shared MFC DLL." Autodesk does not support this type of application.

Universal 8-Byte Alignment

In previous releases, some AutoCAD classes and structures used an 8-byte alignment (which is the MSVC++ default) and others used a single-byte alignment. In this release, all classes and structures use an 8-byte alignment.

Also in previous releases, the symbol RADPACK was defined to turn on this 1-byte alignment in header files that declared certain interface structures. This RADPACK symbol is no longer defined in any headers, and it is no longer necessary for developers to define it in their makefiles.

The most commonly used structure that this affects is the `resbuf` struct. Because its first field (`restype`) is a short, the location of the next field (`resval`) is different in this release from previous releases. This should not cause any problems in most cases, but there are a few things to watch out for:

- All modules that access `resbuf` structures should be recompiled with the new headers. (Of course you should be recompiling all of your modules anyway.)
- If you were declaring your own structure that mirrored the ObjectARX `resbuf` struct (and casting back and forth between the two types), you would need to modify your own structure definition to have the default alignment.
- If you are writing `resbuf` structures directly to a file and then reading them back in, this change in the `resbuf` layout will render old files incompatible with new ones. This is an unlikely scenario but is mentioned here for completeness.

AutoLISP List Return Values in ObjectARX

AutoLISP list return values from ObjectARX work differently in AutoCAD 2000 than they did in AutoCAD Release 14. This difference appears in the use of the `acedGetArgs()` (formerly `ads_getargs()`) and `acedRetList()` (formerly `ads_retlist()`) functions.

In Release 14, entity data lists were treated specially by `ads_getargs()`. For an entity data list, the linked list of resbuf structures returned by `ads_getargs()` had no resbuf with type code RTLB at the head of the list and matching resbuf with type code RTLE at the end of the list to act as a bracketing "wrapper" pair indicating that the data was all within a single list. If the list was not an entity data list, then when a list was passed to ObjectARX, `ads_getargs()` returned a resbuf chain with an RTLB/RTLE pair.

This special treatment of entity data lists was not desirable - all lists should be treated the same by `acedGetArgs()` and `acedRetList()`.

Another problem with the way Release 14 treated lists was that `ads_retlist()` always added an RTLB/RTLE "wrapper" pair. When passing a list back and forth between AutoLISP and ObjectARX, this caused the list to be repeatedly nested one level deeper in a set of nested lists. For example, if (list "a" "b") was passed to ObjectARX from AutoLISP, then returned to AutoLISP, then passed to ObjectARX, then returned to AutoLISP, the result would have been:

```
(((“a” “b”)))
```

but the original was simply:

```
(“a” “b”)
```

In AutoCAD 2000 the behavior of `acedGetArgs()` and `acedRetList()` has changed so that they treat all lists the same and in a fashion that does not alter the list during a round trip.

The rule that AutoCAD 2000 follows for `acedGetArgs()` is that if the data passed out to ObjectARX is a list, then there will be an RTLB/RTLE for each list nesting level. This is the way Release 14 treated non-entity data lists.

The rules AutoCAD 2000 follows for `acedRetList()` are that if the resbuf chain passed to `acedRetList()` is already a valid list (that is, it has a “wrapper” RTLB/RTLE resbuf pair), then no RTLB/RTLE wrapper resbuf pair will be added. If the resbuf chain is not a valid list, and `acedRetList()` is returning the list to AutoLISP (as opposed to another ObjectARX program) then an RTLB/RTLE resbuf wrapper pair will be added internally (not to the actual resbuf linked list passed into `acedRetList()`) to make it a valid list. When returning the data to AutoLISP, `acedRetList()` must add an RTLB/RTLE wrapper to a resbuf chain that is not a valid list because AutoLISP cannot handle the data otherwise.

These rules apply to all data passed to ObjectARX and obtained via `acedGetArgs()` and all resbuf chains passed to `acedRetList()`, so there is no more special casing just for entity data lists.

If you need to have the old Release 14 behavior, you can write a utility function that will add the extra RTLB/RTLE wrapper as Release 14 did. You can also write a similar utility function to strip the RTLB/RTLE resbuf wrapper pair off of entity data list resbuf chains returned by `acedGetArgs()`.

New DocLockMode Setting

There is a new `AcAp::DocLockMode` document lock setting, `AcAp::kProtectedAutoWrite`, defined in *acdocman.h*.

If your application tests for document lock mode settings, the `kProtectedAutoWrite` lock mode should be treated the same as the `kWrite` and `kAutoWrite` modes. If you test for those modes by ORing with the lock setting, `kProtectedAutoWrite` will match the `kWrite` bit, as well as have a separate bit for a distinct identity. If you test for these modes by equality, or with cases in a switch statement, you must add consideration for `kProtectedAutoWrite`.

Here is a sample function which is affected by this change, and what must be done to accommodate the new mode:

```
// lockDocForWriteIfNecessary
//
// Returns true if the document is indeed locked sufficiently for
// write, false if the document wasn't locked and couldn't be
// locked. If true is returned, then output parameter lockedIt
// indicates that this function did the locking, thus calling for
// an unlock later.

static
bool lockDocForWriteIfNecessary(AcApDocument* pDoc, bool& lockedIt)
{
    Acad::ErrorStatus es;
    switch (pDoc->myLockMode()) {
    case AcAp::kNone:
    case AcAp::kNotLocked:
    case AcAp::kRead:
        es = acDocManager->lockDocument(pDoc);
        lockedIt = true;
        return es == Acad::eOk;
    case AcAp::kWrite:
    case AcAp::kAutoWrite:
   case AcAp::kProtectedAutoWrite: // *** New with AutoCAD 2000 ***
    case AcAp::kXWrite:
        lockedIt = false;
        return true;
    }
    // New Lock Mode? Need to classify it accordingly.
    acrx_abort("Internal Error 1");
   return false; // Will not be executed - prevents compiler errors.
}
```

Unless you treat `kWrite` and `kAutoWrite` modes differently in your logic, or you actually set the `kAutoWrite` document lock mode, all you need is to treat `kProtectedAutoWrite`, `kAutoWrite` and `kWrite` the same way. See the `DocLockMode` section in the *ObjectARX Reference* for more information.

Class Changes

This section describes the changes that have been made to ObjectARX classes but don't belong to a particular feature or command.

AcArray Template Class

All the array classes used in ObjectARX are now created with the template class `AcArray`, so all ObjectARX arrays of specific objects (such as `AcDbIntArray` or `AcDbObjectIdArray`) use `AcArray` to obtain their definition. Arrays created this way are dynamic arrays. The term "dynamic array" means that the array can grow without bounds, unlike declaring an array of items in the usual manner. For example, an array declared by

```
AcDbObjectId myArray[10];
```

is limited to holding only ten `AcDbObjectId` objects.

Since `AcArray` is a template class, it is useful when you need to create an array of any object type. For example, to declare a dynamic array of `WxyzMyObj` objects you can make the following declaration:

```
typedef AcArray<WxyzMyObj> WxyzMyObjArray;
```

Note that a forward declaration of the form

```
class AcDbObjectIdArray;
```

is not valid for classes created with the `AcArray` template. Instead, include the header file with the proper class definition.

See the *ObjectARX Reference* for basic concepts and method definitions of `AcArray`.

AcDbCurve Class

To allow the OFFSET command to support linear entities properly, the new method `getOffsetCurvesGivenPlaneNormal()` has been added at the `AcDbCurve` class level. This method was needed because a linear entity has no concept of a plane on which to be offset, so a plane must be provided.

This new method has been implemented for the `AcDbRay`, `AcDbXline`, and `AcDbLine` classes. The OFFSET command code has also been updated to make use of this new method for linear entities—if the entity's `getPlane()` method returns a planarity value of `AcDb::kLinear`, then the new method will be used. For entities with planarity of `AcDb::kPlanar`, the `getOffsetCurves()` method will still be used:

```
virtual Acad::ErrorStatus
AcDbCurve::getOffsetCurvesGivenPlaneNormal(
    const AcGeVector3d& normal,
    double offsetDist,
    AcDbVoidPtrArray& offsetCurves) const;
```

normal	Input the normal vector of the offset plane
offsetDist	Input the distance to offset the curve
offsetCurves	Returns an array of pointers to the resulting curve(s)

The method creates one or more entities that together make up the result of offsetting the curve by the distance `offsetDist` in the plane with normal vector `normal`. For many curves, the result will be a single new curve, but the new curve may not be of the same class as the original curve. For example, offsetting an `AcDbEllipse` will result in an `AcDbSpline` since the result of offsetting an ellipse does not fit the equation of an ellipse. In some cases it might be necessary for the offset result to be several curves. To allow for this possibility, a dynamic array of void pointers is used to hold pointers to the resulting entity or entities.

To use this array of pointers, your application needs to cast the pointer(s) to the appropriate object type(s). If only a specific entity type or set of types is to be handled, then using the cast method of the class(es) desired to see if each pointer can be safely cast to that object type will work. If the actual object type is desired, then each object's `isA()` method may be used to get a pointer to the object's `AcRxClass` object, which has a `name()` method to get the `classname` of the object.

If the `offsetDist` value is negative, this is usually interpreted as being an offset to make a "smaller" curve. For example, for an arc the method would offset to a radius that is `offsetDist` less than the starting curve's radius. Or, if "smaller" has no meaning, then a negative `offsetDist` may be interpreted as an offset in the direction of smaller *X,Y,Z* WCS coordinates. However, this is not enforced, so custom entities can interpret the sign of the `offsetDist` value however they wish.

The entities that are returned in the `offsetCurves` array are dynamically allocated but have not been added to an `AcDbDatabase` yet, so the application that calls this function is responsible for their memory. If they are subsequently appended to a database, then the database takes over responsibility for their memory; otherwise, the application is responsible for deleting them when they are no longer needed.

The method returns `Acad::eOk` if offsetting is successfully completed. If the offset distance is invalid (for example, if you are offsetting an arc such that the offset result would be a negative radius), or if `normal` is a zero-length vec-

tor, or if the entity is linear and `normal` is not perpendicular with the entity, then `Acad::eInvalidInput` is returned. For the AutoCAD built-in classes that use ACIS (`AcDbEllipse`, `AcDbSpline`, `AcDbBody`, `AcDbRegion`, and `AcDb3dSolid`), `Acad::eGeneralModelingFailure` will be returned if an error occurs in the ACIS modeler. Other `ErrorStatus` return values are implementation-dependent. The method's default implementation returns `Acad::eNotImplemented`.

AcDbDatabase Class

The signature of `AcDbDatabase::readDwgFile()` has been changed to take two additional arguments: an integer for specifying an access Share mode, and an `Adesk::Boolean` for doing codepage conversion:

```
Acad::ErrorStatus
readDwgFile(
    const char* fileName,
    int shmode = _SH_DENYWR,
    Adesk::Boolean bAllowCPConversion = Adesk::kFalse);
```

fileName — Name of the drawing file to read

shmode — Share mode to use when opening the drawing file

bAllowCPConversion — Controls automatically converting between codepages

Share Mode

The Share mode describes how other applications may access the file when the same file has already been opened. Share mode values are defined in the MSVC++ header file *share.h*. The following are Share modes supported by AutoCAD:

Share modes supported by AutoCAD

Mode	Function
_SH_DENYWR	Opens an existing file as read-only Denies write-access to the file by other sessions
_SH_DENYRW	Opens a file for reading and writing Denies read-write access to the file by other sessions
_SH_DENYNO	Opens an existing file as read-only Permits read-write access to the file by other sessions

The `shmode` argument defaults to `_SH_DENYWR`. This default value gives the same behavior as in previous releases of ObjectARX, and prevents other applications from writing to the file. Also, if the file was already opened by another application for writing, this call will fail. You may also choose to open a file with the other Share modes as shown in the following examples:

- `readDwgFile(fName, _SH_DENYRW)`

  Prevents other applications from reading and writing to the file when the file is opened by this call. A `readDwgFile()` call using this Share mode will fail if any other application has the drawing file open, or if the drawing file is read-only.

- `readDwgFile(fName, _SH_DENYNO)`

  Allows other applications full access to the file when the file is opened by this call. This call will open a file for reading when the same file is already opened for writing.

Using `_SH_DENYNO` does not lock out other applications from writing to the file. It is not safe to do a lazy load in such circumstances, so your application should call the `closeInput()` method for this Share mode to force a full load.

NOTE Performing a lazy load means that pieces of the drawing are read-in only as needed. This feature, combined with the possibility of other applications updating the drawing file under `_SH_DENYNO`, can cause inconsistent reads and other errors without the call to `closeInput()`. Also, the call to `closeInput()` will release the file handle when it finishes reading from the file.

Current Document Association

The following `AcDbDatabase` methods have a new argument, `noDocument`:

```
AcDbDatabase(
    Adesk::Boolean buildDefaultDrawing = Adesk::kTrue,
    Adesk::Boolean noDocument = Adesk::kFalse);

static Acad::ErrorStatus
open(
    AcDbDatabase*& pNewDb,
    AcDbDwgFiler* pInputFiler,
    Adesk::Boolean enableUndo = Adesk::kFalse,
    Adesk::Boolean noDocument = Adesk::kFalse);

static Acad::ErrorStatus
salvage(
    AcDbDatabase*& pNewDb,
    AcDbDwgFiler* pInputFiler,
    Adesk::Boolean noDocument = Adesk::kFalse);
```

If `noDocument` is `Adesk::kTrue`, the database won't have any association with the currently active document. That is, the database will not use the default transaction manager, undo filer, locking, or other services of the current document.

NOTE The `open()` and `salvage()` methods are primarily for AutoCAD internal use.

Language Conversion

Under AutoCAD Release 14, if a DWG file was in a codepage for which the National Language Support (NLS) file was not available on the user's system (for example, a Kanji drawing file on a standard English system), the action

taken was to display a message box asking users if they wanted to do some kind of default conversion anyway, and warning them that a possible loss of data may occur.

This behavior did not allow third-party applications the option of not displaying the dialog, or controlling how to perform any conversion. The new `bAllowCPConversion` argument adds this option. If the argument is set to true, the conversion will silently happen. The default value is `Adesk::kFalse`, which keeps the behavior consistent for those who choose not to pass this new argument. In addition, for AutoCAD only, if the argument is false, the message box querying the user will continue to be displayed.

Undo Recording

```
Adesk::Boolean
AcDbDatabase::undoRecording() const
```

This new method returns the undo recording status of the database.

AcDbDatabaseReactor Class

Two new methods have been added to `AcDbDatabaseReactor` to handle header system variable (sysvar) notifications. The new methods are

```
virtual void
headerSysVarWillChange(
    const AcDbDatabase* dwg,
    const char* name);

virtual void
headerSysVarChanged(
    const AcDbDatabase* dwg,
    const char* name,
    Adesk::Boolean bSuccess);
```

The sysvar name in the `name` argument must be in uppercase. To find out if a sysvar is part of the drawing database, check for set and get methods for the sysvar in `AcDbDatabase` class. In the ObjectDBX environment, you should add a database reactor to get notifications for sysvars that are part of `AcDbDatabase`.

Now, when an `AcDbDatabase` sysvar is changed, both a database notification and also the old editor sysvar notification are sent. This is done to preserve backward compatibility. The editor notification won't be sent in the next release of AutoCAD.

AcDbLayerTableRecord Class

The enum `AcDb::kDxfLayerPlotBit` has been moved from the `AcDb` struct (in *acdb.h*) to the `AcDbLayerTableRecord` class definition (in *dbsymtb.h*). The enum is now accessible as `AcDbLayerTableRecord::kDxfLayerPlotBit`. This enum is a mnemonic name for the DXF group code that layer records use for storing the `isPlottable` value of a table record. This enum is equal to `AcDb::kDxfBool` (group code 290).

AcDbLongTransaction Class

The `AcDbLongTransaction::addToWorkSet()` method now performs a self-reference check if the object to be added is an `AcDbBlockReference`. If the reference would cause a self-reference, it is not added to the workset, and the method returns `eSelfReference`.

AcDbObject Class

There have always been a series of xdata data types that are meant to be transformed whenever the object on which they attached is transformed. The data groups are

- 1011, 1021, 1031 – World Space Position that is moved, scaled, rotated, and mirrored along with the parent object.
- 1012, 1022, 1032 – World Space Displacement that is scaled, rotated, and mirrored along with the parent object (but not moved or stretched).
- 1013, 1023, 1033 – World Direction that is rotated and mirrored along with the parent object (but not moved, scaled, or stretched).
- 1041 – Distance that is scaled along with the parent object.
- 1042 – Scale Factor that is scaled along with the parent object.

These xdata types were not being transformed by the `transformBy()` methods on the various entities in AutoCAD, nor was there any convenient way to make the `transformBy()` methods transform the xdata, which caused a problem. To address this, a new method of `AcDbObject` has been introduced that will do the transformation of the object's xdata. This new method is used in all the `transformBy()` methods of the built-in entity classes.

NOTE This method should also be called in all custom entity `transformBy()` methods in your application.

The new method is

```
Acad::ErrorStatus
xDataTransformBy(
    AcGeMatrix& xform);
```

xform	The transformation matrix to be applied to the xdata

This method applies the xform transformation matrix to any xdata data types 1011, 1021, 1031, 1012, 1022, 1032, 1013, 1023, 1033, 1041, and 1042 in the object's xdata, and will return `Acad::eOk` if successful. See the documentation on xdata in the *ObjectARX Reference* for more information on the way the transformation matrix is applied to the different data types.

This method should be called from within the `transformBy()` method on custom entities so that such entities will transform xdata in the same way that the built-in AcDb entities do.

AcDbObjectId Class

A new method in the `AcDbObjectId` class provides a direct way to get an object's erased status:

```
Adesk::Boolean
isErased() const;
```

This method returns `Adesk::kTrue` if the object identified by this ObjectId is currently erased, or `Adesk::kFalse` if not.

In earlier releases of AutoCAD, it was necessary to open an object and then query its erased status to determine if the object was erased or not. That could have involved the use of a transaction if the object was already open for write. The new `isErased()` method provides the object's erased status without opening the object.

AcEditorReactor Class

There is a new callback function in `AcEditorReactor` that indicates that the drawing database was not saved, and that the save operation failed:

```
virtual void
abortSave(
    AcDbDatabase* pDb);
```

The `pDb` argument is a pointer to the source drawing database. This callback is in the same group as the existing `beginSave()` and `saveComplete()` notifications.

AcGeCompositeCurve2d Class

```
AcGeCompositeCurve2d&
setCurveList(
    const AcGeVoidPointerArray& curveList,
    const AcGeIntArray& isOwnerOfCurves);
```

This method is similar to the class constructor with the same arguments, and constructs a composite curve from the list of input curves. The input curves should be connected at their endpoints, although no check of this is made by the constructor. That is, the start point of each curve in the list (excluding the first curve) should be equal to the endpoint of the previous curve in the list. The number of entries in `isOwnerOfCurves` must be the same as the number of entries in `curves`. Each entry in `isOwnerOfCurves` specifies whether the corresponding curve in `curves` should be deleted when this composite is deleted. If the entry in `isOwnerOfCurves` is zero, then the composite curve destructor will not delete the corresponding curve. If the entry in `isOwnerOfCurves` is any non-zero value, then the corresponding curve will be deleted by the composite curve destructor.

AcGeCompositeCurve3d Class

```
AcGeCompositeCurve3d&
setCurveList(
    const AcGeVoidPointerArray& curves,
    const AcGeIntArray& isOwnerOfCurves);
```

This method is similar to the class constructor with the same arguments, and constructs a composite curve from the list of input curves. The input curves should be connected at their endpoints, although no check of this is made by the constructor. That is, the start point of each curve in the list (excluding the first curve) should be equal to the endpoint of the previous curve in the list. The number of entries in `isOwnerOfCurves` must be the same as the number of entries in `curves`. Each entry in `isOwnerOfCurves` specifies whether the corresponding curve in `curves` should be deleted when this composite is deleted. If the entry in `isOwnerOfCurves` is zero, then the composite curve destructor will not delete the corresponding curve. If the entry in `isOwnerOfCurves` is any non-zero value, then the corresponding curve will be deleted by the composite curve destructor.

Global Function Changes

This section describes the changes to global functions that have been made to ObjectARX but that don't belong to a particular feature or command.

extern "C" Linkage Removed

The `extern "C"` has been removed from the following headers:

- *rxregsvc.h*
- *adslib.h*

This gives the functions in those headers C++ linkage instead of their previous C linkage.

acdbTextFind() Function

The `acdbTextFind()` function will search a drawing database for the specified text string. You may restrict the search to certain types of text or certain object IDs, and also replace found text with new text:

```
Adesk::Boolean
acdbTextFind(
    AcDbDatabase* pDatabase,
    AcDbObjectIdArray& resultSet,
    const char* findString,
    const char* replaceString = NULL,
    Adesk::UInt8 searchOptions = AC_SRCH_DEFAULT
    const AcDbObjectIdArray& selSet = 0);
```

pDatabase	Pointer to the database to be searched.
resultSet	Array of object IDs that satisfy the search.
findString	String to search for.
replaceString	String to replace the `findString` text with. The default value is `NULL`, in which case no replacement is done.
searchOpts	Specifies the search options. The options can be any of the values shown in the following table. The default search option is to search all text without regard to case or to whole words. This value is represented by `AC_SRCH_DEFAULT`.
selSet	An array of object IDs to be searched. This argument is optional, and, if not supplied, the whole database is searched.

The following table lists the available search options:

Search options		
Option Name	**Value**	**Meaning**
AC_SRCH_BLOCK	0x01	Only search text in blocks.
AC_SRCH_DIM_TEXT	0x02	Only search dimension text.
AC_SRCH_TEXT	0x04	Only search plain text.
AC_SRCH_LINK_NAME	0x08	Only search hyperlink names.
AC_SRCH_LINK_URL	0x10	Only search URLs.
AC_SRCH_MATCH_CASE	0x20	Match letter case when searching.
AC_SRCH_WHOLE_WORD	0x40	Match only whole words when searching.
AC_SRCH_DEFAULT	0x1F	An aggregate of the default search options.

`acdbTextFind()` will return `Adesk::kTrue` if it succeeds, and `Adesk::kFalse` otherwise. Success means that the function did not encounter any errors while looking at the entities. Therefore, it is possible to have a return value of `Adesk::kTrue` even if no match was found.

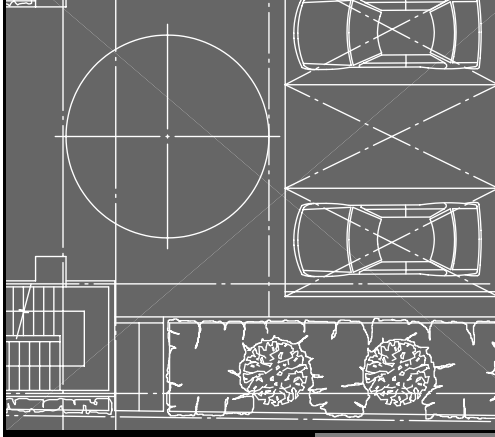
acedOleSelected() Function

A new API has been defined for retrieving selected OLE objects:

```
BOOL
acedOleSelected(
    ObjectID& oid);
```

This function returns `False` if no OLE object is selected. Otherwise, `True` is returned, and the `ObjectID` is passed out through the reference to `oid`.

ObjectARX Changes



In This Appendix

A

- New Items
- Changed Items
- Deprecated Items
- Removed Items

This appendix lists the changes in behavior from earlier ObjectARX releases and all of the individual API changes made to the ObjectARX SDK.

New Items

This section lists the enumerated types, classes, and global functions that have been added to ObjectARX for AutoCAD 2000.

New Enumerated Types

Several new enumerations were added to ObjectARX to make your code easier to write and clearer to read. The new enums are listed below in alphabetical order.

The line spacing value controls whether or not the spacing will be automatically adjusted:

LineSpacingStyle (used with mtext)		
Name	**Value**	**Usage**
kAtLeast	1	Allows the spacing between different lines of text to adjust automatically based on the height of the largest character in a line of text.
kExactly	2	Forces the line spacing to be the same size for all the lines in the mtext object regardless of formatting overrides.

The `LineWeight` enum lists all the built-in lineweights, covering the requirements for most drafting disciplines:

AcDb::LineWeight			
Name	**Value**	**Size in mm**	**Size in Inches**
kLnWt000	0		
kLnWt005	5	0.05	.002
kLnWt009	9	0.09	.003
kLnWt010	10	0.10	.004
kLnWt013	13	0.13	.005
kLnWt015	15	0.15	.006

AcDb::LineWeight (*continued*)			
Name	**Value**	**Size in mm**	**Size in Inches**
kLnWt018	18	0.18	.007
kLnWt020	20	0.20	.008
kLnWt025	25	0.25	.010
kLnWt030	30	0.30	.012
kLnWt035	35	0.35	.014
kLnWt040	40	0.40	.016
kLnWt050	50	0.50	.020
kLnWt053	53	0.53	.021
kLnWt060	60	0.60	.024
kLnWt070	70	0.70	.028
kLnWt080	80	0.80	.031
kLnWt090	90	0.90	.035
kLnWt100	100	1.00	.039
kLnWt106	106	1.06	.042
kLnWt120	120	1.20	.047
kLnWt140	140	1.40	.056
kLnWt158	158	1.58	.062
kLnWt200	200	2.00	.078
kLnWt211	211	2.11	.083
kLnWtByLayer	–1		
kLnWtByBlock	–2		
kLnWtByLwDefault	–3		

New Global Functions

This section lists the new global functions that are available with this release of ObjectARX for AutoCAD. The functions are grouped by feature.

These are the functions for creating earlier versions of DXF files:

DXF functions

Function	Usage
acdbDxfOutAsR12()	Acad::ErrorStatus acdbDxfOutAsR12(AcDbDatabase* pdb, const char* filename, int precision=16);
acdbDxfOutAsR13()	Acad::ErrorStatus acdbDxfOutAsR13(AcDbDatabase* pdb, const char* filename, int precision=16);
acdbDxfOutAsR14()	Acad::ErrorStatus acdbDxfOutAsR14(AcDbDatabase* pdb, const char* filename, int precision=16);
acdbGetBitmapPreviewFromDxfFile()	Acad::ErrorStatus acdbGetBitmapPreviewFromDxfFile(const char* filename, void*& pBitmap);

This is the list of the new functions for working with mtext:

Mtext functions

Function	Usage
setLineSpacingFactor	Acad::ErrorStatus setLineSpacingFactor(double dFactor);
lineSpacingFactor	double lineSpacingFactor() const;

Mtext functions (*continued*)	
Function	**Usage**
setLineSpacingStyle	Acad::ErrorStatus setLineSpacingStyle(LineSpacingStyle eStyle);
lineSpacingStyle	LineSpacingStyle lineSpacingStyle() const;
setAttachmentMovingLocation	Acad::ErrorStatus setAttachmentMovingLocation(AttachmentPoint type);

These are the functions for working with Shortcut menus:

Shortcut menu functions	
Function	**Usage**
acedAddDefaultContextMenu	Adesk::Boolean acedAddDefaultContextMenu(AcEdUIContext* pContext);
acedAddObjectContextMenu	Adesk::Boolean acedAddObjectContextMenu(const AcRxClass* pClass, AcEdUIContext* pContext);
acedRemoveDefaultContextMenu	Adesk::Boolean acedRemoveDefaultContextMenu(AcEdUIContext* pContext);
acedRemoveObjectContextMenu	Adesk::Boolean acedRemoveObjectContextMenu(const AcRxClass* pClass, AcEdUIContext* pContext);

New Classes

This is a list of the new classes added to ObjectARX to support the new features in AutoCAD. The classes are listed alphabetically with their header files, and you can use the index to find classes associated with a given feature if the class is described in the *Migration Guide for Applications*. Refer to the *ObjectARX Reference* for complete information on these classes:

New classes

Class Name	Header File
AcApDocManager	*acdocman.h*
AcApDocManagerReactor	*acdocman.h*
AcApDocument	*acdocman.h*
AcApDocumentIterator	*acdocman.h*
AcApLongTransactionManager	*lngtrans.h*
AcApLongTransactionReactor	*lngtrans.h*
AcApProfileManager	*acprofile.h*
AcApProfileManagerReactor	*acprofile.h*
AcArray	*acarray.h*
AcArrayMemCopyReallocator	*acarray.h*
AcArrayObjectCopyReallocator	*acarray.h*
AcDbAppSystemVariables	*acappvar.h*
AcDbBlockTablePointer	*dbobjptr.h*
AcDbBlockTableRecordPointer	*dbobjptr.h*
AcDbCustomOsnapInfo	*dbosnap.h*
AcDbCustomOsnapManager	*dbosnap.h*
AcDbCustomOsnapMode	*dbosnap.h*
AcDbDatabaseSummaryInfo	*summinfo.h*

New classes (*continued*)	
Class Name	**Header File**
AcDbDiametricDimension	*dbdim.h*
AcDbDictionaryWithDefault	*dbdictdflt.h*
AcDbDimStyleTablePointer	*dbobjptr.h*
AcDbDimStyleTableRecordPointer	*dbobjptr.h*
AcDbEntityHyperlinkPE	*achapi.h*
AcDbEntityPointer	*dbobjptr.h*
AcDbHostApplicationProgressMeter	*dbapserv.h*
AcDbHostApplicationServices	*dbapserv.h*
AcDbHyperlink	*achapi.h*
AcDbHyperlinkCollection	*achapi.h*
AcDbLayerTablePointer	*dbobjptr.h*
AcDbLayerTableRecordPointer	*dbobjptr.h*
AcDbLinetypeTablePointer	*dbobjptr.h*
AcDbLinetypeTableRecordPointer	*dbobjptr.h*
AcDbLongTransaction	*dbltrans.h*
AcDbLongTransWorkSetIterator	*dbltrans.h*
AcDbObjectPointerBase	*dbobjptr.h*
AcDbTextStyleTablePointer	*dbobjptr.h*
AcDbTransactionManager	*dbtrans.h*
AcEdInputContextReactor	*acedinpt.h*
AcEdInputPointFilter	*acedinpt.h*
AcEdInputPointManager	*acedinpt.h*
AcEdInputPointMonitor	*acedinpt.h*

New classes (*continued*)	
Class Name	**Header File**
AcEdUIContext	*aced.h*
AcGiCommonDraw	*acgi.h*
AcGiContext	*acgi.h*
AcGiDrawable	*drawable.h*
AcGiDrawableTraits	*acgi.h*
AcGiGeometry	*acgi.h*
AcGiGlyph	*dbosnap.h*
AcGsClassFactory	*gs.h*
AcGsConfig	*gs.h*
AcGsDCPoint	*gs.h*
AcGsDCRect	*gs.h*
AcGsDevice	*gs.h*
AcGsManager	*AcGsManager.h*
AcGsModel	*gs.h*
AcGsReactor	*gs.h*
AcGsView	*gs.h*
AcReCallout	*scene.h*
AcRxEvent	*rxevent.h*
AcRxEventReactor	*rxevent.h*
CAcExtensionModule	*AcExtensionModule.h*
CAcModuleResourceOverride	*AcExtensionModule.h*
CAcUiAngleComboBox	*acuiComboBox.h*
CAcUiAngleEdit	*acuiEdit.h*

New classes (*continued*)	
Class Name	**Header File**
CAcUiArrowHeadComboBox	*acuiComboBox.h*
CAcUiBitmapButton	*acuiButton.h*
CAcUiBitmapStatic	*acuiButton.h*
CAcUiColorComboBox	*acuiComboBox.h*
CAcUiComboBox	*acuiComboBox.h*
CAcUiDialog	*acuiDialog.h*
CAcUiDialogBar	*acuiDialog.h*
CAcUiDropSite	*acuiButton.h*
CAcUiEdit	*acuiEdit.h*
CAcUiFileDialog	*acuiDialog.h*
CAcUiHeaderCtrl	*acuiHeaderCtrl.h*
CAcUiLineWeightComboBox	*acuiComboBox.h*
CAcUiListBox	*acuiListBox.h*
CAcUiListCtrl	*acuiListCtrl.h*
CAcUiMRUComboBox	*acuiComboBox.h*
CAcUiMRUListBox	*acuiComboBox.h*
CAcUiNumericComboBox	*acuiComboBox.h*
CAcUiNumericEdit	*acuiEdit.h*
CAcUiOwnerDrawButton	*acuiButton.h*
CAcUiPickButton	*acuiButton.h*
CAcUiPlotStyleTablesComboBox	*acuiComboBox.h*
CAcUiSelectButton	*acuiButton.h*
CAcUiString	*acuiString.h*

New classes (*continued*)	
Class Name	**Header File**
CAcUiStringComboBox	*acuiComboBox.h*
CAcUiStringEdit	*acuiEdit.h*
CAcUiSymbolComboBox	*acuiComboBox.h*
CAcUiSymbolEdit	*acuiEdit.h*
CAcUiTab	*acuiTabCtrl.h*
CAcUiTabChildDialog	*acuiDialog.h*
CAcUiTabMainDialog	*acuiDialog.h*
CAcUiToolButton	*acuiButton.h*
CAdUiBalloonTip	*aduiTextTip.h*
CAdUiBaseDialog	*aduiBaseDialog.h*
CAdUiBitmapButton	*aduiButton.h*
CAdUiBitmapStatic	*aduiButton.h*
CAdUiComboBox	*aduiComboBox.h*
CAdUiDialog	*aduiDialog.h*
CAdUiDialogBar	*aduiDialogBar.h*
CAdUiDockControlBar	*aduiDock.h*
CAdUiDockFrame	*aduiDock.h*
CAdUiDrawTipText	*aduiMessage.h*
CAdUiDropSite	*aduiButton.h*
CAdUiEdit	*aduiEdit.h*
CAdUiFileDialog	*aduiFileDialog.h*
CAdUiHeaderCtrl	*aduiHeaderCtrl.h*
CAdUiListBox	*aduiListBox.h*

New classes (*continued*)	
Class Name	**Header File**
CAdUiListCtrl	*aduiListCtrl.h*
CAdUiOwnerDrawButton	*aduiButton.h*
CAdUiRegistryDeleteAccess	*aduiRegistryAccess.h*
CAdUiRegistryWriteAccess	*aduiRegistryAccess.h*
CAdUiTab	*aduiTabCtrl.h*
CAdUiTabChildDialog	*aduiTabChildDialog.h*
CAdUiTabExtensionManager	*aduiTabExtension.h*
CAdUiTabMainDialog	*aduiTabMainDialog.h*
CAdUiTextTip	*aduiTextTip.h*
CAdUiTipWindow	*aduiTextTip.h*
CAdUiToolButton	*aduiButton.h*

Changed Items

This section lists the changes that have been made to existing ObjectARX classes, methods, and functions. For more complete information about these changes, please see the *ObjectARX Reference*.

Changed Messages

The `kOleUnloadAppMsg`, formerly used by applications with ActiveX (OLE) Automation, is being deprecated. Applications that use this message are usually also COM servers, and as such should rely on reference counts to determine when to unload. Supporting this message under these circumstances requires the application to hardcode the name of the COM server to be able to identify it, defeating one of the purposes of COM. You should no longer use `kOleUnloadAppMsg` in your application.

Changed Global Functions

AutoCAD 2000 renames several existing global functions in the interest of clarity, though the arguments and return values are unchanged. If you wish to continue to use the earlier names, a set of `#define` directives that map the old names to the new names is provided in the *migrtion.h* header file. You should still replace your old names with the new names as soon as possible, however, because the mapping is only provided for transitioning to AutoCAD 2000 and may not be available in future releases. See "ADS" on page 155 for a list of the renamed ADS functions, and refer to the *migration.h* header file for other names.

Changed Classes

This is a list of modified classes. The classes are listed alphabetically with their header files, and you can use the index to find classes associated with a given feature if the class is described in the *Migration Guide for Applications*. Refer to the *ObjectARX Reference* for complete information on these classes:

Changed classes

Class Name	Header File
AcadAppInfo	*appinfo.h*
AcDb2dPolyline	*dbents.h*
AcDb2dVertex	*dbents.h*
AcDb3dPolyline	*dbents.h*
AcDb3dPolylineVertex	*dbents.h*
AcDb3dSolid	*dbsol3d.h*
AcDbAbstractViewTable	*dbsymtb.h*
AcDbAbstractViewTableIterator	*dbsymtb.h*
AcDbAbstractViewTableRecord	*dbsymtb.h*
AcDbArc	*dbents.h*
AcDbAttribute	*dbents.h*
AcDbBlockEnd	*dbents.h*

Changed classes (*continued*)	
Class Name	**Header File**
AcDbBlockReference	*dbents.h*
AcDbBlockReferenceIdIterator	*dbsymtb.h*
AcDbBlockTable	*dbsymtb.h*
AcDbBlockTableIterator	*dbsymtb.h*
AcDbBlockTableRecord	*dbsymtb.h*
AcDbBlockTableRecordIterator	*dbsymtb.h*
AcDbBody	*dbbody.h*
AcDbCircle	*dbents.h*
AcDbCurve	*dbcurve.h*
AcDbDatabase	*dbmain.h*
AcDbDatabaseReactor	*dbmain.h*
AcDbDate	*dbdate.h*
AcDbDeepCloneFiler	*dbcfilrs.h*
AcDbDictionary	*dbdict.h*
AcDbDictionaryIterator	*dbdict.h*
AcDbDimension	*dbdim.h*
AcDbDimStyleTable	*dbsymtb.h*
AcDbDimStyleTableIterator	*dbsymtb.h*
AcDbDimStyleTableRecord	*dbsymtb.h*
AcDbDwgFiler	*dbfiler.h*
AcDbDxfFiler	*dbfiler.h*
AcDbEllipse	*dbelipse.h*
AcDbEntity	*dbmain.h*

Changed classes (*continued*)	
Class Name	**Header File**
AcDbEntityReactor	*dbmain.h*
AcDbExtents	*dbmain.h*
AcDbFace	*dbents.h*
AcDbFaceRecord	*dbents.h*
AcDbFcf	*dbfcf.h*
AcDbFrame	*dbframe.h*
AcDbGraph	*graph.h*
AcDbGraphNode	*graph.h*
AcDbGraphStack	*graph.h*
AcDbGroup	*dbgroup.h*
AcDbGroupIterator	*dbgroup.h*
AcDbHardOwnershipId	*dbid.h*
AcDbHardPointerId	*dbid.h*
AcDbHatch	*dbhatch.h*
AcDbIdMapping	*dbidmap.h*
AcDbIdMappingIter	*dbidmap.h*
AcDbIdPair	*dbidmap.h*
AcDbImage	*dbimage.h*
AcDbLayerTable	*dbsymtb.h*
AcDbLayerTableIterator	*dbsymtb.h*
AcDbLayerTableRecord	*dbsymtb.h*
AcDbLeader	*dblead.h*
AcDbLine	*dbents.h*

Changed classes (*continued*)	
Class Name	**Header File**
AcDbLinetypeTable	*dbsymtb.h*
AcDbLinetypeTableIterator	*dbsymtb.h*
AcDbLinetypeTableRecord	*dbsymtb.h*
AcDbMatchProperties	*dbmatch.h*
AcDbMInsertBlock	*dbents.h*
AcDbMline	*dbmline.h*
AcDbMlineStyle	*dbmstyle.h*
AcDbMText	*dbmtext.h*
AcDbObject	*dbmain.h*
AcDbObjectId	*dbid.h*
AcDbObjectIterator	*dbmain.h*
AcDbObjectReactor	*dbmain.h*
AcDbOle2Frame	*dbole.h*
AcDbOleFrame	*dbole.h*
AcDbPoint	*dbents.h*
AcDbPolyFaceMesh	*dbents.h*
AcDbPolyFaceMeshVertex	*dbents.h*
AcDbPolygonMesh	*dbents.h*
AcDbPolygonMeshVertex	*dbents.h*
AcDbPolyline	*dbpl.h*
AcDbProxyEntity	*dbproxy.h*
AcDbProxyObject	*dbproxy.h*
AcDbR13ObjectId	*dbidapps.h*

Changed classes (*continued*)	
Class Name	**Header File**
AcDbRasterImage	*imgent.h*
AcDbRasterImageDef	*imgdef.h*
AcDbRasterImageDefFileAccessReactor	*imgdef.h*
AcDbRasterImageDefReactor	*imgdef.h*
AcDbRasterImageDefTransReactor	*imgdef.h*
AcDbRasterVariables	*imgvars.h*
AcDbRay	*dbray.h*
AcDbRecover	*dbaudita.h*
AcDbRecoverCallBack	*dbaudita.h*
AcDbRegAppTable	*dbsymtb.h*
AcDbRegAppTableIterator	*dbsymtb.h*
AcDbRegAppTableRecord	*dbsymtb.h*
AcDbRegion	*dbregion.h*
AcDbSequenceEnd	*dbents.h*
AcDbServices	*dbmain.h*
AcDbShape	*dbents.h*
AcDbSoftOwnershipId	*dbid.h*
AcDbSoftPointerId	*dbid.h*
AcDbSolid	*dbents.h*
AcDbSpline	*dbspline.h*
AcDbSubentId	*dbsubeid.h*
AcDbSymbolTable	*dbsymtb.h*
AcDbSymbolTableIterator	*dbsymtb.h*

Changed classes (*continued*)	
Class Name	**Header File**
AcDbSymbolTableRecord	*dbsymtb.h*
AcDbText	*dbents.h*
AcDbTextStyleTable	*dbsymtb.h*
AcDbTextStyleTableIterator	*dbsymtb.h*
AcDbTextStyleTableRecord	*dbsymtb.h*
AcDbTrace	*dbents.h*
AcDbUCSTable	*dbsymtb.h*
AcDbUCSTableIterator	*dbsymtb.h*
AcDbUCSTableRecord	*dbsymtb.h*
AcDbVertex	*dbents.h*
AcDbViewport	*dbents.h*
AcDbViewportTable	*dbsymtb.h*
AcDbViewportTableIterator	*dbsymtb.h*
AcDbViewportTableRecord	*dbsymtb.h*
AcDbViewTable	*dbsymtb.h*
AcDbViewTableIterator	*dbsymtb.h*
AcDbViewTableRecord	*dbsymtb.h*
AcDbWblockCloneFiler	*dbcfilrs.h*
AcDbXline	*dbxline.h*
AcDbXObject	*dbmain.h*
AcDbXrecord	*dbxrecrd.h*
AcDbXrecordIterator	*dbxrecrd.h*
AcDbXrefGraph	*xgraph.h*

Changed classes (*continued*)	
Class Name	**Header File**
AcDbXrefGraphNode	*xgraph.h*
AcEdCommand	*accmd.h*
AcEdCommandIterator	*accmd.h*
AcEdCommandStack	*accmd.h*
AcEditor	*aced.h*
AcEditorReactor	*aced.h*
AcEdJig	*dbjig.h*
AcEdServices	*aced.h*
AcGeAugPolyline3d	*geapln3d.h*
AcGeBoundBlock2d	*geblok2d.h*
AcGeBoundBlock3d	*geblok3d.h*
AcGeBoundedPlane	*gebndpln.h*
AcGeCircArc2d	*gearc2d.h*
AcGeCircArc3d	*gearc3d.h*
AcGeClipBoundary2d	*geclip2d.h*
AcGeCompositeCurve2d	*gecomp2d.h*
AcGeCompositeCurve3d	*gecomp3d.h*
AcGeCone	*gecone.h*
AcGeCubicSplineCurve2d	*gecspl2d.h*
AcGeCubicSplineCurve3d	*gecspl3d.h*
AcGeCurve2d	*gearc2d.h*
AcGeCurve3d	*gearc3d.h*
AcGeCurveBoundary	*gecbndry.h*

Changed classes (*continued*)	
Class Name	**Header File**
AcGeCurveCurveInt2d	*gecint2d.h*
AcGeCurveCurveInt3d	*gecint3d.h*
AcGeCylinder	*gecylndr.h*
AcGeDwgIO	*gedwgio.h*
AcGeDxfIO	*gedxfio.h*
AcGeEllipArc2d	*geell2d.h*
AcGeEllipArc3d	*geell3d.h*
AcGeEntity2d	*geent2d.h*
AcGeEntity3d	*geent3d.h*
AcGeExternalBoundedSurface	*gexbndsf.h*
AcGeExternalCurve2d	*geextc2d.h*
AcGeExternalCurve3d	*geextc3d.h*
AcGeExternalSurface	*geextsf.h*
AcGeFileIO	*gefileio.h*
AcGeFiler	*gefiler.h*
AcGeInterval	*geintrvl.h*
AcGeKnotVector	*gekvec.h*
AcGeLibVersion	*gelibver.h*
AcGeLine2d	*geline2d.h*
AcGeLine3d	*geline3d.h*
AcGeLinearEnt2d	*gelent2d.h*
AcGeLinearEnt3d	*gelent3d.h*
AcGeLineSeg2d	*gelnsg2d.h*

Changed classes (*continued*)	
Class Name	**Header File**
AcGeLineSeg3d	*gelnsg3d.h*
AcGeMatrix2d	*gemat2d.h*
AcGeMatrix3d	*gemat3d.h*
AcGeNurbCurve2d	*genurb2d.h*
AcGeNurbCurve3d	*genurb3d.h*
AcGeNurbSurface	*genurbsf.h*
AcGeOffsetCurve2d	*geoffc2d.h*
AcGeOffsetCurve3d	*geoffc3d.h*
AcGeOffsetSurface	*geoffsf.h*
AcGePlanarEnt	*geplanar.h*
AcGePlane	*geplane.h*
AcGePointEnt2d	*gepent2d.h*
AcGePointEnt3d	*gepent3d.h*
AcGePointOnCurve2d	*geponc2d.h*
AcGePointOnCurve3d	*geponc3d.h*
AcGePointOnSurface	*geponsrf.h*
AcGePolyline2d	*geplin2d.h*
AcGePolyline3d	*geplin3d.h*
AcGePosition2d	*gepos2d.h*
AcGePosition3d	*gepos3d.h*
AcGeRay2d	*geray2d.h*
AcGeRay3d	*geray3d.h*
AcGeScale2d	*gescl2d.h*

Changed classes (*continued*)	
Class Name	**Header File**
AcGeScale3d	*gescl3d.h*
AcGeSphere	*gesphere.h*
AcGeSplineEnt2d	*gesent2d.h*
AcGeSplineEnt3d	*gesent3d.h*
AcGeSurface	*gesurf.h*
AcGeTol	*getol.h*
AcGeTorus	*getorus.h*
AcGeVector2d	*gevec2d.h*
AcGeVector3d	*gevec3d.h*
AcGiEdgeData	*acgi.h*
AcGiFaceData	*acgi.h*
AcGiSubEntityTraits	*acgi.h*
AcGiTextStyle	*acgi.h*
AcGiVertexData	*acgi.h*
AcGiViewport	*acgi.h*
AcGiViewportDraw	*acgi.h*
AcGiViewportGeometry	*acgi.h*
AcGiWorldDraw	*acgi.h*
AcGiWorldGeometry	*acgi.h*
AcRxClass	*rxclass.h*
AcRxDictionary	*rxdict.h*
AcRxDictionaryIterator	*rxditer.h*
AcRxDLinkerReactor	*rxdlinkr.h*

Changed classes (*continued*)	
Class Name	**Header File**
AcRxDynamicLinker	*rxdlinkr.h*
AcRxIterator	*rxiter.h*
AcRxKernel	*rxkernel.h*
AcRxObject	*rxobject.h*
AcTransaction	*dbtrans.h*
AcTransactionManager	*actrans.h*
AcTransactionReactor	*dbtrans.h*

Classes That Cannot Be Derived From

You cannot derive from the following classes. Doing so will cause your application to halt AutoCAD:

- `AcDb2dPolyline`
- `AcDb3dPolyline`
- `AcDbPolygonMesh`
- `AcDbPolyFaceMesh`
- `AcDbSequenceEnd`
- `AcDbBlockBegin`
- `AcDbBlockEnd`
- `AcDbVertex`
- `AcDbFaceRecord`
- `AcDb2dVertex`
- `AcDb3dPolylineVertex`
- `AcDbPolygonMeshVertex`
- `AcDbPolyFaceMeshVertex`
- `AcDbMInsertBlock`

Deprecated Items

The following items are still provided in AutoCAD 2000 but will probably be removed in the following release of AutoCAD. Instead, use the suggested replacement, if applicable.

The `AcDbDatabase` methods listed here are being deprecated because the data used by these methods is now stored in the system registry instead of the drawing header:

- `osmode()`
- `setOsmode()`
- `pickstyle()`
- `setPickstyle()`
- `coords()`
- `setCoords()`
- `attdia()`
- `setAttdia()`

Removed Items

The following items are no longer provided in AutoCAD 2000. Instead, use the suggested replacement:

Removed global functions

Function	Replacement
acdbCurDwg()	acdbHostApplicationServices()-> workingDatabase()
acedSetTilemode()	AcDbDatabase::setTilemode()
freeAcDbBuffer()	delBuffer()
freeAcdbString()	delString()
newAcDbBuffer()	newBuffer()
newAcDbString()	newString()
updAcDbString()	updString()

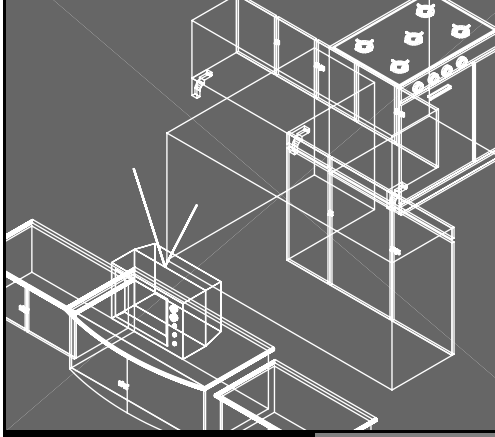
The following methods have been removed from `AcRxEventReactor`:

```
virtual void
sysVarChanged(
    const char* varName,
    Adesk::Boolean success);

virtual void
sysVarWillChange(
    const char* varName);
```

Use the new `AcDbDatabaseReactor` methods `headerSysVarWillChange()` and `headerSysVarChanged()` instead. See chapter 10, "AcDbDatabaseReactor Class" section, for more information.

ActiveX Changes



In This Appendix

B

- New Items
- Changed Items
- Removed Items

This appendix lists the new objects, methods, properties, and events for the ActiveX Automation interface. It also lists the methods and properties that have changed, and those that have been removed from the system.

New Items

The following section lists all ActiveX objects along with their new methods, events, and properties.

This section lists all previously existing, new, and hidden objects in ActiveX. (Hidden objects are exposed as COM interfaces but not as VBA objects. The methods and properties on hidden objects are available to VBA users through other VBA objects.)

In the following table, new objects are identified with an asterisk (*) and hidden objects are identified with a plus sign (+):

New methods, properties, and events since AutoCAD Release 14.01

Object	Method / Property / Event Name
3dFace	Coordinate, Coordinates, Delete, Document, GetExtensionDictionary, HasExtensionDictionary, Hyperlinks, Lineweight, Modified, ObjectName, OwnerID, PlotStyleName, VisibilityEdge1, VisibilityEdge2, VisibilityEdge3, VisibilityEdge4
3dPolyline	Coordinate, Delete, Document, GetExtensionDictionary, HasExtensionDictionary, Hyperlinks, Lineweight, Modified, ObjectName, OwnerID, PlotStyleName, Type
3dSolid	Delete, Document, GetExtensionDictionary, HasExtensionDictionary, Hyperlinks, Lineweight, Modified, ObjectName, OwnerID, PlotStyleName
Application	AppActivate, AppDeactivate, ARXLoaded, ARXUnloaded, BeginCommand, BeginFileDrop, BeginLisp, BeginModal, BeginOpen, BeginPlot, BeginQuit, BeginSave, Documents, EndCommand, EndLisp, EndModal, EndOpen, EndPlot, EndSave, Eval, LispCancelled, LoadDVB, MenuBar, MenuGroups, NewDrawing, RunMacro, StatusId, SysVarChanged, UnloadDVB, VBE, WindowChanged, WindowLeft, WindowMovedOrResized, WindowState, WindowTop, ZoomAll, ZoomCenter, ZoomExtents, ZoomPickWindow, ZoomScaled, ZoomWindow
Arc	ArcLength, Delete, Document, GetExtensionDictionary, HasExtensionDictionary, Hyperlinks, Lineweight, Modified, ObjectName, OwnerID, PlotStyleName, TotalAngle
Attribute	Alignment, Backward, Constant, Delete, Document, GetExtensionDictionary, HasExtensionDictionary, Hyperlinks, Invisible, Lineweight, Modified, ObjectName, OwnerID, PlotStyleName, Preset, UpsideDown, Verify

New methods, properties, and events since AutoCAD Release 14.01

Object	Method / Property / Event Name
AttributeReference	Alignment, ArrayPolar, ArrayRectangular, Backward, Constant, Copy, Delete, Document, GetExtensionDictionary, HasExtensionDictionary, Hyperlinks, Invisible, Lineweight, Mirror, Mirror3D, Modified, ObjectName, OwnerID, PlotStyleName, UpsideDown
Block	AddDim3PointAngular, AddMInsertBlock, AddMLine, AddPolyfaceMesh, AttachExternalReference, Bind, Detach, Document, GetExtensionDictionary, GetXData, HasExtensionDictionary, IsLayout, IsXRef, Layout, Modified, ObjectID, ObjectName, OwnerID, Reload, SetXData, Unload, XRefDatabase
BlockReference	Delete, Document, GetConstantAttributes, GetExtensionDictionary, HasExtensionDictionary, Hyperlinks, Lineweight, Modified, ObjectName, OwnerID, PlotStyleName
Blocks	Delete, Document, GetExtensionDictionary, GetXData, Handle, HasExtensionDictionary, Modified, ObjectID, ObjectName, OwnerID, SetXData
Circle	Circumference, Delete, Diameter, Document, GetExtensionDictionary, HasExtensionDictionary, Hyperlinks, Lineweight, Modified, ObjectName, OwnerID, PlotStyleName
Database*	Blocks, CopyObjects, Dictionaries, DimStyles, ElevationModelspace, ElevationPaperspace, Groups, HandleToObject, Layers, Layouts, Limits, Linetypes , ModelSpace, ObjectIdToObject, PaperSpace, PlotConfigurations, Preferences, RegisteredApplications, TextStyles, UserCoordinateSystems, Viewports, Views
DatabasePreferences*	AllowLongSymbolNames, Application, ContourLinesPerSurface, DisplaySilhouette, Lineweight, LineWeightDisplay, MaxActiveViewports, ObjectSortByPlotting, ObjectSortByPSOutput, ObjectSortByRedraws, ObjectSortByRegens, ObjectSortBySelection, ObjectSortBySnap, OLELaunch, PickGroup, RenderSmoothness, SegmentPerPolyline, SolidFill, TextFrameDisplay, XRefEdit, XRefLayerVisibility
Dictionaries	Delete, Document, GetExtensionDictionary, GetXData, Handle, HasExtensionDictionary, Modified, ObjectID, ObjectName, OwnerID, SetXData
Dictionary	AddXRecord, Count, Document, GetExtensionDictionary, HasExtensionDictionary, Item, Modified, ObjectID, ObjectName, OwnerID

New methods, properties, and events since AutoCAD Release 14.01	
Object	**Method / Property / Event Name**
Dim3PointAngular*	AngleFormat, AngleVertex, Application, ArrayPolar, ArrayRectangular, Arrowhead1Block, Arrowhead1Type, Arrowhead2Block, Arrowhead2Type, ArrowheadSize, Color, Copy, DecimalSeparator, Delete, DimensionLineColor, DimensionLineWeight, DimLine1Suppress, DimLine2Suppress, DimLineInside, Document, ExtensionLineColor, ExtensionLineExtend, ExtensionLineOffset, ExtensionLineWeight, ExtLine1EndPoint, ExtLine1Suppress, ExtLine2EndPoint, ExtLine2Suppress, Fit, ForceLineInside, GetBoundingBox, GetExtensionDictionary, GetXdata, Handle, HasExtensionDictionary, Highlight, HorizontalTextPosition, Hyperlinks, IntersectWith, Layer, Linetype, LinetypeScale, Lineweight, Measurement, Mirror, Mirror3D, Modified, Move, Normal, ObjectID, ObjectName, OwnerID, PlotStyleName, Rotate, Rotate3D, Rotation, ScaleEntity, ScaleFactor, SetXdata, StyleName, SuppressLeadingZeros, SuppressTrailingZeros, TextColor, TextGap, TextHeight, TextInside, TextInsideAlign, TextMovement, TextOutsideAlign, TextOverride, TextPosition, TextPrecision, TextPrefix, TextRotation, TextStyle, TextSuffix, ToleranceDisplay, ToleranceHeightScale, ToleranceJustification, ToleranceLowerLimit, TolerancePrecision, ToleranceSuppressLeadingZeros, ToleranceSuppressTrailingZeros, ToleranceUpperLimit, TransformBy, Update, VerticalTextPosition, Visible
DimAligned	AltRoundDistance, AltSuppressLeadingZeros, AltSuppressTrailingZeros, AltSuppressZeroFeet, AltSuppressZeroInches, AltTextPrefix, AltTextSuffix, AltTolerancePrecision, AltToleranceSuppressLeadingZeros, AltToleranceSuppressTrailingZeros, AltToleranceSuppressZeroFeet, AltToleranceSuppressZeroInches, AltUnits, AltUnitsFormat, AltUnitsPrecision, AltUnitsScale, Arrowhead1Block, Arrowhead1Type, Arrowhead2Block, Arrowhead2Type, ArrowheadSize, DecimalSeparator, Delete, DimensionLineColor, DimensionLineExtend, DimensionLineWeight, DimLine1Suppress, DimLine2Suppress, DimLineInside, Document, ExtensionLineColor, ExtensionLineExtend, ExtensionLineOffset, ExtensionLineWeight, ExtLine1Suppress, ExtLine2Suppress, Fit, ForceLineInside, FractionFormat, GetExtensionDictionary, HasExtensionDictionary, HorizontalTextPosition, Hyperlinks, LinearScaleFactor, Lineweight, Measurement, Modified, ObjectName, OwnerID, PlotStyleName, PrimaryUnitsPrecision, RoundDistance, ScaleFactor, SuppressLeadingZeros, SuppressTrailingZeros, SuppressZeroFeet, SuppressZeroInches, TextColor, TextGap, TextHeight, TextInside, TextInsideAlign, TextMovement, TextOutsideAlign, TextOverride, TextPrefix, TextStyle, TextSuffix, ToleranceDisplay, ToleranceHeightScale, ToleranceJustification, ToleranceLowerLimit, TolerancePrecision, ToleranceSuppressLeadingZeros, ToleranceSuppressTrailingZeros, ToleranceSuppressZeroFeet, ToleranceSuppressZeroInches, ToleranceUpperLimit, UnitsFormat, VerticalTextPosition

New methods, properties, and events since AutoCAD Release 14.01	
Object	**Method / Property / Event Name**
DimAngular	AngleFormat, Arrowhead1Block, Arrowhead1Type, Arrowhead2Block, Arrowhead2Type, ArrowheadSize, DecimalSeparator, Delete, DimensionLineColor, DimensionLineWeight, DimLine1Suppress, DimLine2Suppress, DimLineInside, Document, ExtensionLineColor, ExtensionLineExtend, ExtensionLineOffset, ExtensionLineWeight, ExtLine1Suppress, ExtLine2Suppress, Fit, ForceLineInside, GetExtensionDictionary, HasExtensionDictionary, HorizontalTextPosition, Hyperlinks, Lineweight, Measurement, Modified, ObjectName, OwnerID, PlotStyleName, ScaleFactor, SuppressLeadingZeros, SuppressTrailingZeros, TextColor, TextGap, TextHeight, TextInside, TextInsideAlign, TextMovement, TextOutsideAlign, TextOverride, TextPrecision, TextPrefix, TextStyle, TextSuffix, ToleranceDisplay, ToleranceHeightScale, ToleranceJustification, ToleranceLowerLimit, TolerancePrecision, ToleranceSuppressLeadingZeros, ToleranceSuppressTrailingZeros, ToleranceUpperLimit, VerticalTextPosition
DimDiametric	AltRoundDistance, AltSuppressLeadingZeros, AltSuppressTrailingZeros, AltSuppressZeroFeet, AltSuppressZeroInches, AltTextPrefix, AltTextSuffix, AltTolerancePrecision, AltToleranceSuppressLeadingZeros, AltToleranceSuppressTrailingZeros, AltToleranceSuppressZeroFeet, AltToleranceSuppressZeroInches, AltUnits, AltUnitsFormat, AltUnitsPrecision, AltUnitsScale, Arrowhead1Block, Arrowhead1Type, Arrowhead2Block, Arrowhead2Type, ArrowheadSize, CenterMarkSize, CenterType, DecimalSeparator, Delete, DimensionLineColor, DimensionLineWeight, DimLine1Suppress, DimLine2Suppress, Document, Fit, ForceLineInside, FractionFormat, GetExtensionDictionary, HasExtensionDictionary, Hyperlinks, LinearScaleFactor, Lineweight, Measurement, Modified, ObjectName, OwnerID, PlotStyleName, PrimaryUnitsPrecision, RoundDistance, ScaleFactor, SuppressLeadingZeros, SuppressTrailingZeros, SuppressZeroFeet, SuppressZeroInches, TextColor, TextGap, TextHeight, TextInside, TextInsideAlign, TextMovement, TextOutsideAlign, TextOverride, TextPrefix, TextStyle, TextSuffix, ToleranceDisplay, ToleranceHeightScale, ToleranceJustification, ToleranceLowerLimit, TolerancePrecision, ToleranceSuppressLeadingZeros, ToleranceSuppressTrailingZeros, ToleranceSuppressZeroFeet, ToleranceSuppressZeroInches, ToleranceUpperLimit, UnitsFormat, VerticalTextPosition

New methods, properties, and events since AutoCAD Release 14.01	
Object	**Method / Property / Event Name**
Dimension+	Application, ArrayPolar, ArrayRectangular, Color, Copy, DecimalSeparator, Delete, Document, GetBoundingBox, GetExtensionDictionary, GetXdata, Handle, HasExtensionDictionary, Highlight, Hyperlinks, IntersectWith, Layer, Linetype, LinetypeScale, Lineweight, Mirror, Mirror3D, Modified, Move, Normal, ObjectID, ObjectName, OwnerID, PlotStyleName, Rotate, Rotate3D, Rotation, ScaleEntity, ScaleFactor, SetXdata, StyleName, SuppressLeadingZeros, SuppressTrailingZeros, TextColor, TextGap, TextHeight, TextMovement, TextOverride, TextPosition, TextPrefix, TextRotation, TextStyle, TextSuffix, ToleranceDisplay, ToleranceHeightScale, ToleranceJustification, ToleranceLowerLimit, TolerancePrecision, ToleranceSuppressLeadingZeros, ToleranceSuppressTrailingZeros, ToleranceUpperLimit, TransformBy, Update, VerticalTextPosition, Visible
DimOrdinate	AltRoundDistance, AltSuppressLeadingZeros, AltSuppressTrailingZeros, AltSuppressZeroFeet, AltSuppressZeroInches, AltTextPrefix, AltTextSuffix, AltTolerancePrecision, AltToleranceSuppressLeadingZeros, AltToleranceSuppressTrailingZeros, AltToleranceSuppressZeroFeet, AltToleranceSuppressZeroInches, AltUnits, AltUnitsFormat, AltUnitsPrecision, AltUnitsScale, ArrowheadSize, DecimalSeparator, Delete, Document, ExtensionLineColor, ExtensionLineOffset, ExtensionLineWeight, FractionFormat, GetExtensionDictionary, HasExtensionDictionary, Hyperlinks, LinearScaleFactor, Lineweight, Measurement, Modified, ObjectName, OwnerID, PlotStyleName, PrimaryUnitsPrecision, RoundDistance, ScaleFactor, SuppressLeadingZeros, SuppressTrailingZeros, SuppressZeroFeet, SuppressZeroInches, TextColor, TextGap, TextHeight, TextMovement, TextOverride, TextPrefix, TextStyle, TextSuffix, ToleranceDisplay, ToleranceHeightScale, ToleranceJustification, ToleranceLowerLimit, TolerancePrecision, ToleranceSuppressLeadingZeros, ToleranceSuppressTrailingZeros, ToleranceSuppressZeroFeet, ToleranceSuppressZeroInches, ToleranceUpperLimit, UnitsFormat, VerticalTextPosition

New methods, properties, and events since AutoCAD Release 14.01

Object	Method / Property / Event Name
DimRadial	AltRoundDistance, AltSuppressLeadingZeros, AltSuppressTrailingZeros, AltSuppressZeroFeet, AltSuppressZeroInches, AltTextPrefix, AltTextSuffix, AltTolerancePrecision, AltToleranceSuppressLeadingZeros, AltToleranceSuppressTrailingZeros, AltToleranceSuppressZeroFeet, AltToleranceSuppressZeroInches, AltUnits, AltUnitsFormat, AltUnitsPrecision, AltUnitsScale, ArrowheadBlock, ArrowheadSize, ArrowheadType, CenterMarkSize, CenterType, DecimalSeparator, Delete, DimensionLineColor, DimensionLineWeight, DimLineSuppress, Document, Fit, ForceLineInside, FractionFormat, GetExtensionDictionary, HasExtensionDictionary, Hyperlinks, LinearScaleFactor, Lineweight, Measurement, Modified, ObjectName, OwnerID, PlotStyleName, PrimaryUnitsPrecision, RoundDistance, ScaleFactor, SuppressLeadingZeros, SuppressTrailingZeros, SuppressZeroFeet, SuppressZeroInches, TextColor, TextGap, TextHeight, TextInside, TextInsideAlign, TextMovement, TextOutsideAlign, TextOverride, TextPrefix, TextStyle, TextSuffix, ToleranceDisplay, ToleranceHeightScale, ToleranceJustification, ToleranceLowerLimit, TolerancePrecision, ToleranceSuppressLeadingZeros, ToleranceSuppressTrailingZeros, ToleranceSuppressZeroFeet, ToleranceSuppressZeroInches, ToleranceUpperLimit, UnitsFormat, VerticalTextPosition

New methods, properties, and events since AutoCAD Release 14.01	
Object	**Method / Property / Event Name**
DimRotated	AltRoundDistance, AltSuppressLeadingZeros, AltSuppressTrailingZeros, AltSuppressZeroFeet, AltSuppressZeroInches, AltTextPrefix, AltTextSuffix, AltTolerancePrecision, AltToleranceSuppressLeadingZeros, AltToleranceSuppressTrailingZeros, AltToleranceSuppressZeroFeet, AltToleranceSuppressZeroInches, AltUnits, AltUnitsFormat, AltUnitsPrecision, AltUnitsScale, Arrowhead1Block, Arrowhead1Type, Arrowhead2Block, Arrowhead2Type, ArrowheadSize, DecimalSeparator, Delete, DimensionLineColor, DimensionLineExtend, DimensionLineWeight, DimLine1Suppress, DimLine2Suppress, DimLineInside, Document, ExtensionLineColor, ExtensionLineExtend, ExtensionLineOffset, ExtensionLineWeight, ExtLine1Suppress, ExtLine2Suppress, Fit, ForceLineInside, FractionFormat, GetExtensionDictionary, HasExtensionDictionary, HorizontalTextPosition, Hyperlinks, LinearScaleFactor, Lineweight, Measurement, Modified, ObjectName, OwnerID, PlotStyleName, PrimaryUnitsPrecision, RoundDistance, ScaleFactor, SuppressLeadingZeros, SuppressTrailingZeros, SuppressZeroFeet, SuppressZeroInches, TextColor, TextGap, TextHeight, TextInside, TextInsideAlign, TextMovement, TextOutsideAlign, TextOverride, TextPrefix, TextStyle, TextSuffix, ToleranceDisplay, ToleranceHeightScale, ToleranceJustification, ToleranceLowerLimit, TolerancePrecision, ToleranceSuppressLeadingZeros, ToleranceSuppressTrailingZeros, ToleranceSuppressZeroFeet, ToleranceSuppressZeroInches, ToleranceUpperLimit, UnitsFormat, VerticalTextPosition
DimStyle	Document, GetExtensionDictionary, HasExtensionDictionary, Modified, ObjectID, ObjectName, OwnerID
DimStyles	Delete, Document, GetExtensionDictionary, GetXData, Handle, HasExtensionDictionary, Modified, ObjectID, ObjectName, OwnerID, SetXData
Document	Activate, Active, ActiveLayout, BeginClose, BeginDoubleClick, BeginLisp, BeginPlot, BeginRightClick, BeginShortcutMenuCommand, BeginShortcutMenuDefault, BeginShortcutMenuEdit, BeginShortcutMenuGrip, BeginShortcutMenuOsnap, Close, CopyObjects, Database, Deactivate, EndLisp, EndPlot, EndShortcutMenu, Height, HWND, Layouts, LayoutSwitched, LispCancelled, ObjectAdded, ObjectErased, ObjectModified, PickfirstSelectionSet, PlotConfigurations, Preferences, SelectionChanged, SendCommand, Width, WindowChanged, WindowMovedOrResized, WindowState, WindowTitle
Documents*	Add, Application, Close, Count, Item, Open

New methods, properties, and events since AutoCAD Release 14.01	
Object	**Method / Property / Event Name**
Ellipse	Delete, Document, GetExtensionDictionary, HasExtensionDictionary, Hyperlinks, Lineweight, MajorRadius, MinorRadius, Modified, ObjectName, OwnerID, PlotStyleName
Entity+	Delete, Document, GetExtensionDictionary, HasExtensionDictionary, Hyperlinks, Lineweight, Modified, ObjectName, OwnerID, PlotStyleName
ExternalReference*	Application, ArrayPolar, ArrayRectangular, Color, Copy, Delete, Document, Explode, GetAttributes, GetBoundingBox, GetConstantAttributes, GetExtensionDictionary, GetXdata, Handle, HasAttributes, HasExtensionDictionary, Highlight, Hyperlinks, InsertionPoint, IntersectWith, Layer, Linetype, LinetypeScale, Lineweight, Mirror, Mirror3D, Modified, Move, Name, Normal, ObjectID, ObjectName, OwnerID, Path, PlotStyleName, Rotate, Rotate3D, Rotation, ScaleEntity, SetXdata, TransformBy, Update, Visible, XScaleFactor, YScaleFactor, ZScaleFactor
Group	Document, GetExtensionDictionary, HasExtensionDictionary, Lineweight, Modified, ObjectName, OwnerID, PlotStyleName
Groups	Delete, Document, GetExtensionDictionary, GetXData, Handle, HasExtensionDictionary, Modified, ObjectID, ObjectName, OwnerID, SetXData
Hatch	Delete, Document, GetExtensionDictionary, HasExtensionDictionary, Hyperlinks, ISOPenWidth, Lineweight, Modified, ObjectName, OwnerID, PlotStyleName
Hyperlink*	Application, Delete, URL, URLDescription, URLNamedLocation
Hyperlinks*	Add, Application, Count, Item
IDPair*	Application, IsCloned, IsOwnerXlated, IsPrimary, Key, Value
Layer	Document, GetExtensionDictionary, HasExtensionDictionary, Modified, ObjectID, ObjectName, OwnerID, Plottable, ViewportDefault
Layers	Delete, Document, GetExtensionDictionary, GetXData, Handle, HasExtensionDictionary, Modified, ObjectID, ObjectName, OwnerID, SetXData

New methods, properties, and events since AutoCAD Release 14.01	
Object	**Method / Property / Event Name**
Layout*	Application, Block, CanonicalMediaName, CenterPlot, ConfigName, CopyFrom, Delete, Document, GetCustomScale, GetExtensionDictionary, GetPaperMargins, GetPaperSize, GetWindowToPlot, GetXdata, Handle, HasExtensionDictionary, ModelType, Modified, Name, ObjectID, ObjectName, OwnerID, PaperUnits, PlotHidden, PlotOrigin, PlotRotation, PlotType, PlotViewportBorders, PlotViewportsFirst, PlotWithLineweights, PlotWithPlotStyles, ScaleLineweights, SetCustomScale, SetWindowToPlot, SetXdata, ShowPlotStyles, StandardScale, StyleSheet, TabOrder, UseStandardScale, ViewToPlot
Layouts*	Add, Application, Count, Delete, Document, GetExtensionDictionary, GetXdata, Handle, HasExtensionDictionary, Item, Modified, ObjectID, ObjectName, OwnerID, SetXdata
Leader	Annotation, ArrowheadBlock, ArrowheadSize, ArrowheadType, Coordinate, Delete, DimensionLineColor, DimensionLineWeight, Document, GetExtensionDictionary, HasExtensionDictionary, Hyperlinks, Lineweight, Modified, ObjectName, OwnerID, PlotStyleName, ScaleFactor, TextGap, VerticalTextPosition
Line	Angle, Delete, Delta, Document, GetExtensionDictionary, HasExtensionDictionary, Hyperlinks, Length, Lineweight, Modified, ObjectName, OwnerID, PlotStyleName
Linetype	Document, GetExtensionDictionary, HasExtensionDictionary, Modified, ObjectID, ObjectName, OwnerID
Linetypes	Delete, Document, GetExtensionDictionary, GetXData, Handle, HasExtensionDictionary, Modified, ObjectID, ObjectName, OwnerID, SetXData
LWPolyline	ConstantWidth, Coordinate, Delete, Document, Elevation, GetExtensionDictionary, HasExtensionDictionary, Hyperlinks, LinetypeGeneration, Lineweight, Modified, ObjectName, OwnerID, PlotStyleName
MenuBar*	Application, Count, Item, Parent
MenuGroup*	Application, MenuFileName, Menus, Name, Parent, Save, SaveAs, Toolbars, Type, Unload
MenuGroups*	Application, Count, Item, Load, Parent

New methods, properties, and events since AutoCAD Release 14.01	
Object	**Method / Property / Event Name**
MInsertBlock*	Application, ArrayPolar, ArrayRectangular, Color, Columns, ColumnSpacing, Copy, Delete, Document, Explode, GetAttributes, GetBoundingBox, GetConstantAttributes, GetExtensionDictionary, GetXdata, Handle, HasAttributes, HasExtensionDictionary, Highlight, Hyperlinks, InsertionPoint, IntersectWith, Layer, Linetype, LinetypeScale, Lineweight, Mirror, Mirror3D, Modified, Move, Name, Normal, ObjectID, ObjectName, OwnerID, PlotStyleName, Rotate, Rotate3D, Rotation, Rows, RowSpacing, ScaleEntity, SetXdata, TransformBy, Update, Visible, XScaleFactor, YScaleFactor, ZScaleFactor
MLine*	Application, ArrayPolar, ArrayRectangular, Color, Coordinates, Copy, Delete, Document, GetBoundingBox, GetExtensionDictionary, GetXdata, Handle, HasExtensionDictionary, Highlight, Hyperlinks, IntersectWith, Layer, Linetype, LinetypeScale, Lineweight, Mirror, Mirror3D, Modified, Move, ObjectID, ObjectName, OwnerID, PlotStyleName, Rotate, Rotate3D, ScaleEntity, SetXdata, StyleName, TransformBy, Update, Visible
ModelSpace	AddDim3PointAngular, AddMInsertBlock, AddMLine, AddPolyfaceMesh, AttachExternalReference, Bind, Delete, Detach, Document, GetExtensionDictionary, GetXData, HasExtensionDictionary, IsLayout, IsXRef, Layout, Modified, ObjectID, ObjectName, Origin, OwnerID, Reload, SetXData, Unload, XRefDatabase
Mtext	Delete, Document, GetExtensionDictionary, HasExtensionDictionary, Hyperlinks, LineSpacingFactor, LineSpacingStyle, Lineweight, Modified, ObjectName, OwnerID, PlotStyleName
Object+	Delete, Document, GetExtensionDictionary, HasExtensionDictionary, Modified, ObjectName, OwnerID
PaperSpace	AddDim3PointAngular, AddMInsertBlock, AddMLine, AddPolyfaceMesh, AttachExternalReference, Bind, Delete, Detach, Document, GetExtensionDictionary, GetXData, HasExtensionDictionary, IsLayout, IsXRef, Layout, Modified, ObjectID, ObjectName, Origin, OwnerID, Reload, SetXData, Unload, XRefDatabase
Plot	BatchPlotProgress, DisplayPlotPreview, NumberOfCopies, QuietErrorMode, SetLayoutsToPlot, StartBatchMode

New methods, properties, and events since AutoCAD Release 14.01	
Object	**Method / Property / Event Name**
PlotConfiguration*	Application, CanonicalMediaName, CenterPlot, ConfigName, CopyFrom, Delete, Document, GetCustomScale, GetExtensionDictionary, GetPaperMargins, GetPaperSize, GetWindowToPlot, GetXdata, Handle, HasExtensionDictionary, ModelType, Modified, Name, ObjectID, ObjectName, OwnerID, PaperUnits, PlotHidden, PlotOrigin, PlotRotation, PlotType, PlotViewportBorders, PlotViewportsFirst, PlotWithLineweights, PlotWithPlotStyles, ScaleLineweights, SetCustomScale, SetWindowToPlot, SetXdata, ShowPlotStyles, StandardScale, StyleSheet, UseStandardScale, ViewToPlot
PlotConfigurations*	Add, Application, Count, Delete, Document, GetExtensionDictionary, GetXdata, Handle, HasExtensionDictionary, Item, Modified, ObjectID, ObjectName, OwnerID, SetXdata
Point	Delete, Document, GetExtensionDictionary, HasExtensionDictionary, Hyperlinks, Lineweight, Modified, ObjectName, OwnerID, PlotStyleName
PolyfaceMesh*	Application, ArrayPolar, ArrayRectangular, Color, Coordinate, Coordinates, Copy, Delete, Document, GetBoundingBox, GetExtensionDictionary, GetXdata, Handle, HasExtensionDictionary, Highlight, Hyperlinks, IntersectWith, Layer, Linetype, LinetypeScale, Lineweight, Mirror, Mirror3D, Modified, Move, NumberOfFaces, NumberOfVertices, ObjectID, ObjectName, OwnerID, PlotStyleName, Rotate, Rotate3D, ScaleEntity, SetXdata, TransformBy, Update, Visible
Polygonmesh	Coordinate, Delete, Document, GetExtensionDictionary, HasExtensionDictionary, Hyperlinks, Lineweight, Modified, ObjectName, OwnerID, PlotStyleName
Polyline	ConstantWidth, Coordinate, Delete, Document, Elevation, GetExtensionDictionary, HasExtensionDictionary, Hyperlinks, LinetypeGeneration, Lineweight, Modified, ObjectName, OwnerID, PlotStyleName
PopupMenu*	AddMenuItem, AddSeparator, AddSubMenu, Application, Count, InsertInMenuBar, Item, Name, NameNoMnemonic, OnMenuBar, Parent, RemoveFromMenuBar, ShortcutMenu, TagString
PopupMenuItem*	Application, Caption, Check, Delete, Enable, HelpString, Index, Label, Macro, Parent, SubMenu, TagString, Type
PopupMenus*	Add, Application, Count, InsertMenuInMenuBar, Item, Parent, RemoveMenuFromMenuBar

New methods, properties, and events since AutoCAD Release 14.01

Object	Method / Property / Event Name
Preferences	Display, Drafting, Files, OpenSave, Output, Profiles, Selection, System, User
PreferencesDisplay*	Application, AutoTrackingVecColor, CursorSize, DisplayLayoutTabs, DisplayScreenMenu, DisplayScrollBars, DockedVisibleLines, GraphicsWinLayoutBackgrndColor, GraphicsWinModelBackgrndColor, HistoryLines, ImageFrameHighlight, LayoutCreateViewport, LayoutCrosshairColor, LayoutDisplayMargins, LayoutDisplayPaper, LayoutDisplayPaperShadow, LayoutShowPlotSetup, MaxAutoCADWindow, ModelCrosshairColor, ShowRasterImage, TextFont, TextFontSize, TextFontStyle, TextWinBackgrndColor, TextWinTextColor, TrueColorImages, XRefFadeIntensity
PreferencesDrafting*	AlignmentPointAcquisition, Application, AutoSnapAperture, AutoSnapApertureSize, AutoSnapMagnet, AutoSnapMarker, AutoSnapMarkerColor, AutoSnapMarkerSize, AutoSnapTooltip, AutoTrackTooltip, FullScreenTrackingVector, PolarTrackingVector
PreferencesFiles*	AltFontFile, AltTabletMenuFile, Application, AutoSavePath, ConfigFile, CustomDictionary, DefaultInternetURL, DriversPath, FontFileMap, GetProjectFilePath, HelpFilePath, LicenseServer, LogFilePath, MainDictionary, MenuFile, ObjectARXPath, PostScriptPrologFile, PrinterConfigPath, PrinterDescPath, PrinterStyleSheetPath, PrintFile, PrintSpoolerPath, PrintSpoolExecutable, SetProjectFilePath, SupportPath, TempFIlePath, TemplateDwgPath, TempXrefPath, TextEditor, TextureMapPath, WorkspacePath
PreferencesOpenSave*	Application, AutoAudit, AutoSaveInterval, CreateBackup, DemandLoadArxApp, FullCrcValidation, IncrementalSavePercent, LogFileOn, MRUNumber, ProxyImage, SaveAsType, SavePreviewThumbnail, ShowProxyDialogBox, TempFileExtension, XrefDemandLoad
PreferencesOutput*	Application, DefaultOutputDevice, DefaultPlotStyleForLayer, DefaultPlotStyleForObjects, NewStyleSheet, OLEQuality, PlotLegacy, PlotPolicy, PrinterPaperSizeAlert, PrinterSpoolAlert, UseLastPlotSettings
PreferencesProfiles*	ActiveProfile, Application, CopyProfile, DeleteProfile, ExportProfile, ImportProfile, RenameProfile, ResetProfile
PreferencesSelection*	Application, DisplayGrips, DisplayGripsWithinBlocks, GripColorSelected, GripColorUnselected, GripSize, PickAdd, PickAuto, PickBoxSize, PickDrag, PickFirst, PickGroup

New methods, properties, and events since AutoCAD Release 14.01	
Object	**Method / Property / Event Name**
PreferencesSystem*	AcadLspInDoc, Application, BeepOnError, DisplayOLEScale, EnableStartupDialog, ShowWarningMessages, SingleDocumentMode, StoreSQLIndex, TablesReadOnly
PreferencesUser*	ADCInsertUnitsDefaultSource, ADCInsertUnitsDefaultTarget, Application, HyperlinkDisplayCursor, HyperlinkDisplayTooltip, KeyboardAccelerator, KeyboardPriority, SCMCommandMode, SCMDefaultMode, SCMEditMode, ShortCutMenuInDrawingArea
Pviewport	ArcSmoothness, Clipped, CustomScale, Delete, DisplayLocked, Document, GetExtensionDictionary, HasExtensionDictionary, Hyperlinks, Lineweight, Modified, ObjectName, OwnerID, PlotStyleName, StandardScale, StyleSheet, UCSPerViewport, ViewportOn
RasterImage	Delete, Document, GetExtensionDictionary, HasExtensionDictionary, Hyperlinks, ImageHeight, ImageWidth, Lineweight, Modified, Name, ObjectName, OrthoEnabled, OwnerID, PlotStyleName, Rotation, ScaleFactor
Ray	Delete, Document, GetExtensionDictionary, HasExtensionDictionary, Hyperlinks, Lineweight, Modified, ObjectName, OwnerID, PlotStyleName, SecondPoint
Region	Delete, Document, GetExtensionDictionary, HasExtensionDictionary, Hyperlinks, Lineweight, Modified, ObjectName, OwnerID, PlotStyleName
RegisteredApplication	Document, GetExtensionDictionary, HasExtensionDictionary, Modified, ObjectID, ObjectName, OwnerID
RegisteredApplications	Delete, Document, GetExtensionDictionary, GetXData, Handle, HasExtensionDictionary, Modified, ObjectID, ObjectName, OwnerID, SetXData
Shape	Delete, Document, GetExtensionDictionary, HasExtensionDictionary, Hyperlinks, Lineweight, Modified, ObjectName, OwnerID, PlotStyleName
Solid	Coordinate, Delete, Document, GetExtensionDictionary, HasExtensionDictionary, Hyperlinks, Lineweight, Modified, ObjectName, OwnerID, PlotStyleName
Spline	ControlPoints, Delete, Document, FitPoints, GetExtensionDictionary, HasExtensionDictionary, Hyperlinks, IsPeriodic, IsPlanar, Knots, Lineweight, Modified, ObjectName, OwnerID, PlotStyleName, Weights

New methods, properties, and events since AutoCAD Release 14.01

Object	Method / Property / Event Name
Text	Alignment, Backward, Delete, Document, GetExtensionDictionary, HasExtensionDictionary, Hyperlinks, Lineweight, Modified, ObjectName, OwnerID, PlotStyleName, UpsideDown
TextStyle	Document, GetExtensionDictionary, GetFont, HasExtensionDictionary, Modified, ObjectID, ObjectName, OwnerID, SetFont
TextStyles	Delete, Document, GetExtensionDictionary, GetXData, Handle, HasExtensionDictionary, Modified, ObjectID, ObjectName, OwnerID, SetXData
Tolerance	Delete, DimensionLineColor, Document, GetExtensionDictionary, HasExtensionDictionary, Hyperlinks, Lineweight, Modified, ObjectName, OwnerID, PlotStyleName, ScaleFactor, TextColor, TextGap, TextHeight, TextStyle
ToolBar*	AddSeparator, AddToolbarButton, Application, Count, Delete, Dock, DockStatus, Float, FloatingRows, Height, HelpString, Item, LargeButtons, Left, Name, Parent, TagString, Top, Visible, Width
ToolbarItem*	Application, AttachToolbarToFlyout, Delete, Flyout, GetBitmaps, HelpString, Index, Macro, Name, Parent, SetBitmaps, TagString, Type
Toolbars*	Add, Application, Count, Item, LargeButtons, Parent

Changed Items

The following section briefly describes the existing methods and properties that have been changed since Release 14.01:

Changed methods and properties

Release 14 Method/Property Name	AutoCAD 2000 Signature	Description of Changes
ArcSmoothness	object.ArcSmoothness	Moved from the Preferences object to the PViewport and Viewport object.
AutoSaveFile	object.AutoSavePath	The name of this property has been changed to AutoSavePath. This propety has also moved from the Preferences object to the PreferencesFiles object.
InsertBlock	RetVal = object.InsertBlock(InsertionPoint, Name, Xscale, Yscale, ZScale, Rotation)	The ZScale parameter has been added.
LogFileName	object.LogFilePath	The name of this property has been changed to LogFilePath. This propety has also moved from the Preferences object to the PreferencesFiles object.
TextString (on Dimension objects)	object.TextOverride	The name of this property has been changed to TextOverride.
ZoomAll	object.ZoomAll (VportObj)	The VportObj parameter has been added. This property has moved off the PViewport and Viewport objects to the Application object.
ZoomCenter	object.ZoomCenter VportObj, Center, Magnify	The VportObj parameter has been added. This property has moved off the PViewport and Viewport objects to the Application object.
ZoomExtents	object.ZoomExtents VportObj	The VportObj parameter has been added. This property has moved off the PViewport and Viewport objects to the Application object.
ZoomPickWindow	object.ZoomPickWindow	This property has moved off the PViewport and Viewport objects to the Application object.

Changed methods and properties (*continued*)

Release 14 Method/Property Name	AutoCAD 2000 Signature	Description of Changes
ZoomScaled	object.ZoomScaled VportObj, Scale, ScaleType	The VportObj parameter has been added. This property has moved off the PViewport and Viewport objects to the Application object.
ZoomWindow	object.ZoomWindow VportObj, LowerLeft, UpperRight	The VportObj parameter has been added. This property has moved off the PViewport and Viewport objects to the Application object.

Preferences Object

The `Preferences` object has several major changes:

- The `Preferences` object has been subdivided extensively. The new objects that contain preferences are

  `PreferencesConstructionTools`

  `PreferencesDisplay`

  `PreferencesFiles`

  `PreferencesOpenSave`

  `PreferencesOutput`

  `PreferencesSelection`

  `PreferencesSystem`

  `PreferencesUser`

  `DatabasePreferences`

- All properties and methods previously found on the `Preferences` object have been moved to one of the objects listed above. To find the new object for any given `Preferences` methods and properties, look up the method or property in the *ActiveX and VBA Reference*.
- All properties specifying color have been changed to use the VB constant `OLE_COLOR` instead of the AutoCAD color constants. (Note that this applies only to properties found on the `Preferences` objects listed above.)

Removed Items

The following section briefly describes the existing methods and properties that have been removed since AutoCAD Release 14.01.

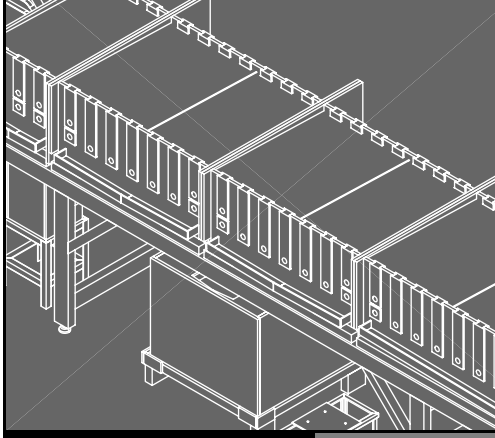
Removed methods and properties

Release 14 Method/Property Name	Migration Options
AdjustAreaFill	Set in a PC3 configuration file (via the Plotter Configuration Editor).
CrosshairColor	Use PreferencesDisplay.ModelSpaceCrosshairColor or PreferencesDisplay.LayoutCrosshairColor.
EntityName	Use the Autocad ActiveX ObjectName property, or in VB and VBA use the TypeOf keyword or TypeName function.
EntityType	See EntityName above.
Erase	Removed from all objects except Selection Set. Use the Delete method instead of Erase.
GraphicsTextBackgrndColor	Graphics text uses the same background color as the graphics. Use PreferencesDisplay.GraphicsWinLayoutBackgrndColor and PreferencesDisplay.GraphicsWinModelBackgrndColor.
GraphicsTextColor	Graphics text uses the same color as the pointer (crosshair). Use PreferencesDisplay.LayoutCrosshairColor and PreferencesDisplay.ModelCrosshairColor.
HideLines	Use AcadLayout.PlotHidden.
LoadPC2	PC2 files are not supported in AutoCAD 2000. You will need to convert your PC2 files to PC3 files via the Add-a-Plotter Wizard.
Origin	Use AcadLayout.PlotOffset.
PaperSize	Use AcadLayout.CanonicalMediaName.
PlotExtents	Use AcadLayout.AreaToPlot.
PlotLimits	Use AcadLayout.AreaToPlot.
PlotOrientation	Use AcadLayout.Rotation.

Removed methods and properties (*continued*)

Release 14 Method/Property Name	Migration Options
PlotScale	Use AcadLayout.Scale, StandardScale, CustomScale.
PlotUnits	Use AcadLayout.PaperUnits.
PlotView	Use AcadLayout.AreaToPlot and ViewToPlot.
PlotWindow	Use AcadLayout.AreaToPlot.
PlotWithConfigFile	Use PlotToDevice instead, and specify a PC3 file as an argument.
Rotation	Use AcadLayout.Rotation.
SavePC2	PC2 files are not supported in AutoCAD 2000. You will need to convert your PC2 files to PC3 files via the Add-a-Plotter Wizard.

Visual LISP Changes



In This Appendix

C

- New Functions
- Changed Functions

This appendix lists the changes between AutoLISP and the new Visual LISP development environment.

New Functions

The following functions are new for AutoLISP. Many are part of the new Visual LISP development environment:

New AutoLISP functions		
Name	**Name**	**Name**
defun-q	defun-q-list-ref	defun-q-list-set
vl-acad-defun	vl-acad-undefun	vl-arx-import
vl-bb-ref	vl-bb-set	vl-cmdf
vl-consp	vl-directory-files	vl-doc-export
vl-doc-import	vl-doc-ref	vl-doc-set
vl-every	vl-exit-with-error	vl-exit-with-value
vl-file-copy	vl-file-delete	vl-file-directory-p
vl-file-rename	vl-file-size	vl-file-systime
vl-filename-base	vl-filename-directory	vl-filename-extension
vl-filename-mktemp	vl-get-resource	vl-list*
vl-list->string	vl-list-exported-functions	vl-list-length
vl-list-loaded-vlx	vl-load-all	vl-load-com
vl-load-dcl-resource	vl-load-fas-resource	vl-load-reactors
vl-load-vba-resource	vl-member-if	vl-member-if-not
vl-position	vl-prin1-to-string	vl-princ-to-string
vl-propagate	vl-registry-delete	vl-registry-descendents
vl-registry-read	vl-registry-write	vl-remove
vl-remove-if	vl-remove-if-not	vl-some
vl-sort	vl-sort-i	vl-string->list

New AutoLISP functions (*continued*)		
Name	**Name**	**Name**
vl-string-elt	vl-string-left-trim	vl-string-mismatch
vl-string-position	vl-string-right-trim	vl-string-search
vl-string-subst	vl-string-translate	vl-string-trim
vl-symbol-name	vl-symbol-value	vl-symbolp
vl-unload-vlx	vl-vbaload	vl-vbarun
vlax-3D-point	vlax-add-cmd	vlax-create-object
vlax-curve-getArea	vlax-curve-getDistAtParam	vlax-curve-getDistAtPoint
vlax-curve-getEndParam	vlax-curve-getEndPoint	vlax-curve-getParamAtDist
vlax-curve-getParamAtPoint	vlax-curve-getPointAtDist	vlax-curve-getPointAtParam
vlax-curve-getStartParam	vlax-curve-getStartPoint	vlax-curve-isClosed
vlax-curve-isPeriodic	vlax-curve-isPlanar	vlax-curve-getClosestPointTo
vlax-curve-getClosestPointToProjection	vlax-curve-getFirstDeriv	vlax-curve-getSecondDeriv
vlax-dump-object	vlax-ename->vla-object	vlax-erased-p
vlax-for	vlax-get-acad-object	vlax-get-object
vlax-get-or-create-object	vlax-get-property	vlax-import-type-library
vlax-invoke-method	vlax-ldata-delete	vlax-ldata-get
vlax-ldata-list	vlax-ldata-put	vlax-ldata-test
vlax-make-safearray	vlax-make-variant	vlax-map-collection
vlax-method-applicable-p	vlax-object-released-p	vlax-product-key
vlax-property-available-p	vlax-put-property	vlax-read-enabled-p
vlax-release-object	vlax-remove-cmd	vlax-safearray-fill
vlax-safearray-get-dim	vlax-safearray-get-element	vlax-safearray-get-l-bound
vlax-safearray-get-u-bound	vlax-safearray-put-element	vlax-safearray-type

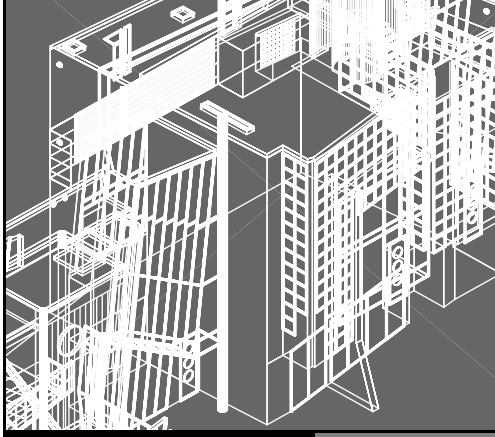
New AutoLISP functions (*continued*)		
Name	**Name**	**Name**
vlax-safearray->list	vlax-tmatrix	vlax-typeinfo-available-p
vlax-variant-change-type	vlax-variant-type	vlax-variant-value
vlax-vla-object->ename	vlax-write-enabled-p	vlisp-compile
vlisp-import-exsubrs	vlr-acdb-reactor	vlr-add
vlr-added-p	vlr-beep-reaction	vlr-command-reactor
vlr-current-reaction-name	vlr-data	vlr-data-set
vlr-docmanager-reactor	vlr-deepclone-reactor	vlr-dwg-reactor
vlr-dxf-reactor	vlr-editor-reactor	vlr-insert-reactor
vlr-linker-reactor	vlr-lisp-reactor	vlr-miscellaneous-reactor
vlr-mouse-reactor	vlr-object-reactor	vlr-owner-add
vlr-owner-remove	vlr-owners	vlr-pers
vlr-pers-p	vlr-pers-release	vlr-reaction-names
vlr-reaction-set	vlr-reactions	vlr-reactors
vlr-remove	vlr-remove-all	vlr-sysvar-reactor
vlr-toolbar-reactor	vlr-trace-reaction	vlr-type
vlr-types	vlr-undo-reactor	vlx-loaded-p

Changed Functions

The arguments, return values, or both for these AutoLISP functions have been changed in AutoCAD 2000:

Changed AutoLISP functions		
Name	**Name**	**Name**
ssget	trace	type

ObjectDBX Changes



In This Appendix

D

- Class Methods
- Global Functions

This appendix lists the changes to the APIs that were only supplied with DWG Unplugged. Many of these APIs are now available in both ObjectARX and ObjectDBX; some have been replaced, and others have been removed. There are no longer APIs specific to ObjectDBX, as there were for DWG Unplugged.

Class Methods

There were many classes in DWG Unplugged with methods that were either specifically available in DWG Unplugged, or that had different behaviors for DWG Unplugged and ObjectARX. These have been resolved as shown in the following table:

Methods unique to DWG Unplugged by class

API	AutoCAD 2000 Disposition	Notes
void AcDb2dVertex::setVertexType();	Deleted	Unnecessary due to Release 12 Objectification
void AcDb3dPolylineVertex::setVertexType();	Deleted	Unnecessary due to Release 12 Objectification
Acad::ErrorStatus AcDbBlockTableRecord:: setHasAttributeDefinitions();	Deleted	Unnecessary due to Release 12 Objectification
Acad::ErrorStatus AcDbBlockTableRecord:: setIsAnonymous();	Deleted	Unnecessary due to Release 12 Objectification
Acad::ErrorStatus AcDbBlockTableRecord:: setIsFromExternalReference();	Deleted	Unnecessary due to Release 12 Objectification
Acad::ErrorStatus AcDbBlockTableRecord:: setIsFromOverlayReference();	Deleted	Unnecessary due to Release 12 Objectification
Acad::ErrorStatus AcDbDatabase::dxfIn();	Added to ObjectARX	Has new arguments in AutoCAD 2000
Acad::ErrorStatus AcDbDatabase::dxfOut();	Added to ObjectARX	Has new arguments in AutoCAD 2000
Acad::ErrorStatus AcDbDatabase::getFilename() const;	Added to ObjectARX	
Acad::ErrorStatus AcDbDatabase::getProjectName() const;	Added to ObjectARX	
void* AcDbDatabase::getThumbnailBitmap() const;	Renamed	Renamed to AcDbDatabase:: thumbnailBitmap()

Methods unique to DWG Unplugged by class (*continued*)

API	AutoCAD 2000 Disposition	Notes
Acad::ErrorStatus AcDbDatabase::readDwgFile();	Added to ObjectARX	
void AcDbDatabase::resetThumbnailImage();	Deleted	Use AcDbDatabase:: setThumbnailBitmap(NULL);
Acad::ErrorStatus AcDbDatabase::save();	Added to ObjectARX	
Acad::ErrorStatus AcDbDatabase::setHandseed();	Added to ObjectARX	
Acad::ErrorStatus AcDbDatabase::setProjectName() const;	Added to ObjectARX	
Acad::ErrorStatus AcDbDatabase::setThumbnailBitmap();	Added to ObjectARX	
void AcDbDatabase::setThumbnailClearOnSave();	Deleted	Use AcDbDatabase:: [set]RetainOriginalBitmap();
Adesk::UInt32 AcDbHandle::high();	Added to ObjectARX	
Adesk::UInt32 AcDbHandle::low();	Added to ObjectARX	
void AcDbHandle::setHigh();	Added to ObjectARX	
void AcDbHandle::setLow();	Added to ObjectARX	
double AcDbLeader::annoHeight() const;	Added to ObjectARX	
AnnoType AcDbLeader::annoType() const;	Added to ObjectARX	
double AcDbLeader::annoWidth() const;	Added to ObjectARX	
AcGeVector3d AcDbMline::axisAt() const;	Added to ObjectARX	

Methods unique to DWG Unplugged by class (*continued*)

API	AutoCAD 2000 Disposition	Notes
Acad::ErrorStatus AcDbMline::getParametersAt();	Added to ObjectARX	
AcGeVector3d AcDbMline::miterAt() const;	Added to ObjectARX	
void AcDbPolyFaceMeshVertex::setVertexType();	Deleted	Unnecessary due to Release 12 Objectification
void AcDbPolygonMeshVertex::setVertexType();	Deleted	Unnecessary due to Release 12 Objectification
Adesk::Boolean AcDbViewport::isOn() const;	Added to ObjectARX	
void AcDbShape::setShapeIndex();	Added to ObjectARX	
AcDbHardPointerId AcDbShape::shapeIndex() const;	Added to ObjectARX	
Adesk::Int16 AcDbShape::shapeNumber() const;	Added to ObjectARX	
virtual AcDbBlockReference* AcGiContext::blockReference() const = 0;	Deleted	
virtual void AcGiContext::candidateWCSPoint() = 0;	Deleted	
virtual AcDbDatabase* AcGiContext::database() const = 0;	Added to AutoCAD 2000 AcGi	
virtual const AcGeMatrix3d& AcGiContext::inverseModelTransform() const = 0;	Deleted	
virtual Adesk::Boolean AcGiContext::isPlotGeneration() const = 0;	Added to AutoCAD 2000 AcGi	
virtual Adesk::Boolean AcGiContext::isPsOut() const = 0;	Added to AutoCAD 2000 AcGi	

Methods unique to DWG Unplugged by class (*continued*)

API	AutoCAD 2000 Disposition	Notes
virtual Adesk::Boolean AcGiContext::isRegenForExtents() const = 0;	Deleted	
virtual const AcGeMatrix3d& AcGiContext::modelTransform() const = 0;	Deleted	
virtual void AcGiContext::popBlockReference() = 0;	Deleted	
virtual void AcGiContext::popModelTransform() = 0;	Deleted	
virtual void AcGiContext::pushBlockReference() = 0;	Deleted	
virtual void AcGiContext::pushModelTransform() = 0;	Added to AutoCAD 2000 AcGi	
virtual void AcGiContext::pushModelTransform() = 0;	Added to AutoCAD 2000 AcGi	
virtual Adesk::Boolean AcGiViewport::generateFull3d() const = 0;	Deleted	
virtual Adesk::Boolean AcGiViewport::layerVisible() const = 0;	Added to AutoCAD 2000 AcGi	
virtual double AcGiViewport::linetypeGenerationCriterion() const = 0;	Added to AutoCAD 2000 AcGi	
virtual double AcGiViewport::linetypeScaleMultiplier() const = 0;	Added to AutoCAD 2000 AcGi	
virtual double AcGiViewport::logicalToScreenRatio() const = 0;	Deleted	
virtual const AcGeVector3d& AcGiViewport::viewDir() const = 0;	Added to AutoCAD 2000 AcGi	

Methods unique to DWG Unplugged by class (*continued*)

API	AutoCAD 2000 Disposition	Notes
virtual AcGiContext& AcGiViewportDraw::context() const = 0;	Added to AutoCAD 2000 AcGi	
virtual void AcGiViewportDraw::regenerate() = 0;	Deleted	
virtual AcGiContext& AcGiWorldDraw::context() = 0;	Added to AutoCAD 2000 AcGi	
virtual void AcGiWorldDraw::regenerate() = 0;	Deleted	

Global Functions

The DWG Unplugged product contained many global functions that were not available in ObjectARX. These functions have been rationalized for ObjectDBX and ObjectARX as shown in the following table:

Global functions unique to DWG Unplugged

API	AutoCAD 2000 Disposition	Notes
void acdbAcisDeleteModelerBulletins();	Added to ObjectARX	
void acdbCleanUp();	Added to ObjectARX	Only required for use by ObjectDBX
void acdbDeleteResBuf();	Added to ObjectARX	Equivalent to ads_relrb()
const char* acdbDWGUnpluggedRegistryKey();	Added to ObjectARX	Registry location for the custom object portion of applications
const char* acdbGetBuild();	Deleted	Right-click versioning

Global functions unique to DWG Unplugged (*continued*)

API	AutoCAD 2000 Disposition	Notes
AcDbObjectId acdbGetCurVportId();	Added to ObjectARX	
int acdbGetObjectSize();	Deleted	
void* acdbGetPreviewFromDwg();	Deleted	Use the ObjectARX equivalents
const char* acdbGetProdname();	Deleted	Use AcDbHostApplicationServices ::product()
Acad::ErrorStatus acdbGetProxyInfo();	Added to ObjectARX	
const char* acdbGetVersion();	Deleted	Right-click versioning
int acdbIsFullAcis();	Deleted	Multiple ACIS versions are no longer supported
void acdbMakeDatabaseCurrent();	Deleted	Use AcDbHostApplicationServices ::setWorkingDatabase()
struct resbuf* acdbNewResbuf();	Added to ObjectARX	Equivalent to ads_newrb()
void acdbSetDatabaseFromCurrent();	Deleted	Use AcDbHostApplicationServices ::setWorkingDatabase()
void acdbSetHostApplicationServices();	Added to ObjectARX	
void acdbVersionDialog();	Deleted	Right-click versioning
void acdbWblockAsR13();	Deleted	Use acdbSaveAsR13() and acdbSaveAsR14()
BOOL clientItem2Data();	Deleted	Use new AcDbHostApplicationServices features

Global functions unique to DWG Unplugged (*continued*)

API	AutoCAD 2000 Disposition	Notes
BOOL data2ClientItem();	Deleted	Use new AcDbHostApplicationServices features
BOOL getRectFromData();	Deleted	Use new AcDbHostApplicationServices features

Subject Index

B

C

D

E

F

G

H

I

N

O

P

Q

R

Code Index

Header Files

Libraries

ActiveX/VBA

Visual LISP