AutoCAD .NET

Developer's Guide

Refresh

View Code

Show All Files

View Designer

Properties

View Class Diagram

Solution

Solution Explorer - Solution 'VbMdgAcad'...

Solution 'VbMdgAcad' (2 projects)

CsMgdAcad

VbMdgAcad — Project

My Project

References

AcCui

acdbmgd

acmgd

AutoCAD

CsMgdAcad

PresentationCore

System

System.Core

System.Drawing

System.Windows.Forms

bin

obj

Class1.vb — Class File

Code File — Commands.vb

Form1.vb — Windows Form

Stephen Preton
Autodesk Developer Technical Services Team (DevTech)

AutoCAD .NET Developers Guide

Author: Stephen Preton
              Autodesk Developer Technical Services Team (DevTech)

Contents

_

1   Introduction

This introduction describes the concepts of exposing AutoCAD® objects through a managed .NET application programming interface (API). The AutoCAD .NET API allows you to automate tasks such as creating and modifying objects stored in the database of a drawing file or change the contents of a customization file. This guide covers using Microsoft® Visual Studio® 2008, and the programming languages Microsoft® Visual Basic® .NET (referred to in this guide as VB.NET) and Microsoft® Visual C#® with the AutoCAD .NET API.

Topics in this section

- Guide Organization
- Overview of the AutoCAD .NET API
- Components of the AutoCAD .NET API
- Overview of Microsoft Visual Studio
- For More Information
- Sample Code
- Transition from ActiveX Automation to .NET

Guide Organization

This guide provides information on how to use the AutoCAD .NET API with Microsoft Visual Studio and the programming languages VB.NET and C#. Information specific to developing applications using Microsoft Visual Studio can be found under the topics "Getting Started with Microsoft Visual Studio" and "Develop Applications with Microsoft Visual Studio."

Programmers developing with the .NET Framework using a development environment other than Microsoft Visual Studio can skip these two chapters. However, all of the example code in this guide is presented in VB.NET and C#.

Overview of the AutoCAD .NET API

The AutoCAD .NET API enables you to manipulate AutoCAD and drawing files programmatically with the assemblies or libraries that are exposed. With these objects exposed, they can be accessed by many different programming languages and environments.

There are several advantages to implementing a .NET API for AutoCAD:

- Programmatic access to AutoCAD drawings is opened up to more programming environments. Before the .NET API, developers were limited to ActiveX® Automation and languages that supported COM, AutoLISP®, and C++ with ObjectARX.
- Integrating with other Windows® based applications, such as Microsoft Excel and Word, is made dramatically easier by using an application's native .NET API or exposed ActiveX/COM library.
- The .NET Framework is designed for both 32-bit and 64-bit operating systems. Visual Basic for Applications was only designed for 32-bit operating systems.

- Allows access to advanced programming interfaces with a lower learning curve than those for more traditional programming languages such as C++.

Objects are the main building blocks of the AutoCAD .NET API. Each exposed object represents a precise part of AutoCAD, and they are grouped into different assemblies and namespaces. There are many different types of objects in the AutoCAD .NET API. For example:

- Graphical objects such as lines, arcs, text, and dimensions
- Style settings such as text and dimension styles
- Organizational structures such as layers, groups, and blocks
- The drawing displays such as view and viewport
- The drawing and AutoCAD application

For information on some of the files which make up the AutoCAD .NET API, see Components of the AutoCAD .NET API.

Components of the AutoCAD .NET API

The AutoCAD .NET API is made up of different DLL files that contain a wide range of classes, structures, methods, and events that provide access to objects in a drawing file or the AutoCAD application. Each DLL file defines different namespaces which are used to organize the components of the libraries based on functionality.

The three main DLL files of the AutoCAD .NET API that you will frequently use are:

- *AcDbMgd.dll.* Use when working with objects in a drawing file.
- *AcMgd.dll.* Use when working with the AutoCAD application.
- *AcCui.dll.* Use when working with customization files.

Use an AutoCAD .NET API DLL

Before classes, structures, methods, and events found in one of the AutoCAD .NET API related DLLs can be used, you must reference the DLL to a project. After a DLL is referenced to a project, you can utilize the namespaces and the components in the DLL file in your project.

Once a AutoCAD .NET API DLL is referenced, you must set the Copy Local property of the referenced DLL to False. The Copy Local property determines if Microsoft Visual Studio creates a copy of the referenced DLL file and places it in the same directory as the assembly file of the project when it is built. Since the referenced files already ship with AutoCAD, creating copies of referenced DLL files can cause unexpected results when you load your assembly file in AutoCAD.

Location of AutoCAD .NET API DLL files

The AutoCAD .NET API DLL files can be located at *<drive>:\Program Files\AutoCAD 2010* or as part of the *AutoCAD 2010 ObjectARX SDK* which can be downloaded from http://www.objectarx.com or the Autodesk Developer Network Web site (http://www.autodesk.com/adn).

After the ObjectARX SDK is installed, the DLL files can be found in the *inc-win32* and *inc-x64* folders under the main install folder.

NoteThe DLLs in the ObjectARX SDK are simplified versions of the same files that ship with AutoCAD, as they do not contain dependencies on the AutoCAD user interface. It is recommended

that you download and install the ObjectARX SDK, and then reference the DLL files that come with the SDK instead of those that are found in the install directory of AutoCAD 2010.

Procedures

⊟ To download and install the AutoCAD 2010 ObjectARX SDK

1. Launch your default Internet browser application and browse to http://www.objectarx.com.
2. On the Web page, click *License & Download*.
3. Fill in the required fields and select ObjectARX for AutoCAD 2010. Click *Submit*.
4. On the Download page, click *Download Now* to use the Download Manager or click *Standard Download Method* to use the default download method of your Internet browser.
5. Click *Save* or the option used to save the file to your local drive.
6. Specify a location to download the ObjectARX SDK package file.
7. Once the package file is downloaded, browse to the location you saved it to and double-click it.

   The install wizard is displayed.

8. In the ObjectARX *<Release>* dialog box, specify a new install location or leave the default install location. Click *Install*.

   The install wizard closes after it is finished if no problems were encountered.

⊟ To install the ObjectARX Wizard and the Managed project templates

1. Once the ObjectARX SDK is installed, browse to its install folder which is by default *c:\ObjectARX <Release>*.
2. After browsing to the install folder, open the *Utils* folder and then the *ObjARXWiz* folder.
3. In the *ObjARXWiz* folder, double-click *ArxWizards.msi*.
4. In the ObjectARX Wizards for AutoCAD 2010 dialog box, click *Next*.
5. In the Enter Your Preferred Default RDS Symbol box, enter an abbreviation of your company's name and click *Next*.

   The text you enter for the RDS is used as a prefix to the names of the default class or commands created with the ObjectARX Wizard and project templates.

   NoteRDS stands for Registered Developer Symbol, and is used to create unqiuely named commands and classes to help avoid potential conflicts with other applications.

6. On the Member Variable Wizard page, leave Replace MS Member Variable Wizard by Autodesk Member Variable Wizard selected and click *Next*.
7. On the Select Installation Folder page, click Browse to specify a new installation location for the wizard or leave the default location. Click *Next*.
8. Click *Next* again to install the wizard.
9. Click *Close* to close the installer.

⊟ To reference an AutoCAD .NET API DLL

1. In Microsoft Visual Studio, click View menu ➤ Solution Explorer to display the Solution Explorer if it is not already displayed.
2. In the Solution Explorer, on the toolbar along the top, click Show All Files.
3. Right-click the References node and click Add Reference.
4. In the Add Reference dialog box, Browse tab, select the DLL file that contains the library you want to use and click OK.
5. In the Solution Explorer, click the plus sign to the left the References node to expand it.

6. Select the referenced library from the References node.
7. Right-click over the selected reference and click Properties.
8. In the Properties window, click the Copy Local field and select False from the drop-down list.

Overview of Microsoft Visual Studio

Microsoft Visual Studio is an object-oriented programming environment that runs independently of AutoCAD. While Microsoft Visual Studio is external to AutoCAD and other applications, it is able to interact with applications that expose either a native .NET API or ActiveX/COM library.

Topics in this section

- Which Edition of Microsoft Visual Studio to Use
- Use COM Interoperability with .NET
- Dependencies and Restrictions

Which Edition of Microsoft Visual Studio to Use

Microsoft Visual Studio is available in multiple versions and editions. To use the .NET API for AutoCAD 2010, you need to use:

- Microsoft Visual Studio 2008 with Service Pack 1
- Microsoft .NET Framework 3.5 with Service Pack 1

If you are using AutoCAD 2007 through AutoCAD 2009, you should use:

- Microsoft Visual Studio 2005
- Microsoft .NET Framework 2.0 or later

Microsoft Visual Studio is offered in two editions: for free and for pay. The free edition is known as Microsoft Visual Studio 2008 Express Edition, while the for pay editions vary by name and price due to the different development tools that are incorporated into them. Microsoft Visual Studio 2008 Standard Edition is the entry level edition which provides improved debugging over Microsoft Visual Studio 2008 Express Edition along with a number of other features. The most common edition of Microsoft Visual Studio used by developers is Microsoft Visual Studio 2008 Professional Edition.

NoteWhile it is possible to use Microsoft Visual Studio Express with the AutoCAD .NET API, this guide assumes you are using one of the other versions such as Microsoft Visual Studio 2008 Standard Edition or Microsoft Visual Studio 2008 Professional Edition.

There are four main advantages to using Microsoft Visual Studio:

- Robust and accessible development environment that has a modest learning curve.
- VBA and VB.NET syntax are similar, which makes it an ideal environment for existing VBA users.
- Visually intuitive and extensive dialog box creation tools.

- Projects can be built as a standalone executable or DLL assembly which can then be loaded into AutoCAD for execution.

NoteUnlike VBA projects, .NET applications do not suffer from performance degradation when loaded and run in 64-bit AutoCAD.

For more information on the different editions of Microsoft Visual Studio, see http://www.microsoft.com/vstudio and http://www.microsoft.com/express.

Use COM Interoperability with .NET

Microsoft Visual Studio can utilize both native .NET and COM interfaces in the same project. By utilizing COM interop, you can migrate existing code that might have been written in Visual Basic 6 or VBA without having to completely rewrite it. To access AutoCAD automation objects from a project created in Microsoft Visual Studio, create references to the following files:

- The AutoCAD 2010 type library, *acax18enu.tlb*, located at *<drive>:\Program Files\Common Files\Autodesk Shared*.
- The AutoCAD/ObjectDBX Common 18.0 type library, *axdb18enu.tlb*, located at *<drive>:\Program Files\Common Files\Autodesk Shared*.

NoteThe previous mentioned type libraries are also available as part of the ObjectARX SDK. For information on downloading and installing the ObjectARX SDK, see Components of the AutoCAD .NET API.

These references will make the following primary interop assemblies available:

- Autodesk.AutoCAD.Interop.dll (for AutoCAD-specific types)
- Autodesk.AutoCAD.Interop.Common.dll (for types shared by ObjectDBX™ host applications)

The interop assemblies are located in the global assembly cache; they map automation objects to their .NET counterparts.

After you reference the type libraries, you can declare AutoCAD-based variables in Microsoft Visual Studio, as in the following examples:

VB.NET

```
Dim objAcApp As Autodesk.AutoCAD.Interop.AcadApplication
Dim objLine As Autodesk.AutoCAD.Interop.Common.AcadLine
```

C#

```
Autodesk.AutoCAD.Interop.AcadApplication objAcApp;
Autodesk.AutoCAD.Interop.Common.AcadLine objLine;
```

Utilizing the interop assemblies can make the transitioning your VBA projects over to VB.NET much easier. However, in order to take full advantage of everything that .NET and the AutoCAD .NET API have to offer, you will need to rewrite your existing VBA code.

Create and Reference the AutoCAD Application

AutoCAD 2010 .NET applications can utilize the same type library (*acax18enu.tlb*) as AutoCAD automation projects. The type library is located in *<drive>:\Program Files\Common Files\Autodesk Shared*.

AutoCAD 2010 .NET applications also use the same version-dependent ProgID for the CreateObject, GetObject, and GetInterfaceObject functions. For example, CreateObject ("AutoCAD.Application.18") allows you to create an instance of AutoCAD and get an object that represents the new instance of the application.

Dependencies and Restrictions

Unlike ActiveX Automation, there are fewer issues with library conflicts when other applications are installed, reinstalled, or uninstalled. The reason for fewer compatibility issues is that the .NET Framework is a standardized platform. However, you can still run into dependency issues. To avoid dependency issues with the .NET Framework, be sure to use the same or an earlier version of the .NET Framework with your VB.NET or C# project that AutoCAD 2010 uses. For information on which version of the .NET Framework you should be referencing, see Which Edition of Microsoft Visual Studio to Use.

For More Information

This guide assumes that you have working knowledge of either the VB.NET or C# programming languages, and does not attempt to duplicate or replace the abundance of documentation available on either of these programming languages. If you need more information on VB.NET or C#, and using Microsoft Visual Studio, see the Help system for Microsoft Visual Studio developed by Microsoft, available from the Help menu in Microsoft Visual Studio.

Sample Code

This guide and the *AutoCAD 2010 ObjectARX SDK* together contain a large number of sample projects, subroutines, and functions that demonstrate the use of the classes, structures, methods, properties, and events that make up the AutoCAD .NET API.

You can also find sample projects that demonstrate some of the aspects of the AutoCAD .NET API on the Autodesk website at http://www.autodesk.com/developautocad. These sample projects show a wide range of fuctionality, from extracting AutoCAD drawing data into Microsoft Excel spreadsheets to drawing and performing stress analysis on an electrical transmission tower.

Many of these samples show how to combine various aspects of the VB.NET and C# programming languages with the power of the AutoCAD .NET API to create custom applications.

Additionally, sample code in the *AutoCAD .NET Developer's Guide* can be copied from the Help files, pasted directly into an open code editor window in Microsoft Visual Studio, and then built and loaded in AutoCAD. At the top of most code samples in this guide are the namespaces that are required for that particular sample. Add those that are need to top of the code window.

NoteThe sample code in the Help files contains limited error handling to keep the concepts simple and easy to read. You should apply additional error handling and checking when using any sample code in your project. For more information on error handling, see Handle Errors.

Procedures

⊟ To run the sample code from the Help files

1. In Microsoft Visual Studio, open a code editor window, add the proper namespaces and define a class if one does not already exist.
2. Copy the sample code from the Help file and paste the copied code into the defined class.
3. In Microsoft Visual Studio, click Build menu ➤ Build *<Project>*.
4. In AutoCAD, at the command prompt, enter *netload* and press Enter.
5. In the Choose .NET Assembly dialog box, select the built assembly file. Click Open.
6. At the command prompt, enter the name of the command or AutoLISP function and any required parameters defined in the loaded assembly.

Transition from ActiveX Automation to .NET

AutoCAD 2010 continues to support VBA, but the VBA components must be enabled by downloading and installing them from http://www.autodesk.com/vba-download.

While there is no established release in which VBA will no longer be supported in AutoCAD, we recommend that you begin migrating your existing VBA projects to ensure that you are ready when VBA support is dropped. We recommend that you develop any new applications with Microsoft Visual Studio and the AutoCAD .NET API, AutoLISP, or C++ and ObjectARX.

You can continue to use the AutoCAD COM Automation library in .NET to help make your transition from VBA to VB.NET easier, see Use COM Interoperability with .NET. If you are new to .NET, see Getting Started with Microsoft Visual Studio for basic information on working with Microsoft Visual Studio and the AutoCAD .NET API.

2   Getting Started with Microsoft Visual Studio

This chapter introduces you to the AutoCAD® .NET API and some of the basics of the Microsoft® Visual Studio® development environment. Along with an introduction to the development environment, you will also learn about the NETLOAD command in AutoCAD and some of the terminology used throughout this guide.

Topics in this section

Understand Microsoft Visual Studio Projects

Project files created with Microsoft Visual Studio are not specific to AutoCAD, but do contain specific project settings that you will need to be familiar with in order to create a DLL assembly file that can be loaded into AutoCAD. This guide discusses creating projects for Visual Basic® (VB) .NET and Visual C#® with and without the ObjectARX Wizard for AutoCAD 2010.

A project is a collection of code and resource files; class modules, WPF windows, and Windows forms that work together to perform a given function. When a project is created, a solution is also created which contains the project you are creating. A solution is used to reference one or more projects. Usually you do not have to work with a solution directly unless you want to reference the exposed functionality of other projects.

An example of a multi-project solution is when you have a new project and an existing project that contain a set of methods, functions, and classes that you want to use with a new project.

A solution can contain any combination of VB.NET and C# projects. Using different types of projects in a single solution allows each developer to use the programming language of his or her choice.

Project and solution files have the following file extensions:

- *VBPROJ* - Microsoft Visual Basic (VB) .NET project file
- *CSPROJ* - Microsoft Visual C# project file
- *SLN* - Microsoft Visual Studio solution file

Define the Components in a Project

Each project can contain many different components. The different components of a project can contain class modules, forms, references, and resources.

Class Modules

Components that contain public and private procedures and functions. Class modules are used to define custom namespaces. Within a namespace, you define the procedures used for your program and define the structures to implment custom commands and AutoLISP functions.

Forms

Form components contain custom dialog boxes you lay out for use with your project. Forms in your project are displayed using a procedure or function, unless you build a stand-alone application. Windows forms, WPF windows, and user controls are some of the form types that can be part of a project. For this guide, forms can mean either Windows forms and WPF windows.

References

References are used to indicate which projects or libraries your project uses.

View Project Information

The Solution Explorer, which displays the current solution and a list of all loaded projects, also displays the code, class, and form module files contained in each loaded project, and the references to the external libraries used by each project.

The Solution Explorer has its own toolbar, which can be used to open various project components for editing. Use the View Code button to open the code for a selected module in the Editor Window. Use the View Designer button to display the Visual Designer for a selected form or the View Class Diagram to work with classes.
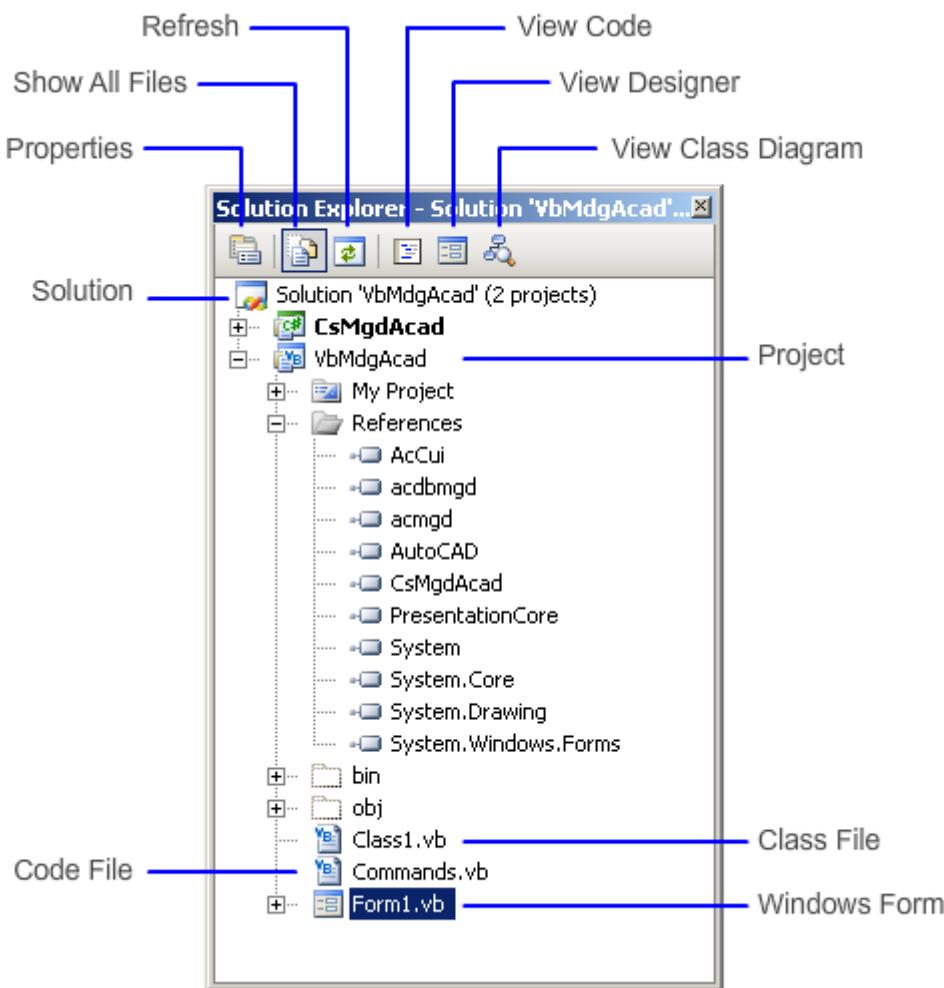
The Solution Explorer is visible by default. If it is not visible, click View menu ⮞ Solution Explorer or press Ctrl+Alt+L.

Work with Microsoft Visual Studio Projects

An open solution or project file can be viewed and accessed from the Solution Explorer. The Solution Explorer allows you to add new items and projects to the current solution and projects, unload a project from the current solution, change the properties of an opened solution or project, and add references to a project in the current solution, among many choices. All available tasks for working with the current solution or project can be accessed via the right-click menu of the Solution Explorer. For information on the Solution Explorer, see View Project Information.

NoteBefore creating a new project, make sure Service Pack 1 for Microsoft Visual Studio 2008 is installed.

Procedures

⊟ To start Microsoft Visual Studio

- On the Start menu, click (All) Programs ⮞ Microsoft Visual Studio 2008 ⮞ Microsoft Visual Studio 2008.

Topics in this section

- [Create a New Project](#)
- [Open an Existing Project or Solution](#)
- [Save a Project or Solution](#)
- [Work with Multiple Projects in a Solution](#)

Create a New Project

New projects are created based on a project template. When creating a new project that will be built into a DLL assembly that will be loaded into AutoCAD, use the Class Library template that comes with Microsoft Visual Studio or one of the AutoCAD Managed project application templates that get installed with the ObjectARX Wizard for AutoCAD 2010. Both types of templates are available for the VB.NET and C# programming languages.

After a new project is created, you will need to reference the files that make up the AutoCAD .NET API. For information on referencing the files associated with the AutoCAD .NET API and installing the ObjectARX Wizard for AutoCAD 2010, see [Components of the AutoCAD .NET API](#).

Procedures

⊟ To create a new project using a standard template

1. In Microsoft Visual Studio, click File menu ➤ New ➤ Project.
2. In the New Project dialog box, Project Types tree, expand Other Languages ➤ Visual Basic or Visual C# and click Windows.
3. Under Templates, select Class Library.
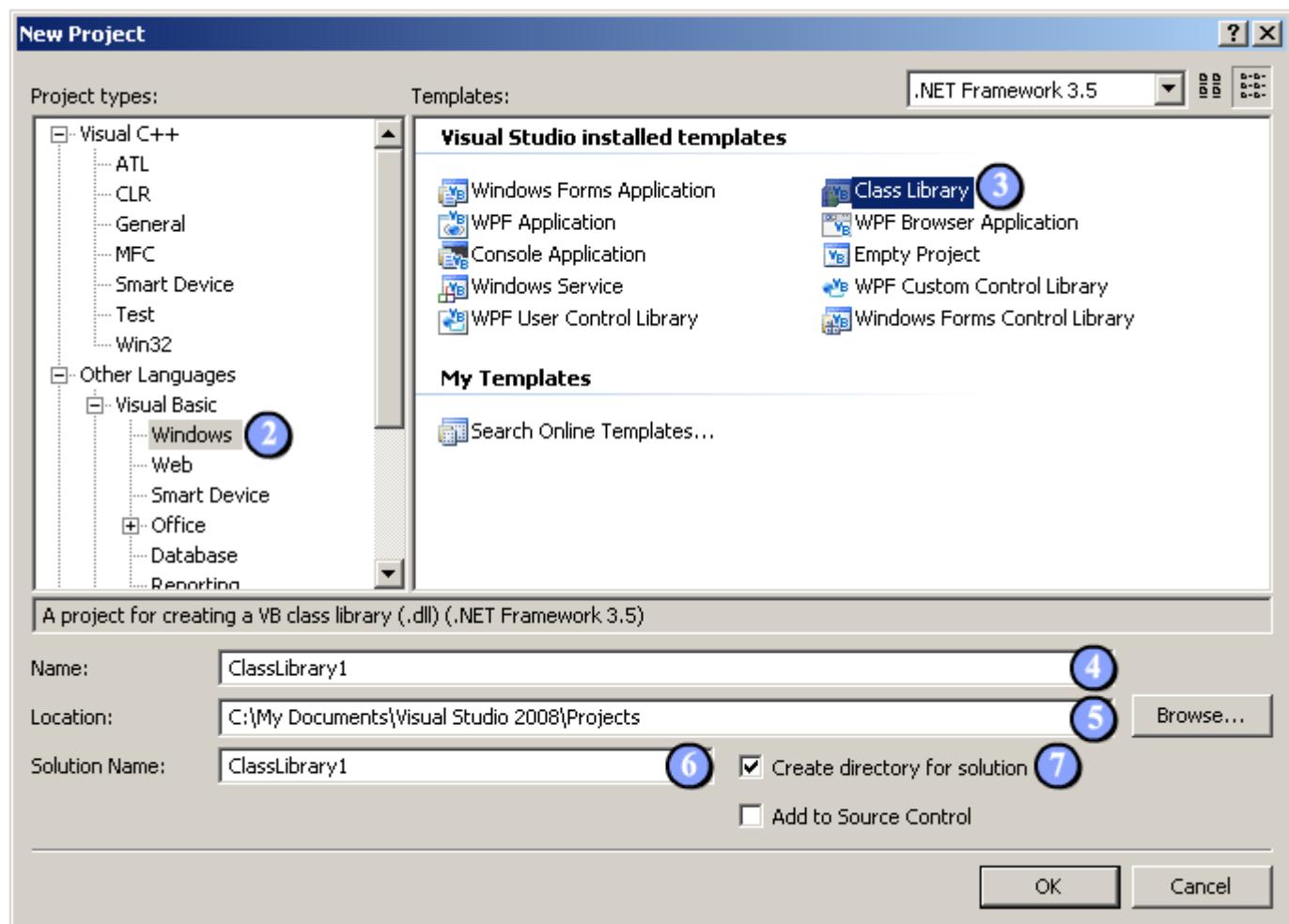
4. In the Name box, enter a name for the new project.
5. In the Location box, enter a location or click Browse to select a folder for the new project.
6. In the Solution Name box, enter a name for the new solution that the project will be added to.
7. Optionally, select the Create Directory for Solution check box to create a sub-folder before creating the new solution and project. Select the Add to Source Control check box to add the solution and project to a source control database.
8. Click OK.

⊟ To create a new project using an AutoCAD Managed template

Before using one of the AutoCAD Managed templates, you must first download and install the AutoCAD 2010 ObjectARX SDK. For information on installing the AutoCAD Managed project templates, see Components of the AutoCAD .NET API.

1. In Microsoft Visual Studio, click File menu ➤ New ➤ Project.
2. In the New Project dialog box, Project Types tree, expand Other Languages and click Visual Basic or Visual C#.
3. Under Templates, select AutoCAD Managed VB Project Application or AutoCAD Managed C# Project Application.
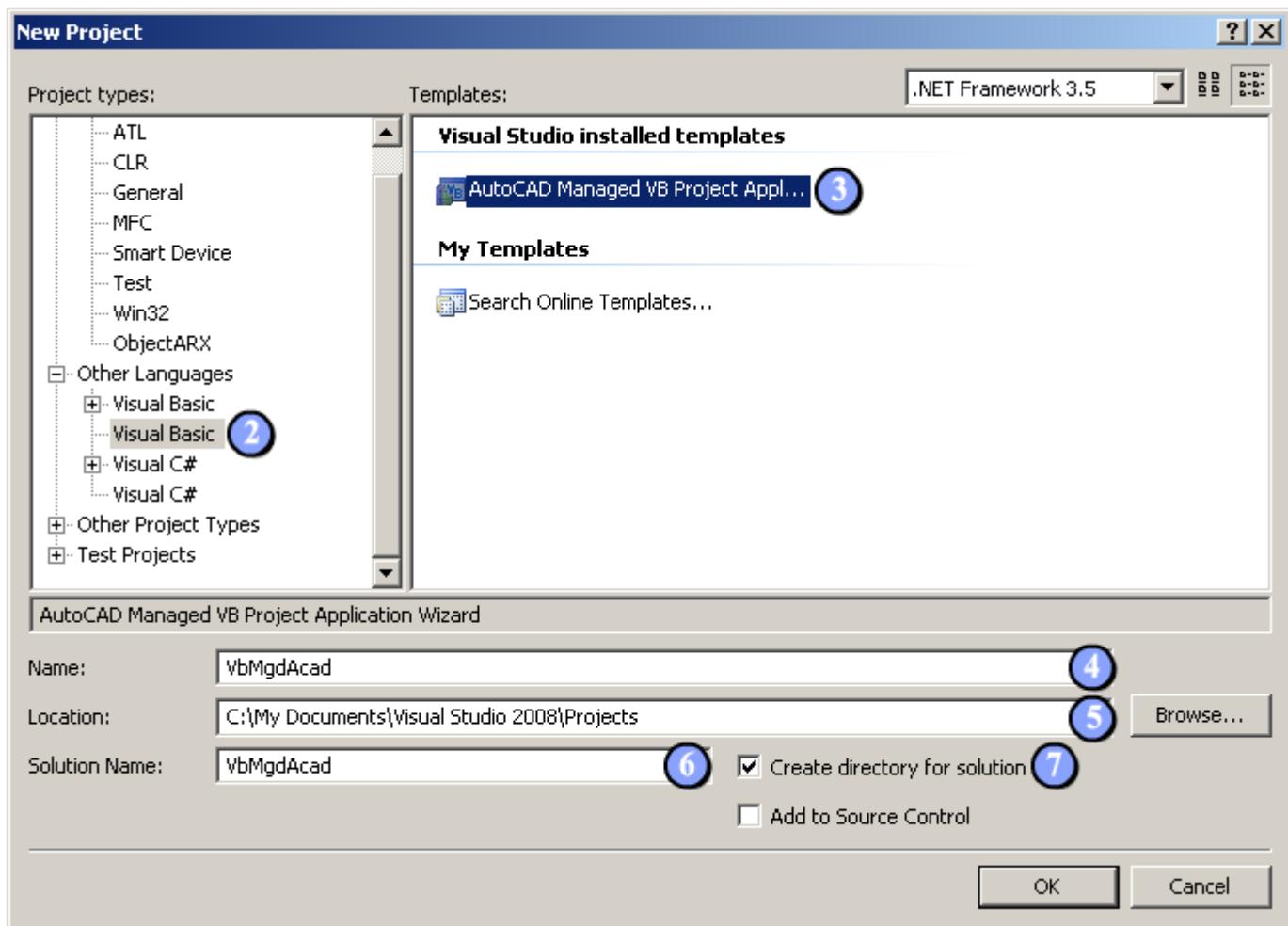
4. In the Name box, enter a name for the new project.
5. In the Location box, enter a location or click Browse to select a folder for the new project.
6. In the Solution Name box, enter a name for the new solution that the project will be added to.
7. Optionally, select Create Directory for Solution to create a sub-folder before creating the new solution and project. Select Add to Source Control to add the solution and project to a source control database.
8. Click OK.
9. In the AutoCAD Managed *<language>* Application Wizard - *<project name>* dialog box, go to the Your Registered Developer Symbol box, and enter a string up to four characters long.

    The symbol entered is used as a prefix for the default class and command that is added to the initial code module, and to avoid potential naming conflicts with other referenced libraries.
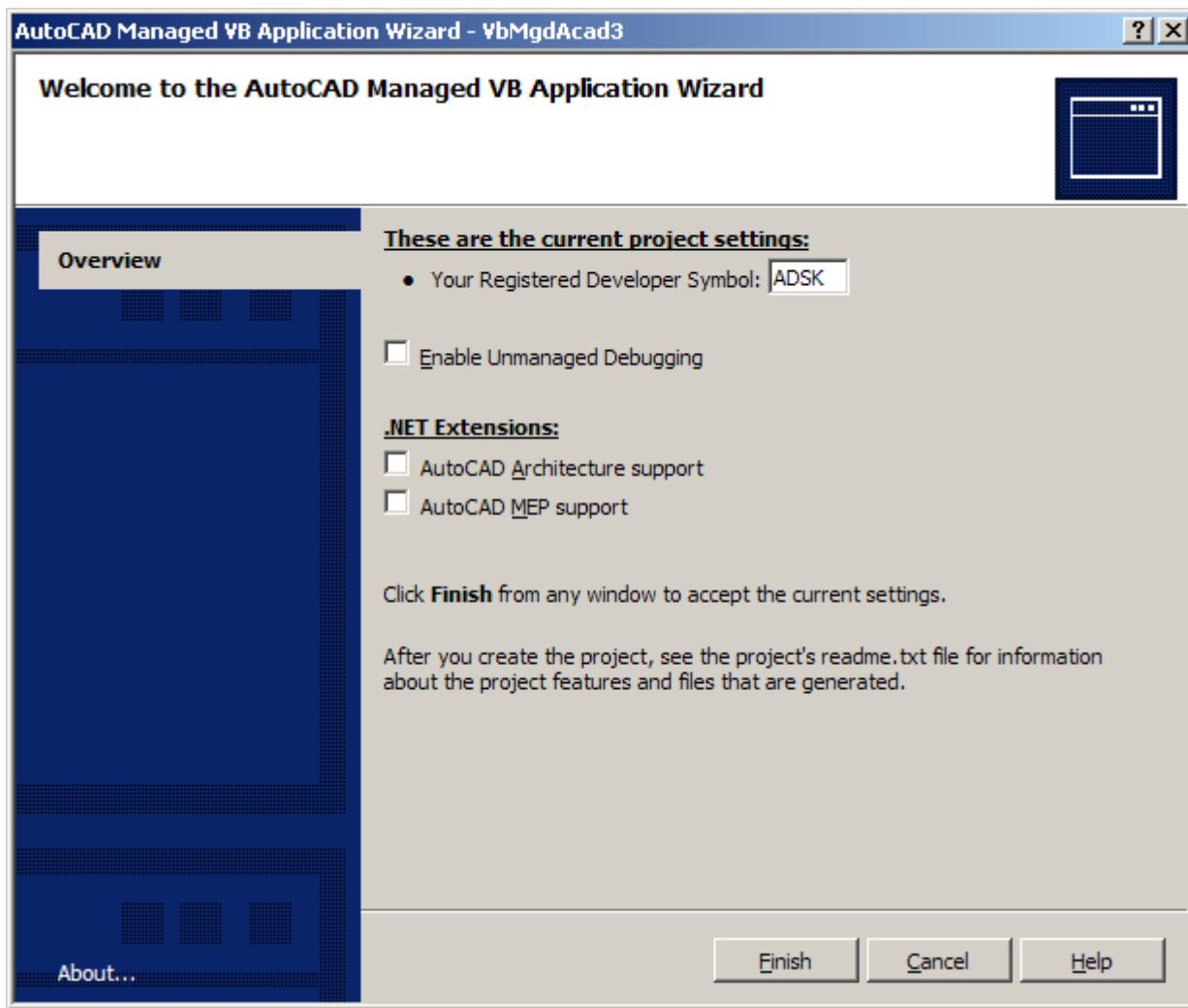
10. Optionally, if you will be working with libraries that utilize unmanaged code, select Enable Unmanaged Debugging.
11. Optionally, select any of the additional .NET extensions you might need for your project.
12. Click Finish.

Open an Existing Project or Solution

When you open a project or solution in Microsoft Visual Studio, the code editor and Windows Form Designer windows are opened in the same state they were in when the project was last saved. Once a project or solution is opened, use the Solution Explorer to navigate between the files of the opened project or solution. For information on working with an open project or solution see, Edit an Existing Project or Solution.

Procedures

⊟ To open an existing project or solution file

1. In Microsoft Visual Studio, click File menu ➤ Open ➤ Project/Solution.
2. In the Open Project dialog box, browse to and select the project file to open.

The Open Project dialog box will allow you to open a wide range of project files which include VB.NET and C# projects, Microsoft Visual Studio .NET solutions, Microsoft Visual C++ projects and even legacy Microsoft Visual Basic 6 projects. Use the Objects of Type drop-down list to control which type of projects or solutions you see in the file selection area of the Open Project dialog box.

NoteYou cannot open a VBA project saved as a DVB file with Microsoft Visual Studio. If you are migrating from VBA to VB.NET or C#, you can copy your code from the AutoCAD VBA development environment to an open code editor window in Microsoft Visual Studio and make the necessary changes.

3. Click Open.

Save a Project or Solution

Microsoft Visual Studio is a contextual based development environment, meaning that the current item selected determines the available features in the development environment. Controlling which changed files you save or how you want to save them is determined by the Solution Explorer.

Once a project or solution is selected from the Solution Explorer, you can save it with its current name, or perform a Save As on it to create a copy of the file. You can select one or more items at a time in the Solution Explorer by pressing and holding the Ctrl key before selecting an item. Most often, you will be saving all changed items at the same time.

Procedures

⊟ To save a project or solution

1. In Microsoft Visual Studio, from the Solution Explorer, select the project or solution to save.
2. Click File menu ➤ Save *<project or solution name>* or Save *<project or solution name>* As.

   If you choose Save *<project or solution name>* As, the Save File As dialog box is displayed.

3. If the Save File As dialog box is displayed, browse to a location to create a new copy of the project or solution and enter a new name. Click Save.

⊟ To save all changed items

- In Microsoft Visual Studio, click File menu ➤ Save All.

Work with Multiple Projects in a Solution

A solution can contain multiple projects. Working with multiple projects is not much different than working with a single project. Use the Solution Explorer to navigate between the projects in the current solution.

Add a project to a solution

You might add a project to a solution to copy code between projects. You might want to reference the procedures, functions and classes in one project and use them in another project. By adding multiple projects to a single solution, it allows you to use a project as a set of common utilities that you might use with more than one project.

Unload a project from a solution

Projects can be unloaded from a solution when they are no longer needed. If you no longer want a project to load with a solution, you can unload the project and then reference the compiled DLL assembly of the project instead. Working with the compiled DLL helps to prevent accidental edits to the source code.

Procedures

⊟ To add a project to a solution

- In Microsoft Visual Studio, do one of the following:
    - ○ Click File menu ➤ Add ➤ New Project to create a new project and add it to the current solution. For information on creating a new project, see Create a New Project.
    - ○ Click File menu ➤ Add ➤ Existing Project to display the Add Existing Project dialog box and add an existing project to the current solution. In the Add Existing Project dialog box, browse to and select the project to add to the current solution.

⊟ To unload a project from a solution

- In Microsoft Visual Studio, Solution Explorer, right-click the project you want to unload from the current solution and click Unload Project.

Edit an Existing Project or Solution

Once you have opened a project or solution into Microsoft Visual Studio, you can edit the projects, class modules, forms, and references using the common development environment. You can also debug and run projects from the common development environment.

Topics in this section

- Add New Items
- Import Existing Items

Add New Items

New items such as class modules and Windows forms can be added to a project. You are responsible for updating the properties of an item (such as name) and for filling in the appropriate code. When naming new items, remember that other developers may want to use your items in future applications. Be sure to follow your company's established naming conventions or the standards established by Microsoft.

Procedures

⊟ To add a new item to your project

1. In Microsoft Visual Studio, Solution Explorer, right-click the project to which you want to add a new item and click Add ➤ New Item.

   Instead of clicking New Item, you can also click Windows Form, Module (VB.NET only), or Class to add one of the main types of items to a project.

2. In the Add New Item - *<Project>* dialog box, select the template that represents the item you want to add.
   - *Code File* - Represents a code module
   - *Windows Form* - Represents a form (or dialog box)
   - *Class* - Represents a class module

Add New Item - VbMdgAcad dialog box
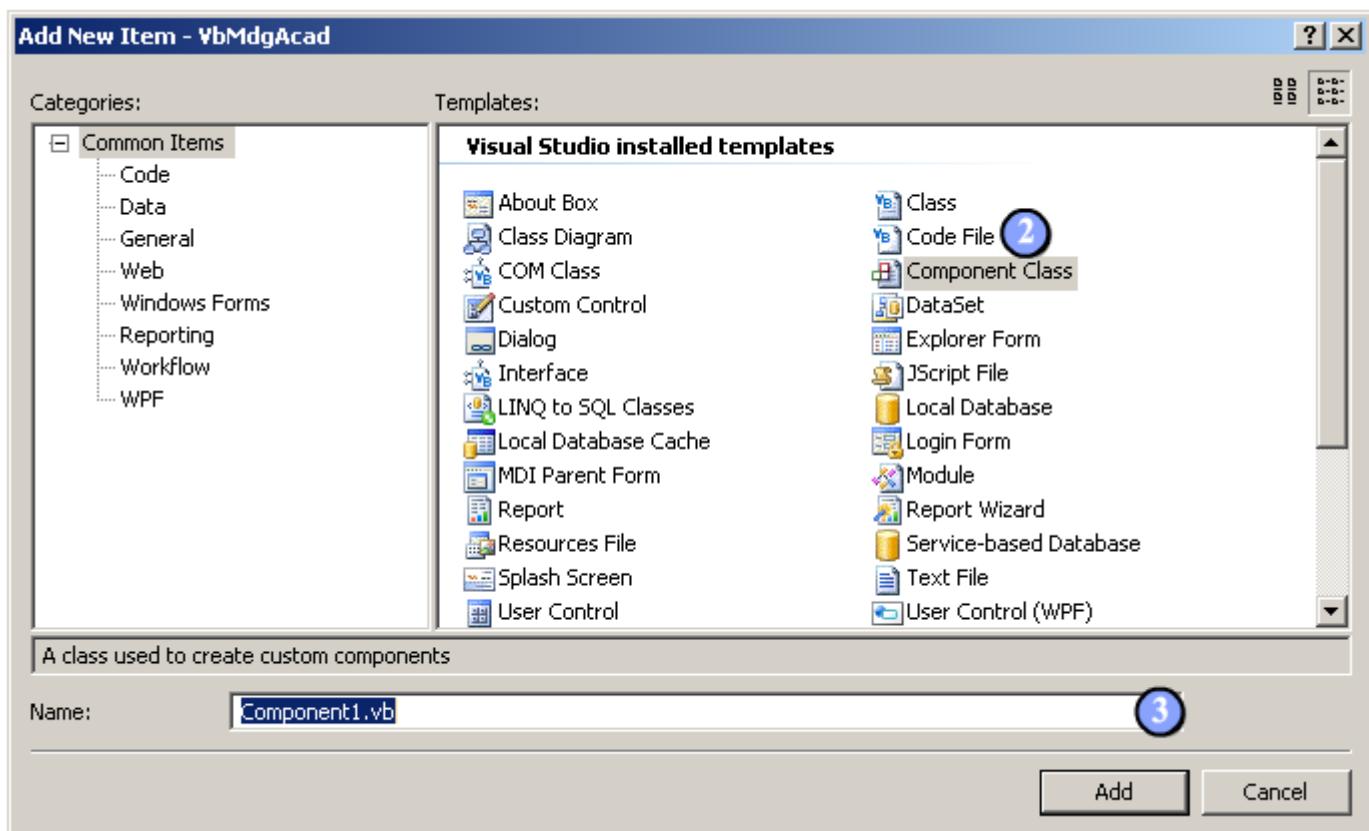
3. Enter a name for the new item and click Add.

Import Existing Items

Importing allows you to add an existing item to your project that might already exist in another project. You can import code or class modules and forms. Files with the VB extension can only be imported in a VB.NET project and CS files in a C# project. Class diagrams are stored as CD files and can be imported in either project type.

When you import an item, a copy of the file or a link to the file you are importing is added to the project. When a copy of the item is created, the original file is left unaltered in its original place. Linked files provide access to the original file, which is great for reusing common code that might be shared between multiple projects.

If you import an item with the same name as an existing item, you are prompted to replace the existing item in the project with the one being imported.

The imported component is added to your project and appears in the Solution Explorer. To edit the properties of the item, select the item in the Solution Explorer and change its properties in the Properties window.

Procedures

⊟ To import an existing item into a project

1. In Microsoft Visual Studio, Solution Explorer, right-click the project to which you want to add an existing item and click Add ➤ Existing Item.
2. In the Add Existing Item - *<Project>* dialog box, browse to and select the file that contains the item you want to add.
3. Click Add to create a copy of the selected file, or click the down arrow to the right of the Add button and click Add As Link to create a link to the file instead of creating a copy of the file.

Edit Items

You edit code and class modules, and forms in the common development environment. Code and class modules are edited in a Code window, while forms are edited in the Windows Form Designer.

You can open as many windows as you have items, allowing you to view the code in different forms or modules, and copy and paste between them.

Procedures

⊟ To edit an item in your project

- In Microsoft Visual Studio, Solution Explorer, double-click the item you want to edit.

  If you want to edit the code of a form or control on a form, while in the Windows Form Designer, double-click the form or control to display the Code window for the form.

Topics in this section

- Use the Code Window
- Use the Windows Form Designer
- Use the Properties Window

Use the Code Window

The Code window contains two drop-down lists, a split bar, and a margin indicator bar.

The two drop-down lists at the top of the Code window display the current object (class or form) on the left and the procedures for the current object on the right. You can move around your project by changing the object or procedure in these drop-down lists.

The splitter bar on the right side of the Code window allows you to split the window horizontally. Simply drag the splitter bar down to create a second window pane. This feature allows you to view two parts of your code simultaneously in the same code module. To close the pane, drag the splitter bar back to its original location or double-click it.

The margin indicator bar is located vertically along the left side of the Code window. It is used to display margin indicators that are used during code editing and debugging.

Current Object ————

Current Procedure

Margin
Indicator ————
Bar

```
Form1.vb [Design]   Start Page   AdskCommands.vb                    ▾ ✕

AdskCommands                    ▾   adskHelloWorld                  ▾

    Imports Autodesk.AutoCAD.Runtime
    Imports Autodesk.AutoCAD.ApplicationServices

  Public Class AdskCommands
        <CommandMethod("adskHelloWorld")> _
        Public Sub adskHelloWorld()
            Dim objEditor As Autodesk.AutoCAD.EditorInput.Editor
            objEditor = Application.DocumentManager.MdiActiveDocument.Editor

            objEditor.WriteMessage(vbLf & "Hello World!!!")
        End Sub

  End Class
```

Splitter
Bar

Use the Windows Form Designer

The Windows Form Designer allows you to create custom dialog boxes for your project.

To add a control to a form, click the desired control from the Toolbox window to activate the tool, and then click over the form in the Windows Form Designer to place and size the control. You can align controls with a grid or other controls by setting the appropriate setting form from the General category under the Windows Form Designer folder in the Options dialog box. The General category allows you to control other grid related settings such as size and layout mode.
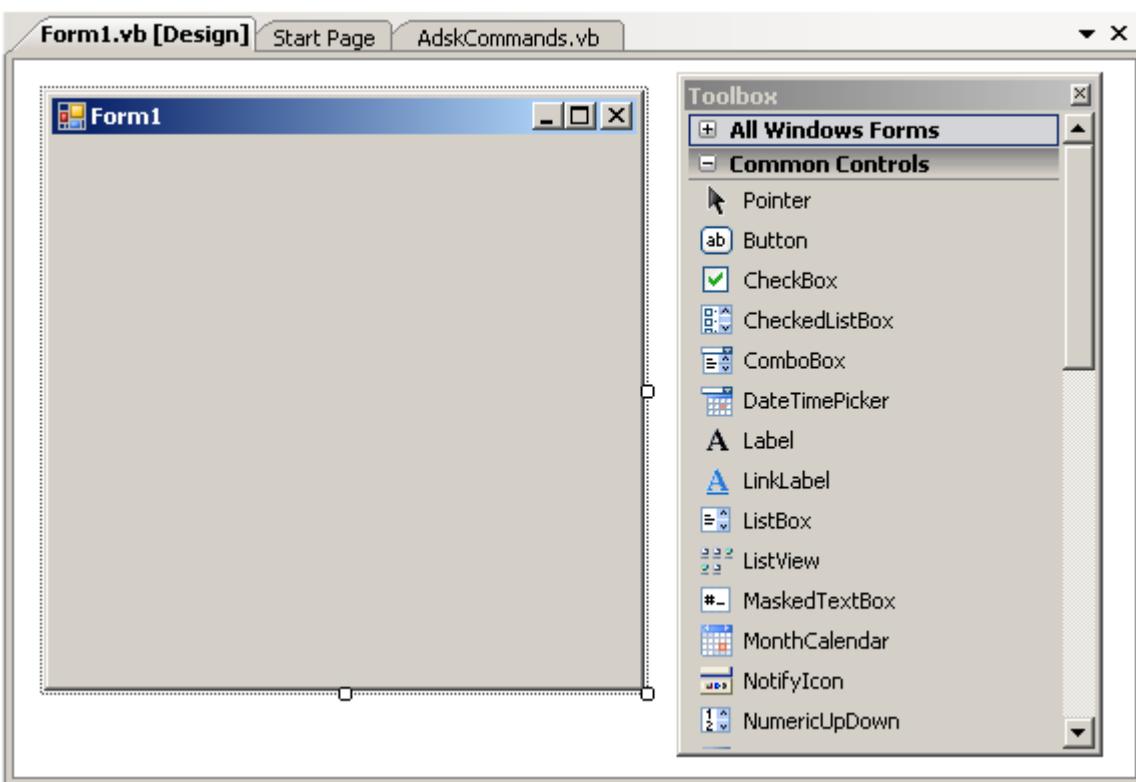
Each form that you design has the standard Maximize, Minimize, and Close buttons. These buttons are implemented as part of the Windows Form item template. You change the appearance and style of the dialog box by selecting the form object in the Windows Form Designer and then using the Properties window.

To add code to the form or a control event, double-click the form or control once it is placed on the form. This opens the Code window for form and adds the default event for the control. Use the right-most drop-down list at the top of the Code window to select a different event when you want to add it to the Code window for the form or control.

Procedures

⊟ To enable snap and set the size for the grid

1. In Microsoft Visual Studio, click Tools menu ➤ Options.
2. In the Options dialog box, expand the Windows Forms Designer folder node in the Options tree and select General.
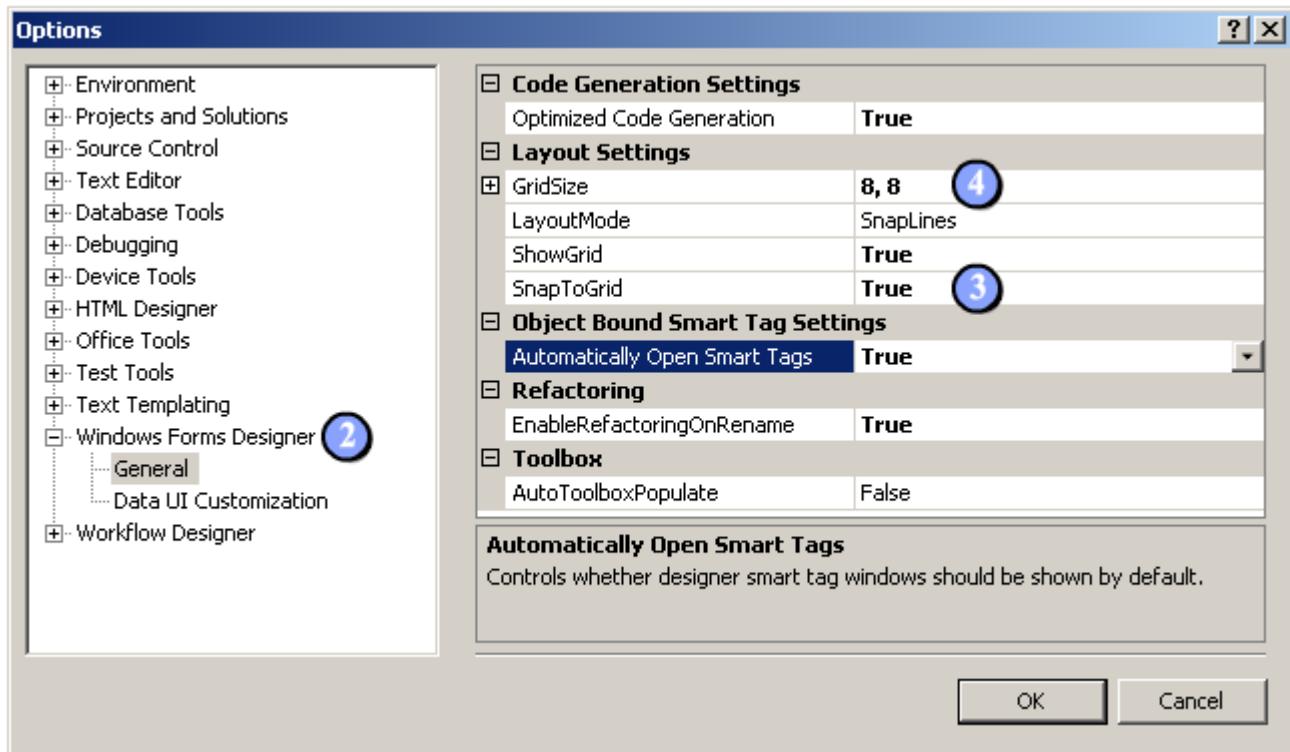
3. In the Properties pane, click SnapToGrid and select True from the drop-down list.
4. Click GridSize and then click in the box to the right of GridSize. Enter a grid spacing value using the format *width,height*.
5. Click OK.

Use the Properties Window

The Properties window is used to change the properties of files in the current project or solution, as well as Windows forms and controls that are placed on a Windows form.
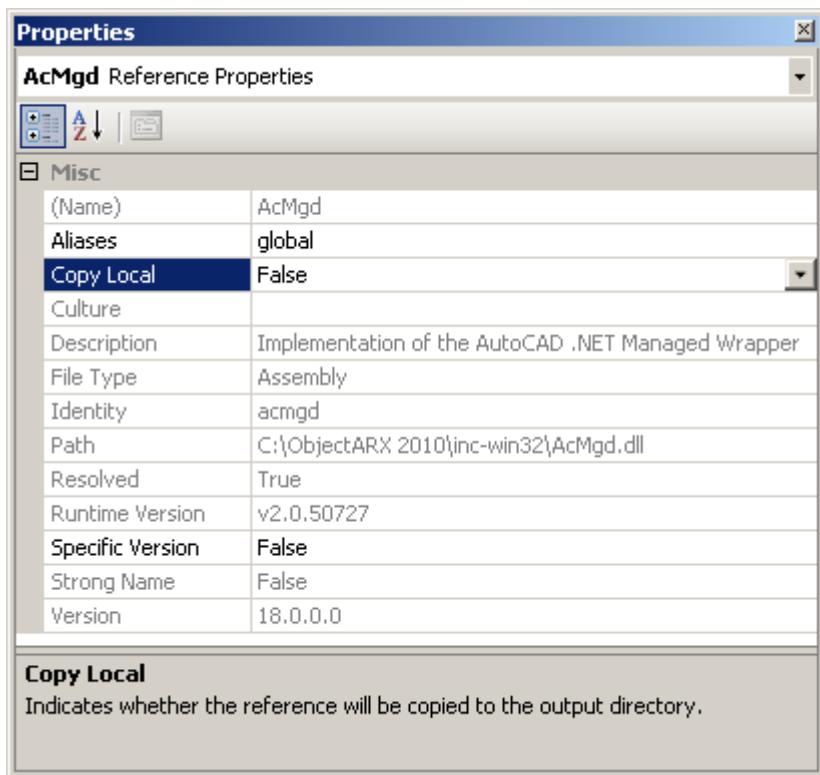
Select the file or object (Windows form or control) that you want to modify. This displays its properties in the Properties window. Select the property in the grid of the Properties window that you want to change, and then enter or select a new value based on the selected property.

Procedures

⊟ To change a property

1. In Microsoft Visual Studio, select a file or reference from the Solution Explorer, or an object (a Windows form in the Windows Form Designer or a control on a Windows form).
2. Display the Properties window if it is not already displayed.

    To display the Properties Window, in Microsoft Visual Studio, do one of the following:

    - o Click View menu ➤ Other Windows ➤ Properties Window
    - o Press Alt+Enter
    - o Right-click the selected file, reference, or object and click Properties
3. On the Properties window, click the property you want to edit and then enter or select a new value in the field to the right of the selected property name.

Rename a Project

When a project is created, the name of the project file is stored with the project. If the name of a project file is changed outside of Visual Studio, the stored project name will not be changed to match. As a best practice, only rename your project or solution from Visual

Studio to maintain consistency. For more information on creating a project, see Create a New Project.

You can rename a project file from the Properties window or through Windows Explorer. The project name is set when the project is first created, but can be renamed through the Solution Explorer.

WarningWhile a project file can be renamed through Windows Explorer, it is not recommended to do this. It can break the reference with the solution file or another project file. If you break the reference with a solution or another project, when you open the solution or project file, you will be notified that a reference is missing.

Procedures

□ To change the name of a project

1. In Microsoft Visual Studio, Solution Explorer, right-click the project you want to rename and click Rename.
2. In the In-place text editor, enter a new name.
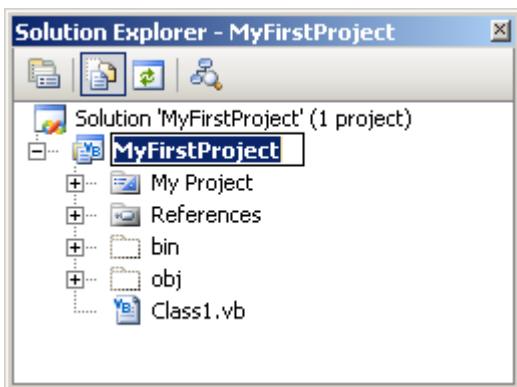
3. Press Enter or click outside the in-place text editor to finish renaming the project.

Add and Reference Other Projects

Adding and referencing projects allows you to share code across multiple projects. You can create centralized libraries of commonly used methods, functions and classes, and then reference the library when needed. Projects can be referenced in the following ways:

- Add the project to a solution and reference it to a project within the solution
- Reference a compiled assembly file to a project

When you add a project to a solution, a new node for the project is displayed in the Solution Explorer. This node is titled the same as the project that was referenced. After a project is added, it must be referenced to the project using the Add Reference dialog box.

If the project has been compiled into an assembly DLL file, you can reference the file to a project using the Add Reference dialog box. After a project or assembly DLL file is

referenced to a project, you use the Imports or using declaration with the namespace or project name to utilize any of the exposed objects (classes or forms) in your project.

Procedures

⊟ To reference another project

1. [Add a New Project or Existing Project to the current solution.](#)
2. Once the project has been added to the current solution, in the Solution Explorer, right-click the project that you want to add the project reference to and click Add Reference.
3. In the Add Reference dialog box, Projects tab, select the project you want to reference. Click OK.
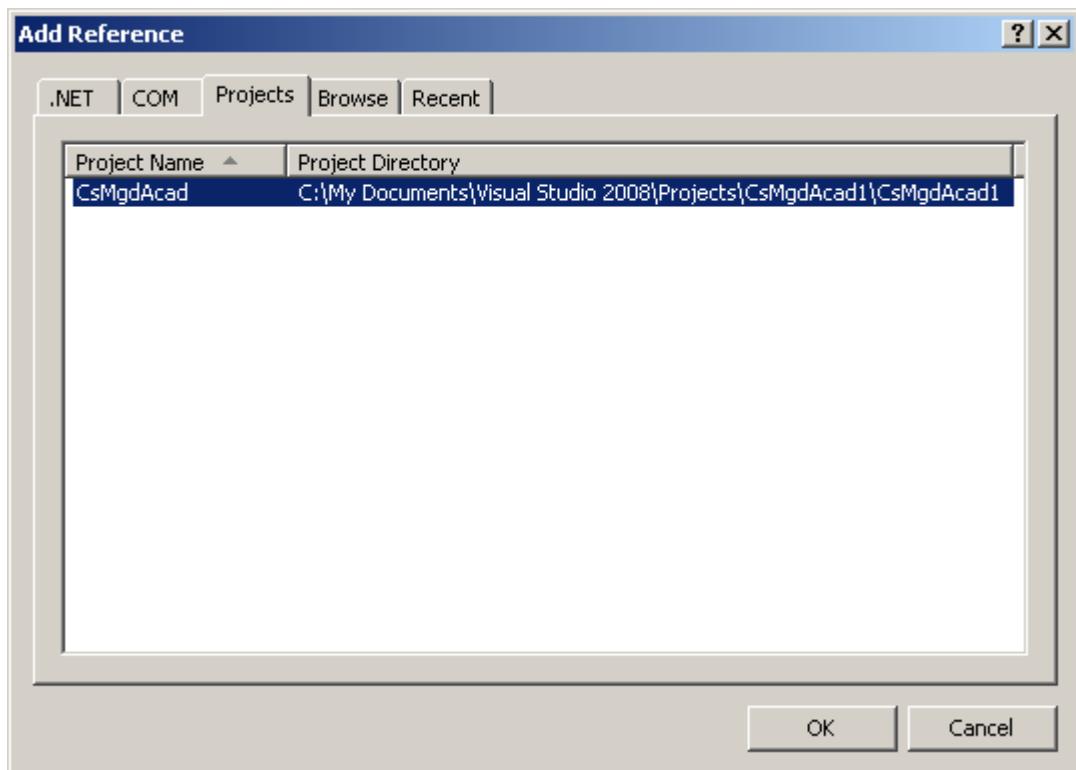
The selected project will appear under the References folder in the Solution Explorer.

4. In the Solution Explorer, double-click the code module that you want to use the publicly exposed functions, methods, or objects that are in the referenced project.
5. At the top of the code module, add an Imports or using declaration for the project or namespace in the project that contains the functions, methods, or objects you want to use.

For example, if a namespace containing an exposed class in the referenced project is named AdskUtilities, you would add the following to the code module to indicate that you are going to use that namespace from the referenced project:

VB.NET

```
Imports AdskUtilities
```

C#

```
using AdskUtilities;
```

6. Use the namespace in the same way you would any other .NET or COM library referenced to your project.

Set the Options for Microsoft Visual Studio

You can change the characteristics of the common development environment by using the Options dialog box.

The Options dialog box contains a variety of settings that affect the development environment, code editor windows, Windows Form Designer, and debugging. Settings are organized in folders that are available from the navigation pane on the left. Each folder contains groups that are used to organize related options. Once a folder or group is selected, the settings in that folder or group are displayed on the right side.

Some of the folders of settings you will work with are:

- *Environment*. Settings used to control the display and behavior of elements within the development environment.
- *Projects and Solutions*. Settings used to create and build projects and solutions.
- *Text Editor*. Settings used to control the behavior of text entered into a code window.
- *Debugging*. Settings used to control the debugging environment.
- *Windows Forms Designer*. Settings used to control the display and behavior of the Windows Form Designer.

For more information on all available settings in the Options dialog box, select a folder from the navigation pane in the Options dialog box and then click the '?' button in the upper-right corner. This displays the related topic in the Help file for Microsoft Visual Studio from Microsoft.

Procedures

To display the Options dialog box

- In Microsoft Visual Studio, click Tools menu ➤ Options.

Load an Assembly into AutoCAD

Once a solution and project are created, a namespace and class are defined, and one or more command or AutoLISP® function structures are implemented, you can use the NETLOAD command to load a .NET assembly into AutoCAD.

Use the Debug Environment

Prior to loading a .NET assembly, you should determine if you need to use the Debug environment of Microsoft Visual Studio to test any logic defined in the procedures and functions you might have created. The Debug enviornment allows you to step through the code in the .NET assembly as it is being executed in real-time. As the code is being executed, you are able to check the values of variables and watch which logic paths of the program are executed.

For more information on using the Debug environment, see the documentation that comes with your development environment.

Procedures

⊟ To load a .NET Assembly through Debug Mode in AutoCAD

1. In Microsoft Visual Studio, Solution Explorer, right-click the project you want to load into AutoCAD. Click Properties.
2. In the <Project> Properties tab, click the Debug tab.
3. On the Debug tab, under Start Action, click Start External Program and then click the ellipsis button to the right of the text box.
4. In the Select File dialog box, browse to *C:\Program Files\AutoCAD 2010* and select *acad.exe*. Click Open.
5. With the project selected in the Solution Explorer, click Debug menu > Start Debugging.
6. In AutoCAD, at the Command prompt, enter *netload* and press Enter.
7. In the Choose .NET Assembly dialog box, browse to the debug version of the assembly file. Click Open.

   TipThe location of the built assembly file is in the Output pane in Microsoft Visual Studio.

8. At the Command prompt, enter the name of the command or AutoLISP function and any required parameters.

⊟ To load a .NET assembly in AutoCAD with NETLOAD

1. In Microsoft Visual Studio, with a solution or project open, click Build menu ➤ Build Solution or Build *<Project name>*.
2. In AutoCAD, at the Command prompt, enter *netload* and press Enter.
3. In the Choose .NET Assembly dialog box, browse to the built assembly file. Click Open.

   TipThe location of the built assembly file is in the Output pane in Microsoft Visual Studio.

4. At the Command prompt, enter the name of the command or AutoLISP function and any required parameters.

Access and Search Referenced Libraries (Object Browser)

The Object Browser is used to view the objects of a library referenced by the projects in a current solution. From the Object Browser, you can navigate through a referenced library like you would a directory structure. You can also use the Search menu and locate an object based on an entered keyword.
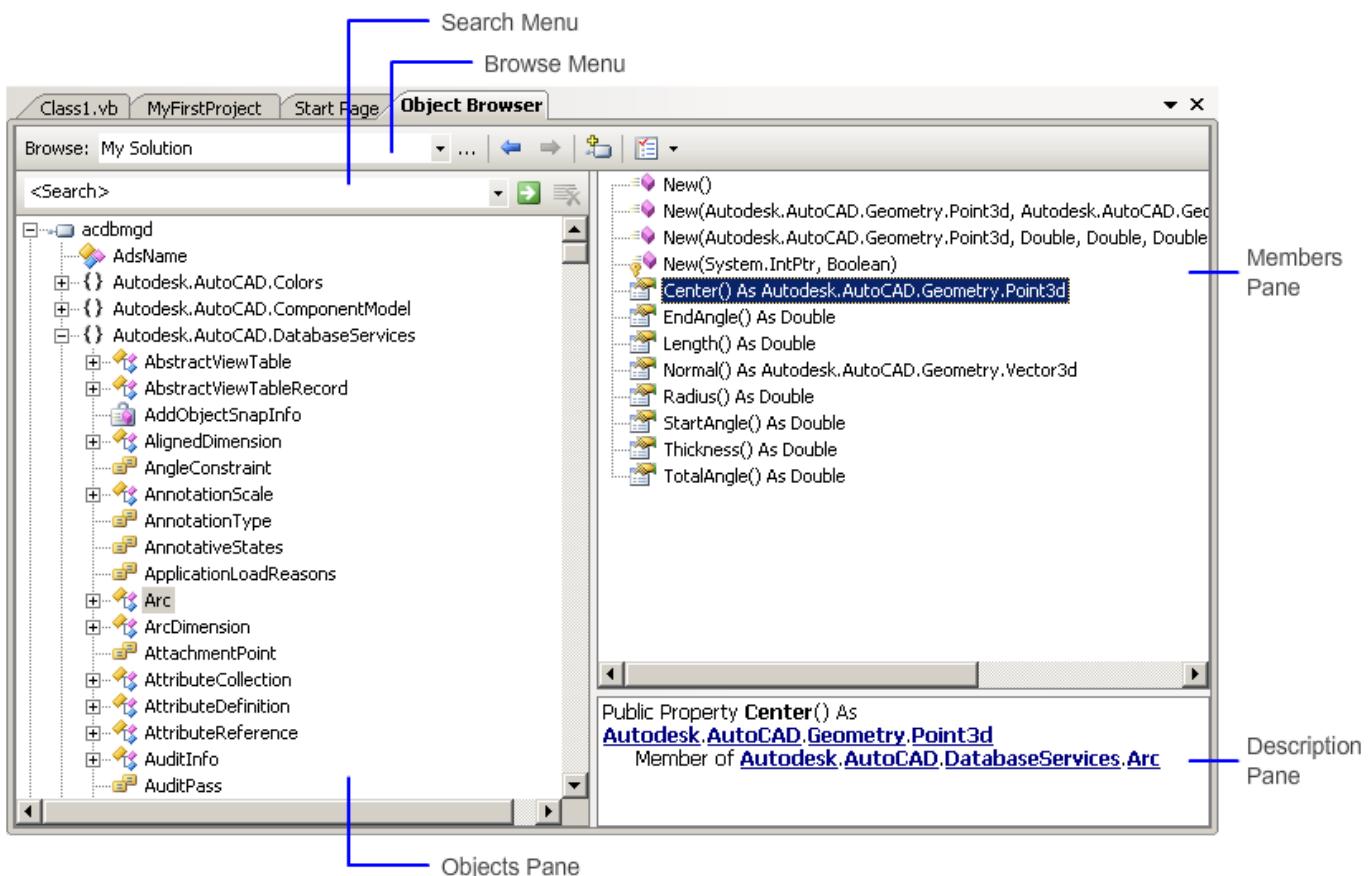
The scope of the libraries you view in the Object Browser is limited by the Browse menu drop-down list. You can limit the scope by the current solution, a specific release of the .NET Framework or a custom component list.

The left side of the Object Browser contains a navigation pane which displays the available libraries that you can navigate. The right side is divided into two halves: upper (Members pane) and lower (Description pane).

Procedures

⊟ To use the Object Browser

1. In Microsoft Visual Studio, click View menu ➤ Object Browser (or press Ctrl+Alt+J).

   The Object Browser should appear as a tab by default in the center of the development environment.

2. Optionally, from the Browse menu, select a scope in which you want to navigate or search for components.
3. Optionally, in the Search menu, enter or select a previous keyword that you want to filter the Objects pane by.
4. In the Objects pane, navigate to the object you want to view and select it.
5. In the Members pane, select one of the displayed members of the selected object to learn more about it.

Exercises: Create Your First Project

Now that you have learned the basics of using Microsoft Visual Studio, let's create a new project that defines a new command. Once the new command is defined, you will learn to load the .NET assembly into AutoCAD.

Topics in this section

Exercise: Create a New Project

In this exercise, you will create a new project named "MyFirstProject" and reference the files of the AutoCAD .NET API.

To create a new project named "MyFirstProject"

1. On the Start menu, click (All) Programs ➤ Microsoft Visual Studio 2008 ➤ Microsoft Visual Studio 2008 to start Microsoft Visual Studio.
2. In Microsoft Visual Studio, click File menu ➤ New ➤ Project.
3. In the New Project dialog box, Project Types tree, expand Other Languages and click Visual Basic or C#.
4. In the Templates list, select Class Library.
5. In the Name box, enter *MyFirstProject*.
6. In the Location box, click Browse to specify a new location or accept the default location.

   TipTo change the default location for new projects, display the Options dialog box and select the Projects and Solutions folder from the navigation pane. Click the ellipsis button to the right of the Visual Studio Projects Location and browse to a new default location.

7. Click OK.

Exercise: Reference the AutoCAD .NET API Files

In this exercise, you will reference the .NET assemblies *acmgd.dll* and *acdbmgd.dll*. After you reference the two files, you will adjust the properties of the referenced files so they are not copied to the build directory.

For more information on referencing AutoCAD .NET API files, see Components of the AutoCAD .NET API.

To reference the AutoCAD .NET API Files

1. In Microsoft Visual Studio, click View menu ➤ Solution Explorer to display the Solution Explorer if it is not already displayed.
2. In the Solution Explorer, on the toolbar along the top, click Show All Files.
3. Right-click the References node and click Add Reference.
4. In the Add Reference dialog box, Browse tab, browse to the install folder of AutoCAD and select *acmgd.dll*. Press and hold Ctrl, and then select *acdbmgd.dll*. Click OK.

   The default install location of AutoCAD is *<drive>:\Program Files\AutoCAD 2010*. If you installed the ObjectARX SDK, you should reference the files from the *inc-win32* or *inc-x64* folder.

5. In the Solution Explorer, click the plus sign to the left the References node to expand it.
6. Press and hold Ctrl, and select AcDbMgd and AcMdg from the References node.
7. Right-click over one of the selected references and click Properties.
8. In the Properties window, click the Copy Local field and then select False from the drop-down list.

   NoteSetting Copy Local to False instructs Microsoft Visual Studio to not include the referenced DLL in the build output for the project. If the referenced DLL is copied to the build output folder, it can cause unexpected results when you load your assembly file in AutoCAD.

Exercise: Create a New Command

Now that you have created a project and added the necessary library references, it is time to create a command. The command will create a new multiline text (MText) object in Model space. These and other concepts are discussed in-depth in later chapters.

To define a new command that creates a new MText object

1. In the Solution Explorer, double-click *Class1.vb* or *Class1.cs* based on the type of project you created.

   A code window is opened for the Class1 module and it should look like the following:

VB.NET

```vb
Public Class Class1

End Class
```

C#

```csharp
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

namespace MyFirstProject1
{
  public class Class1
  {
  }
}
```

2. Change the code in the code window to match the following:

VB.NET

```vb
Imports Autodesk.AutoCAD.Runtime
Imports Autodesk.AutoCAD.ApplicationServices
Imports Autodesk.AutoCAD.DatabaseServices

Public Class Class1
  <CommandMethod("AdskGreeting")> _
  Public Sub AdskGreeting()
      '' Get the current document and database, and start a transaction
      Dim acDoc As Document = Application.DocumentManager.MdiActiveDocument
      Dim acCurDb As Database = acDoc.Database

      Using acTrans As Transaction =
acCurDb.TransactionManager.StartTransaction()
          '' Open the Block table record for read
          Dim acBlkTbl As BlockTable
          acBlkTbl = acTrans.GetObject(acCurDb.BlockTableId, _
                                       OpenMode.ForRead)

          '' Open the Block table record Model space for write
          Dim acBlkTblRec As BlockTableRecord
          acBlkTblRec =
acTrans.GetObject(acBlkTbl(BlockTableRecord.ModelSpace), _
                                       OpenMode.ForWrite)

          '' Creates a new MText object and assigns it a location,
          '' text value and text style
          Dim objText As MText = New MText

          '' Set the default properties for the MText object
          objText.SetDatabaseDefaults()

          '' Specify the insertion point of the MText object
          objText.Location = New Autodesk.AutoCAD.Geometry.Point3d(2, 2, 0)

          '' Set the text string for the MText object
          objText.Contents = "Greetings, Welcome to the AutoCAD .NET
Developer's Guide"

          '' Set the text style for the MText object
```

```
        objText.TextStyleId = acCurDb.Textstyle

        '' Appends the new MText object to model space
        acBlkTblRec.AppendEntity(objText)

        '' Appends to new MText object to the active transaction
        acTrans.AddNewlyCreatedDBObject(objText, True)

        '' Saves the changes to the database and closes the transaction
        acTrans.Commit()
      End Using
    End Sub
End Class
```

C#

```csharp
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

using Autodesk.AutoCAD.Runtime;
using Autodesk.AutoCAD.ApplicationServices;
using Autodesk.AutoCAD.DatabaseServices;

[assembly: CommandClass(typeof(MyFirstProject1.Class1))]

namespace MyFirstProject1
{
  public class Class1
  {
      [CommandMethod("AdskGreeting")]
      public void AdskGreeting()
      {
          // Get the current document and database, and start a transaction
          Document acDoc = Application.DocumentManager.MdiActiveDocument;
          Database acCurDb = acDoc.Database;

          // Starts a new transaction with the Transaction Manager
          using (Transaction acTrans =
acCurDb.TransactionManager.StartTransaction())
          {
              // Open the Block table record for read
              BlockTable acBlkTbl;
              acBlkTbl = acTrans.GetObject(acCurDb.BlockTableId,
                                           OpenMode.ForRead) as BlockTable;

              // Open the Block table record Model space for write
              BlockTableRecord acBlkTblRec;
              acBlkTblRec =
acTrans.GetObject(acBlkTbl[BlockTableRecord.ModelSpace],
                                                  OpenMode.ForWrite) as
BlockTableRecord;

              /* Creates a new MText object and assigns it a location,
              text value and text style */
              MText objText = new MText();

              // Set the default properties for the MText object
              objText.SetDatabaseDefaults();

              // Specify the insertion point of the MText object
              objText.Location = new Autodesk.AutoCAD.Geometry.Point3d(2, 2,
0);
```

```
                // Set the text string for the MText object
                objText.Contents = "Greetings, Welcome to the AutoCAD .NET
Developer's Guide";

                // Set the text style for the MText object
                objText.TextStyleId = acCurDb.Textstyle;

                // Appends the new MText object to model space
                acBlkTblRec.AppendEntity(objText);

                // Appends to new MText object to the active transaction
                acTrans.AddNewlyCreatedDBObject(objText, true);

                // Saves the changes to the database and closes the
transaction
                acTrans.Commit();
            }
        }
    }
}
```

Exercise: Set the Target Framework for a Project

In this exercise, you will specify the target framework that should be used when the project is built.

To set the target framework for the "MyFirstProject"

1. In Microsoft Visual Studio, click View menu ➤ Solution Explorer to display the Solution Explorer if it is not already displayed.
2. In the Solution Explorer, on the toolbar along the top, click Show All Files.
3. Right-click the MyFirstProject project node and click Properties.
4. In the Project Designer, click the Applications tab.
5. Do one of the following:
   o *VB.NET* - On the Compile tab, Advanced Compile Options, Target Framework drop-down list, select .NET Framework 3.5.
   o *C#* - On the Applications tab, Target Framework drop-down list, select .NET Framework 3.5.

Exercise: Build and Load a .NET Assembly in AutoCAD

Now that you have a created a project and defined a command, you are almost ready to execute the command in AutoCAD. Before you can execute the command in AutoCAD, you need to first compile or build a .NET assembly file for your project.

For this exercise, you will be generating a Debug version of the project but you will not learn about using the Debug environment at this time. For more information on debugging a

project, see the documentation that comes with your development environment. To load use the Debug environment with AutoCAD, see [Load an Assembly into AutoCAD](#).

If you want to distribute your project for others to use, you will need to build a .NET assembly from your project. For more information on sharing your .NET assembly with others, see [Distribute Your Application](#).

Once a .NET assembly is built, you can then load it into AutoCAD with the NETLOAD command.

To build a project and load a .NET assembly into AutoCAD

1. In Microsoft Visual Studio, click Build menu ➤ Build MyFirstProject.

   The project should build successfully, unless something is wrong with the code in the project. Look at the Output window for information on the build status of the project. The location of the MyFirstProject.dll file that is built is also displayed in the Output window.

2. Start AutoCAD if it is not already running.
3. In AutoCAD, at the Command prompt, enter *netload* and press Enter.
4. In the Choose .NET Assembly dialog box, browse to the location of the MyFirstProject.dll that is displayed in the Output window in Microsoft Visual Studio and select it. Click Open.
5. At the Command prompt, enter *adskgreeting* and press Enter.

   A new MText object is created at the coordinates 2,2 with the text string "Greetings, Welcome to the AutoCAD .NET Developer's Guide".

Related AutoCAD Commands and Terminology

Commands
NETLOAD

Loads a .NET assembly into AutoCAD.

The Choose .NET Assembly dialog box, a standard file selection dialog box, is displayed.

When FILEDIA is set to 0 (zero), NETLOAD displays the following command prompt:

Assembly file name: *Enter a file name and press Enter.*

Terminology
Assembly

A compiled project which has the DLL file extension.

VB.NET Project

A project file created with Microsoft Visual Studio that has the VBPROJ file extension.

C# Project

A project file created with Microsoft Visual Studio that has the CSPROJ file extension.

Code Editor Window

Window used to edit the code stored in a class module or form.

Solution

File used to manage one or more project files loaded into Microsoft Visual Studio.

Reference

A link to an API library file that is used by a project. A project file can also be referenced to another project.

More Information

More information on the Microsoft Visual Studio development environment, and the VB.NET or C# programming languages, is available from the Help files provided by Microsoft. To access the Help files, choose one of the options from the Help menu in Microsoft Visual Studio. You might also consider purchasing a book published by Microsoft[®] Press or a third-party publisher.

3   Basics of the AutoCAD .NET API

To use the AutoCAD® .NET API effectively you should be familiar with the AutoCAD entities, objects, and features related to the tasks you want to automate. The greater your knowledge of an object's graphical and nongraphical properties, the easier it is for you to manipulate them through the AutoCAD .NET API. For information specific to the members (methods, functions or properties) of an object in the AutoCAD .NET API, see the AutoCAD .NET Reference Guide.

Topics in this section

- Understand the AutoCAD Object Hierarchy
- Access the Object Hierarchy
- Collection Objects
- Understand Properties and Methods
- Out-of-Process versus In-Process
- Define Commands and AutoLISP Functions

Understand the AutoCAD Object Hierarchy

An object is the main building block of the AutoCAD .NET API. Each exposed object represents a precise part of AutoCAD. There are many different types of objects in the AutoCAD .NET API. Some of the objects represented in the AutoCAD .NET API are:

- Graphical objects such as lines, arcs, text, and dimensions
- Style settings such as layers, linetypes, and dimension styles
- Organizational structures such as layers, groups, and blocks
- The drawing display such as view and viewport
- Even the drawing and the AutoCAD application

The objects are structured in a hierarchical fashion, with the AutoCAD Application object at the root. This hierarchical structure is often referred to as the Object Model. The following illustration shows the basic relationships between the Application object and an entity that is in a BlockTableRecord, such as Model space. There are many more objects in the AutoCAD .NET API that are not represented here.
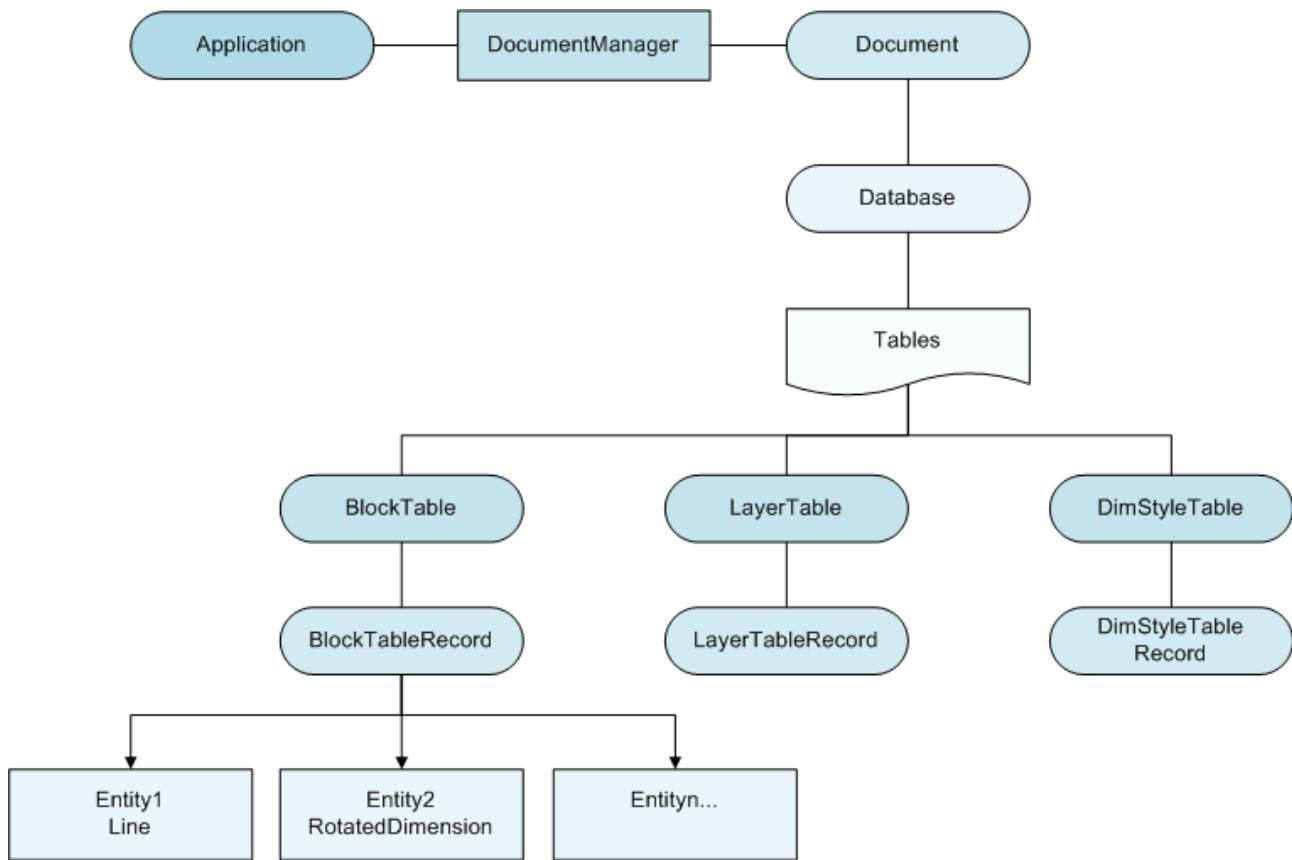
Topics in this section

- [The Application Object](#)
- [The Document Object](#)
- [The Database Object](#)
- [The Graphical and Nongraphical Objects](#)
- [The Collection Objects](#)
- [Non-Native Graphical and Nongraphical Objects](#)

The Application Object

The Application object is the root object of the AutoCAD .NET API. From the Application object, you can access the main window as well as any open drawing. Once you have a drawing, you can then access the objects in the drawing. For information on working with open drawing files (documents) see, The Document Object.

For example, the Application object has a DocumentManager property that returns the DocumentCollection object. This object provides access to the the drawings that are currently open in AutoCAD and allows you to create, save and open drawing files. Other properties of the Application object provide access to the application-specific data such as InfoCenter, the main window, and the status bar.The MainWindow property allows access to the application name, size, location, and visibility.

While most of the properties of the Application object allow access to objects in the AutoCAD .NET API, there are some that reference objects in the AutoCAD ActiveX$^®$ Automation.

These properties include a COM version of the application object (AcadApplication), the menubar (MenuBar), loaded menugroups (MenuGroups), and preferences (Preferences).

DocumentManager

Container for all the document objects (there is a document object for each drawing that is open).

DocumentWindowCollection

Container for all the document window objects (there is a document window object for each document object in the DocumentManager).

InfoCenter

Contains a reference to the InfoCenter toolbar.

MainWindow

Contains a reference to the application window object of AutoCAD.

MenuBar

Contains a reference to the MenuBar COM object for the menubar in AutoCAD.

MenuGroups

Contains a reference to the MenuGroups COM object which contains the customization group name for each loaded CUIx file.

Preferences

Contains a reference to the Preferences COM object which allows you to modify many of the settings in the Options dialog box.

Publisher

Contains a reference to the Publisher object which is used for publishing drawings.

StatusBar

Contains a reference to the StatusBar object for the application window.

UserConfiguration

Contains a reference to the UserConfiguration object

which allows you to work with user saved profiles.

The Document Object

The Document object, which is actually an AutoCAD drawing, is part of the DocumentCollection object and provides access to the Database object which is associated with the Document object. The Database object contains all of the graphical and most of the nongraphical AutoCAD objects. For more information on the Database object, see The Database Object.

Along with the Database object, the Document object provides access to the the drawing status bar, the window the document is opened in, the Editor and Transaction Manager objects. The Editor object provides access to functions used to obtain input from the user in the form of a point or an entered string or numeric value. For more information on requesting input from the user, see Prompt for User Input.

The Transaction Manager object is used to access multiple database objects under a single operation known as a *transaction*. Transactions can be nested, and when you are done with a transaction you can commit or abort the changes made. For more information on transactions and the Transaction Manager object, see Use Transactions with the Transaction Manager.
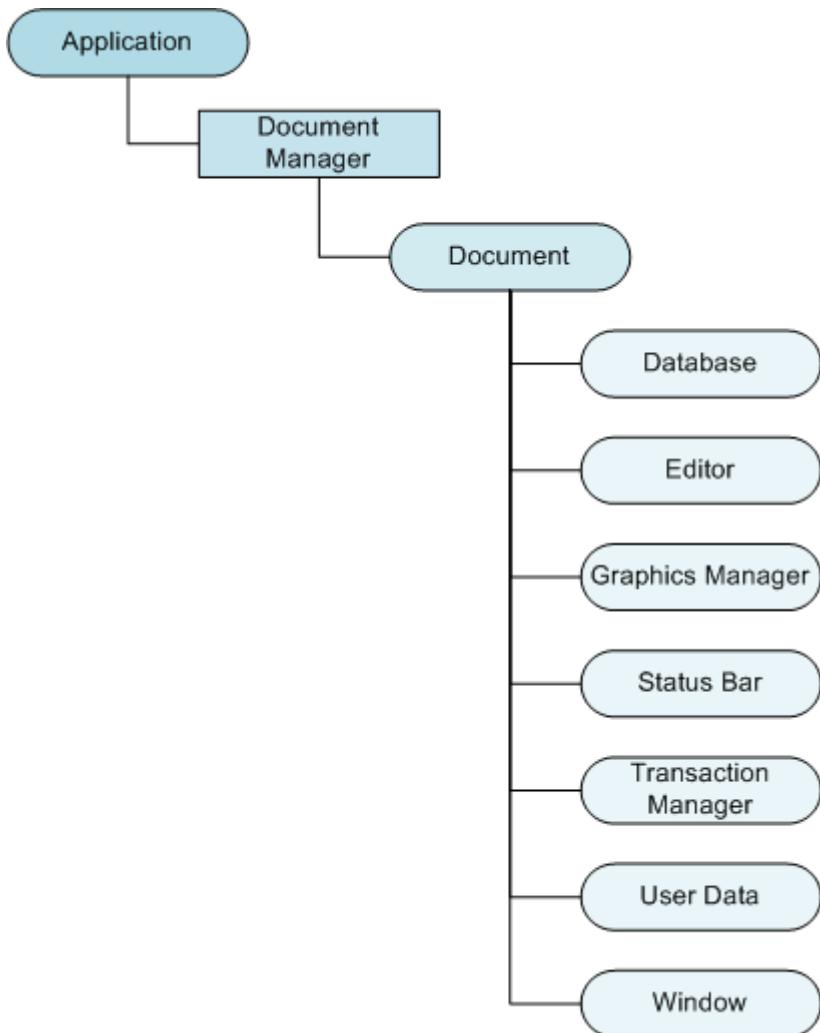
The Database Object

The Database object contains all of the graphical and most of the nongraphical AutoCAD objects. Some of the objects contained in the database are entities, symbol tables, and named dictionaries. Entities in the database represent graphical objects within a drawing. Lines, circles, arcs, text, hatch, and polylines are examples of entities. A user can see an entity on the screen and can manipulate it.

You access the Database object for the current document by the Document object's Database member property.

```
Application.DocumentManager.MdiActiveDocument.Database
```

Symbol Tables and Dictionaries

Symbol table and dictionary objects provide access to nongraphical objects (blocks, layers, linetypes, layouts, and so forth). Each drawing contains a set of nine fixed symbol tables, whereas the number of dictionaries in a drawing can vary based on the features and types of applications used in AutoCAD. New symbol tables cannot be added to a database.

Examples of symbol tables are the layer table (*LayerTable*), which contains layer table records, and the block table (*BlockTable*), which contains block table records. All graphical entities (lines, circles, arcs, and so forth) are owned by a block table record. By default, every drawing contains predefined block table records for Model and Paper space. Each Paper space layout has its own block table record. For more information on working with symbol tables, see Collection Objects.

A dictionary is a container object which can contain any AutoCAD object or an XRecord. Dictionaries are stored either in the database under the named object dictionary or as an extension dictionary of a table record or graphical entity. The named object dictionary is the master table for all of the dictionaries associated with a database. Unlike symbol tables, new dictionaries can be created and added to the named object dictionary. For more information on working with dictionaries, see Collection Objects.

NoteDictionary objects cannot contain drawing entities.

⊟ *VBA/ActiveX Cross Reference*

The Database object in the .NET API is similar to the Document object in the ActiveX Automation library. To access most of the properties that are available in the Document object of the ActiveX Automation library, you will need to work with the Document and Database objects of the .NET API. For more information on the Document object of the .NET API, see The Document Object.

The Graphical and Nongraphical Objects

Graphical objects, also known as entities, are the visible objects (lines, circles, raster images, and so forth) that make up a drawing. Adding graphical objects to a drawing is done by referencing the correct block table record, and then using the AppendEntity method with the new object to append it to the drawing.

To modify or query objects, obtain a reference to the object from the appropriate block table record, and then use the methods or properties of the object itself. Each graphical object has methods that perform most of the same functionality as the AutoCAD editing commands such as Copy, Erase, Move, Mirror, and so forth.

These objects also have methods to retrieve the extended data (xdata), highlight and unhighlight, and set the properties from another entity. Most graphical objects have some properties in common with each other such as LayerId, LinetypeId, Color, and Handle. Each graphical object also has specific properties, such as Center, StartPoint, Radius, and FitTolerance.

Nongraphical objects are the invisible (informational) objects that are part of a drawing, such as Layers, Linetypes, Dimension styles, Table styles, and so forth. To create a new symbol table records, use the Add method on the owner table or use the SetAt method to add a dictionary to the named object dictionary. To modify or query these objects, use the methods or properties of the object itself. Each nongraphical object has methods and properties specific to its purpose; all have methods to retrieve extended data (xdata), and erase themselves.
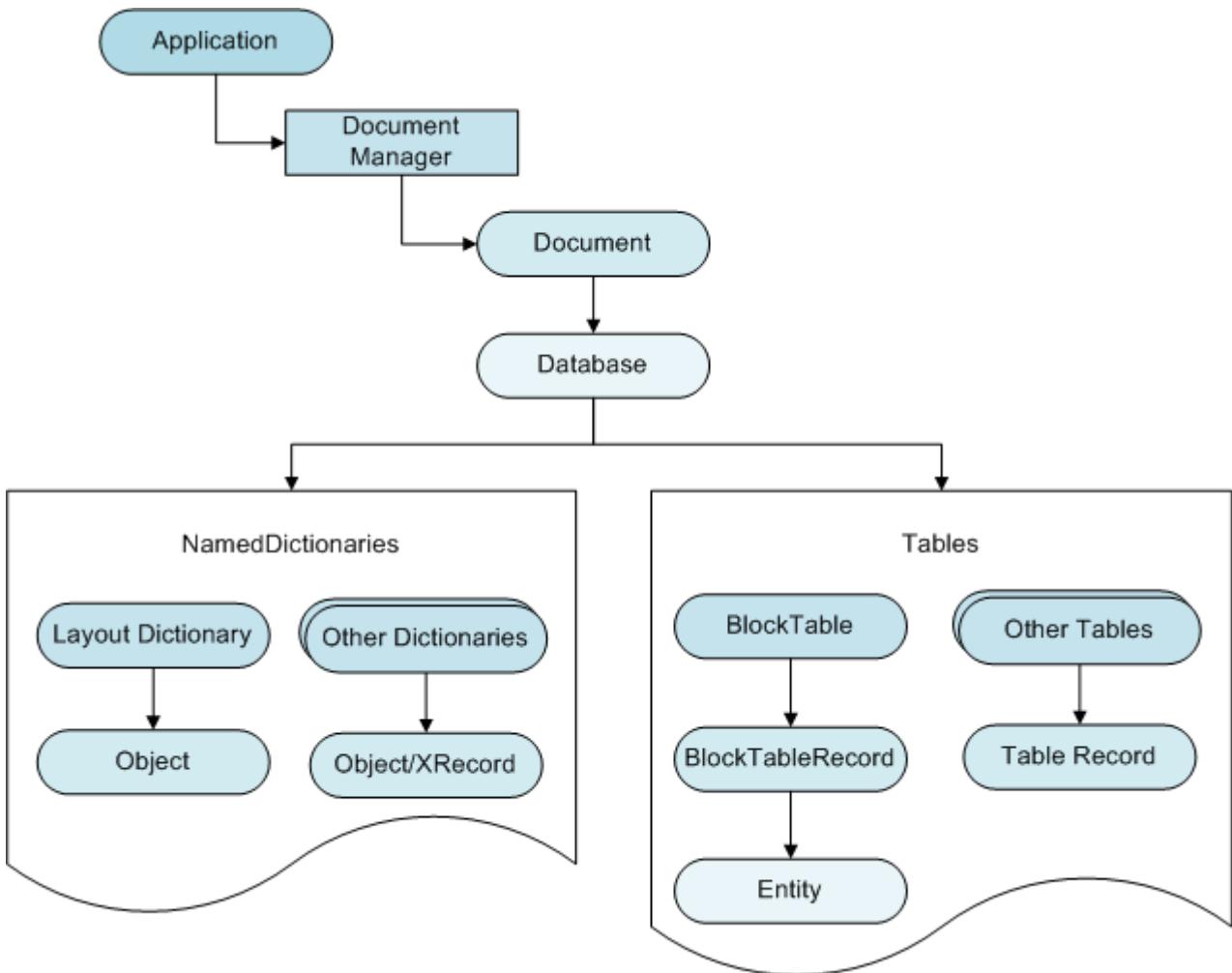
The Collection Objects

AutoCAD groups most graphical and nongraphical objects into collections or container objects. Although collections contain different types of data, they can be processed using similar techniques. Each collection has a method for adding an object to or obtaining an item from a collection. Most collections use the Add or SetAt methods to add an object to a collection.

Most collections offer similar methods and properties to make them easy to use and learn. The Count property returns a zero-based count of the objects in a collection, while the Item function returns an object from a collection. Examples of collection members in the AutoCAD .NET API are:

- Layer table record in the Layers symbol table
- Layout in the ACAD_LAYOUT dictionary
- Document in the DocumentCollection
- Attributes in a block reference

Non-Native Graphical and Nongraphical Objects

The AutoCAD .NET API is a cross implementation of ObjectARX and ActiveX Automation. While you can access ActiveX Automation from ObjectARX, .NET API makes working with the two rather seamless. As you work with objects using the native .NET API, you can access the equivalent COM object from a property. In some cases, the COM object is the only way to access an AutoCAD feature programmatically. Some examples of properties that expose COM objects through the .NET API are, Preferences, Menubar, MenuGroups, AcadObject and AcadApplication.

NoteWhen working with COM objects, you will want to make sure you reference the AutoCAD 2010 type library. For information on COM Interop, see Use COM Interoperability with .NET.

The Preferences property of the Application object provides access to a set of COM objects, each corresponding to a tab in the Options dialog box. Together, these objects provide access to all the registry-stored settings in the Options dialog box. You can also set and modify options (and system variables that are not part of the Options dialog box) with the SetSystemVariable and GetSystemVariable methods of the Application object. For more information about using the Preferences object, see the *ActiveX and VBA Developer's Guide*.

Accessing COM objects is useful if you are working with existing code that might have been originally developed for VB or VBA, or even when working with a third-party library that might work with the AutoCAD ActiveX Automation library with the .NET API. Like the Preferences object, you can also access utilities which translate coordinates or define a new point based on an angle and distance using the Utility object which can be accessed from the AcadApplication COM object which is the equivalent of the Application object in the .NET API.

NoteWhen working with both the AutoCAD .NET API and ActiveX Automation, and you create custom functions that might need to return an object, it is recommended to return an ObjectId instead of the object itself. For information on Object Ids, see Work with ObjectIds.

Access the Object Hierarchy

While the Application is the root object in the AutoCAD .NET API, you commonly will be working with the database of the current drawing. The DocumentManager property of the Application object allows you to access the current document with the MdiActiveDocument property. From the Document object returned by the MdiActiveDocument property, you can access its database with the Database property.

VB.NET

```
Application.DocumentManager.MdiActiveDocument.Database.Clayer
```

C#

```
Application.DocumentManager.MdiActiveDocument.Database.Clayer;
```

⊞ VBA/ActiveX Code Reference

Topics in this section

- [Reference Objects in the Object Hierarchy](#)
- [Access the Application Object](#)

Reference Objects in the Object Hierarchy

When you work with objects in the .NET API, you can reference some objects directly or through a user-defined variable based on the object you are working with. To reference an objects directly, include the object in the calling hierarchy. For example, the following statement attaches a drawing file to the database of the current drawing. Notice that the hierarchy starts with the Application and then goes to the Database object. From the Database object, the AttachXref method is called:

VB.NET

```
Dim strFName As String, strBlkName As String
Dim objId As Autodesk.AutoCAD.DatabaseServices.ObjectId

strFName = "c:\clients\Proj 123\grid.dwg"
strBlkName = System.IO.Path.GetFileNameWithoutExtension(strFName)

objId =
Application.DocumentManager.MdiActiveDocument.Database.AttachXref(strFName,
strBlkName)
```

C#

```
string strFName, strBlkName;
Autodesk.AutoCAD.DatabaseServices.ObjectId objId;

strFName = "c:/clients/Proj 123/grid.dwg";
strBlkName = System.IO.Path.GetFileNameWithoutExtension(strFName);
```

```
objId =
Application.DocumentManager.MdiActiveDocument.Database.AttachXref(strFName,
strBlkName);
```

To reference the objects through a user-defined variable, define the variable as the desired type, then set the variable to the appropriate object. For example, the following code defines a variable (acCurDb) of type Autodesk.AutoCAD.DatabaseServices.Database and sets the variable equal to the current database:

VB.NET
```
Dim acCurDb As Autodesk.AutoCAD.DatabaseServices.Database
acCurDb = Application.DocumentManager.MdiActiveDocument.Database
```

C#
```
Autodesk.AutoCAD.DatabaseServices.Database acCurDb;
acCurDb = Application.DocumentManager.MdiActiveDocument.Database;
```

The following statement then attaches a drawing file to the database using the acCurDb user-defined variable:

VB.NET
```
Dim strFName As String, strBlkName As String
Dim objId As Autodesk.AutoCAD.DatabaseServices.ObjectId

strFName = "c:\clients\Proj 123\grid.dwg"
strBlkName = System.IO.Path.GetFileNameWithoutExtension(strFName)

objId = acCurDb.AttachXref(strFName, strBlkName)
```

C#
```
string strFName, strBlkName;
Autodesk.AutoCAD.DatabaseServices.ObjectId objId;

strFName = "c:/clients/Proj 123/grid.dwg";
strBlkName = System.IO.Path.GetFileNameWithoutExtension(strFName);

objId = acCurDb.AttachXref(strFName, strBlkName);
```

Retrieve entities from Model space

The following example returns the first entity object in Model space. Similar code can do the same for Paper space entities. Note that all graphical objects are defined as an Entity object:

VB.NET
```
Imports Autodesk.AutoCAD.Runtime
Imports Autodesk.AutoCAD.ApplicationServices
Imports Autodesk.AutoCAD.DatabaseServices

<CommandMethod("ListEntities")> _
Public Sub ListEntities()
  '' Get the current document and database, and start a transaction
  Dim acDoc As Document = Application.DocumentManager.MdiActiveDocument
  Dim acCurDb As Database = acDoc.Database

  Using acTrans As Transaction = acCurDb.TransactionManager.StartTransaction()
      '' Open the Block table record for read
      Dim acBlkTbl As BlockTable
      acBlkTbl = acTrans.GetObject(acCurDb.BlockTableId, _
                                   OpenMode.ForRead)
```

```vb
        '' Open the Block table record Model space for read
        Dim acBlkTblRec As BlockTableRecord
        acBlkTblRec = acTrans.GetObject(acBlkTbl(BlockTableRecord.ModelSpace), _
                                        OpenMode.ForRead)

        Dim nCnt As Integer = 0
        acDoc.Editor.WriteMessage(vbLf & "Model space objects: ")

        '' Step through each object in Model space and
        '' display the type of object found
        For Each acObjId As ObjectId In acBlkTblRec
            acDoc.Editor.WriteMessage(vbLf & acObjId.ObjectClass().DxfName)

            nCnt = nCnt + 1
        Next

        '' If no objects are found then display the following message
        If nCnt = 0 Then
            acDoc.Editor.WriteMessage(vbLf & "  No objects found")
        End If

        '' Dispose of the transaction
    End Using
End Sub
```

C#

```csharp
using Autodesk.AutoCAD.Runtime;
using Autodesk.AutoCAD.ApplicationServices;
using Autodesk.AutoCAD.DatabaseServices;

[CommandMethod("ListEntities")]
public static void ListEntities()
{
  // Get the current document and database, and start a transaction
  Document acDoc = Application.DocumentManager.MdiActiveDocument;
  Database acCurDb = acDoc.Database;

  using (Transaction acTrans = acCurDb.TransactionManager.StartTransaction())
  {
      // Open the Block table record for read
      BlockTable acBlkTbl;
      acBlkTbl = acTrans.GetObject(acCurDb.BlockTableId,
                                   OpenMode.ForRead) as BlockTable;

      // Open the Block table record Model space for read
      BlockTableRecord acBlkTblRec;
      acBlkTblRec = acTrans.GetObject(acBlkTbl[BlockTableRecord.ModelSpace],
                                      OpenMode.ForRead) as BlockTableRecord;

      int nCnt = 0;
      acDoc.Editor.WriteMessage("\nModel space objects: ");

      // Step through each object in Model space and
      // display the type of object found
      foreach (ObjectId acObjId in acBlkTblRec)
      {
          acDoc.Editor.WriteMessage("\n" + acObjId.ObjectClass.DxfName);

          nCnt = nCnt + 1;
      }

      // If no objects are found then display a message
      if (nCnt == 0)
      {
```

```
            acDoc.Editor.WriteMessage("\n No objects found");
        }

        // Dispose of the transaction
    }
}
```

```
Sub ListEntities()
    ' This example returns the first entity in model space
    On Error Resume Next
    Dim entity As AcadEntity
    If ThisDrawing.ModelSpace.count <> 0 Then
        Set entity = ThisDrawing.ModelSpace.Item(0)
        MsgBox entity.ObjectName + _
                " is the first entity in model space."
    Else
        MsgBox "There are no objects in model space."
    End If
End Sub
```

Access the Application Object

The Application object is at the root of the object hierarchy and it provides access to the main window of AutoCAD. For example, the following line of code updates the application:

VB.NET

```
Autodesk.AutoCAD.ApplicationServices.Application.UpdateScreen()
```

C#

```
Autodesk.AutoCAD.ApplicationServices.Application.UpdateScreen();
```

```
ThisDrawing.Application.Update
```

Collection Objects

A collection is a type of object that contains many instances of similar objects. The following list contains some of the collection objects that are found in the AutoCAD .NET API:

Block Table Record

Contains all entities within a specific block definition.

Block Table

Contains all blocks in the drawing.

Named Objects Dictionary

Contains all dictionaries in the drawing.

Dimension Style Table

Contains all dimension styles in the drawing.

Document Collection

Contains all open drawings in the current session.

File Dependency Collection

Contains all items in the File Dependency List.

Group Dictionary

Contains all groups in the drawing.

Hyperlink Collection

Contains all hyperlinks for a given entity.

Layer Table

Contains all layers in the drawing.

Layout Dictionary

Contains all layouts in the drawing.

Linetype Table

Contains all linetypes in the drawing.

MenuBar Collection

Contains all menus currently displayed in AutoCAD.

MenuGroup Collection

Contains all customization groups currently loaded in AutoCAD. A customization group represents a loaded CUIx file which can contain menus, toolbars, and ribbon tabs among other elements that define the user interface.

Plot Configuration Dictionary

Contains named plot settings in the drawing.

Registered Application Table

Contains all registered applications in the drawing.

Text Style Table

Contains all text styles in the drawing.

UCS Table

Contains all user coordinate systems (UCS's) in the drawing.

View Table

Contains all views in the drawing.

Viewport Table

Contains all viewports in the drawing.

Topics in this section

Access a Collection

Most collection and container objects are accessed through the Document or Database objects. The Document and Database objects contain a property to access an object or object id for most of the available Collection objects. For example, the following code defines a variable and retrieves the LayersTable object which represents the collection of Layers in the current drawing:

VB.NET

```
'' Get the current document and start the Transaction Manager
Dim acCurDb As Database = Application.DocumentManager.MdiActiveDocument.Database
Using acTrans As Transaction = acCurDb.TransactionManager.StartTransaction()

    '' This example returns the layer table for the current database
    Dim acLyrTbl As LayerTable
    acLyrTbl = acTrans.GetObject(acCurDb.LayerTableId, _
                                 OpenMode.ForRead)

    '' Dispose of the transaction
End Using
```

C#

```
// Get the current document and start the Transaction Manager
Database acCurDb = Application.DocumentManager.MdiActiveDocument.Database;
using (Transaction acTrans = acCurDb.TransactionManager.StartTransaction())
{
    // This example returns the layer table for the current database
```

```
    LayerTable acLyrTbl;
    acLyrTbl = acTrans.GetObject(acCurDb.LayerTableId,
                                 OpenMode.ForRead) as LayerTable;

     // Dispose of the transaction
}
```

```
Dim layerCollection as AcadLayers
Set layerCollection = ThisDrawing.Layers
```

Add a New Member to a Collection Object

To add a new member to the collection, use the Add method. For example, the following code creates a new layer and adds it to the Layer table:

VB.NET

```
Imports Autodesk.AutoCAD.Runtime
Imports Autodesk.AutoCAD.ApplicationServices
Imports Autodesk.AutoCAD.DatabaseServices

<CommandMethod("AddMyLayer")> _
Public Sub AddMyLayer()
  '' Get the current document and database, and start a transaction
  Dim acDoc As Document = Application.DocumentManager.MdiActiveDocument
  Dim acCurDb As Database = acDoc.Database

  Using acTrans As Transaction = acCurDb.TransactionManager.StartTransaction()
      '' Returns the layer table for the current database
      Dim acLyrTbl As LayerTable
      acLyrTbl = acTrans.GetObject(acCurDb.LayerTableId, _
                                   OpenMode.ForRead)

      '' Check to see if MyLayer exists in the Layer table
      If Not acLyrTbl.Has("MyLayer") Then
          '' Open the Layer Table for write
          acLyrTbl.UpgradeOpen()

          '' Create a new layer table record and name the layer "MyLayer"
          Dim acLyrTblRec As LayerTableRecord = New LayerTableRecord
          acLyrTblRec.Name = "MyLayer"

          '' Add the new layer table record to the layer table and the transaction
          acLyrTbl.Add(acLyrTblRec)
          acTrans.AddNewlyCreatedDBObject(acLyrTblRec, True)

          '' Commit the changes
          acTrans.Commit()
      End If

      '' Dispose of the transaction
  End Using
End Sub
```

C#

```
using Autodesk.AutoCAD.Runtime;
```

```
using Autodesk.AutoCAD.ApplicationServices;
using Autodesk.AutoCAD.DatabaseServices;

[CommandMethod("AddMyLayer")]
public static void AddMyLayer()
{
  // Get the current document and database, and start a transaction
  Document acDoc = Application.DocumentManager.MdiActiveDocument;
  Database acCurDb = acDoc.Database;

  using (Transaction acTrans = acCurDb.TransactionManager.StartTransaction())
  {
      // Returns the layer table for the current database
      LayerTable acLyrTbl;
      acLyrTbl = acTrans.GetObject(acCurDb.LayerTableId,
                                   OpenMode.ForRead) as LayerTable;

      // Check to see if MyLayer exists in the Layer table
      if (acLyrTbl.Has("MyLayer") != true)
      {
          // Open the Layer Table for write
          acLyrTbl.UpgradeOpen();

          // Create a new layer table record and name the layer "MyLayer"
          LayerTableRecord acLyrTblRec = new LayerTableRecord();
          acLyrTblRec.Name = "MyLayer";

          // Add the new layer table record to the layer table and the transaction
          acLyrTbl.Add(acLyrTblRec);
          acTrans.AddNewlyCreatedDBObject(acLyrTblRec, true);

          // Commit the changes
          acTrans.Commit();
      }

      // Dispose of the transaction
  }
}
```

⊟ *VBA/ActiveX Code Reference*

```
Sub AddMyLayer()
    Dim newLayer as AcadLayer
    Set newLayer = ThisDrawing.Layers.Add("MyLayer")
End Sub
```

Iterate through a Collection Object

To select a specific member of a Collection object, use the Item or GetAt method. The Item and GetAt methods require a key in the form of a string in which represents the name of the item. With most collections, the Item method is implied, meaning you do not actually need to use method.

With some Collection objects, you can also use an index number to specify the location of the item within the collection you want to retrieve. The method you can use varies based on the language you are using as well as if you are working with a symbol table or dictionary.

The following statements show how to access the "MyLayer" table record in Layer symbol table.

VB.NET

```
acObjId = acLyrTbl.Item("MyLayer")

acObjId = acLyrTbl("MyLayer")
```

C#

```
acObjId = acLyrTbl["MyLayer"];
```

⊟ *VBA/ActiveX Code Reference*

```
acLayer = ThisDrawing.Layers.Item("MyLayer")

acLayer = ThisDrawing.Layers("MyLayer")
```

Iterate through the LayerTable object

The following example iterates through the LayerTable object and displays the names of all its layer table records:

VB.NET

```
Imports Autodesk.AutoCAD.Runtime
Imports Autodesk.AutoCAD.ApplicationServices
Imports Autodesk.AutoCAD.DatabaseServices

<CommandMethod("IterateLayers")> _
Public Sub IterateLayers()
  '' Get the current document and database, and start a transaction
  Dim acDoc As Document = Application.DocumentManager.MdiActiveDocument
  Dim acCurDb As Database = acDoc.Database

  Using acTrans As Transaction = acCurDb.TransactionManager.StartTransaction()
      '' This example returns the layer table for the current database
      Dim acLyrTbl As LayerTable
      acLyrTbl = acTrans.GetObject(acCurDb.LayerTableId, _
                                   OpenMode.ForRead)

      '' Step through the Layer table and print each layer name
      For Each acObjId As ObjectId In acLyrTbl
          Dim acLyrTblRec As LayerTableRecord
          acLyrTblRec = acTrans.GetObject(acObjId, OpenMode.ForRead)

          acDoc.Editor.WriteMessage(vbLf & acLyrTblRec.Name)
      Next

      '' Dispose of the transaction
  End Using
End Sub
```

C#

```
using Autodesk.AutoCAD.Runtime;
using Autodesk.AutoCAD.ApplicationServices;
using Autodesk.AutoCAD.DatabaseServices;

[CommandMethod("IterateLayers")]
public static void IterateLayers()
```

```
{
  // Get the current document and database, and start a transaction
  Document acDoc = Application.DocumentManager.MdiActiveDocument;
  Database acCurDb = acDoc.Database;

  using (Transaction acTrans = acCurDb.TransactionManager.StartTransaction())
  {
      // This example returns the layer table for the current database
      LayerTable acLyrTbl;
      acLyrTbl = acTrans.GetObject(acCurDb.LayerTableId,
                                   OpenMode.ForRead) as LayerTable;

      // Step through the Layer table and print each layer name
      foreach (ObjectId acObjId in acLyrTbl)
      {
          LayerTableRecord acLyrTblRec;
          acLyrTblRec = acTrans.GetObject(acObjId,
                                          OpenMode.ForRead) as LayerTableRecord;

          acDoc.Editor.WriteMessage("\n" + acLyrTblRec.Name);
      }

      // Dispose of the transaction
  }
}
```

VBA/ActiveX Code Reference

```
Sub IterateLayers()
    ' Iterate through the collection
    On Error Resume Next

    Dim lay As AcadLayer
    Dim msg As String
    msg = ""
    For Each lay In ThisDrawing.Layers
        msg = msg + lay.Name + vbCrLf
    Next

    ThisDrawing.Utility.prompt msg
End Sub
```

Find the layer table record named MyLayer in the LayerTable object

The following example checks the LayerTable object to determine if the layer named MyLayer exists or not, and displays the appropriate message:

VB.NET

```
Imports Autodesk.AutoCAD.Runtime
Imports Autodesk.AutoCAD.ApplicationServices
Imports Autodesk.AutoCAD.DatabaseServices

<CommandMethod("FindMyLayer")> _
Public Sub FindMyLayer()
  '' Get the current document and database, and start a transaction
  Dim acDoc As Document = Application.DocumentManager.MdiActiveDocument
  Dim acCurDb As Database = acDoc.Database

  Using acTrans As Transaction = acCurDb.TransactionManager.StartTransaction()
```

```
        '' Returns the layer table for the current database
        Dim acLyrTbl As LayerTable
        acLyrTbl = acTrans.GetObject(acCurDb.LayerTableId, _
                                     OpenMode.ForRead)

        '' Check to see if MyLayer exists in the Layer table
        If Not acLyrTbl.Has("MyLayer") Then
            acDoc.Editor.WriteMessage(vbLf & "'MyLayer' does not exist")
        Else
            acDoc.Editor.WriteMessage(vbLf & "'MyLayer' exists")
        End If

        '' Dispose of the transaction
    End Using
End Sub
```

C#

```
using Autodesk.AutoCAD.Runtime;
using Autodesk.AutoCAD.ApplicationServices;
using Autodesk.AutoCAD.DatabaseServices;

[CommandMethod("FindMyLayer")]
public static void FindMyLayer()
{
  // Get the current document and database, and start a transaction
  Document acDoc = Application.DocumentManager.MdiActiveDocument;
  Database acCurDb = acDoc.Database;

  using (Transaction acTrans = acCurDb.TransactionManager.StartTransaction())
  {
      // Returns the layer table for the current database
      LayerTable acLyrTbl;
      acLyrTbl = acTrans.GetObject(acCurDb.LayerTableId,
                                   OpenMode.ForRead) as LayerTable;

      // Check to see if MyLayer exists in the Layer table
      if (acLyrTbl.Has("MyLayer") != true)
      {
          acDoc.Editor.WriteMessage("\n'MyLayer' does not exist");
      }
      else
      {
          acDoc.Editor.WriteMessage("\n'MyLayer' exists");
      }

      // Dispose of the transaction
  }
}
```

⊟ *VBA/ActiveX Code Reference*

```
Sub FindMyLayer()
    ' Use the Item method to find a layer named MyLayer
    On Error Resume Next
    Dim ABCLayer As AcadLayer
    Set ABCLayer = ThisDrawing.Layers("MyLayer")
    If Err <> 0 Then
        ThisDrawing.Utility.prompt "'MyLayer' does not exist"
    Else
        ThisDrawing.Utility.prompt "'MyLayer' exists"
    End If
```

```
End Sub
```

Erase a Member of a Collection Object

Members from a collection object can be erased using the Erase method found on the member object. For example, the following code erases the layer MyLayer from the LayerTable object.

Before you erase a layer from a drawing, you should make sure it can be safely removed. To determine if a layer or another named object such as a block or text style can be erased, you should use the Purge method. For information on the Purge method, see Purge Unreferenced Named Objects.

VB.NET

```
Imports Autodesk.AutoCAD.Runtime
Imports Autodesk.AutoCAD.ApplicationServices
Imports Autodesk.AutoCAD.DatabaseServices

<CommandMethod("RemoveMyLayer")> _
Public Sub RemoveMyLayer()
  '' Get the current document and database, and start a transaction
  Dim acDoc As Document = Application.DocumentManager.MdiActiveDocument
  Dim acCurDb As Database = acDoc.Database

  Using acTrans As Transaction = acCurDb.TransactionManager.StartTransaction()
      '' Returns the layer table for the current database
      Dim acLyrTbl As LayerTable
      acLyrTbl = acTrans.GetObject(acCurDb.LayerTableId, _
                                   OpenMode.ForRead)

      '' Check to see if MyLayer exists in the Layer table
      If acLyrTbl.Has("MyLayer") = True Then
          Dim acLyrTblRec As LayerTableRecord
          acLyrTblRec = acTrans.GetObject(acLyrTbl("MyLayer"), _
                                          OpenMode.ForWrite)

          Try
              acLyrTblRec.Erase()
              acDoc.Editor.WriteMessage(vbLf & "'MyLayer' was erased")

              '' Commit the changes
              acTrans.Commit()
          Catch
              acDoc.Editor.WriteMessage(vbLf & "'MyLayer' could not be erased")
          End Try
      Else
          acDoc.Editor.WriteMessage(vbLf & "'MyLayer' does not exist")
      End If

      '' Dispose of the transaction
  End Using
End Sub
```

C#

```csharp
using Autodesk.AutoCAD.Runtime;
using Autodesk.AutoCAD.ApplicationServices;
using Autodesk.AutoCAD.DatabaseServices;

[CommandMethod("RemoveMyLayer")]
public static void RemoveMyLayer()
{
  // Get the current document and database, and start a transaction
  Document acDoc = Application.DocumentManager.MdiActiveDocument;
  Database acCurDb = acDoc.Database;

  using (Transaction acTrans = acCurDb.TransactionManager.StartTransaction())
  {
      // Returns the layer table for the current database
      LayerTable acLyrTbl;
      acLyrTbl = acTrans.GetObject(acCurDb.LayerTableId,
                                   OpenMode.ForRead) as LayerTable;

      // Check to see if MyLayer exists in the Layer table
      if (acLyrTbl.Has("MyLayer") == true)
      {
          LayerTableRecord acLyrTblRec;
          acLyrTblRec = acTrans.GetObject(acLyrTbl["MyLayer"],
                                          OpenMode.ForWrite) as LayerTableRecord;

          try
          {
              acLyrTblRec.Erase();
              acDoc.Editor.WriteMessage("\n'MyLayer' was erased");

              // Commit the changes
              acTrans.Commit();
          }
          catch
          {
              acDoc.Editor.WriteMessage("\n'MyLayer' could not be erased");
          }
      }
      else
      {
          acDoc.Editor.WriteMessage("\n'MyLayer' does not exist");
      }

      // Dispose of the transaction
  }
}
```

VBA/ActiveX Code Reference

```vba
Sub RemoveMyLayer()
  On Error Resume Next

  '' Get the layer "MyLayer" from the Layers collection
  Dim ABCLayer As AcadLayer
  Set ABCLayer = ThisDrawing.Layers.Item("MyLayer")

  '' Check for an error, if no error occurs the layer exists
  If Err = 0 Then

    '' Delete the layer
    ABCLayer.Delete

    '' Clear the current error
```

```
      Err.Clear

    '' Get the layer again if it is found the layer could not be removed
    Set ABCLayer = ThisDrawing.Layers.Item("MyLayer")

    '' Check for error, if an error is encountered the layer was removed
    If Err <> 0 Then
      ThisDrawing.Utility.prompt "'MyLayer' was removed"
    Else
      ThisDrawing.Utility.prompt "'MyLayer' could not be removed"
    End If
  Else
    ThisDrawing.Utility.prompt "'MyLayer' does not exist"
  End If
End Sub
```

Once an object has been erased, you should not attempt to access the object again later in the program; otherwise an error will occur. The above sample tests to see if the object exists before it is accessed again. When a request to erase an object is made, you should check to see if the object exists with the Has method or use a Try statement to catch any exceptions that occur. For more information on handling exceptions, see Handle Errors.

Understand Properties and Methods

Each object has associated properties and methods. Properties describe aspects of the individual object, while methods are actions that can be performed on the individual object. Once an object is created, you can query and edit the object through its properties and methods.

For example, a Circle object has a Center property. This property represents the point in the world coordinate system (WCS) at the center of that circle. To change the center of the circle, simply set the Center property to a new point. The Circle object also has a method called GetOffsetCurves. This method creates a new object at a specified offset distance from the existing circle.

To see a list of all properties and methods for the Circle object, refer to the Circle object in the *AutoCAD .NET Reference Guide* or use the Object Browser in Microsoft® Visual Studio®. For more information on the Object Browser in Microsoft Visual Studio, see Access and Search Referenced Libraries (Object Browser).

Out-of-Process versus In-Process

When you develop a new application, it can either run in or out-of-process. The AutoCAD .NET API is designed to run in-process only, which is different from the ActiveX Automation library which can be used in or -out-of-process.

- In-process applications are designed to run in the same process space as the host application. In this case, a DLL assembly is loaded into AutoCAD which is the host application.
- Out-of-process applications do not run in the same space as the host application. These applications are often built as stand-alone executables.

If you need to create a stand-alone application to drive AutoCAD, it is best to create an application that uses the CreateObject and GetObject methods to create a new instance of an AutoCAD application or return one of the instances that is currently running. Once a reference to an AcadApplication is returned, you can then load your in-process .NET application into AutoCAD by using the SendCommand method that is a member of the ActiveDocument property of the AcadApplication.

As an alternative to executing your .NET application in-process, could use COM interop for your application.

NoteThe ProgID for COM application access to AutoCAD 2010 is *AutoCAD.Application.18*.

VB.NET

```
Imports System
Imports System.Runtime.InteropServices

Imports Autodesk.AutoCAD.Interop
Imports Autodesk.AutoCAD.Runtime
Imports Autodesk.AutoCAD.ApplicationServices

<CommandMethod("ConnectToAcad")> _
Public Sub ConnectToAcad()
  Dim acAppComObj As AcadApplication
  Dim strProgId As String = "AutoCAD.Application.18"

  On Error Resume Next

  '' Get a running instance of AutoCAD
  acAppComObj = GetObject(, strProgId)

  '' An error occurs if no instance is running
  If Err.Number > 0 Then
     Err.Clear()

     '' Create a new instance of AutoCAD
     acAppComObj = CreateObject("AutoCAD.Application.18")

     '' Check to see if an instance of AutoCAD was created
     If Err.Number > 0 Then
        Err.Clear()

        '' If an instance of AutoCAD is not created then message and exit
        MsgBox("Instance of 'AutoCAD.Application' could not be created.")

        Exit Sub
     End If
  End If

  '' Display the application and return the name and version
  acAppComObj.Visible = True
  MsgBox("Now running " & acAppComObj.Name & " version " & acAppComObj.Version)

  '' Get the active document
  Dim acDocComObj As AcadDocument
```

```
   acDocComObj = acAppComObj.ActiveDocument

   '' Optionally, load your assembly and start your command or if your assembly
   '' is demandloaded, simply start the command of your in-process assembly.
   acDocComObj.SendCommand("(command " & Chr(34) & "NETLOAD" & Chr(34) & " " & _
                           Chr(34) & "c:/myapps/mycommands.dll" & Chr(34) & ") ")

   acDocComObj.SendCommand("MyCommand ")
End Sub
```

C#

```
using System;
using System.Runtime.InteropServices;

using Autodesk.AutoCAD.Interop;
using Autodesk.AutoCAD.Runtime;
using Autodesk.AutoCAD.ApplicationServices;

[CommandMethod("ConnectToAcad")]
public static void ConnectToAcad()
{

   AcadApplication acAppComObj = null;
   const string strProgId = "AutoCAD.Application.18";

   // Get a running instance of AutoCAD
   try
   {
       acAppComObj = (AcadApplication)Marshal.GetActiveObject(strProgId);
   }
   catch // An error occurs if no instance is running
   {
       try
       {
           // Create a new instance of AutoCAD
           acAppComObj =
(AcadApplication)Activator.CreateInstance(Type.GetTypeFromProgID(strProgId),
true);
       }
       catch
       {
           // If an instance of AutoCAD is not created then message and exit
           System.Windows.Forms.MessageBox.Show("Instance of 'AutoCAD.Application'"
+
                                               " could not be created.");

           return;
       }
   }

   // Display the application and return the name and version
   acAppComObj.Visible = true;
   System.Windows.Forms.MessageBox.Show("Now running " + acAppComObj.Name +
                                        " version " + acAppComObj.Version);

   // Get the active document
   AcadDocument acDocComObj;
   acDocComObj = acAppComObj.ActiveDocument;

   // Optionally, load your assembly and start your command or if your assembly
   // is demandloaded, simply start the command of your in-process assembly.
   acDocComObj.SendCommand("(command " + (char)34 + "NETLOAD" + (char)34 + " " +
```

```
                             (char)34 + "c:/myapps/mycommands.dll" + (char)34 + ")
");

    acDocComObj.SendCommand("MyCommand ");
}
```

```
Sub ConnectToAcad()
    Dim acadApp As AcadApplication
    On Error Resume Next

    Set acadApp = GetObject(, "AutoCAD.Application.18")
    If Err Then
        Err.Clear
        Set acadApp = CreateObject("AutoCAD.Application.18")
        If Err Then
            MsgBox Err.Description
            Exit Sub
        End If
    End If

    acadApp.Visible = True
    MsgBox "Now running " + acadApp.Name + _
            " version " + acadApp.Version

    Dim acadDoc as AcadDocument
    Set acadDoc = acadApp.ActiveDocument

    '' Optionally, load your assembly and start your command or if your assembly
    '' is demandloaded, simply start the command of your in-process assembly.
    acadDoc.SendCommand("(command " & Chr(34) & "NETLOAD" & Chr(34) & " " & _
                        Chr(34) & "c:/myapps/mycommands.dll" & Chr(34) & ") ")

    acadDoc.SendCommand("MyCommand ")
End Sub
```

Define Commands and AutoLISP Functions

Commands and AutoLISP® functions can be defined with the AutoCAD .NET API through the use of two attributes: CommandMethod and LispFunction. You place one of the two attributes before the method that should be called when the command or AutoLISP function is executed in AutoCAD.

Methods used for commands should not be defined with arguments. However, a method used to define an AutoLISP function should be defined with a single argument of the ResultBuffer object type.

Topics in this section

- Command Definition
- AutoLISP Function Definition

Command Definition

When defining a command, you use the CommandMethod attribute. The CommandMethod attribute expects a string value to use as the global name of the command that is being defined. Along with a global command name, the CommandMethod attribute can accept the following values:

- **Command Flags** -Defines the behavior of the command
- **Group Name** - Command group name
- **Local Name** - Local command name, usually language specific
- **Help Topic Name** - Help topic name that should be displayed when F1 is pressed
- **Context Menu Type Flags** - Defines the context menu behavior when the command is active
- **Help File Name** -Help file that contains the help topic that should be displayed when the command is active and F1 is pressed

The following table lists the available flags that can be used to define the behavior of a command.

Enum Value	Description
ActionMacro	Command can be recorded as an action with the Action Recorder.
DocReadLock	Document will be read locked when command is invoked.
Interruptible	The command may be interrupted when prompting for user input.
Modal	Command cannot be invoked while another command is active.
NoActionRecording	Command cannot be recorded as action with the Action Recorder.
NoBlockEditor	Command cannot be used from the Block Editor.
NoHistory	Command is not added to the repeat-last-command history list.
NoPaperSpace	Command cannot be used from Paper space.
NoTileMode	Command cannot be used when TILEMODE is set to 1.
NoUndoMarker	Command does not support undo markers. This is intended for commands that do not modify the database, and therefore should not show up in the undo file.
Redraw	When the pickfirst set or grip set are retrieved, they are not cleared.
Session	Command is executed in the context of the application rather than the current document context.
Transparent	Command can be used while another command is active.
Undefined	Command can only be used via its Global Name.
UsePickSet	When the pickfirst set is retrieved, it is cleared.

Syntax to Define a Command

The following demonstrates the creation of a CommandMethod attribute that defines a command named CheckForPickfirstSelection. The attribute also uses the command flag UsePickSet to indicate that the command is allowed to use the objects that are selected before the command is started.

VB.NET

```
<CommandMethod("CheckForPickfirstSelection", CommandFlags.UsePickSet)> _
Public Sub CheckForPickfirstSelection()
 . . .
End Sub
```

C#

```
[CommandMethod("CheckForPickfirstSelection", CommandFlags.UsePickSet)]
public static void CheckForPickfirstSelection()
{
 . . .
}
```

You can specify the use of more than one flag by using the + operator in VB.NET and the & operator in C#.

VB.NET

```
<CommandMethod("CheckForPickfirstSelection", CommandFlags.UsePickSet + _
                                             CommandFlags.NoBlockEditor)> _
Public Sub CheckForPickfirstSelection()
 . . .
End Sub
```

C#

```
[CommandMethod("CheckForPickfirstSelection", CommandFlags.UsePickSet &
                                             CommandFlags.NoBlockEditor)]
public static void CheckForPickfirstSelection()
{
 . . .
}
```

AutoLISP Function Definition

When defining an AutoLISP function, you use the LispFunction attribute. The LispFunction attribute expects a string value to use as the global name of the AutoLISP function that is being defined. Along with a global function name, the LispFunction structure can accept the following values:

- *Local Name* - Local function name, usually language specific
-
- *Help Topic Name* - Help topic name that should be associated with the AutoLISP function
-
- *Help File Name* -Help file that contains the help topic that should be displayed when the command is active and F1 is pressed

Syntax to Define an AutoLISP Function

The following demonstrates the creation of a LispFunction attribute that defines an AutoLISP function named InsertDynamicBlock.

VB.NET

```
<LispFunction("InsertDynamicBlock")> _
Public Sub InsertDynamicBlock(ByVal rbArgs As ResultBuffer)
```

```
   . . .
End Sub
```

C#

```csharp
[LispFunction("DisplayFullName")]
public static void DisplayFullName(ResultBuffer rbArgs)
{
   . . .
}
```

Retrieve Values Passed into an AutoLISP Function

Use a Foreach loop to step through the values returned in the ResultBuffer by the AutoLISP function. A ResultBuffer is a collection of TypedValue objects. The TypeCode property of a TypedValue object can be used to determine the value type for each value passed into the AutoLISP function. The Value property is used to return the value of the TypedValue object.

To define an AutoLISP Function

This example code defines an AutoLISP function named DisplayFullName. While the method defined in the .NET project accepts a single value, the AutoLISP function expects two string values to produce the correct output.

Load the .NET project into AutoCAD and enter the following at the Command prompt:

```
(displayfullname "First" "Last")
```

The following is the output displayed after the AutoLISP function is executed:

```
Name: First Last
```

VB.NET

```vb
Imports Autodesk.AutoCAD.Runtime
Imports Autodesk.AutoCAD.ApplicationServices

<LispFunction("DisplayFullName")> _
Public Sub DisplayFullName(ByVal rbArgs As ResultBuffer)
  If Not rbArgs = Nothing Then
      Dim strVal1 As String = "", strVal2 As String = ""

      Dim nCnt As Integer = 0
      For Each rb As TypedValue In rbArgs
          If (rb.TypeCode = Autodesk.AutoCAD.Runtime.LispDataType.Text) Then
              Select Case nCnt
                  Case 0
                      strVal1 = rb.Value.ToString()
                  Case 1
                      strVal2 = rb.Value.ToString()
              End Select

              nCnt = nCnt + 1
          End If
      Next

      Application.DocumentManager.MdiActiveDocument.Editor. _
          WriteMessage(vbLf & "Name: " & strVal1 & " " & strVal2)
  End If
End Sub
```

C#

```csharp
using Autodesk.AutoCAD.Runtime;
using Autodesk.AutoCAD.ApplicationServices;

[LispFunction("DisplayFullName")]
public static void DisplayFullName(ResultBuffer rbArgs)
{
  if (rbArgs != null)
  {
      string strVal1 = "";
      string strVal2 = "";

      int nCnt = 0;
      foreach (TypedValue rb in rbArgs)
      {
          if (rb.TypeCode == (int)Autodesk.AutoCAD.Runtime.LispDataType.Text)
          {
              switch(nCnt)
              {
                  case 0:
                      strVal1 = rb.Value.ToString();
                      break;
                  case 1:
                      strVal2 = rb.Value.ToString();
                      break;
              }

              nCnt = nCnt + 1;
          }
      }

      Application.DocumentManager.MdiActiveDocument.Editor.
          WriteMessage("\nName: " + strVal1 + " " + strVal2);
  }
}
```

4   Control the AutoCAD Environment

This chapter describes the fundamentals for developing an application that runs in-process with AutoCAD. It explains many of the concepts to control and work effectively with the AutoCAD environment.

Topics in this section

- Control the Application Window
- Control the Drawing Windows
- Create, Open, Save, and Close Drawings
- Lock and Unlock a Document
- Set AutoCAD Preferences
- Set and Return System Variables
- Draw with Precision
- Prompt for User Input
- Access the AutoCAD Command Line

Control the Application Window

The ability to control the Application window allows developers the flexibility to create effective and intelligent applications. There will be times when it is appropriate for your application to minimize the AutoCAD window, perhaps while your code is performing work in another application such as Microsoft® Excel®. Additionally, you will often want to verify the state of the AutoCAD window before performing such tasks as prompting for input from the user.

Using methods and properties found on the Application object, you can change the position, size, and visibility of the Application window. You can also use the WindowState property to minimize, maximize, and check the current state of the Application window.

Position and size the Application window

This example uses the Location and Size properties to position the AutoCAD Application window in the upper-left corner of the screen and size it to 400 pixels wide by 400 pixels high.

NoteThe following examples require that the PresentationCore (*PresentationCore.dll*) library to be referenced to the project. Use the Add Reference dialog box and select PresentationCore from the .NET tab.

VB.NET

```
Imports System.Drawing
Imports Autodesk.AutoCAD.Runtime
Imports Autodesk.AutoCAD.ApplicationServices

<CommandMethod("PositionApplicationWindow")> _
```

```
Public Sub PositionApplicationWindow()
  '' Set the position of the Application window
  Dim ptApp As Point = New Point(0, 0)
  Application.MainWindow.Location = ptApp

  '' Set the size of the Application window
  Dim szApp As Size = New Size(400, 400)
  Application.MainWindow.Size = szApp
End Sub
```

C#

```
using System.Drawing;
using Autodesk.AutoCAD.Runtime;
using Autodesk.AutoCAD.ApplicationServices;

[CommandMethod("PositionApplicationWindow")]
public static void PositionApplicationWindow()
{
  // Set the position of the Application window
  Point ptApp = new Point(0, 0);
  Application.MainWindow.Location = ptApp;

  // Set the size of the Application window
  Size szApp = new Size(400, 400);
  Application.MainWindow.Size = szApp;
}
```

⊟ VBA/ActiveX Code Reference

```
Sub PositionApplicationWindow()
    '' Set the position of the Application window
    ThisDrawing.Application.WindowTop = 0
    ThisDrawing.Application.WindowLeft = 0

    '' Set the size of the Application window
    ThisDrawing.Application.width = 400
    ThisDrawing.Application.height = 400
End Sub
```

Minimize and maximize the Application window

NoteThe following examples require that the PresentationCore (*PresentationCore.dll*) library
to be referenced to the project. Use the Add Reference dialog box and select
PresentationCore from the .NET tab.

VB.NET

```
Imports System.Drawing
Imports Autodesk.AutoCAD.Runtime
Imports Autodesk.AutoCAD.ApplicationServices
Imports Autodesk.AutoCAD.Windows

<CommandMethod("MinMaxApplicationWindow")> _
Sub MinMaxApplicationWindow()
  '' Minimize the Application window
  Application.MainWindow.WindowState = Window.State.Minimized
  MsgBox("Minimized", MsgBoxStyle.SystemModal, "MinMax")

  '' Maximize the Application window
```

```
  Application.MainWindow.WindowState = Window.State.Maximized
  MsgBox("Maximized", MsgBoxStyle.SystemModal, "MinMax")
End Sub
```

C#

```
using System.Drawing;
using Autodesk.AutoCAD.Runtime;
using Autodesk.AutoCAD.ApplicationServices;
using Autodesk.AutoCAD.Windows;

[CommandMethod("MinMaxApplicationWindow")]
public static void MinMaxApplicationWindow()
{
  // Minimize the Application window
  Application.MainWindow.WindowState = Window.State.Minimized;
  System.Windows.Forms.MessageBox.Show("Minimized", "MinMax",
              System.Windows.Forms.MessageBoxButtons.OK,
              System.Windows.Forms.MessageBoxIcon.None,
              System.Windows.Forms.MessageBoxDefaultButton.Button1,
              System.Windows.Forms.MessageBoxOptions.ServiceNotification);

  // Maximize the Application window
  Application.MainWindow.WindowState = Window.State.Maximized;
  System.Windows.Forms.MessageBox.Show("Maximized", "MinMax");
}
```
⊟ VBA/ActiveX Code Reference
```
Sub MinMaxApplicationWindow()
    '' Minimize the Application window
    ThisDrawing.Application.WindowState = acMin
    MsgBox "Minimized"

    '' Maximize the Application window
    ThisDrawing.Application.WindowState = acMax
    MsgBox "Maximized"
End Sub
```

Find the current state of the Application window

This example queries the state of the Application window and displays the state in a message box to the user.

VB.NET

```
Imports Autodesk.AutoCAD.Runtime
Imports Autodesk.AutoCAD.ApplicationServices
Imports Autodesk.AutoCAD.Windows

<CommandMethod("CurrentWindowState")> _
Public Sub CurrentWindowState()
  System.Windows.Forms.MessageBox.Show("The application window is " + _
                                        Application.MainWindow.WindowState.ToString
(), _
                                        "Window State")
End Sub
```

C#

```
using Autodesk.AutoCAD.Runtime;
using Autodesk.AutoCAD.ApplicationServices;
using Autodesk.AutoCAD.Windows;
```

```
[CommandMethod("CurrentWindowState")]
public static void CurrentWindowState()
{
   System.Windows.Forms.MessageBox.Show("The application window is " +
                                        Application.MainWindow.WindowState.ToString
(),
                                        "Window State");
}
```

```
Sub CurrentWindowState()
    Dim CurrWindowState As Integer
    Dim msg As String
    CurrWindowState = ThisDrawing.Application.WindowState
    msg = Choose(CurrWindowState, "Normal", _
                 "Minimized", "Maximized")
    MsgBox "The application window is " + msg
End Sub
```

Make the Application window invisible and visible

The following code uses the Visible property to make the AutoCAD application invisible and then visible again.

VB.NET

```
Imports Autodesk.AutoCAD.Runtime
Imports Autodesk.AutoCAD.ApplicationServices
Imports Autodesk.AutoCAD.Windows

<CommandMethod("HideWindowState")> _
Public Sub HideWindowState()
  '' Hide the Application window
  Application.MainWindow.Visible = False
  MsgBox("Invisible", MsgBoxStyle.SystemModal, "Show/Hide")

  '' Show the Application window
  Application.MainWindow.Visible = True
  MsgBox("Visible", MsgBoxStyle.SystemModal, "Show/Hide")
End Sub
```

C#

```
using Autodesk.AutoCAD.Runtime;
using Autodesk.AutoCAD.ApplicationServices;
using Autodesk.AutoCAD.Windows;

[CommandMethod("HideWindowState")]
public static void HideWindowState()
{
  // Hide the Application window
  Application.MainWindow.Visible = false;
  System.Windows.Forms.MessageBox.Show("Invisible", "Show/Hide");

  // Show the Application window
  Application.MainWindow.Visible = true;
  System.Windows.Forms.MessageBox.Show("Visible", "Show/Hide");
}
```

```
Sub HideWindowState()
    '' Hide the Application window
    ThisDrawing.Application.Visible = False
    MsgBox "Invisible"

    '' Show the Application window
    ThisDrawing.Application.Visible = True
    MsgBox "Visible"
End Sub
```

Control the Drawing Windows

Like the AutoCAD Application window, you can minimize, maximize, reposition, resize, and check the state of any Document window. But you can also change the way the drawing is displayed within a window by using views, viewports, and zooming methods.

The AutoCAD .NET API provides many ways to display your drawing. You can control the drawing display to move quickly to different areas of your drawing while you track the overall effect of your changes. You can zoom to change magnification or pan to reposition the view in the graphics area, save a named view and then restore it when you need to plot or refer to specific details, or display several views at one time by splitting the screen into several tiled viewports.

Topics in this section

- Position and Size the Document Window
- Zoom and Pan the Current View
- Use Named Views
- Use Tiled Viewports
- Update the Geometry in the Document Window

Position and Size the Document Window

Use the Document object to modify the position and size of any document window. The Document window can be minimized or maximized by using the WindowState property, and you can find the current state of the Document window by using the WindowState property.

Size the active Document window

This example uses the Location and Size Height properties to set the placement and size of Document window to 400 pixels wide by 400 pixels high.

VB.NET

```
Imports Autodesk.AutoCAD.Runtime
Imports Autodesk.AutoCAD.ApplicationServices
```

```
Imports Autodesk.AutoCAD.Windows

<CommandMethod("SizeDocumentWindow")> _
Public Sub SizeDocumentWindow()
  '' Size the Document window
  Dim acDoc As Document = Application.DocumentManager.MdiActiveDocument
  acDoc.Window.WindowState = Window.State.Normal

  '' Set the position of the Document window
  Dim ptDoc As Point = New Point(0, 0)
  acDoc.Window.Location = ptDoc

  '' Set the size of the Document window
  Dim szDoc As Size = New Size(400, 400)
  acDoc.Window.Size = szDoc
End Sub
```

C#

```
using Autodesk.AutoCAD.Runtime;
using Autodesk.AutoCAD.ApplicationServices;
using Autodesk.AutoCAD.Windows;

[CommandMethod("SizeDocumentWindow")]
public static void SizeDocumentWindow()
{
  // Size the Document window
  Document acDoc = Application.DocumentManager.MdiActiveDocument;
  acDoc.Window.WindowState = Window.State.Normal;

  // Set the position of the Document window
  Point ptDoc  = new Point(0, 0);
  acDoc.Window.Location = ptDoc;

  // Set the size of the Document window
  Size szDoc = new Size(400, 400);
  acDoc.Window.Size = szDoc;
}
```

⊟ VBA/ActiveX Code Reference

```
Sub SizeDocumentWindow()
    ThisDrawing.Width = 400
    ThisDrawing.Height = 400
End Sub
```

Minimize and maximize the active Document window

VB.NET

```
Imports Autodesk.AutoCAD.Runtime
Imports Autodesk.AutoCAD.ApplicationServices
Imports Autodesk.AutoCAD.Windows

<CommandMethod("MinMaxDocumentWindow")> _
Public Sub MinMaxDocumentWindow()
  Dim acDoc As Document = Application.DocumentManager.MdiActiveDocument

  '' Minimize the Document window
  acDoc.Window.WindowState = Window.State.Minimized
  MsgBox("Minimized", MsgBoxStyle.SystemModal, "MinMax")
```

```
    '' Maximize the Document window
    acDoc.Window.WindowState = Window.State.Maximized
    MsgBox("Maximized", MsgBoxStyle.SystemModal, "MinMax")
End Sub
```

C#

```
using Autodesk.AutoCAD.Runtime;
using Autodesk.AutoCAD.ApplicationServices;
using Autodesk.AutoCAD.Windows;

[CommandMethod("MinMaxDocumentWindow")]
public static void MinMaxDocumentWindow()
{
  Document acDoc = Application.DocumentManager.MdiActiveDocument;

  // Minimize the Document window
  acDoc.Window.WindowState = Window.State.Minimized;
  System.Windows.Forms.MessageBox.Show("Minimized" , "MinMax");

  // Maximize the Document window
  acDoc.Window.WindowState = Window.State.Maximized;
  System.Windows.Forms.MessageBox.Show("Maximized" , "MinMax");
}
```

VBA/ActiveX Code Reference

```
Sub MinMaxDocumentWindow()
    '' Minimize the Document window
    ThisDrawing.WindowState = acMin
    MsgBox "Minimized"

    '' Minimize the Document window
    ThisDrawing.WindowState = acMax
    MsgBox "Maximized"
End Sub
```

Find the current state of the active Document window

VB.NET

```
Imports Autodesk.AutoCAD.Runtime
Imports Autodesk.AutoCAD.ApplicationServices
Imports Autodesk.AutoCAD.Windows

<CommandMethod("CurrentDocWindowState")> _
Public Sub CurrentDocWindowState()
  Dim acDoc As Document = Application.DocumentManager.MdiActiveDocument

  System.Windows.Forms.MessageBox.Show("The document window is " & _
  acDoc.Window.WindowState.ToString(), "Window State")
End Sub
```

C#

```
using Autodesk.AutoCAD.Runtime;
using Autodesk.AutoCAD.ApplicationServices;
using Autodesk.AutoCAD.Windows;

[CommandMethod("CurrentDocWindowState")]
```

```
public static void CurrentDocWindowState()
{
  Document acDoc = Application.DocumentManager.MdiActiveDocument;

  System.Windows.Forms.MessageBox.Show("The document window is " +
  acDoc.Window.WindowState.ToString(), "Window State");
}
```

⊟ *VBA/ActiveX Code Reference*

```
Sub CurrentDocWindowState()
    Dim CurrWindowState As Integer
    Dim msg As String
    CurrWindowState = ThisDrawing.WindowState
    msg = Choose(CurrWindowState, "Normal", _
                 "Minimized", "Maximized")
    MsgBox "The document window is " + msg
End Sub
```

Zoom and Pan the Current View

A view is a specific magnification, position, and orientation of a drawing in the drawing window. You change a view of a drawing by changing the height, width and center point of the current view. Increasing or decreasing the width or height of a view affects the size in which a drawing is displayed. Panning a view is done by adjusting the center of the current view. For more information on zooming in AutoCAD, see "Change Views" in the *AutoCAD User's Guide*.

Topics in this section

- [Manipulate the Current View](#)
- [Define to Window](#)
- [Scale a View](#)
- [Center Objects](#)
- [Display Drawing Extents and Limits](#)

Manipulate the Current View

You access the current view of a viewport in Model or Paper space by using the GetCurrentView method of the Editor object. The GetCurrentView method returns a ViewTableRecord object. You use the ViewTableRecord object to manipulate the magnification, position, and orientation of the view in the active viewport. Once the ViewTableRecord object has been changed, you update the current view of the active viewport with the SetCurrentView method.

Some of the common properties that you will use to manipulate the current view are:

- **CenterPoint** - Center point of a view in DCS coordinates.

- **Height** - Height of a view in DCS coordinates. Increase the height to zoom out; decrease the height to zoom in.

- **Target** - Target of a view in WCS coordinates.

- **ViewDirection** - Vector from the target to the camera of a view in WCS coordinates.

- **ViewTwist** - Twist angle in radians of a view.

- **Width** - Width of a view in DCS coordinates. Increase the width to zoom out; decrease the width to zoom in.

VBA Code Cross Reference

The .NET API does not offer methods to directly manipulate the current view of a drawing like those found in the ActiveX Automation library. For example, if you want to zoom to the extents of the objects in a drawing or the limits of a drawing, you must manipulate the Width, Height and CenterPoint properties of the current view. To get the extents of limits of a drawing, you use the Extmin, Extmax, Limmin, and Limmax properties of the Database object.

Function used to manipulate the current view

This example code is a common procedure that is used by later examples. The Zoom procedure accepts four parameters to accomplish zooming to a boundary, panning or centering the view of a drawing, and scaling the view of a drawing by a given factor. The Zoom procedure expects all coordinate values to be provided in WCS coordinates.

The parameters of the Zoom procedure are:

- **Minimum point** - 3D point used to define the lower-left corner of the area to display.

- **Maximum point** - 3D point used to define the upper-right corner of the area to display.

- **Center point** - 3D point used to define the center of a view.

- **Scale factor** - Real number used to specify the scale to increase or decrease the size of a view.

VB.NET

```
Imports Autodesk.AutoCAD.ApplicationServices
Imports Autodesk.AutoCAD.DatabaseServices
Imports Autodesk.AutoCAD.Runtime
Imports Autodesk.AutoCAD.Geometry

Public Sub Zoom(ByVal pMin As Point3d, ByVal pMax As Point3d, _
                ByVal pCenter As Point3d, ByVal dFactor As Double)
  '' Get the current document and database
  Dim acDoc As Document = Application.DocumentManager.MdiActiveDocument
  Dim acCurDb As Database = acDoc.Database
```

```vbnet
    Dim nCurVport As Integer = _
System.Convert.ToInt32(Application.GetSystemVariable("CVPORT"))

    '' Get the extents of the current space when no points
    '' or only a center point is provided
    '' Check to see if Model space is current
    If acCurDb.TileMode = True Then
        If pMin.Equals(New Point3d()) = True And _
            pMax.Equals(New Point3d()) = True Then

            pMin = acCurDb.Extmin
            pMax = acCurDb.Extmax
        End If
    Else
        '' Check to see if Paper space is current
        If nCurVport = 1 Then
            If pMin.Equals(New Point3d()) = True And _
                pMax.Equals(New Point3d()) = True Then

                pMin = acCurDb.Pextmin
                pMax = acCurDb.Pextmax
            End If
        Else
            '' Get the extents of Model space
            If pMin.Equals(New Point3d()) = True And _
                pMax.Equals(New Point3d()) = True Then

                pMin = acCurDb.Extmin
                pMax = acCurDb.Extmax
            End If
        End If
    End If

    '' Start a transaction
    Using acTrans As Transaction = acCurDb.TransactionManager.StartTransaction()
        '' Get the current view
        Using acView As ViewTableRecord = acDoc.Editor.GetCurrentView()
            Dim eExtents As Extents3d

            '' Translate WCS coordinates to DCS
            Dim matWCS2DCS As Matrix3d
            matWCS2DCS = Matrix3d.PlaneToWorld(acView.ViewDirection)
            matWCS2DCS = Matrix3d.Displacement(acView.Target - Point3d.Origin) * _
matWCS2DCS
            matWCS2DCS = Matrix3d.Rotation(-acView.ViewTwist, _
                                          acView.ViewDirection, _
                                          acView.Target) * matWCS2DCS

            '' If a center point is specified, define the
            '' min and max point of the extents
            '' for Center and Scale modes
            If pCenter.DistanceTo(Point3d.Origin) <> 0 Then
                pMin = New Point3d(pCenter.X - (acView.Width / 2), _
                                   pCenter.Y - (acView.Height / 2), 0)

                pMax = New Point3d((acView.Width / 2) + pCenter.X, _
                                   (acView.Height / 2) + pCenter.Y, 0)
            End If

            '' Create an extents object using a line
            Using acLine As Line = New Line(pMin, pMax)
                eExtents = New Extents3d(acLine.Bounds.Value.MinPoint, _
                                         acLine.Bounds.Value.MaxPoint)
            End Using
```

```vb
            '' Calculate the ratio between the width and height of the current view
            Dim dViewRatio As Double
            dViewRatio = (acView.Width / acView.Height)

            '' Tranform the extents of the view
            matWCS2DCS = matWCS2DCS.Inverse()
            eExtents.TransformBy(matWCS2DCS)

            Dim dWidth As Double
            Dim dHeight As Double
            Dim pNewCentPt As Point2d

            '' Check to see if a center point was provided (Center and Scale modes)
            If pCenter.DistanceTo(Point3d.Origin) <> 0 Then
                dWidth = acView.Width
                dHeight = acView.Height

                If dFactor = 0 Then
                    pCenter = pCenter.TransformBy(matWCS2DCS)
                End If

                pNewCentPt = New Point2d(pCenter.X, pCenter.Y)
            Else '' Working in Window, Extents and Limits mode
                '' Calculate the new width and height of the current view
                dWidth = eExtents.MaxPoint.X - eExtents.MinPoint.X
                dHeight = eExtents.MaxPoint.Y - eExtents.MinPoint.Y

                '' Get the center of the view
                pNewCentPt = New Point2d(((eExtents.MaxPoint.X +
eExtents.MinPoint.X) * 0.5), _
                                        ((eExtents.MaxPoint.Y +
eExtents.MinPoint.Y) * 0.5))
            End If

            '' Check to see if the new width fits in current window
            If dWidth > (dHeight * dViewRatio) Then dHeight = dWidth / dViewRatio

            '' Resize and scale the view
            If dFactor <> 0 Then
                acView.Height = dHeight * dFactor
                acView.Width = dWidth * dFactor
            End If

            '' Set the center of the view
            acView.CenterPoint = pNewCentPt

            '' Set the current view
            acDoc.Editor.SetCurrentView(acView)
        End Using

        '' Commit the changes
        acTrans.Commit()
    End Using
End Sub
```

C#

```csharp
using Autodesk.AutoCAD.ApplicationServices;
using Autodesk.AutoCAD.DatabaseServices;
using Autodesk.AutoCAD.Runtime;
using Autodesk.AutoCAD.Geometry;

static void Zoom(Point3d pMin, Point3d pMax, Point3d pCenter, double dFactor)
{
```

```csharp
    // Get the current document and database
    Document acDoc = Application.DocumentManager.MdiActiveDocument;
    Database acCurDb = acDoc.Database;

    int nCurVport = System.Convert.ToInt32(Application.GetSystemVariable("CVPORT"));

    // Get the extents of the current space no points
    // or only a center point is provided
    // Check to see if Model space is current
    if (acCurDb.TileMode == true)
    {
        if (pMin.Equals(new Point3d()) == true &&
            pMax.Equals(new Point3d()) == true)
        {
            pMin = acCurDb.Extmin;
            pMax = acCurDb.Extmax;
        }
    }
    else
    {
        // Check to see if Paper space is current
        if (nCurVport == 1)
        {
            // Get the extents of Paper space
            if (pMin.Equals(new Point3d()) == true &&
                pMax.Equals(new Point3d()) == true)
            {
                pMin = acCurDb.Pextmin;
                pMax = acCurDb.Pextmax;
            }
        }
        else
        {
            // Get the extents of Model space
            if (pMin.Equals(new Point3d()) == true &&
                pMax.Equals(new Point3d()) == true)
            {
                pMin = acCurDb.Extmin;
                pMax = acCurDb.Extmax;
            }
        }
    }

    // Start a transaction
    using (Transaction acTrans = acCurDb.TransactionManager.StartTransaction())
    {
        // Get the current view
        using (ViewTableRecord acView = acDoc.Editor.GetCurrentView())
        {
            Extents3d eExtents;

            // Translate WCS coordinates to DCS
            Matrix3d matWCS2DCS;
            matWCS2DCS = Matrix3d.PlaneToWorld(acView.ViewDirection);
            matWCS2DCS = Matrix3d.Displacement(acView.Target - Point3d.Origin) *
matWCS2DCS;
            matWCS2DCS = Matrix3d.Rotation(-acView.ViewTwist,
                                          acView.ViewDirection,
                                          acView.Target) * matWCS2DCS;

            // If a center point is specified, define the min and max
            // point of the extents
            // for Center and Scale modes
            if (pCenter.DistanceTo(Point3d.Origin) != 0)
            {
```

```
            pMin = new Point3d(pCenter.X - (acView.Width / 2),
                                pCenter.Y - (acView.Height / 2), 0);

            pMax = new Point3d((acView.Width / 2) + pCenter.X,
                                (acView.Height / 2) + pCenter.Y, 0);
        }

        // Create an extents object using a line
        using (Line acLine = new Line(pMin, pMax))
        {
            eExtents = new Extents3d(acLine.Bounds.Value.MinPoint,
                                     acLine.Bounds.Value.MaxPoint);
        }

        // Calculate the ratio between the width and height of the current view
        double dViewRatio;
        dViewRatio = (acView.Width / acView.Height);

        // Tranform the extents of the view
        matWCS2DCS = matWCS2DCS.Inverse();
        eExtents.TransformBy(matWCS2DCS);

        double dWidth;
        double dHeight;
        Point2d pNewCentPt;

        // Check to see if a center point was provided (Center and Scale modes)
        if (pCenter.DistanceTo(Point3d.Origin) != 0)
        {
            dWidth = acView.Width;
            dHeight = acView.Height;

            if (dFactor == 0)
            {
                pCenter = pCenter.TransformBy(matWCS2DCS);
            }

            pNewCentPt = new Point2d(pCenter.X, pCenter.Y);
        }
        else // Working in Window, Extents and Limits mode
        {
            // Calculate the new width and height of the current view
            dWidth = eExtents.MaxPoint.X - eExtents.MinPoint.X;
            dHeight = eExtents.MaxPoint.Y - eExtents.MinPoint.Y;

            // Get the center of the view
            pNewCentPt = new Point2d(((eExtents.MaxPoint.X +
eExtents.MinPoint.X) * 0.5),
                                     ((eExtents.MaxPoint.Y +
eExtents.MinPoint.Y) * 0.5));
        }

        // Check to see if the new width fits in current window
        if (dWidth > (dHeight * dViewRatio)) dHeight = dWidth / dViewRatio;

        // Resize and scale the view
        if (dFactor != 0)
        {
            acView.Height = dHeight * dFactor;
            acView.Width = dWidth * dFactor;
        }

        // Set the center of the view
        acView.CenterPoint = pNewCentPt;
```

```
        // Set the current view
        acDoc.Editor.SetCurrentView(acView);
    }

    // Commit the changes
    acTrans.Commit();
  }
}
```

Define to Window

in AutoCAD, you use the Window option of the ZOOM command to define the area of the drawing that should be displayed in the drawing window. When you define the area to be displayed, the Width and Height properties of the current view are adjusted to match the area defined by the two points specified. Based on the specified points, the CenterPoint property of the view is also moved.

Zoom to an area defined by two points

This example code demonstrates how to zoom to a defined area using the Zoom procedure defined under <u>Manipulate the Current View</u>. The Zoom procedure is passed the coordinates (1.3,7.8,0) and (13.7,-2.6,0) for the first two arguments to define the area to display.

No new center point is needed, so a new Point3d object is passed to the procedure. The last argument is used to scale the new view. Scaling the view can be used to create a gap between the area displayed and the edge of the drawing window.

VB.NET

```
<CommandMethod("ZoomWindow")> _
Public Sub ZoomWindow()
  '' Zoom to a window boundary defined by 1.3,7.8 and 13.7,-2.6
  Dim pMin As Point3d = New Point3d(1.3, 7.8, 0)
  Dim pMax As Point3d = New Point3d(13.7, -2.6, 0)

  Zoom(pMin, pMax, New Point3d(), 1)
End Sub
```

C#

```
[CommandMethod("ZoomWindow")]
static public void ZoomWindow()
{
  // Zoom to a window boundary defined by 1.3,7.8 and 13.7,-2.6
  Point3d pMin = new Point3d(1.3, 7.8, 0);
  Point3d pMax = new Point3d(13.7, -2.6, 0);

  Zoom(pMin, pMax, new Point3d(), 1);
}
```

```
Sub ZoomWindow()
    Dim point1(0 To 2) As Double
    Dim point2(0 To 2) As Double
    point1(0) = 1.3: point1(1) = 7.8: point1(2) = 0
    point2(0) = 13.7: point2(1) = -2.6: point2(2) = 0

    ThisDrawing.Application.ZoomWindow point1, point2
End Sub
```

Scale a View

If you need to increase or decrease the magnification of the image in the drawing window, you change the Width and Height properties of the current view. When resizing a view, make sure to change the Width and Height properties by the same factor. The scale factor you calculate when resizing the current view will commonly be based on one of the following situations:

- Relative to the drawing limits

- Relative to the current view

- Relative to paper space units

Zoom in on the active drawing using a specified scale

This example code demonstrates how to reduce the current view by 50% using the Zoom procedure defined under [Manipulate the Current View](#).

While the Zoom procedure is passed a total of four values, the first two are new 3D points which are not used. The third value passed is the center point to use in resizing the view and the last value passed is the scale factor to use in resizing the view.

VB.NET

```
<CommandMethod("ZoomScale")> _
Public Sub ZoomScale()
  '' Get the current document
  Dim acDoc As Document = Application.DocumentManager.MdiActiveDocument

  '' Get the current view
  Using acView As ViewTableRecord = acDoc.Editor.GetCurrentView()
      '' Get the center of the current view
      Dim pCenter As Point3d = New Point3d(acView.CenterPoint.X, _
                                           acView.CenterPoint.Y, 0)

      '' Set the scale factor to use
      Dim dScale As Double = 0.5
```

```
        '' Scale the view using the center of the current view
        Zoom(New Point3d(), New Point3d(), pCenter, 1 / dScale)
    End Using
End Sub
```

C#

```
[CommandMethod("ZoomScale")]
static public void ZoomScale()
{
  // Get the current document
  Document acDoc = Application.DocumentManager.MdiActiveDocument;

  // Get the current view
  using (ViewTableRecord acView = acDoc.Editor.GetCurrentView())
  {
      // Get the center of the current view
      Point3d pCenter = new Point3d(acView.CenterPoint.X,
                                    acView.CenterPoint.Y, 0);

      // Set the scale factor to use
      double dScale = 0.5;

      // Scale the view using the center of the current view
      Zoom(new Point3d(), new Point3d(), pCenter, 1 / dScale);
  }
}
```

⊟ *VBA/ActiveX Code Reference*

```
Sub ZoomScale()
    Dim scalefactor As Double
    Dim scaletype As Integer

    scalefactor = 0.5
    scaletype = acZoomScaledRelative

    ThisDrawing.Application.ZoomScaled scalefactor, scaletype
End Sub
```

Center Objects

You can reposition the image in the drawing window by changing the center point of a view using the CenterPoint property. When the center point of a view is changed and the size of the view is not changed, the view is panned parallel to the screen. For information on resizing the image in the drawing window, see Scale a View.

Zoom in on the active drawing to a specified center

This example code demonstrates how to change the center point of the current view using the Zoom procedure defined under Manipulate the Current View.

While the Zoom procedure is passed a total of four values, the first two values are defined as new 3D points and are ignored by the procedure. The third value is the point (5,5,0) to define the new center point of the view and 1 is passed in for the last value to retain the size of the current view.

VB.NET

```
<CommandMethod("ZoomCenter")> _
Public Sub ZoomCenter()
  '' Center the view at 5,5,0
  Zoom(New Point3d(), New Point3d(), New Point3d(5, 5, 0), 1)
End Sub
```

C#

```
[CommandMethod("ZoomCenter")]
static public void ZoomCenter()
{
  // Center the view at 5,5,0
  Zoom(new Point3d(), new Point3d(), new Point3d(5, 5, 0), 1);
}
```

⊟ VBA/ActiveX Code Reference

```
Sub ZoomCenter()
    Dim Center(0 To 2) As Double
    Dim magnification As Double

    Center(0) = 5: Center(1) = 5: Center(2) = 0
    magnification = 1

    ThisDrawing.Application.ZoomCenter Center, magnification
End Sub
```

Display Drawing Extents and Limits

The extents or limits of a drawing are used to define the boundary in which the outermost objects appear in or the area defined by the limits of the current space.

See "Magnify a View (Zoom)" in the *AutoCAD User's Guide* for illustrations of how zooming works.

Calculate the extents of the current space

The extents of the current space can be accessed from the Database object using the following properties:

- *Extmin and Extmax* - Returns the extents of Model space.
- *Pextmin and Pextmax* - Returns the extents of the current Paper space layout.

Once the extents of the current space is obtained, you can calculate the new values for the Width and Height properties of the current view. The new width for the view is calculated using the following formula:

```
dWidth = MaxPoint.X - MinPoint.X
```

The new height for the view is calculated using the following formula:

```
dHeight = MaxPoint.Y - MinPoint.Y
```

After the width and height of the view are calculated, the center point of the view can be calculated. The center point of the view can be obtained using the following formula:

```
dCenterX = (MaxPoint.X + MinPoint.X) * 0.5
dCenterY = (MaxPoint.Y + MinPoint.Y) * 0.5
```

Calculate the limits of the current space

To change the display of a drawing based on the limits of the current space, you use the `Limmin` and `Limmax` , and `Plimmin` and `Plimmax` properties of the Database object. After the points that define the limits of the current space are returned, you can use the previously mentioned formulas to calculate the width, height and center points of the new view.

Zoom in to the extents and limits of the current space

This example code demonstrates how to display the extents of limits of the current space using the Zoom procedure defined under Manipulate the Current View.

While the Zoom procedure is passed a total of four values, the first two values passed should be the points that define the minimum and maximum points of the area to be displayed. The third value is defined as a new 3D point and is ignored by the procedure, while the last value is used to resize the image of the drawing so it is not completely fill the entire drawing window.

VB.NET

```vbnet
<CommandMethod("ZoomExtents")> _
Public Sub ZoomExtents()
  '' Zoom to the extents of the current space
  Zoom(New Point3d(), New Point3d(), New Point3d(), 1.01075)
End Sub

<CommandMethod("ZoomLimits")> _
Public Sub ZoomLimits()
  Dim acDoc As Document = Application.DocumentManager.MdiActiveDocument
  Dim acCurDb As Database = acDoc.Database

  '' Zoom to the limits of Model space
  Zoom(New Point3d(acCurDb.Limmin.X, acCurDb.Limmin.Y, 0), _
       New Point3d(acCurDb.Limmax.X, acCurDb.Limmax.Y, 0), _
       New Point3d(), 1)
End Sub
```

C#

```csharp
[CommandMethod("ZoomExtents")]
static public void ZoomExtents()
{
```

```
  // Zoom to the extents of the current space
  Zoom(new Point3d(), new Point3d(), new Point3d(), 1.01075);
}

[CommandMethod("ZoomLimits")]
static public void ZoomLimits()
{
  Document acDoc = Application.DocumentManager.MdiActiveDocument;
  Database acCurDb = acDoc.Database;

  // Zoom to the limits of Model space
  Zoom(new Point3d(acCurDb.Limmin.X, acCurDb.Limmin.Y, 0),
       new Point3d(acCurDb.Limmax.X, acCurDb.Limmax.Y, 0),
       new Point3d(), 1);
}
```

VBA/ActiveX Code Reference

```
Sub ZoomExtents()
    ThisDrawing.Application.ZoomExtents
End Sub

Sub ZoomLimits()
    Dim point1(0 To 2) As Double
    Dim point2(0 To 2) As Double

    point1(0) = ThisDrawing.GetVariable("LIMMIN")(0)
    point1(1) = ThisDrawing.GetVariable("LIMMIN")(1)
    point1(2) = 0#

    point2(0) = ThisDrawing.GetVariable("LIMMAX")(0)
    point2(1) = ThisDrawing.GetVariable("LIMMAX")(1)
    point2(2) = 0#

    ThisDrawing.Application.ZoomWindow point1, point2
End Sub
```

Use Named Views

You can name and save a view you want to reuse. When you no longer need the view, you can remove it.

Named views are stored in the View table, one of the named symbol tables in a drawing database. A named view is created with the Add method to add a new view to the View table. When you add the new named view to the View table, a default model space view is created.

You name the view when you create it. The name of the view can be up to 255 characters long and contain letters, digits, and the special characters dollar sign ($), hyphen (-), and underscore (_).

A named view can be removed from the View table by simply use the Erase method of the ViewTableRecord object you want to remove.

Add a named view and set it current

The following example adds a named view to the drawing and sets it current.

VB.NET

```
Imports Autodesk.AutoCAD.ApplicationServices
Imports Autodesk.AutoCAD.DatabaseServices
Imports Autodesk.AutoCAD.Runtime

<CommandMethod("CreateNamedView")> _
Public Sub CreateNamedView()
  '' Get the current database
  Dim acDoc As Document = Application.DocumentManager.MdiActiveDocument
  Dim acCurDb As Database = acDoc.Database

  '' Start a transaction
  Using acTrans As Transaction = acCurDb.TransactionManager.StartTransaction()

      '' Open the View table for read
      Dim acViewTbl As ViewTable
      acViewTbl = acTrans.GetObject(acCurDb.ViewTableId, OpenMode.ForRead)

      '' Check to see if the named view 'View1' exists
      If (acViewTbl.Has("View1") = False) Then
          '' Open the View Table for write
          acViewTbl.UpgradeOpen()

          '' Create a new View table record and name the view "View1"
          Dim acViewTblRec As ViewTableRecord = New ViewTableRecord()
          acViewTblRec.Name = "View1"

          '' Add the new View table record to the View table and the transaction
          acViewTbl.Add(acViewTblRec)
          acTrans.AddNewlyCreatedDBObject(acViewTblRec, True)

          '' Set 'View1' current
          acDoc.Editor.SetCurrentView(acViewTblRec)

          '' Commit the changes
          acTrans.Commit()
      End If

      '' Dispose of the transaction
  End Using
End Sub
```

C#

```
using Autodesk.AutoCAD.ApplicationServices;
using Autodesk.AutoCAD.DatabaseServices;
using Autodesk.AutoCAD.Runtime;

[CommandMethod("CreateNamedView")]
public static void CreateNamedView()
{
  // Get the current database
  Document acDoc = Application.DocumentManager.MdiActiveDocument;
  Database acCurDb = acDoc.Database;

  // Start a transaction
  using (Transaction acTrans = acCurDb.TransactionManager.StartTransaction())
  {
```

```
        // Open the View table for read
        ViewTable acViewTbl;
        acViewTbl = acTrans.GetObject(acCurDb.ViewTableId,
                                      OpenMode.ForRead) as ViewTable;

        // Check to see if the named view 'View1' exists
        if (acViewTbl.Has("View1") == false)
        {
            // Open the View table for write
            acViewTbl.UpgradeOpen();

            // Create a new View table record and name the view 'View1'
            ViewTableRecord acViewTblRec = new ViewTableRecord();
            acViewTblRec.Name = "View1";

            // Add the new View table record to the View table and the transaction
            acViewTbl.Add(acViewTblRec);
            acTrans.AddNewlyCreatedDBObject(acViewTblRec, true);

            // Set 'View1' current
            acDoc.Editor.SetCurrentView(acViewTblRec);

            // Commit the changes
            acTrans.Commit();
        }

        // Dispose of the transaction
    }
}
```

⊟ *VBA/ActiveX Code Reference*

```
Sub CreateNamedView()
    ' Add a named view to the views collection
    Dim viewObj As AcadView
    Set viewObj = ThisDrawing.Views.Add("View1")

    ThisDrawing.ActiveViewport.SetView viewObj
End Sub
```

Erase a named view

The following example erases a named view from the drawing.

VB.NET

```
Imports Autodesk.AutoCAD.ApplicationServices
Imports Autodesk.AutoCAD.DatabaseServices
Imports Autodesk.AutoCAD.Runtime

<CommandMethod("EraseNamedView")> _
Public Sub EraseNamedView()
  '' Get the current database
  Dim acDoc As Document = Application.DocumentManager.MdiActiveDocument
  Dim acCurDb As Database = acDoc.Database

  '' Start a transaction
  Using acTrans As Transaction = acCurDb.TransactionManager.StartTransaction()

      '' Open the View table for read
      Dim acViewTbl As ViewTable
```

```
      acViewTbl = acTrans.GetObject(acCurDb.ViewTableId, OpenMode.ForRead)

      '' Check to see if the named view 'View1' exists
      If (acViewTbl.Has("View1") = True) Then
          '' Open the View table for write
          acViewTbl.UpgradeOpen()

          '' Get the named view
          Dim acViewTblRec As ViewTableRecord
          acViewTblRec = acTrans.GetObject(acViewTbl("View1"), OpenMode.ForWrite)

          '' Remove the named view from the View table
          acViewTblRec.Erase()

          '' Commit the changes
          acTrans.Commit()
      End If

      '' Dispose of the transaction
  End Using
End Sub
```

C#

```
using Autodesk.AutoCAD.ApplicationServices;
using Autodesk.AutoCAD.DatabaseServices;
using Autodesk.AutoCAD.Runtime;

[CommandMethod("EraseNamedView")]
public static void EraseNamedView()
{
  // Get the current database
  Document acDoc = Application.DocumentManager.MdiActiveDocument;
  Database acCurDb = acDoc.Database;

  // Start a transaction
  using (Transaction acTrans = acCurDb.TransactionManager.StartTransaction())
  {
      // Open the View table for read
      ViewTable acViewTbl;
      acViewTbl = acTrans.GetObject(acCurDb.ViewTableId,
                                    OpenMode.ForRead) as ViewTable;

      // Check to see if the named view 'View1' exists
      if (acViewTbl.Has("View1") == true)
      {
          // Open the View table for write
          acViewTbl.UpgradeOpen();

          // Get the named view
          ViewTableRecord acViewTblRec;
          acViewTblRec = acTrans.GetObject(acViewTbl["View1"],
                                           OpenMode.ForWrite) as ViewTableRecord;

          // Remove the named view from the View table
          acViewTblRec.Erase();

          // Commit the changes
          acTrans.Commit();
      }

      // Dispose of the transaction
  }
}
```

```
Sub EraseNamedView()
    On Error Resume Next
    Dim viewObj As AcadView
    Set viewObj = ThisDrawing.Views("View1")

    If Err = 0 Then
      ' Delete the view
      viewObj.Delete
    End If
End Sub
```

Use Tiled Viewports

AutoCAD usually begins a new drawing using a single tiled viewport that fills the entire graphics area. You can split the drawing area of the Model tab to display several viewports simultaneously. For example, if you keep both the full and the detail views visible, you can see the effects of your detail changes on the entire drawing. In each tiled viewport, you can do the following:

- Zoom, set the Snap, Grid, and UCS icon modes, and restore named views in individual viewports
- Draw from one viewport to another when executing a command
- Name a configuration of viewports so you can reuse it

You can display tiled viewports in various configurations. How you display the viewports depends on the number and size of the views you need to see. Tiled viewports in model space are stored in the Viewport table

For further information and illustrations describing viewports, see "Set Model Space Viewports" in the *AutoCAD User's Guide.*

Tiled viewports are stored in the Viewports table. Each record in the Viewports table represents a single viewport and unlike other table records, there might be multiple Viewport table records with the same name. Each of the records with the same name are used to control which viewports are displayed.

For example, the Viewport table records named "*Active" represent the tiled viewports that are currently displayed on the Model tab.

Topics in this section

- [Identify and Manipulate the Active Viewport](#)
- [Make A Tiled Viewport Current](#)

Identify and Manipulate the Active Viewport

The active viewport is represented in the Viewports table by a record named "*Active", which is not a unique name as all tiled viewports currently displayed on the Model tab are named "*Active". Each tiled viewport that is displayed is assigned a number. The number of the active viewport can be obtained by:

- Retrieving the value of the CVPORT system variable
- Using the ActiveViewportId property of the Editor object to get the object id for the active viewport and then open the Viewport object to access its Number property

Once you have the active viewport, you control its display properties, enable drafting aids for the viewport such as grid and snap, as well as the size of the viewport itself. Tiled viewports are defined by two corner points: lower-left and upper-right. The LowerLeftCorner and UpperRightCorner properties represent the graphic placement of the viewport on the display.

A single tiled viewport configuration has a lower-left corner of (0,0) and an upper-right corner of (1,1). The lower-left corner of the drawing window is always represented by the point of (0,0), and the upper-right corner is presented by (1,1) no matter the number of tiled viewports on the Model tab. When more than one tiled viewport is displayed, the lower-left and upper-right corners will vary but one viewport will have a lower-left corner of (0,0) and another will have an upper-right corner of (1,1)

These properties are defined as follows (using a four-way split as an example):
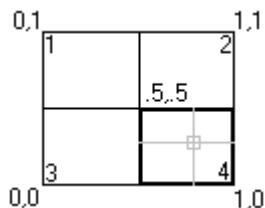
In this example:

- Viewport 1-LowerLeftCorner = (0, .5), UpperRightCorner = (.5, 1)
- Viewport 2-LowerLeftCorner = (.5, .5), UpperRightCorner = (1, 1)
- Viewport 3-LowerLeftCorner = (0, 0), UpperRightCorner = (.5, .5)
- Viewport 4-LowerLeftCorner = (.5, 0), UpperRightCorner = (1, .5)

Create a new tiled viewport configuration with two horizontal windows

The following example creates a two horizontal viewports as a named viewport configuration and redefines the active display.

VB.NET

```
Imports Autodesk.AutoCAD.ApplicationServices
Imports Autodesk.AutoCAD.DatabaseServices
Imports Autodesk.AutoCAD.Runtime
Imports Autodesk.AutoCAD.Geometry
```

```vb
<CommandMethod("CreateModelViewport")> _
Public Sub CreateModelViewport()
    '' Get the current database
    Dim acDoc As Document = Application.DocumentManager.MdiActiveDocument
    Dim acCurDb As Database = acDoc.Database

    '' Start a transaction
    Using acTrans As Transaction = acCurDb.TransactionManager.StartTransaction()

        '' Open the Viewport table for read
        Dim acVportTbl As ViewportTable
        acVportTbl = acTrans.GetObject(acCurDb.ViewportTableId, OpenMode.ForRead)

        '' Check to see if the named view 'TEST_VIEWPORT' exists
        If (acVportTbl.Has("TEST_VIEWPORT") = False) Then
            '' Open the View table for write
            acVportTbl.UpgradeOpen()

            '' Add the new viewport to the Viewport table and the transaction
            Dim acVportTblRecLwr As ViewportTableRecord = New ViewportTableRecord()
            acVportTbl.Add(acVportTblRecLwr)
            acTrans.AddNewlyCreatedDBObject(acVportTblRecLwr, True)

            '' Name the new viewport 'TEST_VIEWPORT' and assign it to be
            '' the lower half of the drawing window
            acVportTblRecLwr.Name = "TEST_VIEWPORT"
            acVportTblRecLwr.LowerLeftCorner = New Point2d(0, 0)
            acVportTblRecLwr.UpperRightCorner = New Point2d(1, 0.5)

            '' Add the new viewport to the Viewport table and the transaction
            Dim acVportTblRecUpr As ViewportTableRecord = New ViewportTableRecord()
            acVportTbl.Add(acVportTblRecUpr)
            acTrans.AddNewlyCreatedDBObject(acVportTblRecUpr, True)

            '' Name the new viewport 'TEST_VIEWPORT' and assign it to be
            '' the upper half of the drawing window
            acVportTblRecUpr.Name = "TEST_VIEWPORT"
            acVportTblRecUpr.LowerLeftCorner = New Point2d(0, 0.5)
            acVportTblRecUpr.UpperRightCorner = New Point2d(1, 1)

            '' To assign the new viewports as the active viewports, the
            '' viewports named '*Active' need to be removed and recreated
            '' based on 'TEST_VIEWPORT'.

            '' Step through each object in the symbol table
            For Each acObjId As ObjectId In acVportTbl
                '' Open the object for read
                Dim acVportTblRec As ViewportTableRecord
                acVportTblRec = acTrans.GetObject(acObjId, _
                                                  OpenMode.ForRead)

                '' See if it is one of the active viewports, and if so erase it
                If (acVportTblRec.Name = "*Active") Then
                    acVportTblRec.UpgradeOpen()
                    acVportTblRec.Erase()
                End If
            Next

            '' Clone the new viewports as the active viewports
            For Each acObjId As ObjectId In acVportTbl
                '' Open the object for read
                Dim acVportTblRec As ViewportTableRecord
                acVportTblRec = acTrans.GetObject(acObjId, _
                                                  OpenMode.ForRead)
```

```
                    '' See if it is one of the active viewports, and if so erase it
                    If (acVportTblRec.Name = "TEST_VIEWPORT") Then
                        Dim acVportTblRecClone As ViewportTableRecord
                        acVportTblRecClone = acVportTblRec.Clone()

                        '' Add the new viewport to the Viewport table and the
transaction
                        acVportTbl.Add(acVportTblRecClone)
                        acVportTblRecClone.Name = "*Active"
                        acTrans.AddNewlyCreatedDBObject(acVportTblRecClone, True)
                    End If
                Next

                '' Update the display with the new tiled viewports arrangement
                acDoc.Editor.UpdateTiledViewportsFromDatabase()

                '' Commit the changes
                acTrans.Commit()
        End If

        '' Dispose of the transaction
    End Using
End Sub
```

C#

```csharp
using Autodesk.AutoCAD.ApplicationServices;
using Autodesk.AutoCAD.DatabaseServices;
using Autodesk.AutoCAD.Runtime;
using Autodesk.AutoCAD.Geometry;

[CommandMethod("CreateModelViewport")]
public static void CreateModelViewport()
{
    // Get the current database
    Document acDoc = Application.DocumentManager.MdiActiveDocument;
    Database acCurDb = acDoc.Database;

    // Start a transaction
    using (Transaction acTrans = acCurDb.TransactionManager.StartTransaction())
    {
        // Open the Viewport table for read
        ViewportTable acVportTbl;
        acVportTbl = acTrans.GetObject(acCurDb.ViewportTableId,
                                       OpenMode.ForRead) as ViewportTable;

        // Check to see if the named view 'TEST_VIEWPORT' exists
        if (acVportTbl.Has("TEST_VIEWPORT") == false)
        {
            // Open the View table for write
            acVportTbl.UpgradeOpen();

            // Add the new viewport to the Viewport table and the transaction
            ViewportTableRecord acVportTblRecLwr = new ViewportTableRecord();
            acVportTbl.Add(acVportTblRecLwr);
            acTrans.AddNewlyCreatedDBObject(acVportTblRecLwr, true);

            // Name the new viewport 'TEST_VIEWPORT' and assign it to be
            // the lower half of the drawing window
            acVportTblRecLwr.Name = "TEST_VIEWPORT";
            acVportTblRecLwr.LowerLeftCorner = new Point2d(0, 0);
            acVportTblRecLwr.UpperRightCorner = new Point2d(1, 0.5);

            // Add the new viewport to the Viewport table and the transaction
```

```
            ViewportTableRecord acVportTblRecUpr = new ViewportTableRecord();
            acVportTbl.Add(acVportTblRecUpr);
            acTrans.AddNewlyCreatedDBObject(acVportTblRecUpr, true);

            // Name the new viewport 'TEST_VIEWPORT' and assign it to be
            // the upper half of the drawing window
            acVportTblRecUpr.Name = "TEST_VIEWPORT";
            acVportTblRecUpr.LowerLeftCorner = new Point2d(0, 0.5);
            acVportTblRecUpr.UpperRightCorner = new Point2d(1, 1);

            // To assign the new viewports as the active viewports, the
            // viewports named '*Active' need to be removed and recreated
            // based on 'TEST_VIEWPORT'.

            // Step through each object in the symbol table
            foreach (ObjectId acObjId in acVportTbl)
            {
                // Open the object for read
                ViewportTableRecord acVportTblRec;
                acVportTblRec = acTrans.GetObject(acObjId,
                                                  OpenMode.ForRead) as
ViewportTableRecord;

                // See if it is one of the active viewports, and if so erase it
                if (acVportTblRec.Name == "*Active")
                {
                    acVportTblRec.UpgradeOpen();
                    acVportTblRec.Erase();
                }
            }

            // Clone the new viewports as the active viewports
            foreach (ObjectId acObjId in acVportTbl)
            {
                // Open the object for read
                ViewportTableRecord acVportTblRec;
                acVportTblRec = acTrans.GetObject(acObjId,
                                                  OpenMode.ForRead) as
ViewportTableRecord;

                // See if it is one of the active viewports, and if so erase it
                if (acVportTblRec.Name == "TEST_VIEWPORT")
                {
                    ViewportTableRecord acVportTblRecClone;
                    acVportTblRecClone = acVportTblRec.Clone() as
ViewportTableRecord;

                    // Add the new viewport to the Viewport table and the
transaction
                    acVportTbl.Add(acVportTblRecClone);
                    acVportTblRecClone.Name = "*Active";
                    acTrans.AddNewlyCreatedDBObject(acVportTblRecClone, true);
                }
            }

            // Update the display with the new tiled viewports arrangement
            acDoc.Editor.UpdateTiledViewportsFromDatabase();

            // Commit the changes
            acTrans.Commit();
        }

        // Dispose of the transaction
    }
}
```

```
Sub CreateModelViewport()
    ' Create a new viewport
    Dim vportObj As AcadViewport
    Set vportObj = ThisDrawing.Viewports.Add("TEST_VIEWPORT")

    ' Split vportObj into 2 horizontal windows
    vportObj.Split acViewport2Horizontal

    ' Now set vportObj to be the active viewport
    ThisDrawing.ActiveViewport = vportObj
End Sub
```

Make A Tiled Viewport Current

You enter points and select objects in the current viewport. To make a viewport current, use the CVPORT system variable and specify the viewport by its number that you want to set current.

You can iterate through existing viewports to find a particular viewport. To do this, identify the Viewport table records with the name "*Active" using the Name property.

Split a viewport, then iterate through the windows

This example splits the active viewport into two horizontal windows. It then iterates through all the tiled viewports in the drawing and displays the viewport name and the lower-left and upper-right corner for each viewport.

VB.NET

```
Imports Autodesk.AutoCAD.ApplicationServices
Imports Autodesk.AutoCAD.DatabaseServices
Imports Autodesk.AutoCAD.Runtime
Imports Autodesk.AutoCAD.Geometry

<CommandMethod("SplitAndIterateModelViewports")> _
Public Sub SplitAndIterateModelViewports()
  '' Get the current database
  Dim acDoc As Document = Application.DocumentManager.MdiActiveDocument
  Dim acCurDb As Database = acDoc.Database

  '' Start a transaction
  Using acTrans As Transaction = acCurDb.TransactionManager.StartTransaction()

      '' Open the Viewport table for write
      Dim acVportTbl As ViewportTable
      acVportTbl = acTrans.GetObject(acCurDb.ViewportTableId, _
                                     OpenMode.ForWrite)

      '' Open the active viewport for write
      Dim acVportTblRec As ViewportTableRecord
```

```vbnet
        acVportTblRec = acTrans.GetObject(acDoc.Editor.ActiveViewportId, _
                                          OpenMode.ForWrite)

        Dim acVportTblRecNew As ViewportTableRecord = New ViewportTableRecord()

        '' Add the new viewport to the Viewport table and the transaction
        acVportTbl.Add(acVportTblRecNew)
        acTrans.AddNewlyCreatedDBObject(acVportTblRecNew, True)

        '' Assign the name '*Active' to the new Viewport
        acVportTblRecNew.Name = "*Active"

        '' Use the existing lower left corner for the new viewport
        acVportTblRecNew.LowerLeftCorner = acVportTblRec.LowerLeftCorner
        '' Get half the X of the existing upper corner
        acVportTblRecNew.UpperRightCorner = New
Point2d(acVportTblRec.UpperRightCorner.X, _
                                                  acVportTblRec.LowerLeftCorne
r.Y + _
                                                  ((acVportTblRec.UpperRightCo
rner.Y - _
                                                  acVportTblRec.LowerLeftCor
ner.Y) / 2))
        '' Recalculate the corner of the active viewport
        acVportTblRec.LowerLeftCorner = New Point2d(acVportTblRec.LowerLeftCorner.X,
_
                                                  acVportTblRecNew.UpperRightCorne
r.Y)

        '' Update the display with the new tiled viewports arrangement
        acDoc.Editor.UpdateTiledViewportsFromDatabase()

        '' Step through each object in the symbol table
        For Each acObjId As ObjectId In acVportTbl
            '' Open the object for read
            Dim acVportTblRecCur As ViewportTableRecord
            acVportTblRecCur = acTrans.GetObject(acObjId, _
                                                 OpenMode.ForRead)

            If (acVportTblRecCur.Name = "*Active") Then
                Application.SetSystemVariable("CVPORT", acVportTblRecCur.Number)

                Application.ShowAlertDialog("Viewport: " & acVportTblRecCur.Number &
_
                                           " is now active." & _
                                           vbLf & "Lower left corner: " & _
                                           acVportTblRecCur.LowerLeftCorner.X & ",
" & _
                                           acVportTblRecCur.LowerLeftCorner.Y & _
                                           vbLf & "Upper right corner: " & _
                                           acVportTblRecCur.UpperRightCorner.X & ",
" & _
                                           acVportTblRecCur.UpperRightCorner.Y)
            End If
        Next

        '' Commit the changes and dispose of the transaction
        acTrans.Commit()
    End Using
End Sub
```

C#

```csharp
using Autodesk.AutoCAD.ApplicationServices;
```

```csharp
using Autodesk.AutoCAD.DatabaseServices;
using Autodesk.AutoCAD.Runtime;
using Autodesk.AutoCAD.Geometry;

[CommandMethod("SplitAndIterateModelViewports")]
public static void SplitAndIterateModelViewports()
{
  // Get the current database
  Document acDoc = Application.DocumentManager.MdiActiveDocument;
  Database acCurDb = acDoc.Database;

  // Start a transaction
  using (Transaction acTrans = acCurDb.TransactionManager.StartTransaction())
  {
      // Open the Viewport table for write
      ViewportTable acVportTbl;
      acVportTbl = acTrans.GetObject(acCurDb.ViewportTableId,
                                     OpenMode.ForWrite) as ViewportTable;

      // Open the active viewport for write
      ViewportTableRecord acVportTblRec;
      acVportTblRec = acTrans.GetObject(acDoc.Editor.ActiveViewportId,
                                        OpenMode.ForWrite) as ViewportTableRecord;

      ViewportTableRecord acVportTblRecNew = new ViewportTableRecord();

      // Add the new viewport to the Viewport table and the transaction
      acVportTbl.Add(acVportTblRecNew);
      acTrans.AddNewlyCreatedDBObject(acVportTblRecNew, true);

      // Assign the name '*Active' to the new Viewport
      acVportTblRecNew.Name = "*Active";

      // Use the existing lower left corner for the new viewport
      acVportTblRecNew.LowerLeftCorner = acVportTblRec.LowerLeftCorner;

      // Get half the X of the existing upper corner
      acVportTblRecNew.UpperRightCorner = new
Point2d(acVportTblRec.UpperRightCorner.X,
                                                acVportTblRec.LowerLeftCorne
r.Y +
                                                ((acVportTblRec.UpperRightCo
rner.Y -
                                                acVportTblRec.LowerLeftCor
ner.Y) / 2));

      // Recalculate the corner of the active viewport
      acVportTblRec.LowerLeftCorner = new Point2d(acVportTblRec.LowerLeftCorner.X,
                                                acVportTblRecNew.UpperRightCorne
r.Y);

      // Update the display with the new tiled viewports arrangement
      acDoc.Editor.UpdateTiledViewportsFromDatabase();

      // Step through each object in the symbol table
      foreach (ObjectId acObjId in acVportTbl)
      {
          // Open the object for read
          ViewportTableRecord acVportTblRecCur;
          acVportTblRecCur = acTrans.GetObject(acObjId,
                                               OpenMode.ForRead) as
ViewportTableRecord;

          if (acVportTblRecCur.Name == "*Active")
          {
```

```
                    Application.SetSystemVariable("CVPORT", acVportTblRecCur.Number);

                    Application.ShowAlertDialog("Viewport: " + acVportTblRecCur.Number +
                                                " is now active." +
                                                "\nLower left corner: " +
                                                acVportTblRecCur.LowerLeftCorner.X + ",
" +
                                                acVportTblRecCur.LowerLeftCorner.Y +
                                                "\nUpper right corner: " +
                                                acVportTblRecCur.UpperRightCorner.X + ",
" +
                                                acVportTblRecCur.UpperRightCorner.Y);
            }
        }

        // Commit the changes and dispose of the transaction
        acTrans.Commit();
    }
}
```

⊟ *VBA/ActiveX Code Reference*

```
Sub SplitandInterateModelViewports()
    ' Get the active viewport
    Dim vportObj As AcadViewport
    Set vportObj = ThisDrawing.ActiveViewport

    ' Split the viewport into 2 windows
    vportObj.Split acViewport2Horizontal

    ' Iterate through the viewports,
    ' highlighting each viewport and displaying
    ' the upper right and lower left corners
    ' for each.
    Dim vport As AcadViewport
    Dim LLCorner As Variant
    Dim URCorner As Variant

    For Each vport In ThisDrawing.Viewports
        ThisDrawing.ActiveViewport = vport
        LLCorner = vport.LowerLeftCorner
        URCorner = vport.UpperRightCorner
        MsgBox "Viewport: " & vport.Name & " is now active." & _
                vbCrLf & "Lower left corner: " & _
                LLCorner(0) & ", " & LLCorner(1) & vbCrLf & _
                "Upper right corner: " & _
                URCorner(0) & ", " & URCorner(1)
    Next vport
End Sub
```

Update the Geometry in the Document Window

Many of the actions you perform through the AutoCAD .NET API modify what is displayed in the drawing area. Not all of these actions immediately update the display of the drawing. This is designed so you can make several changes to the drawing without waiting for the display

to be updated after every single action. Instead, you can bundle your actions together and make a single call to update the display when you have finished.

The methods that will update the display are UpdateScreen (Application and Editor objects) and Regen (Editor object).

The UpdateScreen method redraws the application or document windows. The Regen method regenerates the graphical objects in the drawing window, and recomputes the screen coordinates and view resolution for all objects. It also re-indexes the drawing database for optimum display and object selection performance.

VB.NET

```
'' Redraw the drawing
Application.UpdateScreen()
Application.DocumentManager.MdiActiveDocument.Editor.UpdateScreen()

'' Regenerate the drawing
Application.DocumentManager.MdiActiveDocument.Editor.Regen()
```

C#

```
// Redraw the drawing
Application.UpdateScreen();
Application.DocumentManager.MdiActiveDocument.Editor.UpdateScreen();

// Regenerate the drawing
Application.DocumentManager.MdiActiveDocument.Editor.Regen();
```

⊟ *VBA/ActiveX Code Reference*

```
'' Redraw the drawing
ThisDrawing.Application.Update

'' Regenerate the drawing
ThisDrawing.Regen
```

Create, Open, Save, and Close Drawings

The DocumentCollection, Document, and Database objects provide access to the AutoCAD® file methods.

⊟ *VBA/ActiveX Code Reference*

VBA/ActiveX Class	.NET API Class
Documents collection	DocumentCollection
Document	Document and Database
Document.Saved	System.Convert.ToInt16(Application.GetSystemVariable("DBMOD"))

Topics in this section

Create and Open a Drawing

To create a new drawing or open an existing drawing, use the methods of the DocumentCollection object. The Add method creates a new drawing file based on a drawing template and adds that drawing to the DocumentCollection. The Open method opens an existing drawing file.

Create a new drawing

This example uses the Add method to create a new drawing based on the acad.dwt drawing template file.

VB.NET

```
Imports Autodesk.AutoCAD.ApplicationServices
Imports Autodesk.AutoCAD.DatabaseServices
Imports Autodesk.AutoCAD.Runtime

<CommandMethod("NewDrawing", CommandFlags.Session)> _
Public Sub NewDrawing()
  '' Specify the template to use, if the template is not found
  '' the default settings are used.
  Dim strTemplatePath As String = "acad.dwt"

  Dim acDocMgr As DocumentCollection = Application.DocumentManager
  Dim acDoc As Document = acDocMgr.Add(strTemplatePath)
  acDocMgr.MdiActiveDocument = acDoc
End Sub
```

C#

```
using Autodesk.AutoCAD.ApplicationServices;
using Autodesk.AutoCAD.DatabaseServices;
using Autodesk.AutoCAD.Runtime;

[CommandMethod("NewDrawing", CommandFlags.Session)]
public static void NewDrawing()
{
  // Specify the template to use, if the template is not found
  // the default settings are used.
  string strTemplatePath = "acad.dwt";

  DocumentCollection acDocMgr = Application.DocumentManager;
  Document acDoc = acDocMgr.Add(strTemplatePath);
  acDocMgr.MdiActiveDocument = acDoc;
}
```

⊟ *VBA/ActiveX Code Reference*

```
Sub NewDrawing()
    Dim strTemplatePath As String
    strTemplatePath = "acad.dwt"

    Dim docObj As AcadDocument
    Set docObj = ThisDrawing.Application.Documents.Add(strTemplatePath)
End Sub
```

Open an existing drawing

This example uses the Open method to open an existing drawing. Before opening the drawing, the code checks for the existence of the file before trying to open it.

VB.NET

```
Imports System.IO
Imports Autodesk.AutoCAD.ApplicationServices
Imports Autodesk.AutoCAD.DatabaseServices
Imports Autodesk.AutoCAD.Runtime

<CommandMethod("OpenDrawing", CommandFlags.Session)> _
Public Sub OpenDrawing()
  Dim strFileName As String = "C:\campus.dwg"
  Dim acDocMgr As DocumentCollection = Application.DocumentManager

  If (File.Exists(strFileName)) Then
      acDocMgr.Open(strFileName, False)
  Else
      acDocMgr.MdiActiveDocument.Editor.WriteMessage("File " & strFileName & _
                                                " does not exist.")
  End If
End Sub
```

C#

```
using System.IO;
using Autodesk.AutoCAD.ApplicationServices;
using Autodesk.AutoCAD.DatabaseServices;
using Autodesk.AutoCAD.Runtime;

[CommandMethod("OpenDrawing", CommandFlags.Session)]
public static void OpenDrawing()
{
  string strFileName = "C:\\campus.dwg";
  DocumentCollection acDocMgr = Application.DocumentManager;

  if (File.Exists(strFileName))
  {
      acDocMgr.Open(strFileName, false);
  }
  else
  {
      acDocMgr.MdiActiveDocument.Editor.WriteMessage("File " + strFileName +
                                                " does not exist.");
  }
}
```

```
Sub OpenDrawing()
    Dim dwgName As String
    dwgName = "c:\campus.dwg"
    If Dir(dwgName) <> "" Then
        ThisDrawing.Application.Documents.Open dwgName
    Else
        MsgBox "File " & dwgName & " does not exist."
    End If
End Sub
```

Save and Close a Drawing

Use the SaveAs method of the Database object to save the contents of a Database object. When using the SaveAs method, you can specify if the database should be renamed and if a backup of the drawing on disk should be renamed to a backup file by providing True for the bBakAndRename parameter. You can determine if a database is using a default name of Drawing1, Drawing2, etc by checking the value of the DWGTITLED system variable. If DWGTITLED is 0, the drawing has not been renamed.

Occasionally, you will want to check if the active drawing has any unsaved changes. It is a good idea to do this before you quit the AutoCAD session or start a new drawing. To check to see if a drawing file has been changed, you need to check the value of the DBMOD system variable.

Close a Drawing

The CloseAndDiscard or CloseAndSave methods of the Document object are used to close an open drawing and discard or save any changes made. You can use the CloseAll method of the DocumentCollection to close all open drawings in the AutoCAD.

Save the active drawing

This example saves the active drawing to "*c:\MyDrawing.dwg*" if it is currently not saved or under its current name.

VB.NET

```
Imports Autodesk.AutoCAD.ApplicationServices
Imports Autodesk.AutoCAD.Runtime

<CommandMethod("SaveActiveDrawing")> _
Public Sub SaveActiveDrawing()
  Dim acDoc As Document = Application.DocumentManager.MdiActiveDocument
  Dim strDWGName As String = acDoc.Name

  Dim obj As Object = Application.GetSystemVariable("DWGTITLED")

  '' Check to see if the drawing has been named
  If System.Convert.ToInt16(obj) = 0 Then
      '' If the drawing is using a default name (Drawing1, Drawing2, etc)
```

```
      '' then provide a new name
      strDWGName = "c:\MyDrawing.dwg"
   End If

   '' Save the active drawing
   acDoc.Database.SaveAs(strDWGName, True, DwgVersion.Current, _
                         acDoc.Database.SecurityParameters)
End Sub
```

C#

```
using Autodesk.AutoCAD.ApplicationServices;
using Autodesk.AutoCAD.Runtime;

[CommandMethod("SaveActiveDrawing")]
public static void SaveActiveDrawing()
{
   Document acDoc = Application.DocumentManager.MdiActiveDocument;
   string strDWGName = acDoc.Name;

   object obj = Application.GetSystemVariable("DWGTITLED");

   // Check to see if the drawing has been named
   if (System.Convert.ToInt16(obj) == 0)
   {
       // If the drawing is using a default name (Drawing1, Drawing2, etc)
       // then provide a new name
       strDWGName = "c:\\MyDrawing.dwg";
   }

   // Save the active drawing
   acDoc.Database.SaveAs(strDWGName, true, DwgVersion.Current,
                         acDoc.Database.SecurityParameters);
}
```

⊟ *VBA/ActiveX Code Reference*

```
Sub SaveActiveDrawing()
    ' Save the active drawing under a new name
    ThisDrawing.SaveAs "MyDrawing.dwg"
End Sub
```

Determine if a drawing has unsaved changes

This example checks to see if there are unsaved changes and verifies with the user that it is OK to save the drawing (if it is not OK, skip to the end). If OK, use the SaveAs method to save the current drawing, as shown here:

VB.NET

```
Imports Autodesk.AutoCAD.ApplicationServices
Imports Autodesk.AutoCAD.DatabaseServices
Imports Autodesk.AutoCAD.Runtime

<CommandMethod("DrawingSaved")> _
Public Sub DrawingSaved()
  Dim obj As Object = Application.GetSystemVariable("DBMOD")

  '' Check the value of DBMOD, if 0 then the drawing has not been changed
  If Not (System.Convert.ToInt16(obj) = 0) Then
```

```
        If MsgBox("Do you wish to save this drawing?", _
                  MsgBoxStyle.YesNo, _
                  "Save Drawing") = MsgBoxResult.Yes Then
            Dim acDoc As Document = Application.DocumentManager.MdiActiveDocument
            acDoc.Database.SaveAs(acDoc.Name, True, DwgVersion.Current, _
                                  acDoc.Database.SecurityParameters)
        End If
    End If
End Sub
```

C#

```csharp
using Autodesk.AutoCAD.ApplicationServices;
using Autodesk.AutoCAD.DatabaseServices;
using Autodesk.AutoCAD.Runtime;

[CommandMethod("DrawingSaved")]
public static void DrawingSaved()
{
  object obj = Application.GetSystemVariable("DBMOD");

  // Check the value of DBMOD, if 0 then the drawing has no unsaved changes
  if (System.Convert.ToInt16(obj) != 0)
  {
      if (System.Windows.Forms.MessageBox.Show("Do you wish to save this
drawing?",
                                  "Save Drawing",
                                  System.Windows.Forms.MessageBoxButtons.YesNo,
                                  System.Windows.Forms.MessageBoxIcon.Question)
                                  == System.Windows.Forms.DialogResult.Yes)
      {
          Document acDoc = Application.DocumentManager.MdiActiveDocument;
          acDoc.Database.SaveAs(acDoc.Name, true, DwgVersion.Current,
                                  acDoc.Database.SecurityParameters);
      }
  }
}
```

⊟ VBA/ActiveX Code Reference

```
Sub DrawingSaved()
    If Not (ThisDrawing.Saved) Then
        If MsgBox("Do you wish to save this drawing?", _
                  vbYesNo) = vbYes Then
            ThisDrawing.Save
        End If
    End If
End Sub
```

Work with No Documents Open

AutoCAD always starts up with a new or existing document open. It is possible, however, to close all documents during the current session.

If you close all the documents in the AutoCAD user interface, you will notice a few changes to the application window. The Quick Access toolbar and application menu offer limited

options. These limited options are related to creating and opening drawings, displaying the Sheet Set Manager, and recovering drawings. If the menu bar is displayed, simplified File, View, Window, and Help menus are also displayed. You will also notice that there is no command line.

When working in zero document state, you can do the following:

- You can create a new or open an existing document
- You can customize the zero document states of the application menu and menu bar
- You can shutdown AutoCAD

To react to AutoCAD when it enters zero document state, you should use the DocumentDestroyed event. The DocumentDestroyed event is triggered when an open document is closed. The document count when the last document is closed will be 1. Use the Count property of the DocumentManager to determine the number of open documents at the time the DocumentDestroyed event is triggered.

For more information on the using events in AutoCAD, see Use Events.

Customize the application menu

This example code uses the DocumentDestroyed event to monitor when the last drawing is closed and when zero document state is entered. Once zero document state is entered, the Opening event is registered with the application menu. When the application menu is clicked, the Opening event is triggered. During the Opening event, a new menu item is added to the application menu. The new menu item displays a message box.

NoteYou must reference AdWindows.dll to your project in order to use the following example code. AdWindows.dll contains the namespace used to customize the application menu and can be found in the install folder of AutoCAD or part of the ObjectARX SDK. You will also need to reference WindowsBase which can be found on the .NET tab of the Add Reference dialog box.

VB.NET

```
Imports System.Windows.Input

Imports Autodesk.Windows
Imports Autodesk.AutoCAD.Runtime
Imports Autodesk.AutoCAD.ApplicationServices

'' Create the command handler for the custom application menu item
Public Class MyCommandHandler
  Implements ICommand

  Event CanExecuteChanged(ByVal sender As Object, ByVal e As EventArgs) _
                                    Implements ICommand.CanExecuteChanged

  Function CanExecute(ByVal parameter As Object) As Boolean _
                                    Implements ICommand.CanExecute
      Return True
  End Function

  Sub Execute(ByVal parameter As Object) Implements ICommand.Execute
      Application.ShowAlertDialog("MyMenuItem has been clicked")
  End Sub
End Class
```

```vbnet
Public Class Chapter4
  ''Global var for ZeroDocState
  Dim acApMenuItem As ApplicationMenuItem = Nothing

  <CommandMethod("AddZeroDocEvent")> _
  Public Sub AddZeroDocEvent()
      '' Get the DocumentCollection and register the DocumentDestroyed event
      Dim acDocMgr As DocumentCollection = Application.DocumentManager
      AddHandler acDocMgr.DocumentDestroyed, AddressOf docDestroyed
  End Sub

  Public Sub docDestroyed(ByVal obj As Object, _
                          ByVal acDocDesEvtArgs As DocumentDestroyedEventArgs)
      '' Determine if the menu item already exists and the number of documents
open
      If Application.DocumentManager.Count = 1 And IsNothing(acApMenuItem) Then
          '' Add the event handler to watch for when the application menu is
opened
          '' AdWindows.dll must be referenced to the project
          AddHandler ComponentManager.ApplicationMenu.Opening, _
                  AddressOf ApplicationMenu_Opening
      End If
  End Sub

  Public Sub ApplicationMenu_Opening(ByVal sender As Object, _
                                     ByVal e As EventArgs)
      '' Check to see if the custom menu item was added previously
      If IsNothing(acApMenuItem) Then
          '' Get the application menu component
          Dim acApMenu As ApplicationMenu = ComponentManager.ApplicationMenu

          '' Create a new application menu item
          acApMenuItem = New ApplicationMenuItem()
          acApMenuItem.Text = "MyMenuItem"
          acApMenuItem.CommandHandler = New MyCommandHandler()

          '' Append the new menu item
          acApMenu.MenuContent.Items.Add(acApMenuItem)

          '' Remove the application menu Opening event handler
          RemoveHandler ComponentManager.ApplicationMenu.Opening, _
                      AddressOf ApplicationMenu_Opening
      End If
  End Sub
End Class
```

C#

```csharp
using Autodesk.Windows;
using Autodesk.AutoCAD.Runtime;
using Autodesk.AutoCAD.ApplicationServices;

// Create the command handler for the custom  application menu item
public class MyCommandHandler : System.Windows.Input.ICommand
{
  public bool CanExecute(object parameter)
  {
      return true;
  }

  public event EventHandler CanExecuteChanged;

  public void Execute(object parameter)
```

```csharp
        {
            Application.ShowAlertDialog("MyMenuItem has been clicked");
        }
    }

    class Chapter4
    {
        //Global var for ZeroDocState
        ApplicationMenuItem acApMenuItem = null;

        [CommandMethod("AddZeroDocEvent")]
        public void AddZeroDocEvent()
        {
            // Get the DocumentCollection and register the DocumentDestroyed event
            DocumentCollection acDocMgr = Application.DocumentManager;
            acDocMgr.DocumentDestroyed +=
                new DocumentDestroyedEventHandler(docDestroyed);
        }

        public void docDestroyed(object obj,
                                 DocumentDestroyedEventArgs acDocDesEvtArgs)
        {
            // Determine if the menu item already exists and the number of documents
open
            if (Application.DocumentManager.Count == 1 && acApMenuItem == null)
            {
                // Add the event handler to watch for when the application menu is
opened
                // AdWindows.dll must be referenced to the project
                ComponentManager.ApplicationMenu.Opening +=
                    new EventHandler<EventArgs>(ApplicationMenu_Opening);
            }
        }

        void ApplicationMenu_Opening(object sender, EventArgs e)
        {
            // Check to see if the custom menu item was added previously
            if (acApMenuItem == null)
            {
                // Get the application menu component
                ApplicationMenu acApMenu = ComponentManager.ApplicationMenu;

                // Create a new application menu item
                acApMenuItem = new ApplicationMenuItem();
                acApMenuItem.Text = "MyMenuItem";
                acApMenuItem.CommandHandler = new MyCommandHandler();

                // Append the new menu item
                acApMenu.MenuContent.Items.Add(acApMenuItem);

                // Remove the application menu Opening event handler
                ComponentManager.ApplicationMenu.Opening -=
                    new EventHandler<EventArgs>(ApplicationMenu_Opening);
            }
        }
    }
```

Lock and Unlock a Document

Requests to modify objects or access AutoCAD can occur in any context, and coming from any number of applications. To prevent conflicts with other requests, you are responsible for locking a document before you modify it. Failure to lock the document in certain contexts will cause a lock violation during the modification of the database. You want to lock the document when your application:

- Interacts with AutoCAD from a modeless dialog box
- Accesses a loaded document other than the current document
- Used as a COM server
- Registers a command with the Session command flag

For example, when adding an entity to Model or Paper space in a document other than the current document, the document needs to be locked. You use the LockDocument method of the Database object you want to lock. When the LockDocument method is called, a DocumentLock object is returned.

Once you are done modifying the locked database, you need to unlock the database. To unlock the database, you call the Dispose method of the DocumentLock object. You can also use the Using statement with the DocumentLock object, once the Using statement ends the database is unlocked.

NoteWhen working in the context of a command that does not use the Session command flag, you do not need to lock the database for the current document before it is modified.

Lock a database before modifying an object

This example creates a new document and then draws a circle in it. After the document is created, the database for the new document is locked and then a circle is added to it. After the circle is added, the database is unlocked and the associated document window is set current.

VB.NET

```
Imports Autodesk.AutoCAD.ApplicationServices
Imports Autodesk.AutoCAD.DatabaseServices
Imports Autodesk.AutoCAD.Runtime
Imports Autodesk.AutoCAD.Geometry

<CommandMethod("LockDoc", CommandFlags.Session)> _
Public Sub LockDoc()
  '' Create a new drawing
  Dim acDocMgr As DocumentCollection = Application.DocumentManager
  Dim acNewDoc As Document = acDocMgr.Add("acad.dwt")
  Dim acDbNewDoc As Database = acNewDoc.Database

  '' Lock the new document
  Using acLckDoc As DocumentLock = acNewDoc.LockDocument()

    '' Start a transaction in the new database
    Using acTrans As Transaction =
acDbNewDoc.TransactionManager.StartTransaction()

      '' Open the Block table for read
```

```vbnet
            Dim acBlkTbl As BlockTable
            acBlkTbl = acTrans.GetObject(acDbNewDoc.BlockTableId, _
                                         OpenMode.ForRead)

            '' Open the Block table record Model space for write
            Dim acBlkTblRec As BlockTableRecord
            acBlkTblRec = acTrans.GetObject(acBlkTbl(BlockTableRecord.ModelSpace), _
                                            OpenMode.ForWrite)

            '' Create a circle with a radius of 3 at 5,5
            Dim acCirc As Circle = New Circle()
            acCirc.SetDatabaseDefaults()
            acCirc.Center = New Point3d(5, 5, 0)
            acCirc.Radius = 3

            '' Add the new object to Model space and the transaction
            acBlkTblRec.AppendEntity(acCirc)
            acTrans.AddNewlyCreatedDBObject(acCirc, True)

            '' Save the new object to the database
            acTrans.Commit()
        End Using

        '' Unlock the document
    End Using

    '' Set the new document current
    acDocMgr.MdiActiveDocument = acNewDoc
End Sub
```

C#

```csharp
using Autodesk.AutoCAD.ApplicationServices;
using Autodesk.AutoCAD.DatabaseServices;
using Autodesk.AutoCAD.Runtime;
using Autodesk.AutoCAD.Geometry;

[CommandMethod("LockDoc", CommandFlags.Session)]
public static void LockDoc()
{
  // Create a new drawing
  DocumentCollection acDocMgr = Application.DocumentManager;
  Document acNewDoc = acDocMgr.Add("acad.dwt");
  Database acDbNewDoc = acNewDoc.Database;

  // Lock the new document
  using (DocumentLock acLckDoc = acNewDoc.LockDocument())
  {
      // Start a transaction in the new database
      using (Transaction acTrans =
acDbNewDoc.TransactionManager.StartTransaction())
      {
          // Open the Block table for read
          BlockTable acBlkTbl;
          acBlkTbl = acTrans.GetObject(acDbNewDoc.BlockTableId,
                                       OpenMode.ForRead) as BlockTable;

          // Open the Block table record Model space for write
          BlockTableRecord acBlkTblRec;
          acBlkTblRec = acTrans.GetObject(acBlkTbl[BlockTableRecord.ModelSpace],
                                          OpenMode.ForWrite) as BlockTableRecord;

          // Create a circle with a radius of 3 at 5,5
          Circle acCirc = new Circle();
```

```
        acCirc.SetDatabaseDefaults();
        acCirc.Center = new Point3d(5, 5, 0);
        acCirc.Radius = 3;

        // Add the new object to Model space and the transaction
        acBlkTblRec.AppendEntity(acCirc);
        acTrans.AddNewlyCreatedDBObject(acCirc, true);

        // Save the new object to the database
        acTrans.Commit();
    }

    // Unlock the document
  }

  // Set the new document current
  acDocMgr.MdiActiveDocument = acNewDoc;
}
```

Set AutoCAD Preferences

The .NET API does not contain any classes or methods to access the options in which are accessed through the AutoCAD Options dialog box. Access to these options is done through the ActiveX® Automation library. You use the COM object returned from the Preferences property of the Application object.

Once you have the Preferences COM object, you can then access the nine objects pertaining to the options, each representing a tab in the Options dialog box. These objects provide access to all of the registry-stored options in the Options dialog box. You can customize many of the AutoCAD settings by using properties found on these objects. These objects are

- PreferencesDisplay
- PreferencesDrafting
- PreferencesFiles
- PreferencesOpenSave
- PreferencesOutput
- PreferencesProfiles
- PreferencesSelection
- PreferencesSystem
- PreferencesUser

Access the Preferences object

The following example shows how to access the Preferences object through COM interop.

VB.NET

```
Dim acPrefComObj As AcadPreferences = Application.Preferences
```

C#

```
AcadPreferences acPrefComObj = (AcadPreferences)Application.Preferences;
```

```
Dim acadPref as AcadPreferences
Set acadPref = ThisDrawing.Application.Preferences
```

After you reference the Preferences object, you can then access any of the specific Preferences objects using the Display, Drafting, Files, OpenSave, Output, Profile, Selection, System, and User properties.

Set the crosshairs to full screen

VB.NET

```
Imports Autodesk.AutoCAD.ApplicationServices
Imports Autodesk.AutoCAD.Runtime
Imports Autodesk.AutoCAD.Interop

<CommandMethod("PrefsSetCursor")> _
Public Sub PrefsSetCursor()
  '' This example sets the crosshairs of the AutoCAD drawing cursor
  '' to full screen.

  '' Access the Preferences object
  Dim acPrefComObj As AcadPreferences = Application.Preferences

  '' Use the CursorSize property to set the size of the crosshairs
  acPrefComObj.Display.CursorSize = 100
End Sub
```

C#

```
using Autodesk.AutoCAD.ApplicationServices;
using Autodesk.AutoCAD.Runtime;
using Autodesk.AutoCAD.Interop;

[CommandMethod("PrefsSetCursor")]
public static void PrefsSetCursor()
{
  // This example sets the crosshairs for the drawing window
  // to full screen.

  // Access the Preferences object
  AcadPreferences acPrefComObj = (AcadPreferences)Application.Preferences;

  // Use the CursorSize property to set the size of the crosshairs
  acPrefComObj.Display.CursorSize = 100;
}
```

```
Sub PrefsSetCursor()
    ' This example sets the crosshairs of the AutoCAD drawing cursor
    ' to full screen

    ' Access the Preferences object
    Dim acadPref As AcadPreferences
    Set acadPref = ThisDrawing.Application.Preferences

    ' Use the CursorSize property to set the size of the crosshairs
    acadPref.Display.CursorSize = 100
```

```
End Sub
```

Display the screen menu and scroll bars

VB.NET

```
Imports Autodesk.AutoCAD.ApplicationServices
Imports Autodesk.AutoCAD.Runtime
Imports Autodesk.AutoCAD.Interop

<CommandMethod("PrefsSetDisplay")> _
Public Sub PrefsSetDisplay()
  '' This example enables the screen menu and disables the scrolls

  '' Access the Preferences object
  Dim acPrefComObj As AcadPreferences = Application.Preferences

  '' Display the screen menu
  acPrefComObj.Display.DisplayScreenMenu = True

  '' Disable the scroll bars
  acPrefComObj.Display.DisplayScrollBars = False
End Sub
```

C#

```
using Autodesk.AutoCAD.ApplicationServices;
using Autodesk.AutoCAD.Runtime;
using Autodesk.AutoCAD.Interop;

[CommandMethod("PrefsSetDisplay")]
public static void PrefsSetDisplay()
{
  // This example enables the screen menu and disables the scrolls

  // Access the Preferences object
  AcadPreferences acPrefComObj = (AcadPreferences)Application.Preferences;

  // Display the screen menu
  acPrefComObj.Display.DisplayScreenMenu = true;

  // Disable the scroll bars
  acPrefComObj.Display.DisplayScrollBars = false;
}
```

⊟ *VBA/ActiveX Code Reference*

```
Sub PrefsSetDisplay()
    ' This example enables the screen menu and disables the scrolls

    ' Access the Preferences object
    Dim acadPref As AcadPreferences
    Set acadPref = ThisDrawing.Application.Preferences

    ' Display the screen menu
    acadPref.Display.DisplayScreenMenu = True

    ' Disable the scroll bars
    acadPref.Display.DisplayScrollBars = False
End Sub
```

Topics in this section

-

Database Preferences

Along with application level preferences, there are drawing based preferences that are accessed with the Options dialog box that are stored in a drawing file. To access these stored settings, use the appropriate property on the Database object or use the GetSystemVariable and SetSystemVariable methods of the Application object.

⊟ VBA/ActiveX Cross Reference

VBA/ActiveX Class	.NET API Class
DatabasePreferences	Database and drawing based system variables

Set and Return System Variables

The Application object provides the SetSystemVariable and GetSystemVariable methods for setting and retrieving AutoCAD system variables. For example, to assign an integer to the MAXSORT system variable, use the following code:

VB.NET

```
'' Get the current value from a system variable
Dim nMaxSort as Integer = Application.GetSystemVariable("MAXSORT")

'' Set system variable to new value
Application.SetSystemVariable("MAXSORT", 100)
```

C#

```
// Get the current value from a system variable
int nMaxSort = System.Convert.ToInt32(Application.GetSystemVariable("MAXSORT"));

// Set system variable to new value
Application.SetSystemVariable("MAXSORT", 100);
```

⊟ *VBA/ActiveX Code Reference*

```
'' Get the current value from a system variable
Dim nMaxSort as Integer
nMaxSort = ThisDrawing.GetVariable("MAXSORT")

'' Set system variable to new value
```

```
ThisDrawing.SetVariable "MAXSORT", 100
```

Draw with Precision

With AutoCAD you can create your drawings with precise geometry without performing tedious calculations. Often you can specify precise points without knowing the coordinates. Without leaving the drawing screen, you can perform calculations on your drawing and display various types of status information.

Topics in this section

- [Adjust Snap and Grid Alignment](#)
- [Use Ortho Mode](#)
- [Calculate Points and Values](#)
- [Calculate Areas](#)

Adjust Snap and Grid Alignment

The grid is a visual guideline to measure distances, while Snap mode is used to restrict cursor movement. In addition to setting the spacing for the grid and Snap mode, you can adjust the rotation and type of snap used.

If you need to draw along a specific alignment or angle, you can rotate the snap angle. The center point of the snap angle rotation is the snap base point.

NoteAfter changing the snap and grid settings for the active viewport, you should use the UpdateTiledViewportsFromDatabase method of the Editor object to update the display of the drawing area.

Snap and grid do not affect points specified through the .NET API, but do affect points specified in the drawing area by the user if they are requested to enter input using methods such as GetPoint or GetEntity. See "Adjust Grid and Grid Snap" in the *AutoCAD User's Guide* for more information on using and setting snaps and grids.

Change the grid and snap settings

This example changes the snap base point to (1,1) and the snap rotation angle to 30 degrees. The grid is turned on the spacing is adjusted so that the changes are visible.

VB.NET

```
Imports Autodesk.AutoCAD.ApplicationServices
Imports Autodesk.AutoCAD.DatabaseServices
Imports Autodesk.AutoCAD.Runtime
Imports Autodesk.AutoCAD.Geometry
```

```vb
<CommandMethod("ChangeGridAndSnap")> _
Public Sub ChangeGridAndSnap()
  '' Get the current database
  Dim acDoc As Document = Application.DocumentManager.MdiActiveDocument
  Dim acCurDb As Database = acDoc.Database

  '' Start a transaction
  Using acTrans As Transaction = acCurDb.TransactionManager.StartTransaction()

      '' Open the active viewport
      Dim acVportTblRec As ViewportTableRecord
      acVportTblRec = acTrans.GetObject(acDoc.Editor.ActiveViewportId, _
                                        OpenMode.ForWrite)

      '' Turn on the grid for the active viewport
      acVportTblRec.GridEnabled = True

      '' Adjust the spacing of the grid to 1, 1
      acVportTblRec.GridIncrements = New Point2d(1, 1)

      '' Turn on the snap mode for the active viewport
      acVportTblRec.SnapEnabled = True

      '' Adjust the snap spacing to 0.5, 0.5
      acVportTblRec.SnapIncrements = New Point2d(0.5, 0.5)

      '' Change the snap base point to 1, 1
      acVportTblRec.SnapBase = New Point2d(1, 1)

      '' Change the snap rotation angle to 30 degrees (0.524 radians)
      acVportTblRec.SnapAngle = 0.524

      '' Update the display of the tiled viewport
      acDoc.Editor.UpdateTiledViewportsFromDatabase()

      '' Commit the changes and dispose of the transaction
      acTrans.Commit()
  End Using
End Sub
```

C#

```csharp
using Autodesk.AutoCAD.ApplicationServices;
using Autodesk.AutoCAD.DatabaseServices;
using Autodesk.AutoCAD.Runtime;
using Autodesk.AutoCAD.Geometry;

[CommandMethod("ChangeGridAndSnap")]
public static void ChangeGridAndSnap()
{
  // Get the current database
  Document acDoc = Application.DocumentManager.MdiActiveDocument;
  Database acCurDb = acDoc.Database;

  // Start a transaction
  using (Transaction acTrans = acCurDb.TransactionManager.StartTransaction())
  {
      // Open the active viewport
      ViewportTableRecord acVportTblRec;
      acVportTblRec = acTrans.GetObject(acDoc.Editor.ActiveViewportId,
                                        OpenMode.ForWrite) as ViewportTableRecord;

      // Turn on the grid for the active viewport
```

```
        acVportTblRec.GridEnabled = true;

        // Adjust the spacing of the grid to 1, 1
        acVportTblRec.GridIncrements = new Point2d(1, 1);

        // Turn on the snap mode for the active viewport
        acVportTblRec.SnapEnabled = true;

        // Adjust the snap spacing to 0.5, 0.5
        acVportTblRec.SnapIncrements = new Point2d(0.5, 0.5);

        // Change the snap base point to 1, 1
        acVportTblRec.SnapBase = new Point2d(1, 1);

        // Change the snap rotation angle to 30 degrees (0.524 radians)
        acVportTblRec.SnapAngle = 0.524;

        // Update the display of the tiled viewport
        acDoc.Editor.UpdateTiledViewportsFromDatabase();

        // Commit the changes and dispose of the transaction
        acTrans.Commit();
    }
}
```

⊟ *VBA/ActiveX Code Reference*

```
Sub ChangeGridAndSnap()
    ' Turn on the grid for the active viewport
    ThisDrawing.ActiveViewport.GridOn = True

    ' Adjust the spacing of the grid to 1, 1
    ThisDrawing.ActiveViewport.SetGridSpacing 1, 1

    ' Turn on the snap mode for the active viewport
    ThisDrawing.ActiveViewport.SnapOn = True

    ' Adjust the snap spacing to 0.5, 0.5
    ThisDrawing.ActiveViewport.SetSnapSpacing 0.5, 0.5

    ' Change the snap base point to 1, 1
    Dim newBasePoint(0 To 1) As Double
    newBasePoint(0) = 1: newBasePoint(1) = 1
    ThisDrawing.ActiveViewport.SnapBasePoint = newBasePoint

    ' Change the snap rotation angle to 30 degrees (0.524 radians)
    Dim rotationAngle As Double
    rotationAngle = 0.524
    ThisDrawing.ActiveViewport.SnapRotationAngle = rotationAngle

    ' Reset the viewport
    ThisDrawing.ActiveViewport = ThisDrawing.ActiveViewport
End Sub
```

Use Ortho Mode

As you draw lines or move objects, you can use Ortho mode to restrict the cursor to the horizontal or vertical axis. The orthogonal alignment is dependent on the current snap angle and UCS. Ortho mode works with activities that require you to specify a second point, such as when using the GetDistance or GetAngle methods. You can use Ortho not only to establish vertical or horizontal alignment but also to enforce parallelism or create regular offsets.

By allowing AutoCAD to impose orthogonal restraints, you can draw more quickly. For example, you can create a series of perpendicular lines by turning on Ortho mode before you start drawing. Because the lines are constrained to the horizontal and vertical axes, you can draw faster, knowing that the lines are perpendicular.

Ortho mode on   Ortho mode off
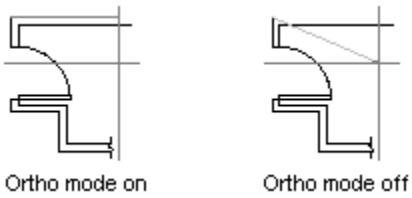
The following statements turn Ortho mode on. Unlike the grid and snap settings, Ortho mode is maintained in the Database object instead of the active viewport.

VB.NET

```
Application.DocumentManager.MdiActiveDocument.Database.Orthomode = True
```

C#

```
Application.DocumentManager.MdiActiveDocument.Database.Orthomode = true;
```

⊟ *VBA/ActiveX Code Reference*

```
ThisDrawing.ActiveViewport.OrthoOn = True
```

Calculate Points and Values

By using the methods provided by the Editor object and the Geometry and Runtime namespaces, you can quickly solve a mathematical problem or locate points in your drawing. Some of the available methods are:

- Get the distance between two 2D or 3D points using the GetDistanceTo and DistanceTo methods
- Get the angle from the X-axis using two 2D points using the GetVectorTo method with the Angle property of the returned value
- Convert an angle as a string to a real (double) value with the StringToAngle method

- Convert an angle from a real (double) value to a string with the AngleToString method
- Convert a distance from a string to a real (double) value with the StringToDistance method
- Find the distance between two points entered by the user with the GetDistance method

NoteThe .NET API does not contain methods to calculate a point based on a distance and angle (polar point) and for translating coordinates between different coordinate systems. If you need these utilities, you will want to utilize the PolarPoint and TranslateCoordinates methods from the ActiveX Automation library.

Get angle from X-axis

This example calculates a vector between two points and determines the angle from the X-axis.

VB.NET

```
Imports Autodesk.AutoCAD.ApplicationServices
Imports Autodesk.AutoCAD.Runtime
Imports Autodesk.AutoCAD.Geometry

<CommandMethod("AngleFromXAxis")> _
Public Sub AngleFromXAxis()
  Dim pt1 As Point2d = New Point2d(2, 5)
  Dim pt2 As Point2d = New Point2d(5, 2)

  Application.ShowAlertDialog("Angle from XAxis: " & _
                              pt1.GetVectorTo(pt2).Angle.ToString())
End Sub
```

C#

```
using Autodesk.AutoCAD.ApplicationServices;
using Autodesk.AutoCAD.Runtime;
using Autodesk.AutoCAD.Geometry;

[CommandMethod("AngleFromXAxis")]
public static void AngleFromXAxis()
{
  Point2d pt1 = new Point2d(2, 5);
  Point2d pt2 = new Point2d(5, 2);

  Application.ShowAlertDialog("Angle from XAxis: " +
                              pt1.GetVectorTo(pt2).Angle.ToString());
}
```

⊟ *VBA/ActiveX Code Reference*

```
Sub AngleFromXAxis()
    ' This example finds the angle, in radians, between the X axis
    ' and a line defined by two points.

    Dim pt1(0 To 2) As Double
    Dim pt2(0 To 2) As Double
    Dim retAngle As Double

    pt1(0) = 2: pt1(1) = 5: pt1(2) = 0
```

```
    pt2(0) = 5: pt2(1) = 2: pt2(2) = 0

    ' Return the angle
    retAngle = ThisDrawing.Utility.AngleFromXAxis(pt1, pt2)

    ' Display the angle found
    MsgBox "The angle in radians between the X axis is " & retAngle
End Sub
```

Calculate Polar Point

This example calculates a point based on a base point, an angle and distance.

VB.NET

```
Imports Autodesk.AutoCAD.ApplicationServices
Imports Autodesk.AutoCAD.Runtime
Imports Autodesk.AutoCAD.Geometry

Public Shared Function PolarPoints(ByVal pPt As Point2d, _
                                   ByVal dAng As Double, _
                                   ByVal dDist As Double)

  Return New Point2d(pPt.X + dDist * Math.Cos(dAng), _
                     pPt.Y + dDist * Math.Sin(dAng))
End Function

Public Shared Function PolarPoints(ByVal pPt As Point3d, _
                                   ByVal dAng As Double, _
                                   ByVal dDist As Double)

  Return New Point3d(pPt.X + dDist * Math.Cos(dAng), _
                     pPt.Y + dDist * Math.Sin(dAng), _
                     pPt.Z)
End Function

<CommandMethod("PolarPoints")> _
Public Sub PolarPoints()
  Dim pt1 As Point2d
  pt1 = PolarPoints(New Point2d(5, 2), 0.785398, 12)

  Application.ShowAlertDialog(vbLf & "PolarPoint: " & _
                              vbLf & "X = " & pt1.X & _
                              vbLf & "Y = " & pt1.Y)

  Dim pt2 As Point3d
  pt2 = PolarPoints(New Point3d(5, 2, 0), 0.785398, 12)

  Application.ShowAlertDialog(vbLf & "PolarPoint: " & _
                              vbLf & "X = " & pt2.X & _
                              vbLf & "Y = " & pt2.Y & _
                              vbLf & "Z = " & pt2.Z)
End Sub
```

C#

```
using Autodesk.AutoCAD.ApplicationServices;
using Autodesk.AutoCAD.Runtime;
using Autodesk.AutoCAD.Geometry;

static Point2d PolarPoints(Point2d pPt, double dAng, double dDist)
{
  return new Point2d(pPt.X + dDist * Math.Cos(dAng),
```

```
                          pPt.Y + dDist * Math.Sin(dAng));
}

static Point3d PolarPoints(Point3d pPt, double dAng, double dDist)
{
  return new Point3d(pPt.X + dDist * Math.Cos(dAng),
                     pPt.Y + dDist * Math.Sin(dAng),
                     pPt.Z);
}

[CommandMethod("PolarPoints")]
public static void PolarPoints()
{
  Point2d pt1 = PolarPoints(new Point2d(5, 2), 0.785398, 12);

  Application.ShowAlertDialog("\nPolarPoint: " +
                              "\nX = " + pt1.X +
                              "\nY = " + pt1.Y);

  Point3d pt2 = PolarPoints(new Point3d(5, 2, 0), 0.785398, 12);

  Application.ShowAlertDialog("\nPolarPoint: " +
                              "\nX = " + pt2.X +
                              "\nY = " + pt2.Y +
                              "\nZ = " + pt2.Z);
}
```

VBA/ActiveX Code Reference

```
Sub PolarPoints()
    ' This example finds the coordinate of a point that is a given
    ' distance and angle from a base point.

    Dim polarPnt As Variant
    Dim basePnt(0 To 2) As Double
    Dim angle As Double
    Dim distance As Double

    basePnt(0) = 2#: basePnt(1) = 2#: basePnt(2) = 0#
    angle = 0.785398
    distance = 6
    polarPnt = ThisDrawing.Utility.PolarPoint(basePnt, angle, distance)

    MsgBox vbLf + "PolarPoint: " + _
           vbLf + "X = " + CStr(polarPnt(0)) + _
           vbLf + "Y = " + CStr(polarPnt(1)) + _
           vbLf + "Z = " + CStr(polarPnt(2))
End Sub
```

Find the distance between two points with the GetDistance method

This example uses the GetDistance method to obtain two points and displays the calculated distance.

VB.NET

```
Imports Autodesk.AutoCAD.ApplicationServices
Imports Autodesk.AutoCAD.EditorInput
Imports Autodesk.AutoCAD.Runtime

<CommandMethod("GetDistanceBetweenTwoPoints")> _
```

```
Public Sub GetDistanceBetweenTwoPoints()
  Dim acDoc As Document = Application.DocumentManager.MdiActiveDocument

  Dim pDblRes As PromptDoubleResult
  pDblRes = acDoc.Editor.GetDistance(vbLf & "Pick two points: ")

  Application.ShowAlertDialog(vbLf & "Distance between points: " & _
                              pDblRes.Value.ToString())
End Sub
```

C#

```
using Autodesk.AutoCAD.ApplicationServices;
using Autodesk.AutoCAD.EditorInput;
using Autodesk.AutoCAD.Runtime;

[CommandMethod("GetDistanceBetweenTwoPoints")]
public static void GetDistanceBetweenTwoPoints()
{
  Document acDoc = Application.DocumentManager.MdiActiveDocument;

  PromptDoubleResult pDblRes;
  pDblRes = acDoc.Editor.GetDistance("\nPick two points: ");

  Application.ShowAlertDialog("\nDistance between points: " +
                              pDblRes.Value.ToString());
}
```

☐ *VBA/ActiveX Code Reference*

```
Sub GetDistanceBetweenTwoPoints()
    Dim returnDist As Double

    ' Return the value entered by user. A prompt is provided.
    returnDist = ThisDrawing.Utility.GetDistance(, "Pick two points.")

    MsgBox "The distance between the two points is: " & returnDist
End Sub
```

Calculate Areas

You can find the area of an arc, circle, ellipse, lightweight polyline, polyline, region, hatch, planar-closed spline or any other entity that is derived from the base type of Curve by using the Area property.

If you need to calculate the combined area of more than one object, you can keep a running total as you add or use the Boolean method on a series of regions to obtain a single region representing the desired area. From this single region you can use the Area property to obtain its area.

The calculated area differs according to the type of object you query. For an explanation of how area is calculated for each object type, see "Obtain Area and Mass Properties Information" in the *AutoCAD User's Guide*.

Topics in this section

- [Calculate a Defined Area](#)

Calculate a Defined Area

If the area you want to calculate is based on user specified points, you might consider creating an in memory object such as a lightweight polyline, and then query of the area of the object before discarding it. The following steps explain how you might accomplish this:

1. Use the GetPoint method in a loop to obtain the points from the user.
2. Create a lightweight polyline from the points provided by the user. Create a new Polyline object. Specify the number of vertices and the points they should be at.
3. Use the Area property to obtain the area of the newly created polyline.
4. Dispose of the polyline using its Dispose method.

Calculate the area defined by points entered from the user

This example prompts the user to enter five points. A polyline is then created out of the points entered. The polyline is closed, and the area of the polyline is displayed in a message box. Since the polyline is not added to a block, it needs to be disposed before the command ends.

VB.NET

```
Imports Autodesk.AutoCAD.ApplicationServices
Imports Autodesk.AutoCAD.DatabaseServices
Imports Autodesk.AutoCAD.Geometry
Imports Autodesk.AutoCAD.EditorInput
Imports Autodesk.AutoCAD.Runtime

<CommandMethod("CalculateDefinedArea")> _
Public Sub CalculateDefinedArea()
  '' Prompt the user for 5 points
  Dim acDoc As Document = Application.DocumentManager.MdiActiveDocument

  Dim pPtRes As PromptPointResult
  Dim colPt As Point2dCollection = New Point2dCollection
  Dim pPtOpts As PromptPointOptions = New PromptPointOptions("")

  '' Prompt for the first point
  pPtOpts.Message = vbLf & "Specify first point: "
  pPtRes = acDoc.Editor.GetPoint(pPtOpts)
  colPt.Add(New Point2d(pPtRes.Value.X, pPtRes.Value.Y))

  '' Exit if the user presses ESC or cancels the command
  If pPtRes.Status = PromptStatus.Cancel Then Exit Sub

  Dim nCounter As Integer = 1

  While (nCounter <= 4)
      '' Prompt for the next points
      Select Case nCounter
          Case 1
              pPtOpts.Message = vbLf & "Specify second point: "
```

```
        Case 2
            pPtOpts.Message = vbLf & "Specify third point: "
        Case 3
            pPtOpts.Message = vbLf & "Specify fourth point: "
        Case 4
            pPtOpts.Message = vbLf & "Specify fifth point: "
    End Select

    '' Use the previous point as the base point
    pPtOpts.UseBasePoint = True
    pPtOpts.BasePoint = pPtRes.Value

    pPtRes = acDoc.Editor.GetPoint(pPtOpts)
    colPt.Add(New Point2d(pPtRes.Value.X, pPtRes.Value.Y))

    If pPtRes.Status = PromptStatus.Cancel Then Exit Sub

    '' Increment the counter
    nCounter = nCounter + 1
  End While

  '' Create a polyline with 5 points
  Using acPoly As Polyline = New Polyline()
      acPoly.AddVertexAt(0, colPt(0), 0, 0, 0)
      acPoly.AddVertexAt(1, colPt(1), 0, 0, 0)
      acPoly.AddVertexAt(2, colPt(2), 0, 0, 0)
      acPoly.AddVertexAt(3, colPt(3), 0, 0, 0)
      acPoly.AddVertexAt(4, colPt(4), 0, 0, 0)

      '' Close the polyline
      acPoly.Closed = True

      '' Query the area of the polyline
      Application.ShowAlertDialog("Area of polyline: " & _
                                  acPoly.Area.ToString())

      '' Dispose of the polyline
  End Using
End Sub
```

C#

```csharp
using Autodesk.AutoCAD.ApplicationServices;
using Autodesk.AutoCAD.DatabaseServices;
using Autodesk.AutoCAD.Geometry;
using Autodesk.AutoCAD.EditorInput;
using Autodesk.AutoCAD.Runtime;

[CommandMethod("CalculateDefinedArea")]
public static void CalculateDefinedArea()
{
  // Prompt the user for 5 points
  Document acDoc = Application.DocumentManager.MdiActiveDocument;

  PromptPointResult pPtRes;
  Point2dCollection colPt = new Point2dCollection();
  PromptPointOptions pPtOpts = new PromptPointOptions("");

  // Prompt for the first point
  pPtOpts.Message = "\nSpecify first point: ";
  pPtRes = acDoc.Editor.GetPoint(pPtOpts);
  colPt.Add(new Point2d(pPtRes.Value.X, pPtRes.Value.Y));

  // Exit if the user presses ESC or cancels the command
```

```
        if (pPtRes.Status == PromptStatus.Cancel) return;

    int nCounter = 1;

    while (nCounter <= 4)
    {
        // Prompt for the next points
        switch(nCounter)
        {
            case 1:
                pPtOpts.Message = "\nSpecify second point: ";
                break;
            case 2:
                pPtOpts.Message = "\nSpecify third point: ";
                break;
            case 3:
                pPtOpts.Message = "\nSpecify fourth point: ";
                break;
            case 4:
                pPtOpts.Message = "\nSpecify fifth point: ";
                break;
        }

        // Use the previous point as the base point
        pPtOpts.UseBasePoint = true;
        pPtOpts.BasePoint = pPtRes.Value;

        pPtRes = acDoc.Editor.GetPoint(pPtOpts);
        colPt.Add(new Point2d(pPtRes.Value.X, pPtRes.Value.Y));

        if (pPtRes.Status == PromptStatus.Cancel) return;

        // Increment the counter
        nCounter = nCounter + 1;
    }

    // Create a polyline with 5 points
    using (Polyline acPoly = new Polyline())
    {
        acPoly.AddVertexAt(0, colPt[0], 0, 0, 0);
        acPoly.AddVertexAt(1, colPt[1], 0, 0, 0);
        acPoly.AddVertexAt(2, colPt[2], 0, 0, 0);
        acPoly.AddVertexAt(3, colPt[3], 0, 0, 0);
        acPoly.AddVertexAt(4, colPt[4], 0, 0, 0);

        // Close the polyline
        acPoly.Closed = true;

        // Query the area of the polyline
        Application.ShowAlertDialog("Area of polyline: " +
                                acPoly.Area.ToString());

        // Dispose of the polyline
    }
}
```

This example prompts the user to enter five points. A polyline is then created out of the points entered. The polyline is closed, and the area of the polyline is displayed in a message box. Unlike the .NET API examples, the polyline is not created in memory but as a database

resident object and added to Model space. So after the area of the polyline is obtained, it is removed.

```
Sub CalculateDefinedArea()
    Dim p1 As Variant
    Dim p2 As Variant
    Dim p3 As Variant
    Dim p4 As Variant
    Dim p5 As Variant

    ' Get the points from the user
    p1 = ThisDrawing.Utility.GetPoint(, vbCrLf & "Specify first point: ")
    p2 = ThisDrawing.Utility.GetPoint(p1, vbCrLf & "Specify second point: ")
    p3 = ThisDrawing.Utility.GetPoint(p2, vbCrLf & "Specify third point: ")
    p4 = ThisDrawing.Utility.GetPoint(p3, vbCrLf & "Specify fourth point: ")
    p5 = ThisDrawing.Utility.GetPoint(p4, vbCrLf & "Specify fifth point: ")

    ' Create the 2D polyline from the points
    Dim polyObj As AcadLWPolyline
    Dim vertices(0 To 9) As Double
    vertices(0) = p1(0): vertices(1) = p1(1)
    vertices(2) = p2(0): vertices(3) = p2(1)
    vertices(4) = p3(0): vertices(5) = p3(1)
    vertices(6) = p4(0): vertices(7) = p4(1)
    vertices(8) = p5(0): vertices(9) = p5(1)
    Set polyObj = ThisDrawing.ModelSpace.AddLightWeightPolyline _
                        (vertices)
    polyObj.Closed = True

    ' Display the area for the polyline
    MsgBox "The area defined by the points is " & _
            polyObj.Area, , "Calculate Defined Area"

    ' Remove the polyline
    polyObj.Delete
End Sub
```

Prompt for User Input

The Editor object, which is a child of the Document object, defines the user input methods. The user input methods display a prompt on the AutoCAD command line or in a dynamic input tooltip, and request input of various types. This type of user input is most useful for interactive input of screen coordinates, entity selection, and short-string or numeric values. If your application requires the input of numerous options or values, a Windows form may be more appropriate than individual prompts.

Each user input method displays an optional prompt on the command line and returns a value specific to the type of input requested. For example, GetString returns a PromptResult which allows you to determine the status of the GetString method and retrieve the string the user entered. Each one of the user input methods has a specific return value.

The input methods accept a string for the prompt message to be displayed or a specific object type which controls the input from the user. These object types let you control things such as NULL input (pressing Enter), base point, input of zero or negative numbers, and input of arbitrary text values.

To force the prompt to be displayed on a line by itself, use the carriage return/linefeed constant (vbCrLf) or linefeed constant (vbLf) at the beginning of your prompt strings when using VB.NET, or "\n" with strings in C#.

Topics in this section

- [GetString Method](#)
- [GetPoint Method](#)
- [GetKeywords Method](#)
- [Control User Input](#)

GetString Method

The GetString method prompts the user for the input of a string at the Command prompt. The PromptStringOptions object allows you to control the input entered and how the prompt message appears. The AllowSpaces property of the PromptStringOptions object controls if spaces are allowed or not at the prompt. If set to false, pressing the Spacebar terminates the input.

Get a string value from the user at the AutoCAD command line

The following example displays the Enter Your Name prompt, and requires that the input from the user be terminated by pressing Enter (spaces are allowed in the input string). The entered string is displayed in a message box.

VB.NET

```
Imports Autodesk.AutoCAD.ApplicationServices
Imports Autodesk.AutoCAD.EditorInput
Imports Autodesk.AutoCAD.Runtime

<CommandMethod("GetStringFromUser")> _
Public Sub GetStringFromUser()
  Dim acDoc As Document = Application.DocumentManager.MdiActiveDocument

  Dim pStrOpts As PromptStringOptions = New PromptStringOptions(vbLf & _
                                                "Enter your name: ")
  pStrOpts.AllowSpaces = True
  Dim pStrRes As PromptResult = acDoc.Editor.GetString(pStrOpts)

  Application.ShowAlertDialog("The name entered was: " & _
                             pStrRes.StringResult)
End Sub
```

C#

```
using Autodesk.AutoCAD.ApplicationServices;
using Autodesk.AutoCAD.EditorInput;
using Autodesk.AutoCAD.Runtime;

[CommandMethod("GetStringFromUser")]
public static void GetStringFromUser()
```

```
{
    Document acDoc = Application.DocumentManager.MdiActiveDocument;

    PromptStringOptions pStrOpts = new PromptStringOptions("\nEnter your name: ");
    pStrOpts.AllowSpaces = true;
    PromptResult pStrRes = acDoc.Editor.GetString(pStrOpts);

    Application.ShowAlertDialog("The name entered was: " +
                                pStrRes.StringResult);
}
```

⊟ *VBA/ActiveX Code Reference*

```
Sub GetStringFromUser()
    Dim retVal As String
    retVal = ThisDrawing.Utility.GetString _
                                (1, vbCrLf & "Enter your name: ")
    MsgBox "The name entered was: " & retVal
End Sub
```

GetPoint Method

The GetPoint method prompts the user to specify a point at the Command prompt. The PromptPointOptions object allows you to control the input entered and how the prompt message appears. The UseBasePoint and BasePoint properties of the PromptPointOptions object controls if a rubber-band line is drawn from a base point. The Keywords property of the PromptPointOptions object allows you to define keywords that can be entered at the Command prompt in addition to specifying a point.

Get a point selected by the user

The following example prompts the user for two points, then draws a line using those points as the start point and endpoint.

VB.NET
```
Imports Autodesk.AutoCAD.ApplicationServices
Imports Autodesk.AutoCAD.DatabaseServices
Imports Autodesk.AutoCAD.EditorInput
Imports Autodesk.AutoCAD.Geometry
Imports Autodesk.AutoCAD.Runtime

<CommandMethod("GetPointsFromUser")> _
Public Sub GetPointsFromUser()
  '' Get the current database and start the Transaction Manager
  Dim acDoc As Document = Application.DocumentManager.MdiActiveDocument
  Dim acCurDb As Database = acDoc.Database

  Dim pPtRes As PromptPointResult
  Dim pPtOpts As PromptPointOptions = New PromptPointOptions("")

  '' Prompt for the start point
  pPtOpts.Message = vbLf & "Enter the start point of the line: "
  pPtRes = acDoc.Editor.GetPoint(pPtOpts)
  Dim ptStart As Point3d = pPtRes.Value
```

```vbnet
    '' Exit if the user presses ESC or cancels the command
    If pPtRes.Status = PromptStatus.Cancel Then Exit Sub

    '' Prompt for the end point
    pPtOpts.Message = vbLf & "Enter the end point of the line: "
    pPtOpts.UseBasePoint = True
    pPtOpts.BasePoint = ptStart
    pPtRes = acDoc.Editor.GetPoint(pPtOpts)
    Dim ptEnd As Point3d = pPtRes.Value

    If pPtRes.Status = PromptStatus.Cancel Then Exit Sub

    '' Start a transaction
    Using acTrans As Transaction = acCurDb.TransactionManager.StartTransaction()

        Dim acBlkTbl As BlockTable
        Dim acBlkTblRec As BlockTableRecord

        '' Open Model space for write
        acBlkTbl = acTrans.GetObject(acCurDb.BlockTableId, _
                                     OpenMode.ForRead)

        acBlkTblRec = acTrans.GetObject(acBlkTbl(BlockTableRecord.ModelSpace), _
                                        OpenMode.ForWrite)

        '' Define the new line
        Dim acLine As Line = New Line(ptStart, ptEnd)
        acLine.SetDatabaseDefaults()

        '' Add the line to the drawing
        acBlkTblRec.AppendEntity(acLine)
        acTrans.AddNewlyCreatedDBObject(acLine, True)

        '' Zoom to the extents or limits of the drawing
        acDoc.SendStringToExecute("._zoom _all ", True, False, False)

        '' Commit the changes and dispose of the transaction
        acTrans.Commit()
    End Using
End Sub
```

C#

```csharp
using Autodesk.AutoCAD.ApplicationServices;
using Autodesk.AutoCAD.DatabaseServices;
using Autodesk.AutoCAD.EditorInput;
using Autodesk.AutoCAD.Geometry;
using Autodesk.AutoCAD.Runtime;

[CommandMethod("GetPointsFromUser")]
public static void GetPointsFromUser()
{
  // Get the current database and start the Transaction Manager
  Document acDoc = Application.DocumentManager.MdiActiveDocument;
  Database acCurDb = acDoc.Database;

  PromptPointResult pPtRes;
  PromptPointOptions pPtOpts = new PromptPointOptions("");

  // Prompt for the start point
  pPtOpts.Message = "\nEnter the start point of the line: ";
  pPtRes = acDoc.Editor.GetPoint(pPtOpts);
  Point3d ptStart = pPtRes.Value;
```

```csharp
    // Exit if the user presses ESC or cancels the command
    if (pPtRes.Status == PromptStatus.Cancel) return;

    // Prompt for the end point
    pPtOpts.Message = "\nEnter the end point of the line: ";
    pPtOpts.UseBasePoint = true;
    pPtOpts.BasePoint = ptStart;
    pPtRes = acDoc.Editor.GetPoint(pPtOpts);
    Point3d ptEnd = pPtRes.Value;

    if (pPtRes.Status == PromptStatus.Cancel) return;

    // Start a transaction
    using (Transaction acTrans = acCurDb.TransactionManager.StartTransaction())
    {
        BlockTable acBlkTbl;
        BlockTableRecord acBlkTblRec;

        // Open Model space for write
        acBlkTbl = acTrans.GetObject(acCurDb.BlockTableId,
                                     OpenMode.ForRead) as BlockTable;

        acBlkTblRec = acTrans.GetObject(acBlkTbl[BlockTableRecord.ModelSpace],
                                        OpenMode.ForWrite) as BlockTableRecord;

        // Define the new line
        Line acLine = new Line(ptStart, ptEnd);
        acLine.SetDatabaseDefaults();

        // Add the line to the drawing
        acBlkTblRec.AppendEntity(acLine);
        acTrans.AddNewlyCreatedDBObject(acLine, true);

        // Zoom to the extents or limits of the drawing
        acDoc.SendStringToExecute("._zoom _all ", true, false, false);

        // Commit the changes and dispose of the transaction
        acTrans.Commit();
    }
}
```

⊟ *VBA/ActiveX Code Reference*

```vba
Sub GetPointsFromUser()
    Dim startPnt As Variant
    Dim endPnt As Variant
    Dim prompt1 As String
    Dim prompt2 As String

    prompt1 = vbCrLf & "Enter the start point of the line: "
    prompt2 = vbCrLf & "Enter the end point of the line: "

    ' Get the first point without entering a base point
    startPnt = ThisDrawing.Utility.GetPoint(, prompt1)

    ' Use the point entered above as the base point
    endPnt = ThisDrawing.Utility.GetPoint(startPnt, prompt2)

    ' Create a line using the two points entered
    ThisDrawing.ModelSpace.AddLine startPnt, endPnt
    ThisDrawing.Application.ZoomAll
End Sub
```

GetKeywords Method

The GetKeywords method prompts the user for input of a keyword at the Command prompt. The PromptKeywordOptions object allows you to control the input entered and how the prompt message appears. The Keywords property of the PromptKeywordOptions object allows you to define keywords that can be entered at the Command prompt.

Get a keyword from the user at the AutoCAD command line

The following example forces the user to enter a keyword by setting the property AllowNone to false, which disallows NULL input (pressing Enter). The Keywords property is used to add the valid keywords allowed.

VB.NET

```
Imports Autodesk.AutoCAD.ApplicationServices
Imports Autodesk.AutoCAD.EditorInput
Imports Autodesk.AutoCAD.Runtime

<CommandMethod("GetKeywordFromUser")> _
Public Sub GetKeywordFromUser()
  Dim acDoc As Document = Application.DocumentManager.MdiActiveDocument

  Dim pKeyOpts As PromptKeywordOptions = New PromptKeywordOptions("")
  pKeyOpts.Message = vbLf & "Enter an option "
  pKeyOpts.Keywords.Add("Line")
  pKeyOpts.Keywords.Add("Circle")
  pKeyOpts.Keywords.Add("Arc")
  pKeyOpts.AllowNone = False

  Dim pKeyRes As PromptResult = acDoc.Editor.GetKeywords(pKeyOpts)

  Application.ShowAlertDialog("Entered keyword: " & _
                                pKeyRes.StringResult)
End Sub
```

C#

```
using Autodesk.AutoCAD.ApplicationServices;
using Autodesk.AutoCAD.EditorInput;
using Autodesk.AutoCAD.Runtime;

[CommandMethod("GetKeywordFromUser")]
public static void GetKeywordFromUser()
{
  Document acDoc = Application.DocumentManager.MdiActiveDocument;

  PromptKeywordOptions pKeyOpts = new PromptKeywordOptions("");
  pKeyOpts.Message = "\nEnter an option ";
  pKeyOpts.Keywords.Add("Line");
  pKeyOpts.Keywords.Add("Circle");
  pKeyOpts.Keywords.Add("Arc");
  pKeyOpts.AllowNone = false;

  PromptResult pKeyRes = acDoc.Editor.GetKeywords(pKeyOpts);
```

```
    Application.ShowAlertDialog("Entered keyword: " +
                               pKeyRes.StringResult);
}
```

```
Sub GetKeywordFromUser()
    Dim keyWord As String
    ThisDrawing.Utility.InitializeUserInput 1, "Line Circle Arc"
    keyWord = ThisDrawing.Utility.GetKeyword _
              (vbCrLf & "Enter an option [Line/Circle/Arc]: ")
    MsgBox keyWord, , "GetKeyword Example"
End Sub
```

A more user-friendly keyword prompt is one that provides a default value if the user presses Enter (NULL input). Notice the minor modifications to the following example.

VB.NET

```
Imports Autodesk.AutoCAD.ApplicationServices
Imports Autodesk.AutoCAD.EditorInput
Imports Autodesk.AutoCAD.Runtime

<CommandMethod("GetKeywordFromUser2")> _
Public Sub GetKeywordFromUser2()
  Dim acDoc As Document = Application.DocumentManager.MdiActiveDocument

  Dim pKeyOpts As PromptKeywordOptions = New PromptKeywordOptions("")
  pKeyOpts.Message = vbLf & "Enter an option "
  pKeyOpts.Keywords.Add("Line")
  pKeyOpts.Keywords.Add("Circle")
  pKeyOpts.Keywords.Add("Arc")
  pKeyOpts.Keywords.Default = "Arc"
  pKeyOpts.AllowNone = True

  Dim pKeyRes As PromptResult = acDoc.Editor.GetKeywords(pKeyOpts)

  Application.ShowAlertDialog("Entered keyword: " & _
                             pKeyRes.StringResult)
End Sub
```

C#

```
using Autodesk.AutoCAD.ApplicationServices;
using Autodesk.AutoCAD.EditorInput;
using Autodesk.AutoCAD.Runtime;

[CommandMethod("GetKeywordFromUser2")]
public static void GetKeywordFromUser2()
{
  Document acDoc = Application.DocumentManager.MdiActiveDocument;

  PromptKeywordOptions pKeyOpts = new PromptKeywordOptions("");
  pKeyOpts.Message = "\nEnter an option ";
  pKeyOpts.Keywords.Add("Line");
  pKeyOpts.Keywords.Add("Circle");
  pKeyOpts.Keywords.Add("Arc");
  pKeyOpts.Keywords.Default = "Arc";
  pKeyOpts.AllowNone = true;

  PromptResult pKeyRes = acDoc.Editor.GetKeywords(pKeyOpts);
```

```
        Application.ShowAlertDialog("Entered keyword: " +
                                    pKeyRes.StringResult);
}
```

```
Sub GetKeywordFromUser2()
    Dim keyWord As String
    ThisDrawing.Utility.InitializeUserInput 0, "Line Circle Arc"
    keyWord = ThisDrawing.Utility.GetKeyword _
            (vbCrLf & "Enter an option [Line/Circle/Arc] <Arc>: ")
    If keyWord = "" Then keyWord = "Arc"
    MsgBox keyWord, , "GetKeyword Example"
End Sub
```

Control User Input

When collecting input from the user, you want to make sure you limit the type of information they can enter so you can get the desired response. The various prompt option objects are used to not only define the prompt displayed at the Command prompt, but also restrict the input that the user can provide. With some of the input methods, not only can you get a return value based on the type of method used but also get a keyword.

For example, you can use the GetPoint method to have the user specify a point or respond with a keyword. This is how commands like LINE, CIRCLE, and PLINE work.

Get an integer value or a keyword

The following example prompts the user for a positive non-zero integer value or a keyword.

VB.NET

```
Imports Autodesk.AutoCAD.ApplicationServices
Imports Autodesk.AutoCAD.EditorInput
Imports Autodesk.AutoCAD.Runtime

<CommandMethod("GetIntegerOrKeywordFromUser")> _
Public Sub GetIntegerOrKeywordFromUser()
  Dim acDoc As Document = Application.DocumentManager.MdiActiveDocument

  Dim pIntOpts As PromptIntegerOptions = New PromptIntegerOptions("")
  pIntOpts.Message = vbCrLf & "Enter the size or "

  '' Restrict input to positive and non-negative values
  pIntOpts.AllowZero = False
  pIntOpts.AllowNegative = False

  '' Define the valid keywords and allow Enter
  pIntOpts.Keywords.Add("Big")
  pIntOpts.Keywords.Add("Small")
  pIntOpts.Keywords.Add("Regular")
  pIntOpts.Keywords.Default = "Regular"
  pIntOpts.AllowNone = True
```

```
'' Get the value entered by the user
Dim pIntRes As PromptIntegerResult = acDoc.Editor.GetInteger(pIntOpts)

If pIntRes.Status = PromptStatus.Keyword Then
    Application.ShowAlertDialog("Entered keyword: " & _
                               pIntRes.StringResult)
Else
    Application.ShowAlertDialog("Entered value: " & _
                               pIntRes.Value.ToString())
End If
End Sub
```

C#

```
using Autodesk.AutoCAD.ApplicationServices;
using Autodesk.AutoCAD.EditorInput;
using Autodesk.AutoCAD.Runtime;

[CommandMethod("GetIntegerOrKeywordFromUser")]
public static void GetIntegerOrKeywordFromUser()
{
  Document acDoc = Application.DocumentManager.MdiActiveDocument;

  PromptIntegerOptions pIntOpts = new PromptIntegerOptions("");
  pIntOpts.Message = "\nEnter the size or ";

  // Restrict input to positive and non-negative values
  pIntOpts.AllowZero = false;
  pIntOpts.AllowNegative = false;

  // Define the valid keywords and allow Enter
  pIntOpts.Keywords.Add("Big");
  pIntOpts.Keywords.Add("Small");
  pIntOpts.Keywords.Add("Regular");
  pIntOpts.Keywords.Default = "Regular";
  pIntOpts.AllowNone = true;

  // Get the value entered by the user
  PromptIntegerResult pIntRes = acDoc.Editor.GetInteger(pIntOpts);

  if (pIntRes.Status == PromptStatus.Keyword)
  {
      Application.ShowAlertDialog("Entered keyword: " +
                                 pIntRes.StringResult);
  }
  else
  {
      Application.ShowAlertDialog("Entered value: " +
                                 pIntRes.Value.ToString());
  }
}
```

⊟ *VBA/ActiveX Code Reference*

```
Sub GetIntegerOrKeywordFromUser()

    ' The first parameter of InitializeUserInput (6)
    ' restricts input to positive and non-negative
    ' values. The second parameter is the list of
    ' valid keywords.
    ThisDrawing.Utility.InitializeUserInput 6, "Big Small Regular"
```

```
    ' Set the prompt string variable
    Dim promptStr As String
    promptStr = vbCrLf & "Enter the size or [Big/Small/Regular] <Regular>:"

    ' At the GetInteger prompt, entering a keyword or pressing
    ' ENTER without entering a value results in an error. To allow
    ' your application to continue and check for the error
    ' description, you must set the error handler to resume on error.
    On Error Resume Next

    ' Get the value entered by the user
    Dim returnInteger As Integer
    returnInteger = ThisDrawing.Utility.GetInteger(promptStr)

    ' Check for an error. If the error number matches the
    ' one shown below, then use GetInput to get the returned
    ' string; otherwise, use the value of returnInteger.

    If Err.Number = -2145320928 Then
        Dim returnString As String
        Debug.Print Err.Description
        returnString = ThisDrawing.Utility.GetInput()
        If returnString = "" Then          'ENTER returns null string
            returnString = "Regular"       'Set to default
        End If
        Err.Clear
    Else 'Otherwise,
        returnString = returnInteger       'Use the value entered
    End If

    ' Display the result
    MsgBox returnString, , "InitializeUserInput Example"
End Sub
```

Access the AutoCAD Command Line

You can send commands directly to the AutoCAD command line by using the SendStringToExecute method. The SendStringToExecute method sends a single string to the command line. The string must contain the arguments to the command listed in the order expected by the prompt sequence of the executed command.

A blank space or the ASCII equivalent of a carriage return in the string is equivalent to pressing Enter on the keyboard. Unlike the AutoLISP environment, invoking the SendStringToExecute method with no argument is invalid.

Commands executed with SendStringToExecute are asynchronous and are not invoked until the .NET command has ended. If you need to execute a command immediately (synchronously), you should:

- Use the SendCommand method which is part of the COM Automation library which can be access using .NET COM Interop
- P/Invoke the unmanaged acedCommand or acedCmd method for native AutoCAD commands and commands defined with the ObjectARX or .NET API

- P/Invoke the unmanaged acedInvoke method for commands defined through AutoLISP

Send a command to the AutoCAD command line

The following example creates a circle with a center of (2, 2, 0) and a radius of 4. The drawing is then zoomed to all the geometry in the drawing. Notice that there is a space at the end of the string which represents the final Enter to begin execution of the command.

VB.NET

```
Imports Autodesk.AutoCAD.ApplicationServices
Imports Autodesk.AutoCAD.Runtime

<CommandMethod("SendACommandToAutoCAD")> _
Public Sub SendACommandToAutoCAD()
  Dim acDoc As Document = Application.DocumentManager.MdiActiveDocument

  '' Draws a circle and zooms to the extents or
  '' limits of the drawing
  acDoc.SendStringToExecute("._circle 2,2,0 4 ", True, False, False)
  acDoc.SendStringToExecute("._zoom _all ", True, False, False)
End Sub
```

C#

```
using Autodesk.AutoCAD.ApplicationServices;
using Autodesk.AutoCAD.Runtime;

[CommandMethod("SendACommandToAutoCAD")]
public static void SendACommandToAutoCAD()
{
  Document acDoc = Application.DocumentManager.MdiActiveDocument;

  // Draws a circle and zooms to the extents or
  // limits of the drawing
  acDoc.SendStringToExecute("._circle 2,2,0 4 ", true, false, false);
  acDoc.SendStringToExecute("._zoom _all ", true, false, false);
}
```

⊟ *VBA/ActiveX Code Reference*

```
Sub SendACommandToAutoCAD()
   ' Draws a circle and zooms to the extents or
   ' limits of the drawing
   ThisDrawing.SendCommand "_Circle 2,2,0 4 "
   ThisDrawing.SendCommand "_zoom a "
End Sub
```

5   Create and Edit AutoCAD Entities

You can create a range of objects, from simple lines and circles to spline curves, ellipses, and associative hatch areas. In general, you add objects to a BlockTableRecord object using the AppendEntity function. Once an object is created, you can change its properties such as layer, color, and linetype.

The drawing database is similar to other database programs, you can think of a Line object in Model space as table record and Model space as its database table. When working with a database, you must open and close records before working with them. The objects stored in the Database object are no different, you use the GetObject function to retrieve an object from the database and define how you want to work with the object.

Topics in this section

- Open and Close Objects
- Create Objects
- Work with Selection Sets
- Edit Named and 2D Objects
- Use Layers, Colors, and Linetypes
- Save and Restore Layer States
- Add Text to Drawings

Open and Close Objects

Whether you are working with objects such as lines, circles and polyline or a symbol table and its records, you need to open the object for read or write. When querying an object you want to open the object for read, but if you are going to make changes to the object you will want to open it for write.

Topics in this section

- Work with ObjectIds
- Use Transactions with the Transaction Manager
- Open and Close Objects without the Transaction Manager
- Upgrade and Downgrade Open Objects

Work with ObjectIds

Each object contained with in the Database object is assigned several unique ids. The unique ways you can access objects are:

- Entity handle
- ObjectId

- Instance pointer

The most common method is to access an object by its Object Id. Object Ids work well if your projects utilize both COM interop and the managed .NET API. If you create custom AutoLISP functions, you may need to work with entity handles.

Handles are persistent between AutoCAD sessions, so they are the best way of accessing objects if you need to export drawing information to an external file which might later need to be used to update the drawing. The ObjectId of an object in a database exists only while the database is loaded into memory. Once the database is closed, the Object Ids assigned to an object no longer exist and maybe different the next time the database is opened.

Obtain an Object Id

As you work with objects, you will need to obtain an Object Id first before you can open the object to query or edit it. An Object Id is assigned to an existing object in the database when the drawing file is opened, and new objects are assigned Object Ids when they are first created. An Object ID is commonly obtained for an existing object in the database by:

- Using a member property of the Database object, such as Clayer which retrieves the Object ID for the current layer
- Iterating a symbol table, such as the Layer symbol table

Open an Object

Once an Object Id is obtained, the GetObject function is used to open the object assigned the given Object Id. An object can be opened in one of the following modes:

- *Read.* Opens an object for read.
- *Write.* Opens an object for write if it is not already open.
- *Notify.* Opens an object for notification when it is closed, open for read, or open for write, but not when it is already open for notify. For more information on notifications, see [Use Events](#).

You should open an object in the mode that is best for the situation in which the object will be accessed. Opening an object for write introduces additional overhead than you might need due to the creation of undo records. If you are unsure if the object you are opening is the one you want to work with, you should open it for read and then use the UpgradeOpen function to change from read to write mode. For more information on using the UpgradeOpen function, see [Upgrade and Downgrade Open Objects](#).

Both the GetObject and Open functions return an object. When working with some programming languages, you will need to cast the returned value based on the variable the value is being assigned to. If you are using VB.NET, you do not need to worry about casting the returned value as it is done for you. The following examples show obtaining the LayerTableRecord for Layer Zero of the current database:

VB.NET

The following example manually disposes of the transaction after it is no longer needed.

```
Dim acCurDb As Document = Application.DocumentManager.MdiActiveDocument.Database
Dim acTrans As Transaction = acCurDb.TransactionManager.StartTransaction()
```

```
Dim acLyrTblRec As LayerTableRecord
acLyrTblRec = acTrans.GetObject(acCurDb.LayerZero, OpenMode.ForRead)

acTrans.Dispose()
```

The following example uses the Using statement to dispose of the transaction after it is no longer needed. The Using statement is the preferred coding style.

```
Dim acCurDb As Document = Application.DocumentManager.MdiActiveDocument.Database
Using acTrans As Transaction = acCurDb.TransactionManager.StartTransaction()
    Dim acLyrTblRec As LayerTableRecord
    acLyrTblRec = acTrans.GetObject(acCurDb.LayerZero, OpenMode.ForRead)
End Using
```

C#

The following example manually disposes of the transaction after it is no longer needed.

```
Document acCurDb = Application.DocumentManager.MdiActiveDocument.Database;
Transaction acTrans = acCurDb.TransactionManager.StartTransaction();

LayerTableRecord acLyrTblRec;
acLyrTblRec = acTrans.GetObject(acCurDb.LayerZero,
                                OpenMode.ForRead) as LayerTableRecord;

acTrans.Dispose();
```

The following example uses the Using statement to dispose of the transaction after it is no longer needed. The Using statement is the preferred coding style.

```
Document acCurDb = Application.DocumentManager.MdiActiveDocument.Database;
using (Transaction acTrans = acCurDb.TransactionManager.StartTransaction())
{
    LayerTableRecord acLyrTblRec;
    acLyrTblRec = acTrans.GetObject(acCurDb.LayerZero,
                                    OpenMode.ForRead) as LayerTableRecord;
}
```

Use Transactions with the Transaction Manager

Transactions are used to group multiple operations on multiple objects together as a single operation. Transactions are started and managed through the Transaction Manager. Once a transaction is started, you can then use the GetObject function to open an object.

As you work with objects opened with GetObject, the Transaction manager keeps track of the changes that are being made to the object. Any new objects that you create and add to the database should be added to a transaction as well with the AddNewlyCreatedDBObject function. Once the objects have been edited or added to the database, you can save the changes made to the database and close all the open objects with the Commit function on the Transaction object created with the Transaction Manager. Once you are finished with a transaction, call the Dispose function close the transaction.

Topics in this section

- [Start a New Transaction and Open an Object](#)

-

Start a New Transaction and Open an Object

The Transaction Manager is accessed from the TransactionManager property of the current database. Once a reference to the Transaction Manager is made, you use the StartTransaction method to start a new transaction. StartTransaction creates an instance of a Transaction object and allows you to open objects with the GetObject method.

All open objects opened during a transaction are closed at the end of the transaction. To end a transaction, call the Dispose method of a transaction object. If you use the Using and End Using keywords to indicate the start and end of a transaction, you do not need to call the Dispose method.

Prior to disposing of a transaction, you should commit any changes made with the Commit method. If the changes are not committed before a transaction is disposed, any changes made are rolled back to the state their were in prior to the start of the transaction. For more information on committing or rolling back changes made in a transaction, see [Commit and Rollback Changes](#).

More than one transaction can be started. The number of active transactions can be retrieved with the NumberOfActiveTransactions property of the TransactionManager object while the top most or latest transaction can be retrieved with the TopTransaction property.

Transactions can be nested one inside of another in order to rollback some of the changes made during the execution of a routine. For more information on working with multiple transactions or nesting transactions, see [Nest Transactions](#).

Query objects

The following example demonstrates how to open and read objects within using a transaction. You use the GetObject method to first open the BlockTable and then the Model space record.

VB.NET

```
Imports Autodesk.AutoCAD.Runtime
Imports Autodesk.AutoCAD.ApplicationServices
Imports Autodesk.AutoCAD.DatabaseServices

<CommandMethod("OpenTransactionManager")> _
Public Sub OpenTransactionManager()
  '' Get the current document and database
  Dim acDoc As Document = Application.DocumentManager.MdiActiveDocument
  Dim acCurDb As Database = acDoc.Database

  '' Start a transaction
  Using acTrans As Transaction = acCurDb.TransactionManager.StartTransaction()
```

```
      '' Open the Block table for read
      Dim acBlkTbl As BlockTable
      acBlkTbl = acTrans.GetObject(acCurDb.BlockTableId, OpenMode.ForRead)

      '' Open the Block table record Model space for read
      Dim acBlkTblRec As BlockTableRecord
      acBlkTblRec = acTrans.GetObject(acBlkTbl(BlockTableRecord.ModelSpace), _
                                      OpenMode.ForRead)

      '' Step through the Block table record
      For Each acObjId As ObjectId In acBlkTblRec
          acDoc.Editor.WriteMessage(vbLf & "DXF name: " & _
acObjId.ObjectClass().DxfName)
          acDoc.Editor.WriteMessage(vbLf & "ObjectID: " & acObjId.ToString())
          acDoc.Editor.WriteMessage(vbLf & "Handle: " & acObjId.Handle.ToString())
          acDoc.Editor.WriteMessage(vbLf)
      Next

      '' Dispose of the transaction
  End Using
End Sub
```

C#

```
using Autodesk.AutoCAD.Runtime;
using Autodesk.AutoCAD.ApplicationServices;
using Autodesk.AutoCAD.DatabaseServices;

[CommandMethod("OpenTransactionManager")]
public static void OpenTransactionManager()
{
  // Get the current document and database
  Document acDoc = Application.DocumentManager.MdiActiveDocument;
  Database acCurDb = acDoc.Database;

  // Start a transaction
  using (Transaction acTrans = acCurDb.TransactionManager.StartTransaction())
  {
      // Open the Block table for read
      BlockTable acBlkTbl;
      acBlkTbl = acTrans.GetObject(acCurDb.BlockTableId,
                                   OpenMode.ForRead) as BlockTable;

      // Open the Block table record Model space for read
      BlockTableRecord acBlkTblRec;
      acBlkTblRec = acTrans.GetObject(acBlkTbl[BlockTableRecord.ModelSpace],
                                      OpenMode.ForRead) as BlockTableRecord;

      // Step through the Block table record
      foreach (ObjectId asObjId in acBlkTblRec)
      {
          acDoc.Editor.WriteMessage("\nDXF name: " + asObjId.ObjectClass.DxfName);
          acDoc.Editor.WriteMessage("\nObjectID: " + asObjId.ToString());
          acDoc.Editor.WriteMessage("\nHandle: " + asObjId.Handle.ToString());
          acDoc.Editor.WriteMessage("\n");
      }

      // Dispose of the transaction
  }
}
```

Add a new object to the database

The following example demonstrates how to add a circle object to the database with in a transaction. You use the `GetObject` method to first open the BlockTable for read and then the Model space record for write. After Model space is opened for write, you use the `AppendEntity` and `AddNewlyCreatedDBObject` function to append the new Circle object to Model space as well as the transaction.

VB.NET

```
Imports Autodesk.AutoCAD.Runtime
Imports Autodesk.AutoCAD.ApplicationServices
Imports Autodesk.AutoCAD.DatabaseServices
Imports Autodesk.AutoCAD.Geometry

<CommandMethod("AddNewCircleTransaction")> _
Public Sub AddNewCircleTransaction()
  '' Get the current document and database
  Dim acDoc As Document = Application.DocumentManager.MdiActiveDocument
  Dim acCurDb As Database = acDoc.Database

  '' Start a transaction
  Using acTrans As Transaction = acCurDb.TransactionManager.StartTransaction()

      '' Open the Block table for read
      Dim acBlkTbl As BlockTable
      acBlkTbl = acTrans.GetObject(acCurDb.BlockTableId, OpenMode.ForRead)

      '' Open the Block table record Model space for write
      Dim acBlkTblRec As BlockTableRecord
      acBlkTblRec = acTrans.GetObject(acBlkTbl(BlockTableRecord.ModelSpace), _
                                      OpenMode.ForWrite)

      '' Create a circle with a radius of 3 at 5,5
      Dim acCirc As Circle = New Circle()
      acCirc.SetDatabaseDefaults()
      acCirc.Center = New Point3d(5, 5, 0)
      acCirc.Radius = 3

      '' Add the new object to Model space and the transaction
      acBlkTblRec.AppendEntity(acCirc)
      acTrans.AddNewlyCreatedDBObject(acCirc, True)

      '' Commit the changes and dispose of the transaction
      acTrans.Commit()
  End Using
End Sub
```

C#

```
using Autodesk.AutoCAD.Runtime;
using Autodesk.AutoCAD.ApplicationServices;
using Autodesk.AutoCAD.DatabaseServices;
using Autodesk.AutoCAD.Geometry;

[CommandMethod("AddNewCircleTransaction")]
public static void AddNewCircleTransaction()
{
  // Get the current document and database
  Document acDoc = Application.DocumentManager.MdiActiveDocument;
  Database acCurDb = acDoc.Database;

  // Start a transaction
  using (Transaction acTrans = acCurDb.TransactionManager.StartTransaction())
```

```
    {
        // Open the Block table for read
        BlockTable acBlkTbl;
        acBlkTbl = acTrans.GetObject(acCurDb.BlockTableId,
                                     OpenMode.ForRead) as BlockTable;

        // Open the Block table record Model space for write
        BlockTableRecord acBlkTblRec;
        acBlkTblRec = acTrans.GetObject(acBlkTbl[BlockTableRecord.ModelSpace],
                                        OpenMode.ForWrite) as BlockTableRecord;

        // Create a circle with a radius of 3 at 5,5
        Circle acCirc = new Circle();
        acCirc.SetDatabaseDefaults();
        acCirc.Center = new Point3d(5, 5, 0);
        acCirc.Radius = 3;

        // Add the new object to Model space and the transaction
        acBlkTblRec.AppendEntity(acCirc);
        acTrans.AddNewlyCreatedDBObject(acCirc, true);

        // Commit the changes and dispose of the transaction
        acTrans.Commit();
    }
}
```

Commit and Rollback Changes

When using transactions, you are able to decide when changes to objects are saved to the drawing database. You use the Commit method to save the changes made to the objects opened within a transaction. If your program encounters an error you can rollback any changes made within a transaction with the Abort method.

If Commit is not called before Dispose is called, all changes made within the transaction are rolled back. Whether Commit or Abort are called, you need to call Dispose to signal the end of the transaction. If the transaction object is started with the Using statement, you do not have to call Dispose.

VB.NET

```
'' Commit the changes made within the transaction
<transaction>.Commit()

'' Abort the transaction and rollback to the previous state
<transaction>.Abort()
```

C#

```
// Commit the changes made within the transaction
<transaction>.Commit();

// Abort the transaction and rollback to the previous state
<transaction>.Abort();
```

Nest Transactions

Transactions can be nested one inside another. You might have an outer transaction to undo all the changes made by your routine and inner transactions to undo just portions of the changes made. When you work with nested transactions, you start with a top transaction which is also the outer most transaction.

As you start new transactions, they are added into the previous transaction. Nested transactions must be committed or aborted in the opposite order in which they are created. So if you have three transactions, you must close the third or innermost one before the second and finally the first. If you abort the first transaction, the changes made by all three transactions is undone.

The following illustration shows how transactions appear when nested.
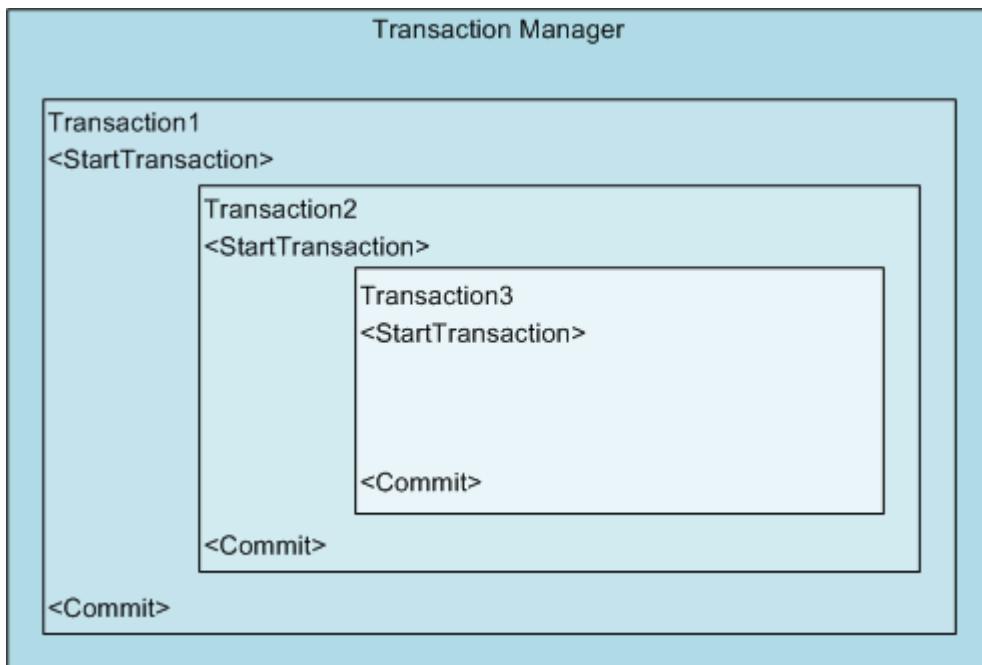
Use nested transactions to create and modify objects

The following example demonstrates using three transactions to create a Circle and Line object, and then change their colors. The color of the circle is changed in the second and third transaction, but since the third transaction is aborted only the changes made in the first and second transactions are saved to the database. Additionally, the number of active transactions is printed in the Command Line window as they are created and closed.

VB.NET

```vbnet
Imports Autodesk.AutoCAD.Runtime
Imports Autodesk.AutoCAD.ApplicationServices
Imports Autodesk.AutoCAD.DatabaseServices
Imports Autodesk.AutoCAD.Geometry
Imports Autodesk.AutoCAD.EditorInput

<CommandMethod("NestedTransactions")> _
Public Sub NestedTransactions()
  '' Get the current document and database
```

```
    Dim acDoc As Document = Application.DocumentManager.MdiActiveDocument
    Dim acCurDb As Database = acDoc.Database

    '' Create a reference to the Transaction Manager
    Dim acTransMgr As Autodesk.AutoCAD.DatabaseServices.TransactionManager
    acTransMgr = acCurDb.TransactionManager

    '' Create a new transaction
    Using acTrans1 As Transaction = acTransMgr.StartTransaction()

        '' Print the current number of active transactions
        acDoc.Editor.WriteMessage(vbLf & "Number of transactions active: " & _
                                  acTransMgr.NumberOfActiveTransactions.ToString())

        '' Open the Block table for read
        Dim acBlkTbl As BlockTable
        acBlkTbl = acTrans1.GetObject(acCurDb.BlockTableId, OpenMode.ForRead)

        '' Open the Block table record Model space for write
        Dim acBlkTblRec As BlockTableRecord
        acBlkTblRec = acTrans1.GetObject(acBlkTbl(BlockTableRecord.ModelSpace), _
                                         OpenMode.ForWrite)

        '' Create a circle with a radius of 3 at 5,5
        Dim acCirc As Circle = New Circle()
        acCirc.SetDatabaseDefaults()
        acCirc.Center = New Point3d(5, 5, 0)
        acCirc.Radius = 3

        '' Add the new object to Model space and the transaction
        acBlkTblRec.AppendEntity(acCirc)
        acTrans1.AddNewlyCreatedDBObject(acCirc, True)

        '' Create the second transaction
        Using acTrans2 As Transaction = acTransMgr.StartTransaction()

            acDoc.Editor.WriteMessage(vbLf & "Number of transactions active: " & _
                                      acTransMgr.NumberOfActiveTransactions.ToString
())

            '' Change the circle's color
            acCirc.ColorIndex = 5

            '' Get the object that was added to Transaction 1 and set it to the
color 5
            Dim acLine As Line = New Line(New Point3d(2, 5, 0), New Point3d(10, 7,
0))
            acLine.SetDatabaseDefaults()
            acLine.ColorIndex = 3

            '' Add the new object to Model space and the transaction
            acBlkTblRec.AppendEntity(acLine)
            acTrans2.AddNewlyCreatedDBObject(acLine, True)

            '' Create the third transaction
            Using acTrans3 As Transaction = acTransMgr.StartTransaction()

                acDoc.Editor.WriteMessage(vbLf & "Number of transactions active: " &
_
                                          acTransMgr.NumberOfActiveTransactions.ToSt
ring())

                '' Change the circle's color
                acCirc.ColorIndex = 3
```

```
                '' Update the display of the drawing
                acDoc.Editor.WriteMessage(vbLf)
                acDoc.Editor.Regen()

                '' Request to keep or discard the changes in the third transaction
                Dim pKeyOpts As PromptKeywordOptions = New PromptKeywordOptions("")
                pKeyOpts.Message = vbLf & "Keep color change "
                pKeyOpts.Keywords.Add("Yes")
                pKeyOpts.Keywords.Add("No")
                pKeyOpts.Keywords.Default = "No"
                pKeyOpts.AllowNone = True

                Dim pKeyRes As PromptResult = acDoc.Editor.GetKeywords(pKeyOpts)

                If pKeyRes.StringResult = "No" Then
                    '' Discard the changes in transaction 3
                    acTrans3.Abort()
                Else
                    '' Save the changes in transaction 3
                    acTrans3.Commit()
                End If

                '' Dispose the transaction
            End Using

            acDoc.Editor.WriteMessage(vbLf & "Number of transactions active: " & _
                                  acTransMgr.NumberOfActiveTransactions.ToString
())

            '' Keep the changes to transaction 2
            acTrans2.Commit()
        End Using

        acDoc.Editor.WriteMessage(vbLf & "Number of transactions active: " & _
                              acTransMgr.NumberOfActiveTransactions.ToString())

        '' Keep the changes to transaction 1
        acTrans1.Commit()
    End Using
End Sub
```

C#

```csharp
using Autodesk.AutoCAD.Runtime;
using Autodesk.AutoCAD.ApplicationServices;
using Autodesk.AutoCAD.DatabaseServices;
using Autodesk.AutoCAD.Geometry;
using Autodesk.AutoCAD.EditorInput;

[CommandMethod("NestedTransactions")]
public static void NestedTransactions()
{
  // Get the current document and database
  Document acDoc = Application.DocumentManager.MdiActiveDocument;
  Database acCurDb = acDoc.Database;

  // Create a reference to the Transaction Manager
  Autodesk.AutoCAD.DatabaseServices.TransactionManager acTransMgr;
  acTransMgr = acCurDb.TransactionManager;

  // Create a new transaction
  using (Transaction acTrans1 = acTransMgr.StartTransaction())
  {
      // Print the current number of active transactions
```

```csharp
        acDoc.Editor.WriteMessage("\nNumber of transactions active: " +
                                  acTransMgr.NumberOfActiveTransactions.ToString());

        // Open the Block table for read
        BlockTable acBlkTbl;
        acBlkTbl = acTrans1.GetObject(acCurDb.BlockTableId,
                                      OpenMode.ForRead) as BlockTable;

        // Open the Block table record Model space for write
        BlockTableRecord acBlkTblRec;
        acBlkTblRec = acTrans1.GetObject(acBlkTbl[BlockTableRecord.ModelSpace],
                                         OpenMode.ForWrite) as BlockTableRecord;

        // Create a circle with a radius of 3 at 5,5
        Circle acCirc = new Circle();
        acCirc.SetDatabaseDefaults();
        acCirc.Center = new Point3d(5, 5, 0);
        acCirc.Radius = 3;

        // Add the new object to Model space and the transaction
        acBlkTblRec.AppendEntity(acCirc);
        acTrans1.AddNewlyCreatedDBObject(acCirc, true);

        // Create the second transaction
        using (Transaction acTrans2 = acTransMgr.StartTransaction())
        {
            acDoc.Editor.WriteMessage("\nNumber of transactions active: " +
                                      acTransMgr.NumberOfActiveTransactions.ToString
());

            // Change the circle's color
            acCirc.ColorIndex = 5;

            // Get the object that was added to Transaction 1 and set it to the
color 5
            Line acLine = new Line(new Point3d(2, 5, 0), new Point3d(10, 7, 0));
            acLine.SetDatabaseDefaults();
            acLine.ColorIndex = 3;

            // Add the new object to Model space and the transaction
            acBlkTblRec.AppendEntity(acLine);
            acTrans2.AddNewlyCreatedDBObject(acLine, true);

            // Create the third transaction
            using (Transaction acTrans3 = acTransMgr.StartTransaction())
            {
                acDoc.Editor.WriteMessage("\nNumber of transactions active: " +
                                          acTransMgr.NumberOfActiveTransactions.ToSt
ring());

                // Change the circle's color
                acCirc.ColorIndex = 3;

                // Update the display of the drawing
                acDoc.Editor.WriteMessage("\n");
                acDoc.Editor.Regen();

                // Request to keep or discard the changes in the third transaction
                PromptKeywordOptions pKeyOpts = new PromptKeywordOptions("");
                pKeyOpts.Message = "\nKeep color change ";
                pKeyOpts.Keywords.Add("Yes");
                pKeyOpts.Keywords.Add("No");
                pKeyOpts.Keywords.Default = "No";
                pKeyOpts.AllowNone = true;
```

```
            PromptResult pKeyRes = acDoc.Editor.GetKeywords(pKeyOpts);

            if (pKeyRes.StringResult == "No")
            {
                // Discard the changes in transaction 3
                acTrans3.Abort();
            }
            else
            {
                // Save the changes in transaction 3
                acTrans3.Commit();
            }

            // Dispose the transaction
        }

        acDoc.Editor.WriteMessage("\nNumber of transactions active: " +
                                acTransMgr.NumberOfActiveTransactions.ToString
());

        // Keep the changes to transaction 2
        acTrans2.Commit();
    }

    acDoc.Editor.WriteMessage("\nNumber of transactions active: " +
                            acTransMgr.NumberOfActiveTransactions.ToString());

    // Keep the changes to transaction 1
    acTrans1.Commit();
  }
}
```

Open and Close Objects without the Transaction Manager

Transactions make it easier to open and work with multiple objects, but they are not the only way to open and edit objects. Other than using a transaction, you can open and close objects using the Open and Close methods. You still need to obtain an object id to use the Open method. Like the GetObject method used with transactions, you need to specify an open mode and the return value is an object. If you make changes to an object after you opened it with the Open method, you can use the Cancel method to rollback all the changes made since it was opened. Cancel must be called on each object in which you want to rollback.

NoteObjects must be paired with an open and close operation. If you use the Open method on an object, you must close it using either the Close or Cancel method. Failure to close the object will lead to read access violations and cause AutoCAD to become unstable.

If you need to work with a single object, using the Open and Close methods can reduce the number of lines of code that you might otherwise have to write compared to working with the Transaction Manager. However, using transactions is the recommended way of opening and closing objects.

WarningYou should not use the Open and Close methods when using transactions, as objects might not get opened or closed properly by the Transaction Manager which could cause AutoCAD to crash.

Query objects

The following example demonstrates how to open and close objects without using a transaction and the GetObject method. To see the same example using the Transaction Manager, see Start a New Transaction and Open an Object.

VB.NET

```
Imports Autodesk.AutoCAD.Runtime
Imports Autodesk.AutoCAD.ApplicationServices
Imports Autodesk.AutoCAD.DatabaseServices

<CommandMethod("OpenCloseObjectId")> _
Public Sub OpenCloseObjectId()
  '' Get the current document and database
  Dim acDoc As Document = Application.DocumentManager.MdiActiveDocument
  Dim acCurDb As Database = acDoc.Database

  '' Open the Block table for read
  Dim acBlkTbl As BlockTable
  acBlkTbl = acCurDb.BlockTableId.Open(OpenMode.ForRead)

  '' Open the Block table record Model space for read
  Dim acBlkTblRec As BlockTableRecord
  acBlkTblRec = acBlkTbl(BlockTableRecord.ModelSpace).Open(OpenMode.ForRead)

  '' Step through the Block table record
  For Each acObjId As ObjectId In acBlkTblRec
      acDoc.Editor.WriteMessage(vbLf & "DXF name: " &
acObjId.ObjectClass().DxfName)
      acDoc.Editor.WriteMessage(vbLf & "ObjectID: " & acObjId.ToString())
      acDoc.Editor.WriteMessage(vbLf & "Handle: " & acObjId.Handle.ToString())
      acDoc.Editor.WriteMessage(vbLf)
  Next

  '' Close the Block table record
  acBlkTblRec.Close()
  acBlkTblRec.Dispose()

  '' Close the Block table
  acBlkTbl.Close()
  acBlkTbl.Dispose()
End Sub
```

C#

```
using Autodesk.AutoCAD.Runtime;
using Autodesk.AutoCAD.ApplicationServices;
using Autodesk.AutoCAD.DatabaseServices;

[CommandMethod("OpenCloseObjectId")]
public static void OpenCloseObjectId()
{
  // Get the current document and database
  Document acDoc = Application.DocumentManager.MdiActiveDocument;
  Database acCurDb = acDoc.Database;

  // Open the Block table for read
  BlockTable acBlkTbl;
  acBlkTbl = acCurDb.BlockTableId.Open(OpenMode.ForRead) as BlockTable;

  // Open the Block table record Model space for read
```

```
   BlockTableRecord acBlkTblRec;
   acBlkTblRec = acBlkTbl[BlockTableRecord.ModelSpace].Open(OpenMode.ForRead) as
BlockTableRecord;

   // Step through the Block table record
   foreach (ObjectId acObjId in acBlkTblRec)
   {
       acDoc.Editor.WriteMessage("\nDXF name: " + acObjId.ObjectClass.DxfName);
       acDoc.Editor.WriteMessage("\nObjectID: " + acObjId.ToString());
       acDoc.Editor.WriteMessage("\nHandle: " + acObjId.Handle.ToString());
       acDoc.Editor.WriteMessage("\n");
   }

   // Close the Block table record
   acBlkTblRec.Close();
   acBlkTblRec.Dispose();

   // Close the Block table
   acBlkTbl.Close();
   acBlkTbl.Dispose();
}
```

Add a new object to the database

This example demonstrates how to create a new object and append it to Model space
without using the Transaction manager. To see the same example using the Transaction
manager, see [Start a New Transaction and Open an Object](#).

VB.NET

```
Imports Autodesk.AutoCAD.Runtime
Imports Autodesk.AutoCAD.ApplicationServices
Imports Autodesk.AutoCAD.DatabaseServices
Imports Autodesk.AutoCAD.Geometry

<CommandMethod("AddNewCircleOpenClose")> _
Public Sub AddNewCircleOpenClose()
  '' Get the current document and database
  Dim acDoc As Document = Application.DocumentManager.MdiActiveDocument
  Dim acCurDb As Database = acDoc.Database

  '' Open the Block table for read
  Dim acBlkTbl As BlockTable
  acBlkTbl = acCurDb.BlockTableId.Open(OpenMode.ForRead)

  '' Open the Block table record Model space for write
  Dim acBlkTblRec As BlockTableRecord
  acBlkTblRec = acBlkTbl(BlockTableRecord.ModelSpace).Open(OpenMode.ForWrite)

  '' Create a circle with a radius of 3 at 5,5
  Dim acCirc As Circle = New Circle()
  acCirc.SetDatabaseDefaults()
  acCirc.Center = New Point3d(5, 5, 0)
  acCirc.Radius = 3

  '' Add the new object to Model space and the transaction
  acBlkTblRec.AppendEntity(acCirc)

  '' Close the circle object
  acCirc.Close()
  acCirc.Dispose()

  '' Close the Block table record
```

```
    acBlkTblRec.Close()
    acBlkTblRec.Dispose()

    '' Close the Block table
    acBlkTbl.Close()
    acBlkTbl.Dispose()
End Sub
```

C#

```csharp
using Autodesk.AutoCAD.Runtime;
using Autodesk.AutoCAD.ApplicationServices;
using Autodesk.AutoCAD.DatabaseServices;
using Autodesk.AutoCAD.Geometry;

[CommandMethod("AddNewCircleOpenClose")]
public static void AddNewCircleOpenClose()
{
    // Get the current document and database
    Document acDoc = Application.DocumentManager.MdiActiveDocument;
    Database acCurDb = acDoc.Database;

    // Open the Block table for read
    BlockTable acBlkTbl;
    acBlkTbl = acCurDb.BlockTableId.Open(OpenMode.ForRead) as BlockTable;

    // Open the Block table record Model space for write
    BlockTableRecord acBlkTblRec;
    acBlkTblRec = acBlkTbl[BlockTableRecord.ModelSpace].Open(OpenMode.ForWrite)
                    as BlockTableRecord;

    // Create a circle with a radius of 3 at 5,5
    Circle acCirc = new Circle();
    acCirc.SetDatabaseDefaults();
    acCirc.Center = new Point3d(5, 5, 0);
    acCirc.Radius = 3;

    // Add the new object to Model space and the transaction
    acBlkTblRec.AppendEntity(acCirc);

    // Close the circle object
    acCirc.Close();
    acCirc.Dispose();

    // Close the Block table record
    acBlkTblRec.Close();
    acBlkTblRec.Dispose();

    // Close the Block table
    acBlkTbl.Close();
    acBlkTbl.Dispose();
}
```

Upgrade and Downgrade Open Objects

Once an object is opened using either the GetObject or Open methods, you can change the current open mode of an object with the UpgradeOpen and DowngradeOpen methods. The UpgradeOpen method changes an object open for read to write mode, while

DowngradeOpen changes an object open for write to read mode. You do not need to pair a call to DowngradeOpen with each UpgradeOpen call, since closing of an object or disposing of a transaction will sufficiently cleanup the open state of an entity.

When you go to open an object, open the object in the mode that you will use the object in. Do not just open an object for write when you might only need to query an object. It is more efficent to open an object for read and query the object's properties than it is to open the object for write and query the object's properties.

Opening an object for write causes undo filing to start for the object. Undo filing is used to track changes to an object, so any changes made can be rolled back. If you are uncertain if you need to modify an object, it is best to open an object for read and then upgrade it for write. This will help to reduce the overhead of your program.

An example of when you might use UpgradeOpen is when you might be querying objects to see if they match a specific condition, and if the condition is met then you would upgrade the object from read to write mode to make modifications to it.

Open Notifications

Similarly, if an object is open for notify and you receive a notification, you use UpgradeFromNotify to upgrade the object for write. Then you would use DowngradeToNotify to downgrade the object back to notify. UpgradeFromNotify and DowngradeFromNotify are reserved for use in methods that are intended to be used by an object to change its own open status so that it can safely modify itself.

VB.NET

```
Imports Autodesk.AutoCAD.Runtime
Imports Autodesk.AutoCAD.ApplicationServices
Imports Autodesk.AutoCAD.DatabaseServices

<CommandMethod("FreezeDoorLayer")> _
Public Sub FreezeDoorLayer()
  '' Get the current document and database
  Dim acDoc As Document = Application.DocumentManager.MdiActiveDocument
  Dim acCurDb As Database = acDoc.Database

  '' Start a transaction
  Using acTrans As Transaction = acCurDb.TransactionManager.StartTransaction()

      '' Open the Layer table for read
      Dim acLyrTbl As LayerTable
      acLyrTbl = acTrans.GetObject(acCurDb.LayerTableId, OpenMode.ForRead)

      '' Step through each layer and update those that start with 'Door'
      For Each acObjId As ObjectId In acLyrTbl
          '' Open the Layer table record for read
          Dim acLyrTblRec As LayerTableRecord
          acLyrTblRec = acTrans.GetObject(acObjId, OpenMode.ForRead)

          '' Check to see if the layer's name starts with 'Door'
          If (acLyrTblRec.Name.StartsWith("Door", _
                                          StringComparison.OrdinalIgnoreCase) =
True) Then
              '' Check to see if the layer is current, if so then do not freeze it
              If acLyrTblRec.ObjectId <> acCurDb.Clayer Then
                  '' Change from read to write mode
                  acLyrTblRec.UpgradeOpen()
```

```
                    '' Freeze the layer
                    acLyrTblRec.IsFrozen = True
                End If
            End If
        Next

        '' Commit the changes and dispose of the transaction
        acTrans.Commit()
    End Using
End Sub
```

C#

```csharp
using Autodesk.AutoCAD.Runtime;
using Autodesk.AutoCAD.ApplicationServices;
using Autodesk.AutoCAD.DatabaseServices;

[CommandMethod("FreezeDoorLayer")]
public static void FreezeDoorLayer()
{
  // Get the current document and database
  Document acDoc = Application.DocumentManager.MdiActiveDocument;
  Database acCurDb = acDoc.Database;

  // Start a transaction
  using (Transaction acTrans = acCurDb.TransactionManager.StartTransaction())
  {
      // Open the Layer table for read
      LayerTable acLyrTbl;
      acLyrTbl = acTrans.GetObject(acCurDb.LayerTableId,
                                   OpenMode.ForRead) as LayerTable;

      // Step through each layer and update those that start with 'Door'
      foreach (ObjectId acObjId in acLyrTbl)
      {
          // Open the Layer table record for read
          LayerTableRecord acLyrTblRec;
          acLyrTblRec = acTrans.GetObject(acObjId,
                                          OpenMode.ForRead) as LayerTableRecord;

          // Check to see if the layer's name starts with 'Door'
          if (acLyrTblRec.Name.StartsWith("Door",
                                          StringComparison.OrdinalIgnoreCase) ==
true)
          {
              // Check to see if the layer is current, if so then do not freeze it
              if (acLyrTblRec.ObjectId != acCurDb.Clayer)
              {
                  // Change from read to write mode
                  acLyrTblRec.UpgradeOpen();

                  // Freeze the layer
                  acLyrTblRec.IsFrozen = true;
              }
          }
      }

      // Commit the changes and dispose of the transaction
      acTrans.Commit();
  }
}
```

Create Objects

AutoCAD often offers several different ways to create the same graphical object. While the .NET API does not offer the same combinations of creating objects, it does offer a basic object constructor for each object type but also offers overrides for many of the object constructors as well.

For example, in AutoCAD there are four different ways you can create a circle: (1) by specifying the center and radius, (2) by two points defining the diameter, (3) by three points defining the circumference, or (4) by two tangents and a radius. However, in .NET API there is two creation methods provided to create a circle. One method accepts no parameters, while the second requires a center point, the normal direction for the circle, and a radius.

NoteObjects are created using the New keyword and then appended to the parent object using Add or AppendEntity based on if you are working with a container (symbol table or dictionary) or a BlockTableRecord object.

Assign default property values to a new object

When a new graphical object is created, the SetDatabaseDefaults method of the new object should be called. The SetDatabaseDefaults method sets the following entity property values based on the current entity values defined in the database of the current document:

- Color
- Layer
- Linetype
- Linetype scale
- Lineweight
- Plot style name
- Visibility

Topics in this section

- [Determine the Parent Object](#)
- [Create Lines](#)
- [Create Curved Objects](#)
- [Create Point Objects](#)
- [Create Solid-Filled Areas](#)
- [Work with Regions](#)
- [Create Hatches](#)

Determine the Parent Object

Graphical objects are appended to a BlockTableRecord object, such as Model or Paper space. You reference the blocks that represent Model and Paper space through the BlockTable object. If you want to work in the current space instead of a specific space, you get the ObjectId for the current space from the current database with the CurrentSpaceId property.

The ObjectId for the block table records of Model and Paper space can be retrieved from the BlockTable object using a property or the GetBlockModelSpaceId and GetBlockPaperSpaceId methods of the SymbolUtilityServices class under the DatabaseServices namespace.

Access Model space, Paper space or the current space

The following example demonstrates how to access the block table records associated with Model space, Paper space or the current space. Once the block table record is referenced, a new line is added to the block table record.

VB.NET

```vb.net
Imports Autodesk.AutoCAD.Runtime
Imports Autodesk.AutoCAD.ApplicationServices
Imports Autodesk.AutoCAD.DatabaseServices
Imports Autodesk.AutoCAD.Geometry
Imports Autodesk.AutoCAD.EditorInput

<CommandMethod("AccessSpace")> _
Public Sub AccessSpace()
  '' Get the current document and database
  Dim acDoc As Document = Application.DocumentManager.MdiActiveDocument
  Dim acCurDb As Database = acDoc.Database

  '' Start a transaction
  Using acTrans As Transaction = acCurDb.TransactionManager.StartTransaction()

      '' Open the Block table for read
      Dim acBlkTbl As BlockTable
      acBlkTbl = acTrans.GetObject(acCurDb.BlockTableId, OpenMode.ForRead)

      '' Open the Block table record for read
      Dim acBlkTblRec As BlockTableRecord

      '' Request which table record to open
      Dim pKeyOpts As PromptKeywordOptions = New PromptKeywordOptions("")
      pKeyOpts.Message = vbLf & "Enter which space to create the line in "
      pKeyOpts.Keywords.Add("Model")
      pKeyOpts.Keywords.Add("Paper")
      pKeyOpts.Keywords.Add("Current")
      pKeyOpts.AllowNone = False
      pKeyOpts.AppendKeywordsToMessage = True

      Dim pKeyRes As PromptResult = acDoc.Editor.GetKeywords(pKeyOpts)

      If pKeyRes.StringResult = "Model" Then
          '' Get the ObjectID for Model space from the Block table
          acBlkTblRec = acTrans.GetObject(acBlkTbl(BlockTableRecord.ModelSpace), _
                                          OpenMode.ForWrite)
      ElseIf pKeyRes.StringResult = "Paper" Then
          '' Get the ObjectID for Paper space from the Block table
          acBlkTblRec = acTrans.GetObject(acBlkTbl(BlockTableRecord.PaperSpace), _
                                          OpenMode.ForWrite)
      Else
          '' Get the ObjectID for the current space from the database
          acBlkTblRec = acTrans.GetObject(acCurDb.CurrentSpaceId, _
                                          OpenMode.ForWrite)
      End If

      '' Create a line that starts at 2,5 and ends at 10,7
      Dim acLine As Line = New Line(New Point3d(2, 5, 0), _
```

```
                                            New Point3d(10, 7, 0))

        acLine.SetDatabaseDefaults()

        '' Add the new object to the block table record and the transaction
        acBlkTblRec.AppendEntity(acLine)
        acTrans.AddNewlyCreatedDBObject(acLine, True)

        '' Save the new line to the database
        acTrans.Commit()
    End Using
End Sub
```

C#

```csharp
using Autodesk.AutoCAD.Runtime;
using Autodesk.AutoCAD.ApplicationServices;
using Autodesk.AutoCAD.DatabaseServices;
using Autodesk.AutoCAD.Geometry;
using Autodesk.AutoCAD.EditorInput;

[CommandMethod("AccessSpace")]
public static void AccessSpace()
{
    // Get the current document and database
    Document acDoc = Application.DocumentManager.MdiActiveDocument;
    Database acCurDb = acDoc.Database;

    // Start a transaction
    using (Transaction acTrans = acCurDb.TransactionManager.StartTransaction())
    {
        // Open the Block table for read
        BlockTable acBlkTbl;
        acBlkTbl = acTrans.GetObject(acCurDb.BlockTableId,
                                     OpenMode.ForRead) as BlockTable;

        // Open the Block table record for read
        BlockTableRecord acBlkTblRec;

        // Request which table record to open
        PromptKeywordOptions pKeyOpts = new PromptKeywordOptions("");
        pKeyOpts.Message = "\nEnter which space to create the line in ";
        pKeyOpts.Keywords.Add("Model");
        pKeyOpts.Keywords.Add("Paper");
        pKeyOpts.Keywords.Add("Current");
        pKeyOpts.AllowNone = false;
        pKeyOpts.AppendKeywordsToMessage = true;

        PromptResult pKeyRes = acDoc.Editor.GetKeywords(pKeyOpts);

        if (pKeyRes.StringResult == "Model")
        {
            // Get the ObjectID for Model space from the Block table
            acBlkTblRec = acTrans.GetObject(acBlkTbl[BlockTableRecord.ModelSpace],
                                            OpenMode.ForWrite) as BlockTableRecord;
        }
        else if (pKeyRes.StringResult == "Paper")
        {
            // Get the ObjectID for Paper space from the Block table
            acBlkTblRec = acTrans.GetObject(acBlkTbl[BlockTableRecord.PaperSpace],
                                            OpenMode.ForWrite) as BlockTableRecord;
        }
        else
        {
```

```
            // Get the ObjectID for the current space from the database
            acBlkTblRec = acTrans.GetObject(acCurDb.CurrentSpaceId,
                                            OpenMode.ForWrite) as BlockTableRecord;
    }

    // Create a line that starts at 2,5 and ends at 10,7
    Line acLine = new Line(new Point3d(2, 5, 0),
                           new Point3d(10, 7, 0));

    acLine.SetDatabaseDefaults();

    // Add the new object to the block table record and the transaction
    acBlkTblRec.AppendEntity(acLine);
    acTrans.AddNewlyCreatedDBObject(acLine, true);

    // Save the new line to the database
    acTrans.Commit();
  }
}
```

VBA/ActiveX Code Reference

```
Public Sub AccessSpace()
    ' Define the valid keywords
    Dim keywordList As String
    keywordList = "Model Paper Current"

    ' Call InitializeUserInput to setup the keywords
    ThisDrawing.Utility.InitializeUserInput 1, keywordList

    ' Get the user input
    Dim retVal As Variant
    retVal = ThisDrawing.Utility.GetKeyword(vbLf & _
                                      "Enter which space to create the line in " & _
                                      "[Model/Paper/Current]: ")

    ' Get the entered keyword
    Dim strVal As String
    strVal = ThisDrawing.Utility.GetInput

    Dim acSpaceObj As Object

    If strVal = "Model" Or _
       (strVal = "Current" And ThisDrawing.ActiveSpace = acModelSpace) Then
        '' Get the Model space object
        Set acSpaceObj = ThisDrawing.ModelSpace
    Else
        '' Get the Paper space object
        Set acSpaceObj = ThisDrawing.PaperSpace
    End If

    '' Create a line that starts at 2,5 and ends at 10,7
    Dim acLine As AcadLine
    Dim dPtStr(0 To 2) As Double
    dPtStr(0) = 2: dPtStr(1) = 5: dPtStr(2) = 0#

    Dim dPtEnd(0 To 2) As Double
    dPtEnd(0) = 10: dPtEnd(1) = 7: dPtEnd(2) = 0#

    Set acLine = acSpaceObj.AddLine(dPtStr, dPtEnd)
End Sub
```

Create Lines

The line is the most basic object in AutoCAD. You can create a variety of lines—single lines, and multiple line segments with and without arcs. In general, you draw lines by specifying coordinate points. Lines when created, inherit the current settings from the drawing database, such as layer, linetype and color.

To create a line, you create a new instance of one of the following objects:

Line

> Creates a line.

Polyline

> Creates a 2D lightweight polyline.

MLine

> Creates a multiline.

Polyline2D

> Creates a 2D polyline.

Polyline3D

> Creates a 3D polyline.

NotePolyline2D objects are the legacy polyline objects that were in AutoCAD prior to Release 14, and the Polyline object represents the new optimized polyline that was introduced with AutoCAD Release 14.

Topics in this section

- [Create a Line Object](#)
- [Create a Polyline object](#)

Create a Line Object

This example adds a line that starts at (5,5,0) and ends at (12,3,0) to Model space.

VB.NET

```
Imports Autodesk.AutoCAD.Runtime
Imports Autodesk.AutoCAD.ApplicationServices
Imports Autodesk.AutoCAD.DatabaseServices
Imports Autodesk.AutoCAD.Geometry
```

```vb
<CommandMethod("AddLine")> _
Public Sub AddLine()
  '' Get the current document and database
  Dim acDoc As Document = Application.DocumentManager.MdiActiveDocument
  Dim acCurDb As Database = acDoc.Database

  '' Start a transaction
  Using acTrans As Transaction = acCurDb.TransactionManager.StartTransaction()

      '' Open the Block table for read
      Dim acBlkTbl As BlockTable
      acBlkTbl = acTrans.GetObject(acCurDb.BlockTableId, OpenMode.ForRead)

      '' Open the Block table record Model space for write
      Dim acBlkTblRec As BlockTableRecord
      acBlkTblRec = acTrans.GetObject(acBlkTbl(BlockTableRecord.ModelSpace), _
                                      OpenMode.ForWrite)

      '' Create a line that starts at 5,5 and ends at 12,3
      Dim acLine As Line = New Line(New Point3d(5, 5, 0), _
                                    New Point3d(12, 3, 0))

      acLine.SetDatabaseDefaults()

      '' Add the new object to the block table record and the transaction
      acBlkTblRec.AppendEntity(acLine)
      acTrans.AddNewlyCreatedDBObject(acLine, True)

      '' Save the new object to the database
      acTrans.Commit()
  End Using
End Sub
```

C#

```csharp
using Autodesk.AutoCAD.Runtime;
using Autodesk.AutoCAD.ApplicationServices;
using Autodesk.AutoCAD.DatabaseServices;
using Autodesk.AutoCAD.Geometry;

[CommandMethod("AddLine")]
public static void AddLine()
{
  // Get the current document and database
  Document acDoc = Application.DocumentManager.MdiActiveDocument;
  Database acCurDb = acDoc.Database;

  // Start a transaction
  using (Transaction acTrans = acCurDb.TransactionManager.StartTransaction())
  {
      // Open the Block table for read
      BlockTable acBlkTbl;
      acBlkTbl = acTrans.GetObject(acCurDb.BlockTableId,
                                   OpenMode.ForRead) as BlockTable;

      // Open the Block table record Model space for write
      BlockTableRecord acBlkTblRec;
      acBlkTblRec = acTrans.GetObject(acBlkTbl[BlockTableRecord.ModelSpace],
                                      OpenMode.ForWrite) as BlockTableRecord;

      // Create a line that starts at 5,5 and ends at 12,3
      Line acLine = new Line(new Point3d(5, 5, 0),
                             new Point3d(12, 3, 0));
```

```
      acLine.SetDatabaseDefaults();

      // Add the new object to the block table record and the transaction
      acBlkTblRec.AppendEntity(acLine);
      acTrans.AddNewlyCreatedDBObject(acLine, true);

      // Save the new object to the database
      acTrans.Commit();
   }
}
```

```
Sub AddLine()
    ' Define the start point
    Dim ptStr(0 To 2) As Double
    ptStr(0) = 5: ptStr(1) = 5: ptStr(2) = 0#

    ' Define the end point
    Dim ptEnd(0 To 2) As Double
    ptEnd(0) = 12: ptEnd(1) = 3: ptEnd(2) = 0#

    ' Create a Line object in model space
    Dim lineObj As AcadLine
    Set lineObj = ThisDrawing.ModelSpace.AddLine(ptStr, ptEnd)

    ThisDrawing.Application.ZoomAll
End Sub
```

Create a Polyline object

This example adds a lightweight polyline with two straight segments using the 2D coordinates (2,4), (4,2), and (6,4) to Model space.

VB.NET

```
Imports Autodesk.AutoCAD.Runtime
Imports Autodesk.AutoCAD.ApplicationServices
Imports Autodesk.AutoCAD.DatabaseServices
Imports Autodesk.AutoCAD.Geometry

<CommandMethod("AddLightweightPolyline")> _
Public Sub AddLightweightPolyline()
  '' Get the current document and database
  Dim acDoc As Document = Application.DocumentManager.MdiActiveDocument
  Dim acCurDb As Database = acDoc.Database

  '' Start a transaction
  Using acTrans As Transaction = acCurDb.TransactionManager.StartTransaction()

      '' Open the Block table for read
      Dim acBlkTbl As BlockTable
      acBlkTbl = acTrans.GetObject(acCurDb.BlockTableId, OpenMode.ForRead)
```

```
    '' Open the Block table record Model space for write
    Dim acBlkTblRec As BlockTableRecord
    acBlkTblRec = acTrans.GetObject(acBlkTbl(BlockTableRecord.ModelSpace), _
                                    OpenMode.ForWrite)

    '' Create a polyline with two segments (3 points)
    Dim acPoly As Polyline = New Polyline()
    acPoly.SetDatabaseDefaults()
    acPoly.AddVertexAt(0, New Point2d(2, 4), 0, 0, 0)
    acPoly.AddVertexAt(1, New Point2d(4, 2), 0, 0, 0)
    acPoly.AddVertexAt(2, New Point2d(6, 4), 0, 0, 0)

    '' Add the new object to the block table record and the transaction
    acBlkTblRec.AppendEntity(acPoly)
    acTrans.AddNewlyCreatedDBObject(acPoly, True)

    '' Save the new object to the database
    acTrans.Commit()
  End Using
End Sub
```

C#

```csharp
using Autodesk.AutoCAD.Runtime;
using Autodesk.AutoCAD.ApplicationServices;
using Autodesk.AutoCAD.DatabaseServices;
using Autodesk.AutoCAD.Geometry;

[CommandMethod("AddLightweightPolyline")]
public static void AddLightweightPolyline()
{
  // Get the current document and database
  Document acDoc = Application.DocumentManager.MdiActiveDocument;
  Database acCurDb = acDoc.Database;

  // Start a transaction
  using (Transaction acTrans = acCurDb.TransactionManager.StartTransaction())
  {
      // Open the Block table for read
      BlockTable acBlkTbl;
      acBlkTbl = acTrans.GetObject(acCurDb.BlockTableId,
                                   OpenMode.ForRead) as BlockTable;

      // Open the Block table record Model space for write
      BlockTableRecord acBlkTblRec;
      acBlkTblRec = acTrans.GetObject(acBlkTbl[BlockTableRecord.ModelSpace],
                                      OpenMode.ForWrite) as BlockTableRecord;

      // Create a polyline with two segments (3 points)
      Polyline acPoly = new Polyline();
      acPoly.SetDatabaseDefaults();
      acPoly.AddVertexAt(0, new Point2d(2, 4), 0, 0, 0);
      acPoly.AddVertexAt(1, new Point2d(4, 2), 0, 0, 0);
      acPoly.AddVertexAt(2, new Point2d(6, 4), 0, 0, 0);

      // Add the new object to the block table record and the transaction
      acBlkTblRec.AppendEntity(acPoly);
      acTrans.AddNewlyCreatedDBObject(acPoly, true);

      // Save the new object to the database
      acTrans.Commit();
  }
}
```

```
Sub AddLightWeightPolyline()
    Dim plineObj As AcadLWPolyline
    Dim points(0 To 5) As Double

    ' Define the 2D polyline points
    points(0) = 2: points(1) = 4
    points(2) = 4: points(3) = 2
    points(4) = 6: points(5) = 4

    ' Create a light weight Polyline object in model space
    Set plineObj = ThisDrawing.ModelSpace. _
                AddLightWeightPolyline(points)
    ThisDrawing.Application.ZoomAll
End Sub
```

Create Curved Objects

You can create a variety of curved objects with AutoCAD, including splines, helixes, circles, arcs, and ellipses. All curves are created on the *XY* plane of the current UCS.

To create a curve, you create a new instance of one of the following objects:

Arc

> Creates an arc given the center point, radius, start and end angles.

Circle

> Creates a circle given the center point and radius.

Ellipse

> Creates an ellipse given the center point, a point on the major axis, and the radius ratio.

Spline

> Creates a quadratic or cubic NURBS (nonuniform rational B-spline) curve.

Helix

> Creates a 2D or 3D helix object.

Topics in this section

- [Create a Circle object](#)
- [Create an Arc object](#)
- [Create a Spline object](#)

Create a Circle object

This example creates a circle in Model space with a center point of (2,3,0) and a radius of 4.25.

VB.NET

```vbnet
Imports Autodesk.AutoCAD.Runtime
Imports Autodesk.AutoCAD.ApplicationServices
Imports Autodesk.AutoCAD.DatabaseServices
Imports Autodesk.AutoCAD.Geometry

<CommandMethod("AddCircle")> _
Public Sub AddCircle()
  '' Get the current document and database
  Dim acDoc As Document = Application.DocumentManager.MdiActiveDocument
  Dim acCurDb As Database = acDoc.Database

  '' Start a transaction
  Using acTrans As Transaction = acCurDb.TransactionManager.StartTransaction()

      '' Open the Block table for read
      Dim acBlkTbl As BlockTable
      acBlkTbl = acTrans.GetObject(acCurDb.BlockTableId, OpenMode.ForRead)

      '' Open the Block table record Model space for write
      Dim acBlkTblRec As BlockTableRecord
      acBlkTblRec = acTrans.GetObject(acBlkTbl(BlockTableRecord.ModelSpace), _
                                      OpenMode.ForWrite)

      '' Create a circle that is at 2,3 with a radius of 4.25
      Dim acCirc As Circle = New Circle()
      acCirc.SetDatabaseDefaults()
      acCirc.Center = New Point3d(2, 3, 0)
      acCirc.Radius = 4.25

      '' Add the new object to the block table record and the transaction
      acBlkTblRec.AppendEntity(acCirc)
      acTrans.AddNewlyCreatedDBObject(acCirc, True)

      '' Save the new object to the database
      acTrans.Commit()
  End Using
End Sub
```

C#

```csharp
using Autodesk.AutoCAD.Runtime;
using Autodesk.AutoCAD.ApplicationServices;
using Autodesk.AutoCAD.DatabaseServices;
using Autodesk.AutoCAD.Geometry;

[CommandMethod("AddCircle")]
public static void AddCircle()
{
  // Get the current document and database
  Document acDoc = Application.DocumentManager.MdiActiveDocument;
  Database acCurDb = acDoc.Database;
```

```
    // Start a transaction
    using (Transaction acTrans = acCurDb.TransactionManager.StartTransaction())
    {
        // Open the Block table for read
        BlockTable acBlkTbl;
        acBlkTbl = acTrans.GetObject(acCurDb.BlockTableId,
                                     OpenMode.ForRead) as BlockTable;

        // Open the Block table record Model space for write
        BlockTableRecord acBlkTblRec;
        acBlkTblRec = acTrans.GetObject(acBlkTbl[BlockTableRecord.ModelSpace],
                                        OpenMode.ForWrite) as BlockTableRecord;

        // Create a circle that is at 2,3 with a radius of 4.25
        Circle acCirc = new Circle();
        acCirc.SetDatabaseDefaults();
        acCirc.Center = new Point3d(2, 3, 0);
        acCirc.Radius = 4.25;

        // Add the new object to the block table record and the transaction
        acBlkTblRec.AppendEntity(acCirc);
        acTrans.AddNewlyCreatedDBObject(acCirc, true);

        // Save the new object to the database
        acTrans.Commit();
    }
}
```

⊟ *VBA/ActiveX Code Reference*

```
Sub AddCircle()
    ' Define the center point
    Dim ptCen(0 To 2) As Double
    ptCen(0) = 2: ptCen(1) = 3: ptCen(2) = 0#

    ' Create a Circle object in model space
    Dim circObj As AcadCircle
    Set circObj = ThisDrawing.ModelSpace.AddCircle(ptCen, 4.25)

    ThisDrawing.Application.ZoomAll
End Sub
```

Create an Arc object

This example creates an arc in Model space with a center point of (6.25,9.125,0), a radius of 6, start angle of 1.117 (64 degrees), and an end angle of 3.5605 (204 degrees).

VB.NET

```
Imports Autodesk.AutoCAD.Runtime
Imports Autodesk.AutoCAD.ApplicationServices
Imports Autodesk.AutoCAD.DatabaseServices
Imports Autodesk.AutoCAD.Geometry

<CommandMethod("AddArc")> _
```

```
Public Sub AddArc()
  '' Get the current document and database
  Dim acDoc As Document = Application.DocumentManager.MdiActiveDocument
  Dim acCurDb As Database = acDoc.Database

  '' Start a transaction
  Using acTrans As Transaction = acCurDb.TransactionManager.StartTransaction()

      '' Open the Block table for read
      Dim acBlkTbl As BlockTable
      acBlkTbl = acTrans.GetObject(acCurDb.BlockTableId, OpenMode.ForRead)

      '' Open the Block table record Model space for write
      Dim acBlkTblRec As BlockTableRecord
      acBlkTblRec = acTrans.GetObject(acBlkTbl(BlockTableRecord.ModelSpace), _
                                      OpenMode.ForWrite)

      '' Create an arc that is at 6.25,9.125 with a radius of 6, and
      '' starts at 64 degrees and ends at 204 degrees
      Dim acArc As Arc = New Arc(New Point3d(6.25, 9.125, 0), _
                                 6, 1.117, 3.5605)

      acArc.SetDatabaseDefaults()

      '' Add the new object to the block table record and the transaction
      acBlkTblRec.AppendEntity(acArc)
      acTrans.AddNewlyCreatedDBObject(acArc, True)

      '' Save the new object to the database
      acTrans.Commit()
  End Using
End Sub
```

C#

```
using Autodesk.AutoCAD.Runtime;
using Autodesk.AutoCAD.ApplicationServices;
using Autodesk.AutoCAD.DatabaseServices;
using Autodesk.AutoCAD.Geometry;

[CommandMethod("AddArc")]
public static void AddArc()
{
  // Get the current document and database
  Document acDoc = Application.DocumentManager.MdiActiveDocument;
  Database acCurDb = acDoc.Database;

  // Start a transaction
  using (Transaction acTrans = acCurDb.TransactionManager.StartTransaction())
  {
      // Open the Block table for read
      BlockTable acBlkTbl;
      acBlkTbl = acTrans.GetObject(acCurDb.BlockTableId,
                                   OpenMode.ForRead) as BlockTable;

      // Open the Block table record Model space for write
      BlockTableRecord acBlkTblRec;
      acBlkTblRec = acTrans.GetObject(acBlkTbl[BlockTableRecord.ModelSpace],
                                      OpenMode.ForWrite) as BlockTableRecord;

      // Create an arc that is at 6.25,9.125 with a radius of 6, and
      // starts at 64 degrees and ends at 204 degrees
      Arc acArc = new Arc(new Point3d(6.25, 9.125, 0),
                          6, 1.117, 3.5605);
```

```
    acArc.SetDatabaseDefaults();

    // Add the new object to the block table record and the transaction
    acBlkTblRec.AppendEntity(acArc);
    acTrans.AddNewlyCreatedDBObject(acArc, true);

    // Save the new line to the database
    acTrans.Commit();
  }
}
```

⊟ *VBA/ActiveX Code Reference*

```
Sub AddArc()
    ' Define the center point
    Dim ptCen(0 To 2) As Double
    ptCen(0) = 6.25: ptCen(1) = 9.125: ptCen(2) = 0#

    ' Create an Arc object in model space
    Dim arcObj As AcadArc
    Set arcObj = ThisDrawing.ModelSpace.AddArc(ptCen, 6#, 1.117, 3.5605)

    ThisDrawing.Application.ZoomAll
End Sub
```

Create a Spline object

This example creates a circle in Model space using three points (0, 0, 0), (5, 5, 0), and (10, 0, 0). The spline has start and end tangents of (0.5, 0.5, 0.0).

VB.NET

```
Imports Autodesk.AutoCAD.Runtime
Imports Autodesk.AutoCAD.ApplicationServices
Imports Autodesk.AutoCAD.DatabaseServices
Imports Autodesk.AutoCAD.Geometry

<CommandMethod("AddSpline")> _
Public Sub AddSpline()
  '' Get the current document and database
  Dim acDoc As Document = Application.DocumentManager.MdiActiveDocument
  Dim acCurDb As Database = acDoc.Database

  '' Start a transaction
  Using acTrans As Transaction = acCurDb.TransactionManager.StartTransaction()

      '' Open the Block table for read
      Dim acBlkTbl As BlockTable
      acBlkTbl = acTrans.GetObject(acCurDb.BlockTableId, OpenMode.ForRead)

      '' Open the Block table record Model space for write
      Dim acBlkTblRec As BlockTableRecord
      acBlkTblRec = acTrans.GetObject(acBlkTbl(BlockTableRecord.ModelSpace), _
                                      OpenMode.ForWrite)
```

```
    '' Define the fit points for the spline
    Dim ptColl As Point3dCollection = New Point3dCollection()
    ptColl.Add(New Point3d(0, 0, 0))
    ptColl.Add(New Point3d(5, 5, 0))
    ptColl.Add(New Point3d(10, 0, 0))

    '' Get a 3D vector from the point (0.5,0.5,0)
    Dim vecTan As Vector3d = New Point3d(0.5, 0.5, 0).GetAsVector

    '' Create a spline through (0, 0, 0), (5, 5, 0), and (10, 0, 0) with a
    '' start and end tangency of (0.5, 0.5, 0.0)
    Dim acSpline As Spline = New Spline(ptColl, vecTan, vecTan, 4, 0.0)

    acSpline.SetDatabaseDefaults()

    '' Add the new object to the block table record and the transaction
    acBlkTblRec.AppendEntity(acSpline)
    acTrans.AddNewlyCreatedDBObject(acSpline, True)

    '' Save the new object to the database
    acTrans.Commit()
  End Using
End Sub
```

C#

```csharp
using Autodesk.AutoCAD.Runtime;
using Autodesk.AutoCAD.ApplicationServices;
using Autodesk.AutoCAD.DatabaseServices;
using Autodesk.AutoCAD.Geometry;

[CommandMethod("AddSpline")]
public static void AddSpline()
{
  // Get the current document and database
  Document acDoc = Application.DocumentManager.MdiActiveDocument;
  Database acCurDb = acDoc.Database;

  // Start a transaction
  using (Transaction acTrans = acCurDb.TransactionManager.StartTransaction())
  {
      // Open the Block table for read
      BlockTable acBlkTbl;
      acBlkTbl = acTrans.GetObject(acCurDb.BlockTableId,
                                   OpenMode.ForRead) as BlockTable;

      // Open the Block table record Model space for write
      BlockTableRecord acBlkTblRec;
      acBlkTblRec = acTrans.GetObject(acBlkTbl[BlockTableRecord.ModelSpace],
                                      OpenMode.ForWrite) as BlockTableRecord;

      // Define the fit points for the spline
      Point3dCollection ptColl = new Point3dCollection();
      ptColl.Add(new Point3d(0, 0, 0));
      ptColl.Add(new Point3d(5, 5, 0));
      ptColl.Add(new Point3d(10, 0, 0));

      // Get a 3D vector from the point (0.5,0.5,0)
      Vector3d vecTan = new Point3d(0.5, 0.5, 0).GetAsVector();

      // Create a spline through (0, 0, 0), (5, 5, 0), and (10, 0, 0) with a
      // start and end tangency of (0.5, 0.5, 0.0)
      Spline acSpline = new Spline(ptColl, vecTan, vecTan, 4, 0.0);
```

```
            acSpline.SetDatabaseDefaults();

            // Add the new object to the block table record and the transaction
            acBlkTblRec.AppendEntity(acSpline);
            acTrans.AddNewlyCreatedDBObject(acSpline, true);

            // Save the new line to the database
            acTrans.Commit();
    }
}
```

⊟ *VBA/ActiveX Code Reference*

```
Sub AddSpline()
    ' This example creates a spline object in model space.
    ' Declare the variables needed
    Dim splineObj As AcadSpline
    Dim startTan(0 To 2) As Double
    Dim endTan(0 To 2) As Double
    Dim fitPoints(0 To 8) As Double

    ' Define the variables
    startTan(0) = 0.5: startTan(1) = 0.5: startTan(2) = 0
    endTan(0) = 0.5: endTan(1) = 0.5: endTan(2) = 0
    fitPoints(0) = 1: fitPoints(1) = 1: fitPoints(2) = 0
    fitPoints(3) = 5: fitPoints(4) = 5: fitPoints(5) = 0
    fitPoints(6) = 10: fitPoints(7) = 0: fitPoints(8) = 0

    ' Create the spline
    Set splineObj = ThisDrawing.ModelSpace.AddSpline _
                        (fitPoints, startTan, endTan)
    ZoomAll
End Sub
```

Create Point Objects

Point objects can be useful, for example, as node or reference points that you can snap to and offset objects from. You can set the style of the point and its size relative to the screen or in absolute units.

The Pdmode and Pdsize properties of the Database object control the appearance of Point objects. A value of 0, 2, 3, and 4 for Pdmode specify a figure to draw through the point. A value of 1 selects nothing to be displayed.

Adding 32, 64, or 96 to the previous value selects a shape to draw around the point in addition to the figure drawn through it:

32 33 34 35 36

64 65 66 67 68

96 97 98 99 100
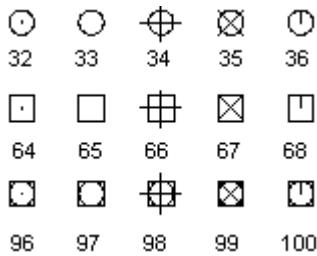
Pdsize controls the size of the point figures, except for when Pdmode is 0 and 1. A 0 setting generates the point at 5 percent of the graphics area height. Setting Pdsize to a positive value specifies an absolute size for the point figures. A negative value is interpreted as a percentage of the viewport size. The size of all points is recalculated when the drawing is regenerated.

After you change Pdmode and Pdsize, the appearance of existing points changes the next time the drawing is regenerated.

Create a Point object and change its appearance

The following example creates a Point object in Model space at the coordinate (5, 5, 0). The Pdmode and Pdsize properties are then updated.

VB.NET

```
Imports Autodesk.AutoCAD.Runtime
Imports Autodesk.AutoCAD.ApplicationServices
Imports Autodesk.AutoCAD.DatabaseServices
Imports Autodesk.AutoCAD.Geometry

<CommandMethod("AddPointAndSetPointStyle")> _
Public Sub AddPointAndSetPointStyle()
  '' Get the current document and database
  Dim acDoc As Document = Application.DocumentManager.MdiActiveDocument
  Dim acCurDb As Database = acDoc.Database

  '' Start a transaction
  Using acTrans As Transaction = acCurDb.TransactionManager.StartTransaction()

      '' Open the Block table for read
      Dim acBlkTbl As BlockTable
      acBlkTbl = acTrans.GetObject(acCurDb.BlockTableId, OpenMode.ForRead)

      '' Open the Block table record Model space for write
      Dim acBlkTblRec As BlockTableRecord
      acBlkTblRec = acTrans.GetObject(acBlkTbl(BlockTableRecord.ModelSpace), _
                                      OpenMode.ForWrite)

      '' Create a point at (4, 3, 0) in Model space
      Dim acPoint As DBPoint = New DBPoint(New Point3d(4, 3, 0))

      acPoint.SetDatabaseDefaults()

      '' Add the new object to the block table record and the transaction
      acBlkTblRec.AppendEntity(acPoint)
      acTrans.AddNewlyCreatedDBObject(acPoint, True)

      '' Set the style for all point objects in the drawing
      acCurDb.Pdmode = 34
      acCurDb.Pdsize = 1
```

```
      '' Save the new object to the database
      acTrans.Commit()
   End Using
End Sub
```

C#

```csharp
using Autodesk.AutoCAD.Runtime;
using Autodesk.AutoCAD.ApplicationServices;
using Autodesk.AutoCAD.DatabaseServices;
using Autodesk.AutoCAD.Geometry;

[CommandMethod("AddPointAndSetPointStyle")]
public static void AddPointAndSetPointStyle()
{
  // Get the current document and database
  Document acDoc = Application.DocumentManager.MdiActiveDocument;
  Database acCurDb = acDoc.Database;

  // Start a transaction
  using (Transaction acTrans = acCurDb.TransactionManager.StartTransaction())
  {
      // Open the Block table for read
      BlockTable acBlkTbl;
      acBlkTbl = acTrans.GetObject(acCurDb.BlockTableId,
                                   OpenMode.ForRead) as BlockTable;

      // Open the Block table record Model space for write
      BlockTableRecord acBlkTblRec;
      acBlkTblRec = acTrans.GetObject(acBlkTbl[BlockTableRecord.ModelSpace],
                                      OpenMode.ForWrite) as BlockTableRecord;

      // Create a point at (4, 3, 0) in Model space
      DBPoint acPoint = new DBPoint(new Point3d(4, 3, 0));

      acPoint.SetDatabaseDefaults();

      // Add the new object to the block table record and the transaction
      acBlkTblRec.AppendEntity(acPoint);
      acTrans.AddNewlyCreatedDBObject(acPoint, true);

      // Set the style for all point objects in the drawing
      acCurDb.Pdmode = 34;
      acCurDb.Pdsize = 1;

      // Save the new object to the database
      acTrans.Commit();
  }
}
```

⊟ *VBA/ActiveX Code Reference*

```vba
Sub AddPointAndSetPointStyle()
    Dim pointObj As AcadPoint
    Dim location(0 To 2) As Double

    ' Define the location of the point
    location(0) = 4#: location(1) = 3#: location(2) = 0#

    ' Create the point
    Set pointObj = ThisDrawing.ModelSpace.AddPoint(location)
    ThisDrawing.SetVariable "PDMODE", 34
```

```
    ThisDrawing.SetVariable "PDSIZE", 1

    ZoomAll
End Sub
```


Create Solid-Filled Areas

You can create triangular and quadrilateral areas filled with a color. When creating filled areas, set the FILLMODE system variable to off to improve performance and back on once the fills have been created.

When you create a quadrilateral solid-filled area, the sequence of the third and fourth points determines its shape. Compare the following illustrations:
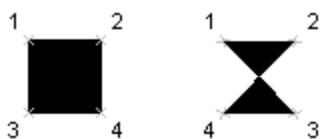
The first two points define one edge of the polygon. The third point is defined diagonally opposite from the second. If the fourth point is set equal to the third point, then a filled triangle is created.

For more information about filling solids, see "Create Solid-Filled Areas" in the *AutoCAD User's Guide*.

Create a solid-filled object

The following example creates a quadrilateral solid (bow-tie) in Model space using the coordinates (0, 0, 0), (5, 0, 0), (5, 8, 0), and (0, 8, 0). It also creates a quadrilateral solid in a rectangular shape using the coordinates (10, 0, 0), (15, 0, 0), (10, 8, 0), and (15, 8, 0).

VB.NET

```
Imports Autodesk.AutoCAD.Runtime
Imports Autodesk.AutoCAD.ApplicationServices
Imports Autodesk.AutoCAD.DatabaseServices
Imports Autodesk.AutoCAD.Geometry

<CommandMethod("Add2DSolid")> _
Public Sub Add2DSolid()
  '' Get the current document and database
  Dim acDoc As Document = Application.DocumentManager.MdiActiveDocument
  Dim acCurDb As Database = acDoc.Database

  '' Start a transaction
  Using acTrans As Transaction = acCurDb.TransactionManager.StartTransaction()

      '' Open the Block table for read
      Dim acBlkTbl As BlockTable
      acBlkTbl = acTrans.GetObject(acCurDb.BlockTableId, OpenMode.ForRead)

      '' Open the Block table record Model space for write
      Dim acBlkTblRec As BlockTableRecord
```

```vb
        acBlkTblRec = acTrans.GetObject(acBlkTbl(BlockTableRecord.ModelSpace), _
                                        OpenMode.ForWrite)

        '' Create a quadrilateral (bow-tie) solid in Model space
        Dim ac2DSolidBow As Solid = New Solid(New Point3d(0, 0, 0), _
                                              New Point3d(5, 0, 0), _
                                              New Point3d(5, 8, 0), _
                                              New Point3d(0, 8, 0))

        ac2DSolidBow.SetDatabaseDefaults()

        '' Add the new object to the block table record and the transaction
        acBlkTblRec.AppendEntity(ac2DSolidBow)
        acTrans.AddNewlyCreatedDBObject(ac2DSolidBow, True)

        '' Create a quadrilateral (square) solid in Model space
        Dim ac2DSolidSqr As Solid = New Solid(New Point3d(10, 0, 0), _
                                              New Point3d(15, 0, 0), _
                                              New Point3d(10, 8, 0), _
                                              New Point3d(15, 8, 0))

        ac2DSolidSqr.SetDatabaseDefaults()

        '' Add the new object to the block table record and the transaction
        acBlkTblRec.AppendEntity(ac2DSolidSqr)
        acTrans.AddNewlyCreatedDBObject(ac2DSolidSqr, True)

        '' Save the new object to the database
        acTrans.Commit()
    End Using
End Sub
```

C#

```csharp
using Autodesk.AutoCAD.Runtime;
using Autodesk.AutoCAD.ApplicationServices;
using Autodesk.AutoCAD.DatabaseServices;
using Autodesk.AutoCAD.Geometry;

[CommandMethod("Add2DSolid")]
public static void Add2DSolid()
{
  // Get the current document and database
  Document acDoc = Application.DocumentManager.MdiActiveDocument;
  Database acCurDb = acDoc.Database;

  // Start a transaction
  using (Transaction acTrans = acCurDb.TransactionManager.StartTransaction())
  {
      // Open the Block table for read
      BlockTable acBlkTbl;
      acBlkTbl = acTrans.GetObject(acCurDb.BlockTableId,
                                   OpenMode.ForRead) as BlockTable;

      // Open the Block table record Model space for write
      BlockTableRecord acBlkTblRec;
      acBlkTblRec = acTrans.GetObject(acBlkTbl[BlockTableRecord.ModelSpace],
                                   OpenMode.ForWrite) as BlockTableRecord;

      // Create a quadrilateral (bow-tie) solid in Model space
      Solid ac2DSolidBow = new Solid(new Point3d(0, 0, 0),
                                     new Point3d(5, 0, 0),
                                     new Point3d(5, 8, 0),
                                     new Point3d(0, 8, 0));
```

```
        ac2DSolidBow.SetDatabaseDefaults();

        // Add the new object to the block table record and the transaction
        acBlkTblRec.AppendEntity(ac2DSolidBow);
        acTrans.AddNewlyCreatedDBObject(ac2DSolidBow, true);

        // Create a quadrilateral (square) solid in Model space
        Solid ac2DSolidSqr = new Solid(new Point3d(10, 0, 0),
                                       new Point3d(15, 0, 0),
                                       new Point3d(10, 8, 0),
                                       new Point3d(15, 8, 0));

        ac2DSolidSqr.SetDatabaseDefaults();

        // Add the new object to the block table record and the transaction
        acBlkTblRec.AppendEntity(ac2DSolidSqr);
        acTrans.AddNewlyCreatedDBObject(ac2DSolidSqr, true);

        // Save the new object to the database
        acTrans.Commit();
    }
}
```

VBA/ActiveX Code Reference

```
Sub Add2DSolid()
    Dim solidObj As AcadSolid
    Dim point1(0 To 2) As Double
    Dim point2(0 To 2) As Double
    Dim point3(0 To 2) As Double
    Dim point4(0 To 2) As Double

    ' Define the solid
    point1(0) = 0#: point1(1) = 0#: point1(2) = 0#
    point2(0) = 5#: point2(1) = 0#: point2(2) = 0#
    point3(0) = 5#: point3(1) = 8#: point3(2) = 0#
    point4(0) = 0#: point4(1) = 8#: point4(2) = 0#
    ' Create the solid object in model space
    Set solidObj = ThisDrawing.ModelSpace.AddSolid _
                                (point1, point2, point3, point4)

    ' Define the solid
    point1(0) = 10#: point1(1) = 0#: point1(2) = 0#
    point2(0) = 15#: point2(1) = 0#: point2(2) = 0#
    point3(0) = 10#: point3(1) = 8#: point3(2) = 0#
    point4(0) = 15#: point4(1) = 8#: point4(2) = 0#
    ' Create the solid object in model space
    Set solidObj = ThisDrawing.ModelSpace.AddSolid _
                                (point1, point2, point3, point4)

    ZoomAll
End Sub
```

Work with Regions

Regions are two-dimensional enclosed areas you create from closed shapes called loops. A loop is a closed boundary that is made up of straight and curved objects which do not intersect themselves. Loops can be combinations of lines, lightweight polylines, 2D and 3D polylines, circles, arcs, ellipses, elliptical arcs, splines, 3D faces, traces, and solids.

The objects that make up the loops must either be closed or form closed areas by sharing endpoints with other objects. They must also be coplanar (on the same plane). The loops that make up a region must be defined as an array of objects.

For more information about working with regions, see "Create and Combine Areas (Regions)" in the *AutoCAD User's Guide*.

Topics in this section

- Create Regions
- Create Composite Regions

Create Regions

Regions are added to a BlockTableRecord object by creating an instance of a Region object and then appending it to a BlockTableRecord. Before you can add it to a BlockTableRecord object, a region needs to be calculated based on the objects that form the closed loop. The CreateFromCurves function creates a region out of every closed loop formed by the input array of objects. The CreateFromCurves method returns and requires a DBObjectCollection object.

AutoCAD converts closed 2D and planar 3D polylines to separate regions, then converts polylines, lines, and curves that form closed planar loops. If more than two curves share an endpoint, the resulting region might be arbitrary. Because of this, several regions may actually be created with the CreateFromCurves method. You need to append each region created to a BlockTableRecord object.

Create a simple region

The following example creates a region from a single circle.

VB.NET

```
Imports Autodesk.AutoCAD.Runtime
Imports Autodesk.AutoCAD.ApplicationServices
Imports Autodesk.AutoCAD.DatabaseServices
Imports Autodesk.AutoCAD.Geometry

<CommandMethod("AddRegion")> _
Public Sub AddRegion()
```

```vb
    '' Get the current document and database
    Dim acDoc As Document = Application.DocumentManager.MdiActiveDocument
    Dim acCurDb As Database = acDoc.Database

    '' Start a transaction
    Using acTrans As Transaction = acCurDb.TransactionManager.StartTransaction()

        '' Open the Block table for read
        Dim acBlkTbl As BlockTable
        acBlkTbl = acTrans.GetObject(acCurDb.BlockTableId, OpenMode.ForRead)

        '' Open the Block table record Model space for write
        Dim acBlkTblRec As BlockTableRecord
        acBlkTblRec = acTrans.GetObject(acBlkTbl(BlockTableRecord.ModelSpace), _
                                        OpenMode.ForWrite)

        '' Create an in memory circle
        Using acCirc As Circle = New Circle()
            acCirc.SetDatabaseDefaults()
            acCirc.Center = New Point3d(2, 2, 0)
            acCirc.Radius = 5

            '' Adds the circle to an object array
            Dim acDBObjColl As DBObjectCollection = New DBObjectCollection()
            acDBObjColl.Add(acCirc)

            '' Calculate the regions based on each closed loop
            Dim myRegionColl As DBObjectCollection = New DBObjectCollection()
            myRegionColl = Region.CreateFromCurves(acDBObjColl)
            Dim acRegion As Region = myRegionColl(0)

            '' Add the new object to the block table record and the transaction
            acBlkTblRec.AppendEntity(acRegion)
            acTrans.AddNewlyCreatedDBObject(acRegion, True)

            '' Dispose of the in memory object not appended to the database
        End Using

        '' Save the new object to the database
        acTrans.Commit()
    End Using
End Sub
```

C#

```csharp
using Autodesk.AutoCAD.Runtime;
using Autodesk.AutoCAD.ApplicationServices;
using Autodesk.AutoCAD.DatabaseServices;
using Autodesk.AutoCAD.Geometry;

[CommandMethod("AddRegion")]
public static void AddRegion()
{
  // Get the current document and database
  Document acDoc = Application.DocumentManager.MdiActiveDocument;
  Database acCurDb = acDoc.Database;

  // Start a transaction
  using (Transaction acTrans = acCurDb.TransactionManager.StartTransaction())
  {
      // Open the Block table for read
      BlockTable acBlkTbl;
      acBlkTbl = acTrans.GetObject(acCurDb.BlockTableId,
                                    OpenMode.ForRead) as BlockTable;
```

```
        // Open the Block table record Model space for write
        BlockTableRecord acBlkTblRec;
        acBlkTblRec = acTrans.GetObject(acBlkTbl[BlockTableRecord.ModelSpace],
                                        OpenMode.ForWrite) as BlockTableRecord;

        // Create an in memory circle
        using (Circle acCirc = new Circle())
        {
            acCirc.SetDatabaseDefaults();
            acCirc.Center = new Point3d(2, 2, 0);
            acCirc.Radius = 5;

            // Adds the circle to an object array
            DBObjectCollection acDBObjColl = new DBObjectCollection();
            acDBObjColl.Add(acCirc);

            // Calculate the regions based on each closed loop
            DBObjectCollection myRegionColl = new DBObjectCollection();
            myRegionColl = Region.CreateFromCurves(acDBObjColl);
            Region acRegion = myRegionColl[0] as Region;

            // Add the new object to the block table record and the transaction
            acBlkTblRec.AppendEntity(acRegion);
            acTrans.AddNewlyCreatedDBObject(acRegion, true);

            // Dispose of the in memory circle not appended to the database
        }

        // Save the new object to the database
        acTrans.Commit();
    }
}
```

⊟ *VBA/ActiveX Code Reference*

```
Sub AddRegion()
    ' Define an array to hold the
    ' boundaries of the region.
    Dim curves(0 To 0) As AcadCircle

    ' Create a circle to become a
    ' boundary for the region.
    Dim center(0 To 2) As Double
    Dim radius As Double
    center(0) = 2
    center(1) = 2
    center(2) = 0
    radius = 5#
    Set curves(0) = ThisDrawing.ModelSpace.AddCircle _
                                (center, radius)

    ' Create the region
    Dim regionObj As Variant
    regionObj = ThisDrawing.ModelSpace.AddRegion(curves)

    ZoomAll
End Sub
```

Create Composite Regions

You can create composite regions by subtracting, combining, or finding the intersection of regions or 3D solids. You can then extrude or revolve composite regions to create complex solids. To create a composite region, use the BooleanOperation method.

Subtract regions

When you subtract one region from another, you call the BooleanOperation method from the first region. This is the region from which you want to subtract. For example, to calculate how much carpeting is needed for a floor plan, call the BooleanOperation method from the outer boundary of the floor space and use the non-carpeted areas, such as pillars and counters, as the object in the Boolean parameter list.

Unite regions

To unite regions, call the BooleanOperation method and use the constant BooleanOperationType.BoolUnite for the operation instead of BooleanOperationType.BoolSubtract. You can combine regions in any order to unite them.

Find the intersection of two regions

To find the intersection of two regions, use the constant BooleanOperationType.BoolIntersect. You can combine regions in any order to intersect them.

Create a composite region

The following example creates two regions from two circles and then subtracts the smaller region from the large one to create a wheel.

VB.NET

```
Imports Autodesk.AutoCAD.Runtime
Imports Autodesk.AutoCAD.ApplicationServices
Imports Autodesk.AutoCAD.DatabaseServices
Imports Autodesk.AutoCAD.Geometry

<CommandMethod("CreateCompositeRegions")> _
Public Sub CreateCompositeRegions()
  '' Get the current document and database
  Dim acDoc As Document = Application.DocumentManager.MdiActiveDocument
  Dim acCurDb As Database = acDoc.Database

  '' Start a transaction
  Using acTrans As Transaction = acCurDb.TransactionManager.StartTransaction()

    '' Open the Block table for read
    Dim acBlkTbl As BlockTable
    acBlkTbl = acTrans.GetObject(acCurDb.BlockTableId, OpenMode.ForRead)

    '' Open the Block table record Model space for write
    Dim acBlkTblRec As BlockTableRecord
    acBlkTblRec = acTrans.GetObject(acBlkTbl(BlockTableRecord.ModelSpace), _
```

```
                                                OpenMode.ForWrite)

      '' Create two in memory circles
      Dim acCirc1 As Circle = New Circle()
      acCirc1.SetDatabaseDefaults()
      acCirc1.Center = New Point3d(4, 4, 0)
      acCirc1.Radius = 2

      Dim acCirc2 As Circle = New Circle()
      acCirc2.SetDatabaseDefaults()
      acCirc2.Center = New Point3d(4, 4, 0)
      acCirc2.Radius = 1

      '' Adds the circle to an object array
      Dim acDBObjColl As DBObjectCollection = New DBObjectCollection()
      acDBObjColl.Add(acCirc1)
      acDBObjColl.Add(acCirc2)

      '' Calculate the regions based on each closed loop
      Dim myRegionColl As DBObjectCollection = New DBObjectCollection()
      myRegionColl = Region.CreateFromCurves(acDBObjColl)
      Dim acRegion1 As Region = myRegionColl(0)
      Dim acRegion2 As Region = myRegionColl(1)

      '' Subtract region 1 from region 2
      If acRegion1.Area > acRegion2.Area Then
          '' Subtract the smaller region from the larger one
          acRegion1.BooleanOperation(BooleanOperationType.BoolSubtract, acRegion2)
          acRegion2.Dispose()

          '' Add the final region to the database
          acBlkTblRec.AppendEntity(acRegion1)
          acTrans.AddNewlyCreatedDBObject(acRegion1, True)
      Else
          '' Subtract the smaller region from the larger one
          acRegion2.BooleanOperation(BooleanOperationType.BoolSubtract, acRegion1)
          acRegion1.Dispose()

          '' Add the final region to the database
          acBlkTblRec.AppendEntity(acRegion2)
          acTrans.AddNewlyCreatedDBObject(acRegion2, True)
      End If

      '' Dispose of the in memory objects not appended to the database
      acCirc1.Dispose()
      acCirc2.Dispose()

      '' Save the new object to the database
      acTrans.Commit()
  End Using
End Sub
```

C#

```csharp
using Autodesk.AutoCAD.Runtime;
using Autodesk.AutoCAD.ApplicationServices;
using Autodesk.AutoCAD.DatabaseServices;
using Autodesk.AutoCAD.Geometry;

[CommandMethod("CreateCompositeRegions")]
public static void CreateCompositeRegions()
{
  // Get the current document and database
  Document acDoc = Application.DocumentManager.MdiActiveDocument;
```

```csharp
Database acCurDb = acDoc.Database;

// Start a transaction
using (Transaction acTrans = acCurDb.TransactionManager.StartTransaction())
{
    // Open the Block table for read
    BlockTable acBlkTbl;
    acBlkTbl = acTrans.GetObject(acCurDb.BlockTableId,
                                 OpenMode.ForRead) as BlockTable;

    // Open the Block table record Model space for write
    BlockTableRecord acBlkTblRec;
    acBlkTblRec = acTrans.GetObject(acBlkTbl[BlockTableRecord.ModelSpace],
                                    OpenMode.ForWrite) as BlockTableRecord;

    // Create two in memory circles
    Circle acCirc1 = new Circle();
    acCirc1.SetDatabaseDefaults();
    acCirc1.Center = new Point3d(4, 4, 0);
    acCirc1.Radius = 2;

    Circle acCirc2 = new Circle();
    acCirc2.SetDatabaseDefaults();
    acCirc2.Center = new Point3d(4, 4, 0);
    acCirc2.Radius = 1;

    // Adds the circle to an object array
    DBObjectCollection acDBObjColl = new DBObjectCollection();
    acDBObjColl.Add(acCirc1);
    acDBObjColl.Add(acCirc2);

    // Calculate the regions based on each closed loop
    DBObjectCollection myRegionColl = new DBObjectCollection();
    myRegionColl = Region.CreateFromCurves(acDBObjColl);
    Region acRegion1 = myRegionColl[0] as Region;
    Region acRegion2 = myRegionColl[1] as Region;

    // Subtract region 1 from region 2
    if (acRegion1.Area > acRegion2.Area)
    {
        // Subtract the smaller region from the larger one
        acRegion1.BooleanOperation(BooleanOperationType.BoolSubtract,
acRegion2);
        acRegion2.Dispose();

        // Add the final region to the database
        acBlkTblRec.AppendEntity(acRegion1);
        acTrans.AddNewlyCreatedDBObject(acRegion1, true);
    }
    else
    {
        // Subtract the smaller region from the larger one
        acRegion2.BooleanOperation(BooleanOperationType.BoolSubtract,
acRegion1);
        acRegion1.Dispose();

        // Add the final region to the database
        acBlkTblRec.AppendEntity(acRegion2);
        acTrans.AddNewlyCreatedDBObject(acRegion2, true);
    }

    // Dispose of the in memory objects not appended to the database
    acCirc1.Dispose();
    acCirc2.Dispose();
```

```
      // Save the new object to the database
      acTrans.Commit();
   }
}
```

```
Sub CreateCompositeRegions()
    ' Create two circles, one representing outside of the wheel,
    ' the other the center of the wheel
    Dim DonutParts(0 To 1) As AcadCircle
    Dim center(0 To 2) As Double
    Dim radius As Double
    center(0) = 4
    center(1) = 4
    center(2) = 0
    radius = 2#
    Set WheelParts(0) = ThisDrawing.ModelSpace. _
                          AddCircle(center, radius)
    radius = 1#
    Set WheelParts(1) = ThisDrawing.ModelSpace. _
                          AddCircle(center, radius)

    ' Create a region from the two circles
    Dim regions As Variant
    regions = ThisDrawing.ModelSpace.AddRegion(WheelParts)

    ' Copy the regions into the region variables for ease of use
    Dim WheelOuter As AcadRegion
    Dim WheelInner As AcadRegion

    If regions(0).Area > regions(1).Area Then
        ' The first region is the outer edge of the wheel
        Set WheelOuter = regions(0)
        Set WheelInner = regions(1)
    Else
        ' The first region is the inner edge of the wheel
        Set WheelInner = regions(0)
        Set WheelOuter = regions(1)
    End If

    ' Subtract the smaller circle from the larger circle
    WheelOuter.Boolean acSubtraction, WheelInner
End Sub
```

Create Hatches

Closed boundaries can be filled with a pattern.

When creating hatch, you do not initially specify the area to be filled. First you must create the Hatch object. Once this is done, you can specify the outer loop, which is the outermost boundary for the hatch. You can then continue to specify any inner loops that may exist in the hatch.

For more information about working with hatches, see "Overview of Hatch Patterns and Fills" in the *AutoCAD User's Guide.*

Topics in this section

Create a Hatch Object

When creating a Hatch object, you specify the hatch pattern type, the hatch pattern name, and the associativity. Once a Hatch object has been created, you will not be able to change the hatch associativity.

To create a Hatch object, you create a new instance of the object and then use the AppendEntity method to add it to a BlockTableRecord object.

Associate a Hatch

You can create associative or nonassociative hatches. Associative hatches are linked to their boundaries and updated when the boundaries are modified. Nonassociative hatches are independent of their boundaries.

To make a hatch associative, set the Associative property of the hatch object created to TRUE. To make a hatch nonassociative, set the Associative property to FALSE.

Associativity for hatch must be set before appending the hatch loop. If a hatch object is nonassociative, you can make it associative again by setting the Associative property to TRUE and reappending the hatch loop.

Assign the Hatch Pattern Type and Name

AutoCAD supplies a solid-fill and more than fifty industry-standard hatch patterns. Hatch patterns highlight a particular feature or area of a drawing. For example, patterns can help differentiate the components of a 3D object or represent the materials that make up an object.

You can use a pattern supplied with AutoCAD or one from an external pattern library. For a table of the hatch patterns supplied with AutoCAD, see the *AutoCAD Command Reference Guide.*

To specify a unique pattern, you must specify both a pattern type and name for the Hatch object. The pattern type specifies where to look up the pattern name. When entering the pattern type, use one of the following constants:

HatchPatternType.PreDefined

Selects the pattern name from those defined in the *acad.pat* or files.

HatchPatternType.**UserDefined**

Defines a pattern of lines using the current linetype.

HatchPatternType.**CustomDefined**

Selects the pattern name from a PAT other than the *acad.pat* or files.

When entering the pattern name, use a name that is valid for the file specified by the pattern type.

Define the Hatch Boundaries

Once the Hatch object is created, the hatch boundaries can be added. Boundaries can be any combination of lines, arcs, circles, 2D polylines, ellipses, splines, and regions.

The first boundary added must be the outer boundary, which defines the outermost limits to be filled by the hatch. To add the outer boundary, use the AppendLoop method with the HatchLoopTypes.Outermost constant for the type of loop to append.

Once the outer boundary is defined, you can continue adding additional boundaries. Add inner boundaries using the AppendLoop method with the HatchLoopTypes.Default constant.

Inner boundaries define islands within the hatch. How these islands are handled by the Hatch object depends on the setting of the HatchStyle property. The HatchStyle property can be set to one of the following conditions:

Hatch style definitions

HatchStyle	Condition	Description
	Normal (HatchStyle.Normal)	Specifies standard style, or normal. This option hatches inward from the outermost area boundary. If AutoCAD encounters an internal boundary, it turns off hatching until it encounters another boundary. This is the default setting for the HatchStyle property.
	Outer (HatchStyle.Outer)	Fills the outermost areas only. This style also hatches inward from the area boundary, but it turns off hatching if it encounters an internal boundary and does not turn it back on again.
	Ignore (HatchStyle.Ignore)	Ignores internal structure. This option hatches through all internal objects.

When you have finished defining the hatch it must be evaluated before it can be displayed. Use the EvaluateHatch method to do this.

Create a Hatch object

The following example creates an associated hatch in Model space. Once the hatch has been created, you can change the size of the circle that the hatch is associated with. The hatch will change to match the size of the circle.

VB.NET

```
Imports Autodesk.AutoCAD.Runtime
Imports Autodesk.AutoCAD.ApplicationServices
Imports Autodesk.AutoCAD.DatabaseServices
Imports Autodesk.AutoCAD.Geometry

<CommandMethod("AddHatch")> _
Public Sub AddHatch()
  '' Get the current document and database
  Dim acDoc As Document = Application.DocumentManager.MdiActiveDocument
  Dim acCurDb As Database = acDoc.Database

  '' Start a transaction
  Using acTrans As Transaction = acCurDb.TransactionManager.StartTransaction()

      '' Open the Block table for read
      Dim acBlkTbl As BlockTable
      acBlkTbl = acTrans.GetObject(acCurDb.BlockTableId, OpenMode.ForRead)

      '' Open the Block table record Model space for write
      Dim acBlkTblRec As BlockTableRecord
      acBlkTblRec = acTrans.GetObject(acBlkTbl(BlockTableRecord.ModelSpace), _
                                 OpenMode.ForWrite)

      '' Create a circle object for the closed boundary to hatch
      Dim acCirc As Circle = New Circle()
      acCirc.SetDatabaseDefaults()
      acCirc.Center = New Point3d(3, 3, 0)
      acCirc.Radius = 1

      '' Add the new circle object to the block table record and the transaction
```

```
        acBlkTblRec.AppendEntity(acCirc)
        acTrans.AddNewlyCreatedDBObject(acCirc, True)

        '' Adds the circle to an object id array
        Dim acObjIdColl As ObjectIdCollection = New ObjectIdCollection()
        acObjIdColl.Add(acCirc.ObjectId)

        '' Create the hatch object and append it to the block table record
        Dim acHatch As Hatch = New Hatch()
        acBlkTblRec.AppendEntity(acHatch)
        acTrans.AddNewlyCreatedDBObject(acHatch, True)

        '' Set the properties of the hatch object
        '' Associative must be set after the hatch object is appended to the
        '' block table record and before AppendLoop
        acHatch.SetDatabaseDefaults()
        acHatch.SetHatchPattern(HatchPatternType.PreDefined, "ANSI31")
        acHatch.Associative = True
        acHatch.AppendLoop(HatchLoopTypes.Outermost, acObjIdColl)
        acHatch.EvaluateHatch(True)

        '' Save the new object to the database
        acTrans.Commit()
    End Using
End Sub
```

C#

```csharp
using Autodesk.AutoCAD.Runtime;
using Autodesk.AutoCAD.ApplicationServices;
using Autodesk.AutoCAD.DatabaseServices;
using Autodesk.AutoCAD.Geometry;

[CommandMethod("AddHatch")]
public static void AddHatch()
{
    // Get the current document and database
    Document acDoc = Application.DocumentManager.MdiActiveDocument;
    Database acCurDb = acDoc.Database;

    // Start a transaction
    using (Transaction acTrans = acCurDb.TransactionManager.StartTransaction())
    {
        // Open the Block table for read
        BlockTable acBlkTbl;
        acBlkTbl = acTrans.GetObject(acCurDb.BlockTableId,
                                     OpenMode.ForRead) as BlockTable;

        // Open the Block table record Model space for write
        BlockTableRecord acBlkTblRec;
        acBlkTblRec = acTrans.GetObject(acBlkTbl[BlockTableRecord.ModelSpace],
                                        OpenMode.ForWrite) as BlockTableRecord;

        // Create a circle object for the closed boundary to hatch
        Circle acCirc = new Circle();
        acCirc.SetDatabaseDefaults();
        acCirc.Center = new Point3d(3, 3, 0);
        acCirc.Radius = 1;

        // Add the new circle object to the block table record and the transaction
        acBlkTblRec.AppendEntity(acCirc);
        acTrans.AddNewlyCreatedDBObject(acCirc, true);

        // Adds the circle to an object id array
```

```
      ObjectIdCollection acObjIdColl = new ObjectIdCollection();
      acObjIdColl.Add(acCirc.ObjectId);

      // Create the hatch object and append it to the block table record
      Hatch acHatch = new Hatch();
      acBlkTblRec.AppendEntity(acHatch);
      acTrans.AddNewlyCreatedDBObject(acHatch, true);

      // Set the properties of the hatch object
      // Associative must be set after the hatch object is appended to the
      // block table record and before AppendLoop
      acHatch.SetDatabaseDefaults();
      acHatch.SetHatchPattern(HatchPatternType.PreDefined, "ANSI31");
      acHatch.Associative = true;
      acHatch.AppendLoop(HatchLoopTypes.Outermost, acObjIdColl);
      acHatch.EvaluateHatch(true);

      // Save the new object to the database
      acTrans.Commit();
  }
}
```

⊟ *VBA/ActiveX Code Reference*

```
Sub AddHatch()
    Dim hatchObj As AcadHatch
    Dim patternName As String
    Dim PatternType As Long
    Dim bAssociativity As Boolean

    ' Define the hatch
    patternName = "ANSI31"
    PatternType = 0
    bAssociativity = True

    ' Create the associative Hatch object
    Set hatchObj = ThisDrawing.ModelSpace.AddHatch _
                        (PatternType, patternName, bAssociativity)

    ' Create the outer boundary for the hatch. (a circle)
    Dim outerLoop(0 To 0) As AcadEntity
    Dim center(0 To 2) As Double
    Dim radius As Double
    center(0) = 3: center(1) = 3: center(2) = 0
    radius = 1
    Set outerLoop(0) = ThisDrawing.ModelSpace. _
                            AddCircle(center, radius)

    ' Append the outerboundary to the hatch
    ' object, and display the hatch
    hatchObj.AppendOuterLoop (outerLoop)
    hatchObj.Evaluate
    ThisDrawing.Regen True
End Sub
```

Work with Selection Sets

A selection set can consist of a single object, or it can be a more complex grouping: for example, the set of objects on a certain layer. A selection set is typically created by requesting a user to select an object in the drawing area before a command is started through pick first selection or at the

Select objects:

prompt when a command is active.

Selection sets are not persistent objects, if you need to maintain a selection set for use between multiple commands or future use, you will need to create a custom dictionary and record the object ids found in the selection set as a record.

Topics in this section

- [Obtain the PickFirst Selection Set](#)
- [Select Objects in the Drawing Area](#)
- [Add To or Merge Multiple Selection Sets](#)
- [Define Rules for Selection Filters](#)
- [Remove Objects From a Selection Set](#)

Obtain the PickFirst Selection Set

The PickFirst selection set is created when you select objects prior to starting a command. Several conditions must be present in order to obtain the objects of a PickFirst selection set, these conditions are:

- PICKFIRST system variable must be set to 1
- UsePickSet command flag must be defined with the command that should use the Pickfirst selection set
- Call the SelectImplied method to obtain the PickFirst selection set

The SetImpliedSelection method is used to clear the current PickFirst selection set.

Get the Pickfirst selection set

This example displays the number of objects in the PickFirst selection set and then requests the user to select additional objects. Before requesting the user to select objects, the current PickFirst selection set is cleared with the SetImpliedSelection method.

VB.NET

```
Imports Autodesk.AutoCAD.Runtime
Imports Autodesk.AutoCAD.ApplicationServices
Imports Autodesk.AutoCAD.DatabaseServices
Imports Autodesk.AutoCAD.EditorInput
```

```vb
<CommandMethod("CheckForPickfirstSelection", CommandFlags.UsePickSet)> _
Public Sub CheckForPickfirstSelection()
  '' Get the current document
  Dim acDocEd As Editor = Application.DocumentManager.MdiActiveDocument.Editor

  '' Get the PickFirst selection set
  Dim acSSPrompt As PromptSelectionResult
  acSSPrompt = acDocEd.SelectImplied()

  Dim acSSet As SelectionSet

  '' If the prompt status is OK, objects were selected before
  '' the command was started
  If acSSPrompt.Status = PromptStatus.OK Then
      acSSet = acSSPrompt.Value

      Application.ShowAlertDialog("Number of objects in Pickfirst selection: " & _
                                 acSSet.Count.ToString())
  Else
      Application.ShowAlertDialog("Number of objects in Pickfirst selection: 0")
  End If

  '' Clear the PickFirst selection set
  Dim idarrayEmpty() As ObjectId
  acDocEd.SetImpliedSelection(idarrayEmpty)

  '' Request for objects to be selected in the drawing area
  acSSPrompt = acDocEd.GetSelection()

  '' If the prompt status is OK, objects were selected
  If acSSPrompt.Status = PromptStatus.OK Then
      acSSet = acSSPrompt.Value

      Application.ShowAlertDialog("Number of objects selected: " & _
                                 acSSet.Count.ToString())
  Else
      Application.ShowAlertDialog("Number of objects selected: 0")
  End If
End Sub
```

C#

```csharp
using Autodesk.AutoCAD.Runtime;
using Autodesk.AutoCAD.ApplicationServices;
using Autodesk.AutoCAD.DatabaseServices;
using Autodesk.AutoCAD.EditorInput;

[CommandMethod("CheckForPickfirstSelection", CommandFlags.UsePickSet)]
public static void CheckForPickfirstSelection()
{
  // Get the current document
  Editor acDocEd = Application.DocumentManager.MdiActiveDocument.Editor;

  // Get the PickFirst selection set
  PromptSelectionResult acSSPrompt;
  acSSPrompt = acDocEd.SelectImplied();

  SelectionSet acSSet;

  // If the prompt status is OK, objects were selected before
  // the command was started
  if (acSSPrompt.Status == PromptStatus.OK)
  {
      acSSet = acSSPrompt.Value;
```

```
            Application.ShowAlertDialog("Number of objects in Pickfirst selection: " +
                                   acSSet.Count.ToString());
  }
  else
  {
      Application.ShowAlertDialog("Number of objects in Pickfirst selection: 0");
  }

  // Clear the PickFirst selection set
  ObjectId[] idarrayEmpty = new ObjectId[0];
  acDocEd.SetImpliedSelection(idarrayEmpty);

  // Request for objects to be selected in the drawing area
  acSSPrompt = acDocEd.GetSelection();

  // If the prompt status is OK, objects were selected
  if (acSSPrompt.Status == PromptStatus.OK)
  {
      acSSet = acSSPrompt.Value;

      Application.ShowAlertDialog("Number of objects selected: " +
                              acSSet.Count.ToString());
  }
  else
  {
      Application.ShowAlertDialog("Number of objects selected: 0");
  }
}
```

```
Sub CheckForPickfirstSelection()
    ' Get the Pickfirst selection set
    Dim acSSet As AcadSelectionSet
    Set acSSet = ThisDrawing.PickfirstSelectionSet

    ' Display the number of selected objects
    MsgBox "Number of objects in Pickfirst selection set: " & acSSet.Count

    ' Create a new selection set
    Dim acSSetUser As AcadSelectionSet
    Set acSSetUser = ThisDrawing.SelectionSets.Add("User")

    ' Select objects in the drawing
    acSSetUser.SelectOnScreen

    ' Display the number of selected objects
    MsgBox "Number of objects selected: " & acSSetUser.Count
    ' Remove the new named selection set
    acSSetUser.Delete
End Sub
```

Select Objects in the Drawing Area

You can select objects through by having the user interactively select objects, or you can
simulate many of the various object selection options through the .NET API. If your routine

performs multiple selection sets, you will need to either track each selection set returned or create an ObjectIdCollection object to keep track of all the selected objects. The following functions allow you to select objects from the drawing:

GetSelection

Prompts the user to pick objects from the screen.

SelectAll

Selects all objects in the current space in which are not locked or frozen.

SelectCrossingPolygon

Selects objects within and crossing a polygon defined by specifying points. The polygon can be any shape but cannot cross or touch itself.

SelectCrossingWindow

Selects objects within and crossing an area defined by two points.

SelectFence

Selects all objects crossing a selection fence. Fence selection is similar to crossing polygon selection except that the fence is not closed, and a fence can cross itself.

SelectLast

Selects the last object created in the current space.

SelectPrevious

Selects all objects selected during the previous Select objects: prompt.

SelectWindow

Selects all objects completely inside a rectangle defined by two points.

SelectWindowPolygon

Selects objects completely inside a polygon defined by points. The polygon can be any shape but cannot cross or touch itself.

SelectAtPoint

Selects objects passing through a given point and places them into the active selection set.

SelectByPolygon

Selects objects within a fence and adds them to the active selection set.

Prompt for objects on screen and iterate the selection set

This example prompts the user to select objects, then changes the color of each object selected to Green or the AutoCAD Color Index of 3.

VB.NET

```
Imports Autodesk.AutoCAD.Runtime
Imports Autodesk.AutoCAD.ApplicationServices
Imports Autodesk.AutoCAD.DatabaseServices
Imports Autodesk.AutoCAD.EditorInput

<CommandMethod("SelectObjectsOnscreen")> _
Public Sub SelectObjectsOnscreen()
  '' Get the current document and database
  Dim acDoc As Document = Application.DocumentManager.MdiActiveDocument
  Dim acCurDb As Database = acDoc.Database

  '' Start a transaction
  Using acTrans As Transaction = acCurDb.TransactionManager.StartTransaction()

      '' Request for objects to be selected in the drawing area
      Dim acSSPrompt As PromptSelectionResult = acDoc.Editor.GetSelection()

      '' If the prompt status is OK, objects were selected
      If acSSPrompt.Status = PromptStatus.OK Then
          Dim acSSet As SelectionSet = acSSPrompt.Value

          '' Step through the objects in the selection set
          For Each acSSObj As SelectedObject In acSSet
              '' Check to make sure a valid SelectedObject object was returned
              If Not IsDBNull(acSSObj) Then
                  '' Open the selected object for write
                  Dim acEnt As Entity = acTrans.GetObject(acSSObj.ObjectId, _
                                                          OpenMode.ForWrite)

                  If Not IsDBNull(acEnt) Then
                      '' Change the object's color to Green
                      acEnt.ColorIndex = 3
                  End If
              End If
          Next

          '' Save the new object to the database
          acTrans.Commit()
      End If

      '' Dispose of the transaction
  End Using
End Sub
```

C#

```
using Autodesk.AutoCAD.Runtime;
using Autodesk.AutoCAD.ApplicationServices;
using Autodesk.AutoCAD.DatabaseServices;
using Autodesk.AutoCAD.EditorInput;

[CommandMethod("SelectObjectsOnscreen")]
public static void SelectObjectsOnscreen()
{
  // Get the current document and database
  Document acDoc = Application.DocumentManager.MdiActiveDocument;
  Database acCurDb = acDoc.Database;
```

```
   // Start a transaction
   using (Transaction acTrans = acCurDb.TransactionManager.StartTransaction())
   {
       // Request for objects to be selected in the drawing area
       PromptSelectionResult acSSPrompt = acDoc.Editor.GetSelection();

       // If the prompt status is OK, objects were selected
       if (acSSPrompt.Status == PromptStatus.OK)
       {
           SelectionSet acSSet = acSSPrompt.Value;

           // Step through the objects in the selection set
           foreach (SelectedObject acSSObj in acSSet)
           {
               // Check to make sure a valid SelectedObject object was returned
               if (acSSObj != null)
               {
                   // Open the selected object for write
                   Entity acEnt = acTrans.GetObject(acSSObj.ObjectId,
                                                    OpenMode.ForWrite) as Entity;

                   if (acEnt != null)
                   {
                       // Change the object's color to Green
                       acEnt.ColorIndex = 3;
                   }
               }
           }

           // Save the new object to the database
           acTrans.Commit();
       }

       // Dispose of the transaction
   }
}
```

VBA/ActiveX Code Reference

```
Sub SelectObjectsOnscreen()
    ' Create a new selection set
    Dim sset As AcadSelectionSet
    Set sset = ThisDrawing.SelectionSets.Add("SS1")

    ' Prompt the user to select objects
    ' and add them to the selection set.
    sset.SelectOnScreen

    Dim acEnt As AcadEntity

    ' Step through the selected objects and change
    ' each object's color to Green
    For Each acEnt In sset
        ' Use the Color property to set the object's color
        acEnt.color = acGreen
    Next acEnt

    ' Remove the selection set at the end
    sset.Delete
End Sub
```

Select objects with crossing window

This example selects the objects within and that intersect a crossing window.

VB.NET

```
Imports Autodesk.AutoCAD.Runtime
Imports Autodesk.AutoCAD.ApplicationServices
Imports Autodesk.AutoCAD.DatabaseServices
Imports Autodesk.AutoCAD.Geometry
Imports Autodesk.AutoCAD.EditorInput

<CommandMethod("SelectObjectsByCrossingWindow")> _
Public Sub SelectObjectsByCrossingWindow()
  '' Get the current document editor
  Dim acDocEd As Editor = Application.DocumentManager.MdiActiveDocument.Editor

  '' Create a crossing window from (2,2,0) to (10,8,0)
  Dim acSSPrompt As PromptSelectionResult
  acSSPrompt = acDocEd.SelectCrossingWindow(New Point3d(2, 2, 0), _
                                            New Point3d(10, 8, 0))

  '' If the prompt status is OK, objects were selected
  If acSSPrompt.Status = PromptStatus.OK Then
      Dim acSSet As SelectionSet = acSSPrompt.Value

      Application.ShowAlertDialog("Number of objects selected: " & _
                                  acSSet.Count.ToString())
  Else
      Application.ShowAlertDialog("Number of objects selected: 0")
  End If
End Sub
```

C#

```
using Autodesk.AutoCAD.Runtime;
using Autodesk.AutoCAD.ApplicationServices;
using Autodesk.AutoCAD.DatabaseServices;
using Autodesk.AutoCAD.Geometry;
using Autodesk.AutoCAD.EditorInput;

[CommandMethod("SelectObjectsByCrossingWindow")]
public static void SelectObjectsByCrossingWindow()
{
  // Get the current document editor
  Editor acDocEd = Application.DocumentManager.MdiActiveDocument.Editor;

  // Create a crossing window from (2,2,0) to (10,8,0)
  PromptSelectionResult acSSPrompt;
  acSSPrompt = acDocEd.SelectCrossingWindow(new Point3d(2, 2, 0),
                                            new Point3d(10, 8, 0));

  // If the prompt status is OK, objects were selected
  if (acSSPrompt.Status == PromptStatus.OK)
  {
      SelectionSet acSSet = acSSPrompt.Value;

      Application.ShowAlertDialog("Number of objects selected: " +
                                  acSSet.Count.ToString());
  }
  else
  {
      Application.ShowAlertDialog("Number of objects selected: 0");
```

```
        }
}
```

```
Sub SelectObjectsByCrossingWindow()
    ' Create a new selection set
    Dim sset As AcadSelectionSet
    Set sset = ThisDrawing.SelectionSets.Add("SS1")

    ' Define the points for the crossing window
    Dim pt1(0 To 2) As Double
    Dim pt2(0 To 2) As Double

    pt1(0) = 2#: pt1(1) = 2#: pt1(2) = 0#:
    pt2(0) = 10#: pt2(1) = 8#: pt2(2) = 0#:

    ' Create a crossing window from (2,2,0) to (10,8,0)
    sset.Select acSelectionSetCrossing, pt1, pt2

    MsgBox "Number of objects selected: " & sset.Count

    ' Remove the selection set at the end
    sset.Delete
End Sub
```

Add To or Merge Multiple Selection Sets

You can merge multiple selection sets be creating an ObjectIdCollection object and then adding the object ids from multiple selection sets together. In addition to adding object ids to an ObjectIdCollection object, you can remove object ids. Once all object ids are added to an ObjectIdCollection object, you can iterate through the collection of object ids and manipulate each object as needed.

Add selected objects to a selection set

This example prompts the user to select objects twice and then merges the two selection sets created into a single selection set.

VB.NET

```
Imports Autodesk.AutoCAD.Runtime
Imports Autodesk.AutoCAD.ApplicationServices
Imports Autodesk.AutoCAD.DatabaseServices
Imports Autodesk.AutoCAD.EditorInput

<CommandMethod("MergeSelectionSets")> _
Public Sub MergeSelectionSets()
  '' Get the current document editor
  Dim acDocEd As Editor = Application.DocumentManager.MdiActiveDocument.Editor

  '' Request for objects to be selected in the drawing area
  Dim acSSPrompt As PromptSelectionResult
  acSSPrompt = acDocEd.GetSelection()
```

```vb
    Dim acSSet1 As SelectionSet
    Dim acObjIdColl As ObjectIdCollection = New ObjectIdCollection()

    '' If the prompt status is OK, objects were selected
    If acSSPrompt.Status = PromptStatus.OK Then
        '' Get the selected objects
        acSSet1 = acSSPrompt.Value

        '' Append the selected objects to the ObjectIdCollection
        acObjIdColl = New ObjectIdCollection(acSSet1.GetObjectIds())
    End If

    '' Request for objects to be selected in the drawing area
    acSSPrompt = acDocEd.GetSelection()

    Dim acSSet2 As SelectionSet

    '' If the prompt status is OK, objects were selected
    If acSSPrompt.Status = PromptStatus.OK Then
        acSSet2 = acSSPrompt.Value

        '' Check the size of the ObjectIdCollection, if zero, then initialize it
        If acObjIdColl.Count = 0 Then
            acObjIdColl = New ObjectIdCollection(acSSet2.GetObjectIds())
        Else
            Dim acObjId As ObjectId

            '' Step through the second selection set
            For Each acObjId In acSSet2.GetObjectIds()
                '' Add each object id to the ObjectIdCollection
                acObjIdColl.Add(acObjId)
            Next
        End If
    End If

    Application.ShowAlertDialog("Number of objects selected: " & _
                               acObjIdColl.Count.ToString())
End Sub
```

C#

```csharp
using Autodesk.AutoCAD.Runtime;
using Autodesk.AutoCAD.ApplicationServices;
using Autodesk.AutoCAD.DatabaseServices;
using Autodesk.AutoCAD.EditorInput;

[CommandMethod("MergeSelectionSets")]
public static void MergeSelectionSets()
{
  // Get the current document editor
  Editor acDocEd = Application.DocumentManager.MdiActiveDocument.Editor;

  // Request for objects to be selected in the drawing area
  PromptSelectionResult acSSPrompt;
  acSSPrompt = acDocEd.GetSelection();

  SelectionSet acSSet1;
  ObjectIdCollection acObjIdColl = new ObjectIdCollection();

  // If the prompt status is OK, objects were selected
  if (acSSPrompt.Status == PromptStatus.OK)
  {
      // Get the selected objects
```

```
        acSSet1 = acSSPrompt.Value;

        // Append the selected objects to the ObjectIdCollection
        acObjIdColl = new ObjectIdCollection(acSSet1.GetObjectIds());
    }

    // Request for objects to be selected in the drawing area
    acSSPrompt = acDocEd.GetSelection();

    SelectionSet acSSet2;

    // If the prompt status is OK, objects were selected
    if (acSSPrompt.Status == PromptStatus.OK)
    {
        acSSet2 = acSSPrompt.Value;

        // Check the size of the ObjectIdCollection, if zero, then initialize it
        if (acObjIdColl.Count == 0)
        {
            acObjIdColl = new ObjectIdCollection(acSSet2.GetObjectIds());
        }
        else
        {
            // Step through the second selection set
            foreach (ObjectId acObjId in acSSet2.GetObjectIds())
            {
                // Add each object id to the ObjectIdCollection
                acObjIdColl.Add(acObjId);
            }
        }
    }

    Application.ShowAlertDialog("Number of objects selected: " +
                               acObjIdColl.Count.ToString());
}
```

⊟ *VBA/ActiveX Code Reference*

```
Sub MergeSelectionSets()
    ' Create a new selection set
    Dim sset As AcadSelectionSet
    Set sset = ThisDrawing.SelectionSets.Add("SS1")

    ' Prompt the user to select objects
    ' and add them to the selection set.
    sset.SelectOnScreen

    ' Prompt the user again to select objects
    ' and add them to the same selection set.
    sset.SelectOnScreen

    MsgBox "Number of total objects selected: " & sset.Count

    ' Remove the selection set at the end
    sset.Delete
End Sub
```

Define Rules for Selection Filters

You can limit which objects are selected and added to a selection set by using a selection filter. A selection filter list can be used to filter selected objects by properties or type. For example, you might want to select only blue objects or objects on a certain layer. You can also combine selection criteria. For example, you can create a selection filter that limits selection to blue circles on the layer named Pattern. Selection filters can be specified as a parameter for the different selection methods in Select Objects in the Drawing Area.

NoteFiltering recognizes values explicitly assigned to objects, not those inherited by the layer. For example, if an object's linetype property is set to ByLayer and the layer it is assigned is set to the Hidden linetype; filtering for objects assigned the Hidden linetype will not select these objects since their linetype property is set to ByLayer.

Topics in this section

- Use Selection Filters to Define Selection Set Rules
- Specify Multiple Criteria in a Selection Filter
- Add Complexity to Your Filter List Conditions
- Use Wild-Card Patterns in Selection Set Filter Criteria
- Filter for Extended Data

Use Selection Filters to Define Selection Set Rules

Selection filters are composed of pairs of arguments in the form of TypedValues. The first argument of a TypedValue identifies the type of filter (for example, an object), and the second argument specifies the value you are filtering on (for example, circles). The filter type is a DXF group code that specifies which filter to use. A few of the most common filter types are listed here.

DXF codes for common filters

DXF code	Filter type
0 (or DxfCode.Start)	Object Type (String)
	Such as "Line," "Circle," "Arc," and so forth.
2 (or DxfCode.BlockName)	Block Name (String)
	The block name of an insert reference.
8 or (DxfCode.LayerName)	Layer Name (String)
	Such as "Layer 0."
60 (DxfCode.Visibility)	Object Visibility (Integer)
	Use 0 = visible, 1 = invisible.

	Color Number (Integer)
62 (or DxfCode.Color)	Numeric index values ranging from 0 to 256.
	Zero indicates BYBLOCK. 256 indicates BYLAYER. A negative value indicates that the layer is turned off.
67	Model/paper space indicator (Integer)
	Use 0 or omitted = model space, 1 = paper space.

For a complete list of DXF group codes, see Group Code Value Types in the *DXF Reference*.

Specify a single selection criterion for a selection set

The following code prompts users to select objects to be included in a selection set, and filters out all objects except for circles.

VB.NET

```
Imports Autodesk.AutoCAD.Runtime
Imports Autodesk.AutoCAD.ApplicationServices
Imports Autodesk.AutoCAD.DatabaseServices
Imports Autodesk.AutoCAD.EditorInput

<CommandMethod("FilterSelectionSet")> _
Public Sub FilterSelectionSet()
  '' Get the current document editor
  Dim acDocEd As Editor = Application.DocumentManager.MdiActiveDocument.Editor

  '' Create a TypedValue array to define the filter criteria
  Dim acTypValAr(0) As TypedValue
  acTypValAr.SetValue(New TypedValue(DxfCode.Start, "CIRCLE"), 0)

  '' Assign the filter criteria to a SelectionFilter object
  Dim acSelFtr As SelectionFilter = New SelectionFilter(acTypValAr)

  '' Request for objects to be selected in the drawing area
  Dim acSSPrompt As PromptSelectionResult
  acSSPrompt = acDocEd.GetSelection(acSelFtr)

  '' If the prompt status is OK, objects were selected
  If acSSPrompt.Status = PromptStatus.OK Then
      Dim acSSet As SelectionSet = acSSPrompt.Value

      Application.ShowAlertDialog("Number of objects selected: " & _
                                  acSSet.Count.ToString())
  Else
      Application.ShowAlertDialog("Number of objects selected: 0")
  End If
End Sub
```

C#

```
using Autodesk.AutoCAD.Runtime;
using Autodesk.AutoCAD.ApplicationServices;
using Autodesk.AutoCAD.DatabaseServices;
using Autodesk.AutoCAD.EditorInput;

[CommandMethod("FilterSelectionSet")]
public static void FilterSelectionSet()
```

```
{
  // Get the current document editor
  Editor acDocEd = Application.DocumentManager.MdiActiveDocument.Editor;

  // Create a TypedValue array to define the filter criteria
  TypedValue[] acTypValAr = new TypedValue[1];
  acTypValAr.SetValue(new TypedValue((int)DxfCode.Start, "CIRCLE"), 0);

  // Assign the filter criteria to a SelectionFilter object
  SelectionFilter acSelFtr = new SelectionFilter(acTypValAr);

  // Request for objects to be selected in the drawing area
  PromptSelectionResult acSSPrompt;
  acSSPrompt = acDocEd.GetSelection(acSelFtr);

  // If the prompt status is OK, objects were selected
  if (acSSPrompt.Status == PromptStatus.OK)
  {
      SelectionSet acSSet = acSSPrompt.Value;

      Application.ShowAlertDialog("Number of objects selected: " +
                                 acSSet.Count.ToString());
  }
  else
  {
      Application.ShowAlertDialog("Number of objects selected: 0");
  }
}
```

⊟ *VBA/ActiveX Code Reference*

```
Sub FilterSelectionSet()
    ' Create a new selection set
    Dim sset As AcadSelectionSet
    Set sset = ThisDrawing.SelectionSets.Add("SS1")

    ' Define the filter list, only Circle objects
    ' will be selectable
    Dim FilterType(0) As Integer
    Dim FilterData(0) As Variant
    FilterType(0) = 0
    FilterData(0) = "Circle"

    ' Prompt the user to select objects
    ' and add them to the selection set
    sset.SelectOnScreen FilterType, FilterData

    MsgBox "Number of objects selected: " & sset.Count

    ' Remove the selection set at the end
    sset.Delete
End Sub
```

Specify Multiple Criteria in a Selection Filter

A selection filter can contain filtering criteria for more than just one property or object. You define the total number of conditions to filter on by declaring an array containing enough elements to represent each of the filter criterion.

Select objects that meet two criterion

The following example specifies two criterion to filter selected objects by: the object must be a circle and it must reside on layer 0.

VB.NET

```
Imports Autodesk.AutoCAD.Runtime
Imports Autodesk.AutoCAD.ApplicationServices
Imports Autodesk.AutoCAD.DatabaseServices
Imports Autodesk.AutoCAD.EditorInput

<CommandMethod("FilterBlueCircleOnLayer0")> _
Public Sub FilterBlueCircleOnLayer0()
  '' Get the current document editor
  Dim acDocEd As Editor = Application.DocumentManager.MdiActiveDocument.Editor

  '' Create a TypedValue array to define the filter criteria
  Dim acTypValAr(2) As TypedValue
  acTypValAr.SetValue(New TypedValue(DxfCode.Color, 5), 0)
  acTypValAr.SetValue(New TypedValue(DxfCode.Start, "CIRCLE"), 1)
  acTypValAr.SetValue(New TypedValue(DxfCode.LayerName, "0"), 2)

  '' Assign the filter criteria to a SelectionFilter object
  Dim acSelFtr As SelectionFilter = New SelectionFilter(acTypValAr)

  '' Request for objects to be selected in the drawing area
  Dim acSSPrompt As PromptSelectionResult
  acSSPrompt = acDocEd.GetSelection(acSelFtr)

  '' If the prompt status is OK, objects were selected
  If acSSPrompt.Status = PromptStatus.OK Then
      Dim acSSet As SelectionSet = acSSPrompt.Value

      Application.ShowAlertDialog("Number of objects selected: " & _
                                  acSSet.Count.ToString())
  Else
      Application.ShowAlertDialog("Number of objects selected: 0")
  End If
End Sub
```

C#

```
using Autodesk.AutoCAD.Runtime;
using Autodesk.AutoCAD.ApplicationServices;
using Autodesk.AutoCAD.DatabaseServices;
using Autodesk.AutoCAD.EditorInput;

[CommandMethod("FilterBlueCircleOnLayer0")]
public static void FilterBlueCircleOnLayer0()
{
  // Get the current document editor
  Editor acDocEd = Application.DocumentManager.MdiActiveDocument.Editor;

  // Create a TypedValue array to define the filter criteria
  TypedValue[] acTypValAr = new TypedValue[3];
  acTypValAr.SetValue(new TypedValue((int)DxfCode.Color, 5), 0);
  acTypValAr.SetValue(new TypedValue((int)DxfCode.Start, "CIRCLE"), 1);
```

```
      acTypValAr.SetValue(new TypedValue((int)DxfCode.LayerName, "0"), 2);

  // Assign the filter criteria to a SelectionFilter object
  SelectionFilter acSelFtr = new SelectionFilter(acTypValAr);

  // Request for objects to be selected in the drawing area
  PromptSelectionResult acSSPrompt;
  acSSPrompt = acDocEd.GetSelection(acSelFtr);

  // If the prompt status is OK, objects were selected
  if (acSSPrompt.Status == PromptStatus.OK)
  {
      SelectionSet acSSet = acSSPrompt.Value;

      Application.ShowAlertDialog("Number of objects selected: " +
                                 acSSet.Count.ToString());
  }
  else
  {
      Application.ShowAlertDialog("Number of objects selected: 0");
  }
}
```

⊟ *VBA/ActiveX Code Reference*

```
Sub FilterBlueCircleOnLayer0()
    Dim sset As AcadSelectionSet
    Set sset = ThisDrawing.SelectionSets.Add("SS1")

    ' Define the filter list, only blue Circle objects
    ' on layer 0
    Dim FilterType(2) As Integer
    Dim FilterData(2) As Variant

    FilterType(0) = 62: FilterData(0) = 3
    FilterType(1) = 0: FilterData(1) = "Circle"
    FilterType(2) = 8: FilterData(2) = "0"

    ' Prompt the user to select objects
    ' and add them to the selection set
    sset.SelectOnScreen FilterType, FilterData

    MsgBox "Number of objects selected: " & sset.Count

    ' Remove the selection set at the end
    sset.Delete
End Sub
```

Add Complexity to Your Filter List Conditions

When you specify multiple selection criteria, AutoCAD assumes the selected object must meet each criterion. You can qualify your criteria in other ways. For numeric items, you can specify relational operations (for example, the radius of a circle must be *greater than or equal* to 5.0). And for all items, you can specify logical operations (for example, Text *or* MText).

Use a -4 DXF code or the constant DxfCode.Operator to indicate a relational operator in a selection filter. The operator is expressed as a string. The allowable relational operators are shown in the following table.

Relational operators for selection set filter lists

Operator	Description
"*"	Anything goes (always true)
"="	Equals
"!="	Not equal to
"/="	Not equal to
"<>"	Not equal to
"<"	Less than
"<="	Less than or equal to
">"	Greater than
">="	Greater than or equal to
"&"	Bitwise AND (integer groups only)
"&="	Bitwise masked equals (integer groups only)

Logical operators in a selection filter are also indicated by a -4 group code or the constant DxfCode.Operator, and the operator is a string, but the operators must be paired. The opening operator is preceded by a less-than symbol (<), and the closing operator is followed by a greater-than symbol (>). The following table lists the logical operators allowed in selection set filtering.

Logical grouping operators for selection set filter lists

Starting operator	Encloses	Ending operator
"<AND"	One or more operands	"AND>"
"<OR"	One or more operands	"OR>"
"<XOR"	Two operands	"XOR>"
"<NOT"	One operand	"NOT>"

Select a circle whose radius is greater than or equal to 5.0

The following example selects circles whose radius is greater than or equal to 5.0.

VB.NET

```
Imports Autodesk.AutoCAD.Runtime
Imports Autodesk.AutoCAD.ApplicationServices
Imports Autodesk.AutoCAD.DatabaseServices
Imports Autodesk.AutoCAD.EditorInput

<CommandMethod("FilterRelational")> _
Public Sub FilterRelational()
  '' Get the current document editor
  Dim acDocEd As Editor = Application.DocumentManager.MdiActiveDocument.Editor

  '' Create a TypedValue array to define the filter criteria
  Dim acTypValAr(2) As TypedValue
  acTypValAr.SetValue(New TypedValue(DxfCode.Start, "CIRCLE"), 0)
  acTypValAr.SetValue(New TypedValue(DxfCode.Operator, ">="), 1)
  acTypValAr.SetValue(New TypedValue(40, 5), 2)

  '' Assign the filter criteria to a SelectionFilter object
```

```
   Dim acSelFtr As SelectionFilter = New SelectionFilter(acTypValAr)

   '' Request for objects to be selected in the drawing area
   Dim acSSPrompt As PromptSelectionResult
   acSSPrompt = acDocEd.GetSelection(acSelFtr)

   '' If the prompt status is OK, objects were selected
   If acSSPrompt.Status = PromptStatus.OK Then
       Dim acSSet As SelectionSet = acSSPrompt.Value

       Application.ShowAlertDialog("Number of objects selected: " & _
                                   acSSet.Count.ToString())
   Else
       Application.ShowAlertDialog("Number of objects selected: 0")
   End If
End Sub
```

C#

```
using Autodesk.AutoCAD.Runtime;
using Autodesk.AutoCAD.ApplicationServices;
using Autodesk.AutoCAD.DatabaseServices;
using Autodesk.AutoCAD.EditorInput;

[CommandMethod("FilterRelational")]
public static void FilterRelational()
{
  // Get the current document editor
  Editor acDocEd = Application.DocumentManager.MdiActiveDocument.Editor;

  // Create a TypedValue array to define the filter criteria
  TypedValue[] acTypValAr = new TypedValue[3];
  acTypValAr.SetValue(new TypedValue((int)DxfCode.Start, "CIRCLE"), 0);
  acTypValAr.SetValue(new TypedValue((int)DxfCode.Operator, ">="), 1);
  acTypValAr.SetValue(new TypedValue(40, 5), 2);

  // Assign the filter criteria to a SelectionFilter object
  SelectionFilter acSelFtr = new SelectionFilter(acTypValAr);

  // Request for objects to be selected in the drawing area
  PromptSelectionResult acSSPrompt;
  acSSPrompt = acDocEd.GetSelection(acSelFtr);

  // If the prompt status is OK, objects were selected
  if (acSSPrompt.Status == PromptStatus.OK)
  {
      SelectionSet acSSet = acSSPrompt.Value;

      Application.ShowAlertDialog("Number of objects selected: " +
                                  acSSet.Count.ToString());
  }
  else
  {
      Application.ShowAlertDialog("Number of objects selected: 0");
  }
}
```

⊟ *VBA/ActiveX Code Reference*

```
Sub FilterRelational()
    Dim sset As AcadSelectionSet
    Dim FilterType(2) As Integer
```

```
    Dim FilterData(2) As Variant
    Set sset = ThisDrawing.SelectionSets.Add("SS1")
    FilterType(0) = 0: FilterData(0) = "Circle"
    FilterType(1) = -4: FilterData(1) = ">="
    FilterType(2) = 40: FilterData(2) = 5#

    sset.SelectOnScreen FilterType, FilterData

    MsgBox "Number of objects selected: " & sset.Count

    ' Remove the selection set at the end
    sset.Delete
End Sub
```

Select either Text or MText

The following example specifies that either Text or MText objects can be selected.

VB.NET

```
Imports Autodesk.AutoCAD.Runtime
Imports Autodesk.AutoCAD.ApplicationServices
Imports Autodesk.AutoCAD.DatabaseServices
Imports Autodesk.AutoCAD.EditorInput

<CommandMethod("FilterForText")> _
Public Sub FilterForText()
  '' Get the current document editor
  Dim acDocEd As Editor = Application.DocumentManager.MdiActiveDocument.Editor

  '' Create a TypedValue array to define the filter criteria
  Dim acTypValAr(3) As TypedValue
  acTypValAr.SetValue(New TypedValue(DxfCode.Operator, "<or"), 0)
  acTypValAr.SetValue(New TypedValue(DxfCode.Start, "TEXT"), 1)
  acTypValAr.SetValue(New TypedValue(DxfCode.Start, "MTEXT"), 2)
  acTypValAr.SetValue(New TypedValue(DxfCode.Operator, "or>"), 3)

  '' Assign the filter criteria to a SelectionFilter object
  Dim acSelFtr As SelectionFilter = New SelectionFilter(acTypValAr)

  '' Request for objects to be selected in the drawing area
  Dim acSSPrompt As PromptSelectionResult
  acSSPrompt = acDocEd.GetSelection(acSelFtr)

  '' If the prompt status is OK, objects were selected
  If acSSPrompt.Status = PromptStatus.OK Then
      Dim acSSet As SelectionSet = acSSPrompt.Value

      Application.ShowAlertDialog("Number of objects selected: " & _
                                  acSSet.Count.ToString())
  Else
      Application.ShowAlertDialog("Number of objects selected: 0")
  End If
End Sub
```

C#

```
using Autodesk.AutoCAD.Runtime;
using Autodesk.AutoCAD.ApplicationServices;
using Autodesk.AutoCAD.DatabaseServices;
using Autodesk.AutoCAD.EditorInput;

[CommandMethod("FilterForText")]
```

```
public static void FilterForText()
{
  // Get the current document editor
  Editor acDocEd = Application.DocumentManager.MdiActiveDocument.Editor;

  // Create a TypedValue array to define the filter criteria
  TypedValue[] acTypValAr = new TypedValue[4];
  acTypValAr.SetValue(new TypedValue((int)DxfCode.Operator, "<or"), 0);
  acTypValAr.SetValue(new TypedValue((int)DxfCode.Start, "TEXT"), 1);
  acTypValAr.SetValue(new TypedValue((int)DxfCode.Start, "MTEXT"), 2);
  acTypValAr.SetValue(new TypedValue((int)DxfCode.Operator, "or>"), 3);

  // Assign the filter criteria to a SelectionFilter object
  SelectionFilter acSelFtr = new SelectionFilter(acTypValAr);

  // Request for objects to be selected in the drawing area
  PromptSelectionResult acSSPrompt;
  acSSPrompt = acDocEd.GetSelection(acSelFtr);

  // If the prompt status is OK, objects were selected
  if (acSSPrompt.Status == PromptStatus.OK)
  {
      SelectionSet acSSet = acSSPrompt.Value;

      Application.ShowAlertDialog("Number of objects selected: " +
                                  acSSet.Count.ToString());
  }
  else
  {
      Application.ShowAlertDialog("Number of objects selected: 0");
  }
}
```

☐ *VBA/ActiveX Code Reference*

```
Sub FilterForText()
    Dim sset As AcadSelectionSet
    Dim FilterType(3) As Integer
    Dim FilterData(3) As Variant
    Set sset = ThisDrawing.SelectionSets.Add("SS1")

    FilterType(0) = -4: FilterData(0) = "<or"
    FilterType(1) = 0: FilterData(1) = "TEXT"
    FilterType(2) = 0: FilterData(2) = "MTEXT"
    FilterType(3) = -4: FilterData(3) = "or>"

    sset.SelectOnScreen FilterType, FilterData

    MsgBox "Number of objects selected: " & sset.Count

    ' Remove the selection set at the end
    sset.Delete
End Sub
```

Use Wild-Card Patterns in Selection Set Filter Criteria

Symbol names and strings in selection filters can include wild-card patterns.

The following table identifies the wild-card characters recognized by AutoCAD, and what each means in the context of a string:

Wild-card characters

Character	Definition
#(pound)	Matches any single numeric digit
@(at)	Matches any single alphabetic character
.(period)	Matches any single non-alphanumeric character
*(asterisk)	Matches any character sequence, including an empty one, and it can be used anywhere in the search pattern: at the beginning, middle, or end
?(question mark)	Matches any single character
~(tilde)	If it is the first character in the pattern, it matches anything except the pattern
[...]	Matches any one of the characters enclosed
[~...]	Matches any single character not enclosed
-(hyphen)	Used inside brackets to specify a range for a single character
,(comma)	Separates two patterns
`(reverse quote)	Escapes special characters (reads next character literally)

Use a reverse quote (`) to indicate that a character is not a wildcard, but is to be taken literally. For example, to specify that only an anonymous block named "*U2" be included in the selection set, use the value"`*U2".

Select MText where a specific word appears in the text

The following example defines a selection filter that selects MText objects that contain the text string of "The".

VB.NET

```
Imports Autodesk.AutoCAD.Runtime
Imports Autodesk.AutoCAD.ApplicationServices
Imports Autodesk.AutoCAD.DatabaseServices
Imports Autodesk.AutoCAD.EditorInput

<CommandMethod("FilterMtextWildcard")> _
Public Sub FilterMtextWildcard()
  '' Get the current document editor
  Dim acDocEd As Editor = Application.DocumentManager.MdiActiveDocument.Editor

  '' Create a TypedValue array to define the filter criteria
  Dim acTypValAr(1) As TypedValue
  acTypValAr.SetValue(New TypedValue(DxfCode.Start, "MTEXT"), 0)
  acTypValAr.SetValue(New TypedValue(DxfCode.Text, "*The*"), 1)

  '' Assign the filter criteria to a SelectionFilter object
  Dim acSelFtr As SelectionFilter = New SelectionFilter(acTypValAr)

  '' Request for objects to be selected in the drawing area
  Dim acSSPrompt As PromptSelectionResult
  acSSPrompt = acDocEd.GetSelection(acSelFtr)

  '' If the prompt status is OK, objects were selected
  If acSSPrompt.Status = PromptStatus.OK Then
      Dim acSSet As SelectionSet = acSSPrompt.Value
```

```
        Application.ShowAlertDialog("Number of objects selected: " & _
                                acSSet.Count.ToString())
    Else
        Application.ShowAlertDialog("Number of objects selected: 0")
    End If
End Sub
```

C#

```csharp
using Autodesk.AutoCAD.Runtime;
using Autodesk.AutoCAD.ApplicationServices;
using Autodesk.AutoCAD.DatabaseServices;
using Autodesk.AutoCAD.EditorInput;

[CommandMethod("FilterMtextWildcard")]
public static void FilterMtextWildcard()
{
  // Get the current document editor
  Editor acDocEd = Application.DocumentManager.MdiActiveDocument.Editor;

  // Create a TypedValue array to define the filter criteria
  TypedValue[] acTypValAr = new TypedValue[2];
  acTypValAr.SetValue(new TypedValue((int)DxfCode.Start, "MTEXT"), 0);
  acTypValAr.SetValue(new TypedValue((int)DxfCode.Text, "*The*"), 1);

  // Assign the filter criteria to a SelectionFilter object
  SelectionFilter acSelFtr = new SelectionFilter(acTypValAr);

  // Request for objects to be selected in the drawing area
  PromptSelectionResult acSSPrompt;
  acSSPrompt = acDocEd.GetSelection(acSelFtr);

  // If the prompt status is OK, objects were selected
  if (acSSPrompt.Status == PromptStatus.OK)
  {
      SelectionSet acSSet = acSSPrompt.Value;

      Application.ShowAlertDialog("Number of objects selected: " +
                              acSSet.Count.ToString());
  }
  else
  {
      Application.ShowAlertDialog("Number of objects selected: 0");
  }
}
```

⊟ VBA/ActiveX Code Reference

```vba
Sub FilterMtextWildcard()
    Dim sset As AcadSelectionSet
    Dim FilterType(1) As Integer
    Dim FilterData(1) As Variant
    Set sset = ThisDrawing.SelectionSets.Add("SS1")

    FilterType(0) = 0
    FilterData(0) = "MTEXT"
    FilterType(1) = 1
    FilterData(1) = "*The*"

    sset.SelectOnScreen FilterType, FilterData
```

```
    MsgBox "Number of objects selected: " & sset.Count

    ' Remove the selection set at the end
    sset.Delete
End Sub
```

Filter for Extended Data

External applications can attach data such as text strings, numeric values, 3D points, distances, and layer names to AutoCAD objects. This data is referred to as extended data, or xdata. You can filter entities containing extended data for a specified application.

Select circles that contain xdata

The following example filters for circles containing xdata added by the "MY_APP" application:

VB.NET

```
Imports Autodesk.AutoCAD.Runtime
Imports Autodesk.AutoCAD.ApplicationServices
Imports Autodesk.AutoCAD.DatabaseServices
Imports Autodesk.AutoCAD.EditorInput

<CommandMethod("FilterXdata")> _
Public Sub FilterXdata()
  '' Get the current document editor
  Dim acDocEd As Editor = Application.DocumentManager.MdiActiveDocument.Editor

  '' Create a TypedValue array to define the filter criteria
  Dim acTypValAr(1) As TypedValue
  acTypValAr.SetValue(New TypedValue(DxfCode.Start, "Circle"), 0)
  acTypValAr.SetValue(New TypedValue(DxfCode.ExtendedDataRegAppName, _
                                     "MY_APP"), 1)

  '' Assign the filter criteria to a SelectionFilter object
  Dim acSelFtr As SelectionFilter = New SelectionFilter(acTypValAr)

  '' Request for objects to be selected in the drawing area
  Dim acSSPrompt As PromptSelectionResult
  acSSPrompt = acDocEd.GetSelection(acSelFtr)

  '' If the prompt status is OK, objects were selected
  If acSSPrompt.Status = PromptStatus.OK Then
      Dim acSSet As SelectionSet = acSSPrompt.Value

      Application.ShowAlertDialog("Number of objects selected: " & _
                                  acSSet.Count.ToString())
  Else
      Application.ShowAlertDialog("Number of objects selected: 0")
  End If
End Sub
```

C#

```
using Autodesk.AutoCAD.Runtime;
using Autodesk.AutoCAD.ApplicationServices;
using Autodesk.AutoCAD.DatabaseServices;
using Autodesk.AutoCAD.EditorInput;
```

```
[CommandMethod("FilterXdata")]
public static void FilterXdata()
{
  // Get the current document editor
  Editor acDocEd = Application.DocumentManager.MdiActiveDocument.Editor;

  // Create a TypedValue array to define the filter criteria
  TypedValue[] acTypValAr = new TypedValue[2];
  acTypValAr.SetValue(new TypedValue((int)DxfCode.Start, "Circle"), 0);
  acTypValAr.SetValue(new TypedValue((int)DxfCode.ExtendedDataRegAppName,
                                     "MY_APP"), 1);

  // Assign the filter criteria to a SelectionFilter object
  SelectionFilter acSelFtr = new SelectionFilter(acTypValAr);

  // Request for objects to be selected in the drawing area
  PromptSelectionResult acSSPrompt;
  acSSPrompt = acDocEd.GetSelection(acSelFtr);

  // If the prompt status is OK, objects were selected
  if (acSSPrompt.Status == PromptStatus.OK)
  {
      SelectionSet acSSet = acSSPrompt.Value;

      Application.ShowAlertDialog("Number of objects selected: " +
                                  acSSet.Count.ToString());
  }
  else
  {
      Application.ShowAlertDialog("Number of objects selected: 0");
  }
}
```

⊟ *VBA/ActiveX Code Reference*

```
Sub FilterXdata()
    Dim sset As AcadSelectionSet
    Dim FilterType(1) As Integer
    Dim FilterData(1) As Variant
    Set sset = ThisDrawing.SelectionSets.Add("SS1")

    FilterType(0) = 0: FilterData(0) = "Circle"
    FilterType(1) = 1001: FilterData(1) = "MY_APP"

    sset.SelectOnScreen FilterType, FilterData

    MsgBox "Number of objects selected: " & sset.Count

    ' Remove the selection set at the end
    sset.Delete
End Sub
```

Remove Objects From a Selection Set

After you create a selection set, you can work with the object ids of the objects selected. Selection sets do not allow you to add or remove object ids from it, but you can use an ObjectIdCollection object to merge multiple selection sets into a single object to work with.

You can add and remove object ids from an ObjectIdCollection object. Use the Remove or RemoveAt methods to remove an object id from an ObjectIdCollection object. For information on merging multiple selection sets and working with an ObjectIdCollection object, see [Add To or Merge Multiple Selection Sets](#).

Edit Named and 2D Objects

Existing objects can be modified with the methods and properties associated with each object. If you modify a visible property of a graphic object, use the Regen method to redraw the object on screen. The Regen method is a member of the Editor object.

Topics in this section

- [Work with Named Objects](#)
- [Erase Objects](#)
- [Copy Objects](#)
- [Offset Objects](#)
- [Transform Objects](#)
- [Array Objects](#)
- [Extend and Trim Objects](#)
- [Explode Objects](#)
- [Edit Polylines](#)
- [Edit Splines](#)
- [Edit Hatches](#)

Work with Named Objects

In addition to the graphical objects used by AutoCAD, there are several types of nongraphical objects stored in a drawing database. These objects have descriptive designations associated with them. For example, blocks, layers, groups, and dimension styles all have a name assigned to them and in most cases can be renamed. The names of symbol table records are displayed in the user interface of AutoCAD, while the object id of an object is used to reference the object in most cases throughout the .NET API.

For example, the object id of a LayerTableRecord object is assigned to a graphical object's Layer property and not the actual name that is assigned to the LayerTableRecord. However, the object id of a LayerTableRecord object can be obtained from the Layer table using the name of the layer you want to access.

Topics in this section

- [Purge Unreferenced Named Objects](#)
- [Rename Objects](#)

Purge Unreferenced Named Objects

Unreferenced named objects can be purged from a database at any time. You cannot purge named objects that are referenced by other objects. For example, a font file might be

referenced by a text style or a layer might be referenced by the objects on that layer. Purging reduces the size of a drawing file when saved to disk.

Unreferenced objects are purged from a drawing database with the Purge method. The Purge method requires a list of objects you want to purge in the form of an ObjectIdCollection or ObjectIdGraph objects. The ObjectIdCollection or ObjectIdGraph objects passed into the Purge method are updated with the objects in which can be erased from the database. After the call to Purge, you must erase each individual object returned.

⊟ VBA/ActiveX Cross Reference***Purge all unreferenced layers***

The following example demonstrates how to purge all unreferenced layers from a database.

VB.NET

```
Imports Autodesk.AutoCAD.Runtime
Imports Autodesk.AutoCAD.ApplicationServices
Imports Autodesk.AutoCAD.DatabaseServices

<CommandMethod("PurgeUnreferencedLayers")> _
Public Sub PurgeUnreferencedLayers()
  '' Get the current document and database
  Dim acDoc As Document = Application.DocumentManager.MdiActiveDocument
  Dim acCurDb As Database = acDoc.Database

  '' Start a transaction
  Using acTrans As Transaction = acCurDb.TransactionManager.StartTransaction()

      '' Open the Layer table for read
      Dim acLyrTbl As LayerTable
      acLyrTbl = acTrans.GetObject(acCurDb.LayerTableId, _
                                   OpenMode.ForRead)

      '' Create an ObjectIdCollection to hold the object ids for each table record
      Dim acObjIdColl As ObjectIdCollection = New ObjectIdCollection()

      '' Step through each layer and add it to the ObjectIdCollection
      For Each acObjId As ObjectId In acLyrTbl
          acObjIdColl.Add(acObjId)
      Next

      '' Remove the layers that are in use and return the ones that can be erased
      acCurDb.Purge(acObjIdColl)

      '' Step through the returned ObjectIdCollection
      For Each acObjId As ObjectId In acObjIdColl
          Dim acSymTblRec As SymbolTableRecord
          acSymTblRec = acTrans.GetObject(acObjId, _
                                          OpenMode.ForWrite)

          Try
              '' Erase the unreferenced layer
              acSymTblRec.Erase(True)
          Catch Ex As Autodesk.AutoCAD.Runtime.Exception
              '' Layer could not be deleted
              Application.ShowAlertDialog("Error:" & vbLf & Ex.Message)
          End Try
      Next

      '' Commit the changes and dispose of the transaction
      acTrans.Commit()
  End Using
```

```
    End Sub
```

C#

```csharp
using Autodesk.AutoCAD.Runtime;
using Autodesk.AutoCAD.ApplicationServices;
using Autodesk.AutoCAD.DatabaseServices;

[CommandMethod("PurgeUnreferencedLayers")]
public static void PurgeUnreferencedLayers()
{
  // Get the current document and database
  Document acDoc = Application.DocumentManager.MdiActiveDocument;
  Database acCurDb = acDoc.Database;

  // Start a transaction
  using (Transaction acTrans = acCurDb.TransactionManager.StartTransaction())
  {
      // Open the Layer table for read
      LayerTable acLyrTbl;
      acLyrTbl = acTrans.GetObject(acCurDb.LayerTableId,
                                   OpenMode.ForRead) as LayerTable;

      // Create an ObjectIdCollection to hold the object ids for each table record
      ObjectIdCollection acObjIdColl = new ObjectIdCollection();

      // Step through each layer and add iterator to the ObjectIdCollection
      foreach (ObjectId acObjId in acLyrTbl)
      {
          acObjIdColl.Add(acObjId);
      }

      // Remove the layers that are in use and return the ones that can be erased
      acCurDb.Purge(acObjIdColl);

      // Step through the returned ObjectIdCollection
      // and erase each unreferenced layer
      foreach (ObjectId acObjId in acObjIdColl)
      {
          SymbolTableRecord acSymTblRec;
          acSymTblRec = acTrans.GetObject(acObjId,
                                          OpenMode.ForWrite) as SymbolTableRecord;

          try
          {
              // Erase the unreferenced layer
              acSymTblRec.Erase(true);
          }
          catch (Autodesk.AutoCAD.Runtime.Exception Ex)
          {
              // Layer could not be deleted
              Application.ShowAlertDialog("Error:\n" + Ex.Message);
          }
      }

      // Commit the changes and dispose of the transaction
      acTrans.Commit();
  }
}
```

Rename Objects

As your drawings become more complex, you can rename objects to keep the names meaningful or to avoid conflicts with names in other drawings you have inserted or attached. The Name property is used to get the current name or change the name of a named object.

You can rename any named object except those reserved by AutoCAD, for example, layer 0 or the CONTINUOUS linetype.

Names can be up to 255 characters long. In addition to letters and numbers, names can contain spaces (although AutoCAD removes spaces that appear directly before and after a name) and any special character not used by Microsoft® Windows® or AutoCAD for other purposes. Special characters that you cannot use include less-than and greater-than symbols (< >), forward slashes and backslashes (/ \), quotation marks ("), colons (:), semicolons (;), question marks (?), commas (,), asterisks (*), vertical bars (|), equal signs (=), and single quotes ('). You also cannot use special characters created with Unicode fonts.

Rename a layer

This example creates a copy of layer "0" and renames the new layer to "MyLayer".

VB.NET

```
Imports Autodesk.AutoCAD.Runtime
Imports Autodesk.AutoCAD.ApplicationServices
Imports Autodesk.AutoCAD.DatabaseServices

<CommandMethod("RenameLayer")> _
Public Sub RenameLayer()
  '' Get the current document and database
  Dim acDoc As Document = Application.DocumentManager.MdiActiveDocument
  Dim acCurDb As Database = acDoc.Database

  '' Start a transaction
  Using acTrans As Transaction = acCurDb.TransactionManager.StartTransaction()

      '' Returns the layer table for the current database
      Dim acLyrTbl As LayerTable
      acLyrTbl = acTrans.GetObject(acCurDb.LayerTableId, _
                                  OpenMode.ForWrite)

      '' Clone layer 0 (copy it and its properties) as a new layer
      Dim acLyrTblRec As LayerTableRecord
      acLyrTblRec = acTrans.GetObject(acLyrTbl("0"), _
                                      OpenMode.ForRead).Clone()

      '' Change the name of the cloned layer
      acLyrTblRec.Name = "MyLayer"

      '' Add the cloned layer to the Layer table and transaction
      acLyrTbl.Add(acLyrTblRec)
      acTrans.AddNewlyCreatedDBObject(acLyrTblRec, True)

      '' Save changes and dispose of the transaction
      acTrans.Commit()
  End Using
End Sub
```

```csharp
using Autodesk.AutoCAD.Runtime;
using Autodesk.AutoCAD.ApplicationServices;
using Autodesk.AutoCAD.DatabaseServices;

[CommandMethod("RenameLayer")]
public static void RenameLayer()
{
  // Get the current document and database
  Document acDoc = Application.DocumentManager.MdiActiveDocument;
  Database acCurDb = acDoc.Database;

  // Start a transaction
  using (Transaction acTrans = acCurDb.TransactionManager.StartTransaction())
  {
      // Returns the layer table for the current database
      LayerTable acLyrTbl;
      acLyrTbl = acTrans.GetObject(acCurDb.LayerTableId,
                                   OpenMode.ForWrite) as LayerTable;

      // Clone layer 0 (copy it and its properties) as a new layer
      LayerTableRecord acLyrTblRec;
      acLyrTblRec = acTrans.GetObject(acLyrTbl["0"],
                                      OpenMode.ForRead).Clone() as
LayerTableRecord;

      // Change the name of the cloned layer
      acLyrTblRec.Name = "MyLayer";

      // Add the cloned layer to the Layer table and transaction
      acLyrTbl.Add(acLyrTblRec);
      acTrans.AddNewlyCreatedDBObject(acLyrTblRec, true);

      // Save changes and dispose of the transaction
      acTrans.Commit();
  }
}
```

Erase Objects

You can delete non-graphical and graphical objects with the Erase method.

WarningWhile many non-graphical objects, such as the Layer table and Model space block table records have an Erase method, it should not be called. If Erase is called on one of these objects, an error will occur.

Create and erase a polyline

This example creates a lightweight polyline, then erases it.

VB.NET

```vbnet
Imports Autodesk.AutoCAD.Runtime
Imports Autodesk.AutoCAD.ApplicationServices
Imports Autodesk.AutoCAD.DatabaseServices
```

```vb
Imports Autodesk.AutoCAD.Geometry

<CommandMethod("EraseObject")> _
Public Sub EraseObject()
  '' Get the current document and database
  Dim acDoc As Document = Application.DocumentManager.MdiActiveDocument
  Dim acCurDb As Database = acDoc.Database

  '' Start a transaction
  Using acTrans As Transaction = acCurDb.TransactionManager.StartTransaction()

      '' Open the Block table for read
      Dim acBlkTbl As BlockTable
      acBlkTbl = acTrans.GetObject(acCurDb.BlockTableId, OpenMode.ForRead)

      '' Open the Block table record Model space for write
      Dim acBlkTblRec As BlockTableRecord
      acBlkTblRec = acTrans.GetObject(acBlkTbl(BlockTableRecord.ModelSpace), _
                                      OpenMode.ForWrite)

      '' Create a lightweight polyline
      Dim acPoly As Polyline = New Polyline()
      acPoly.SetDatabaseDefaults()
      acPoly.AddVertexAt(0, New Point2d(2, 4), 0, 0, 0)
      acPoly.AddVertexAt(1, New Point2d(4, 2), 0, 0, 0)
      acPoly.AddVertexAt(2, New Point2d(6, 4), 0, 0, 0)

      '' Add the new object to the block table record and the transaction
      acBlkTblRec.AppendEntity(acPoly)
      acTrans.AddNewlyCreatedDBObject(acPoly, True)

      '' Update the display and display an alert message
      acDoc.Editor.Regen()
      Application.ShowAlertDialog("Erase the newly added polyline.")

      '' Erase the polyline from the drawing
      acPoly.Erase(True)

      '' Save the new object to the database
      acTrans.Commit()
  End Using
End Sub
```

C#

```csharp
using Autodesk.AutoCAD.Runtime;
using Autodesk.AutoCAD.ApplicationServices;
using Autodesk.AutoCAD.DatabaseServices;

[CommandMethod("EraseObject")]
public static void EraseObject()
{
  // Get the current document and database
  Document acDoc = Application.DocumentManager.MdiActiveDocument;
  Database acCurDb = acDoc.Database;

  // Start a transaction
  using (Transaction acTrans = acCurDb.TransactionManager.StartTransaction())
  {
      // Open the Block table for read
      BlockTable acBlkTbl;
      acBlkTbl = acTrans.GetObject(acCurDb.BlockTableId,
                                   OpenMode.ForRead) as BlockTable;
```

```
        // Open the Block table record Model space for write
        BlockTableRecord acBlkTblRec;
        acBlkTblRec = acTrans.GetObject(acBlkTbl[BlockTableRecord.ModelSpace],
                                        OpenMode.ForWrite) as BlockTableRecord;

        // Create a lightweight polyline
        Polyline acPoly = new Polyline();
        acPoly.SetDatabaseDefaults();
        acPoly.AddVertexAt(0, new Point2d(2, 4), 0, 0, 0);
        acPoly.AddVertexAt(1, new Point2d(4, 2), 0, 0, 0);
        acPoly.AddVertexAt(2, new Point2d(6, 4), 0, 0, 0);

        // Add the new object to the block table record and the transaction
        acBlkTblRec.AppendEntity(acPoly);
        acTrans.AddNewlyCreatedDBObject(acPoly, true);

        // Update the display and display an alert message
        acDoc.Editor.Regen();
        Application.ShowAlertDialog("Erase the newly added polyline.");

        // Erase the polyline from the drawing
        acPoly.Erase(true);

        // Save the new object to the database
        acTrans.Commit();
    }
}
```

⊟ *VBA/ActiveX Code Reference*

```
Sub EraseObject()
    ' Create the polyline
    Dim lwpolyObj As AcadLWPolyline
    Dim vertices(0 To 5) As Double
    vertices(0) = 2: vertices(1) = 4
    vertices(2) = 4: vertices(3) = 2
    vertices(4) = 6: vertices(5) = 4
    Set lwpolyObj = ThisDrawing.ModelSpace. _
                                    AddLightWeightPolyline(vertices)
    ZoomAll
    ' Erase the polyline
    lwpolyObj.Delete
    ThisDrawing.Regen acActiveViewport
End Sub
```

Copy Objects

You can create a copy of most nongraphical and graphical objects in a database. You create a copy of an object with the Clone function. Once an object is cloned, you can modify the returned object before it is added to the database. You can mimic many of the modifying commands in AutoCAD through the use of the Clone and TransformBy methods. For more information about the TransformBy method, see [Transform Objects](#).

Along with creating a direct copy of an object, you can also use the Clone and TransformBy methods to offset, mirror and array objects. For more information about copying objects, see "Copy, Offset, or Mirror Objects" in the *AutoCAD User's Guide*.

Topics in this section

- [Copy an Object](#)
- [Copy Objects between Databases](#)

Copy an Object

To copy an object, use the Clone function provided for that object. This method creates a new object that is a duplicate of the original object. Once the duplicate object is created, you can then modify it prior to adding or appending it to the database. If you do not transform the object or change its position, the new object will be located in the same position as the original.

If you have a large number of objects you might want to copy, you can add each of the object ids to an ObjectIdCollection object and then iterate each of the objects. As you iterate each object, you can then use the Clone function for each object and then add or append the new object to the database.

Copy a single object

The following example creates a new circle and then creates a direct copy of the circle to create a second circle.

VB.NET

```
Imports Autodesk.AutoCAD.Runtime
Imports Autodesk.AutoCAD.ApplicationServices
Imports Autodesk.AutoCAD.DatabaseServices
Imports Autodesk.AutoCAD.Geometry

<CommandMethod("SingleCopy")> _
Public Sub SingleCopy()
  '' Get the current document and database
  Dim acDoc As Document = Application.DocumentManager.MdiActiveDocument
  Dim acCurDb As Database = acDoc.Database
```

```vb
    '' Start a transaction
    Using acTrans As Transaction = acCurDb.TransactionManager.StartTransaction()

        '' Open the Block table for read
        Dim acBlkTbl As BlockTable
        acBlkTbl = acTrans.GetObject(acCurDb.BlockTableId, OpenMode.ForRead)

        '' Open the Block table record Model space for write
        Dim acBlkTblRec As BlockTableRecord
        acBlkTblRec = acTrans.GetObject(acBlkTbl(BlockTableRecord.ModelSpace), _
                                        OpenMode.ForWrite)

        '' Create a circle that is at 2,3 with a radius of 4.25
        Dim acCirc As Circle = New Circle()
        acCirc.SetDatabaseDefaults()
        acCirc.Center = New Point3d(2, 3, 0)
        acCirc.Radius = 4.25

        '' Add the new object to the block table record and the transaction
        acBlkTblRec.AppendEntity(acCirc)
        acTrans.AddNewlyCreatedDBObject(acCirc, True)

        '' Create a copy of the circle and change its radius
        Dim acCircClone As Circle = acCirc.Clone()
        acCircClone.Radius = 1

        '' Add the cloned circle
        acBlkTblRec.AppendEntity(acCircClone)
        acTrans.AddNewlyCreatedDBObject(acCircClone, True)

        '' Save the new object to the database
        acTrans.Commit()
    End Using
End Sub
```

C#

```csharp
using Autodesk.AutoCAD.Runtime;
using Autodesk.AutoCAD.ApplicationServices;
using Autodesk.AutoCAD.DatabaseServices;
using Autodesk.AutoCAD.Geometry;

[CommandMethod("SingleCopy")]
public static void SingleCopy()
{
    // Get the current document and database
    Document acDoc = Application.DocumentManager.MdiActiveDocument;
    Database acCurDb = acDoc.Database;

    // Start a transaction
    using (Transaction acTrans = acCurDb.TransactionManager.StartTransaction())
    {
        // Open the Block table for read
        BlockTable acBlkTbl;
        acBlkTbl = acTrans.GetObject(acCurDb.BlockTableId,
                                     OpenMode.ForRead) as BlockTable;

        // Open the Block table record Model space for write
        BlockTableRecord acBlkTblRec;
        acBlkTblRec = acTrans.GetObject(acBlkTbl[BlockTableRecord.ModelSpace],
                                        OpenMode.ForWrite) as BlockTableRecord;

        // Create a circle that is at 2,3 with a radius of 4.25
        Circle acCirc = new Circle();
```

```
        acCirc.SetDatabaseDefaults();
        acCirc.Center = new Point3d(2, 3, 0);
        acCirc.Radius = 4.25;

        // Add the new object to the block table record and the transaction
        acBlkTblRec.AppendEntity(acCirc);
        acTrans.AddNewlyCreatedDBObject(acCirc, true);

        // Create a copy of the circle and change its radius
        Circle acCircClone = acCirc.Clone() as Circle;
        acCircClone.Radius = 1;

        // Add the cloned circle
        acBlkTblRec.AppendEntity(acCircClone);
        acTrans.AddNewlyCreatedDBObject(acCircClone, true);

        // Save the new object to the database
        acTrans.Commit();
    }
}
```

VBA/ActiveX Code Reference

```
Sub SingleCopy()
    ' Define the Circle object
    Dim centerPoint(0 To 2) As Double
    centerPoint(0) = 2: centerPoint(1) = 3: centerPoint(2) = 0

    ' Define the radius for the initial circle
    Dim radius As Double
    radius = 4.25

    ' Define the radius for the copied circle
    Dim radiusCopy As Double
    radiusCopy = 1#

    ' Add the new circle to model space
    Dim circleObj As AcadCircle
    Set circleObj = ThisDrawing.ModelSpace.AddCircle(centerPoint, radius)

    ' Create a copy of the circle
    Dim circleObjCopy As AcadCircle
    Set circleObjCopy = circleObj.Copy()
    circleObjCopy.radius = radiusCopy
End Sub
```

Copy multiple objects

The following example uses an ObjectIdCollection object to track the objects that should be copied. Once the object ids are added to the collection, the collection is iterated and new objects are created by using the Clone method and then are added to Model space.

VB.NET

```
Imports Autodesk.AutoCAD.Runtime
Imports Autodesk.AutoCAD.ApplicationServices
Imports Autodesk.AutoCAD.DatabaseServices
Imports Autodesk.AutoCAD.Geometry

<CommandMethod("MultipleCopy")> _
Public Sub MultipleCopy()
```

```vbnet
    '' Get the current document and database
    Dim acDoc As Document = Application.DocumentManager.MdiActiveDocument
    Dim acCurDb As Database = acDoc.Database

    '' Start a transaction
    Using acTrans As Transaction = acCurDb.TransactionManager.StartTransaction()

        '' Open the Block table for read
        Dim acBlkTbl As BlockTable
        acBlkTbl = acTrans.GetObject(acCurDb.BlockTableId, OpenMode.ForRead)

        '' Open the Block table record Model space for write
        Dim acBlkTblRec As BlockTableRecord
        acBlkTblRec = acTrans.GetObject(acBlkTbl(BlockTableRecord.ModelSpace), _
                                        OpenMode.ForWrite)

        '' Create a circle that is at (0,0,0) with a radius of 5
        Dim acCirc1 As Circle = New Circle()
        acCirc1.SetDatabaseDefaults()
        acCirc1.Center = New Point3d(0, 0, 0)
        acCirc1.Radius = 5

        '' Add the new object to the block table record and the transaction
        acBlkTblRec.AppendEntity(acCirc1)
        acTrans.AddNewlyCreatedDBObject(acCirc1, True)

        '' Create a circle that is at (0,0,0) with a radius of 7
        Dim acCirc2 As Circle = New Circle()
        acCirc2.SetDatabaseDefaults()
        acCirc2.Center = New Point3d(0, 0, 0)
        acCirc2.Radius = 7

        '' Add the new object to the block table record and the transaction
        acBlkTblRec.AppendEntity(acCirc2)
        acTrans.AddNewlyCreatedDBObject(acCirc2, True)

        '' Add all the objects to clone
        Dim acDBObjColl As DBObjectCollection = New DBObjectCollection()
        acDBObjColl.Add(acCirc1)
        acDBObjColl.Add(acCirc2)

        For Each acEnt As Entity In acDBObjColl
            Dim acEntClone As Entity
            acEntClone = acEnt.Clone()
            acEntClone.ColorIndex = 1

            '' Create a matrix and move each copied entity 15 units
            acEntClone.TransformBy(Matrix3d.Displacement(New Vector3d(15, 0, 0)))

            '' Add the cloned object
            acBlkTblRec.AppendEntity(acEntClone)
            acTrans.AddNewlyCreatedDBObject(acEntClone, True)
        Next

        '' Save the new object to the database
        acTrans.Commit()
    End Using
End Sub
```

C#

```csharp
using Autodesk.AutoCAD.Runtime;
using Autodesk.AutoCAD.ApplicationServices;
using Autodesk.AutoCAD.DatabaseServices;
```

```csharp
using Autodesk.AutoCAD.Geometry;

[CommandMethod("MultipleCopy")]
public static void MultipleCopy()
{
  // Get the current document and database
  Document acDoc = Application.DocumentManager.MdiActiveDocument;
  Database acCurDb = acDoc.Database;

  // Start a transaction
  using (Transaction acTrans = acCurDb.TransactionManager.StartTransaction())
  {
      // Open the Block table for read
      BlockTable acBlkTbl;
      acBlkTbl = acTrans.GetObject(acCurDb.BlockTableId,
                                   OpenMode.ForRead) as BlockTable;

      // Open the Block table record Model space for write
      BlockTableRecord acBlkTblRec;
      acBlkTblRec = acTrans.GetObject(acBlkTbl[BlockTableRecord.ModelSpace],
                                      OpenMode.ForWrite) as BlockTableRecord;

      // Create a circle that is at (0,0,0) with a radius of 5
      Circle acCirc1 = new Circle();
      acCirc1.SetDatabaseDefaults();
      acCirc1.Center = new Point3d(0, 0, 0);
      acCirc1.Radius = 5;

      // Add the new object to the block table record and the transaction
      acBlkTblRec.AppendEntity(acCirc1);
      acTrans.AddNewlyCreatedDBObject(acCirc1, true);

      // Create a circle that is at (0,0,0) with a radius of 7
      Circle acCirc2 = new Circle();
      acCirc2.SetDatabaseDefaults();
      acCirc2.Center = new Point3d(0, 0, 0);
      acCirc2.Radius = 7;

      // Add the new object to the block table record and the transaction
      acBlkTblRec.AppendEntity(acCirc2);
      acTrans.AddNewlyCreatedDBObject(acCirc2, true);

      // Add all the objects to clone
      DBObjectCollection acDBObjColl = new DBObjectCollection();
      acDBObjColl.Add(acCirc1);
      acDBObjColl.Add(acCirc2);

      foreach (Entity acEnt in acDBObjColl)
      {
          Entity acEntClone;
          acEntClone = acEnt.Clone() as Entity;
          acEntClone.ColorIndex = 1;

          // Create a matrix and move each copied entity 15 units
          acEntClone.TransformBy(Matrix3d.Displacement(new Vector3d(15, 0, 0)));

          // Add the cloned object
          acBlkTblRec.AppendEntity(acEntClone);
          acTrans.AddNewlyCreatedDBObject(acEntClone, true);
      }

      // Save the new object to the database
      acTrans.Commit();
  }
}
```

```
Sub MultipleCopy()
    ' Define the Circle object
    Dim centerPoint(0 To 2) As Double
    centerPoint(0) = 0: centerPoint(1) = 0: centerPoint(2) = 0

    Dim radius1 As Double, radius2 As Double
    radius1 = 5#: radius2 = 7#

    ' Add two circles to the current drawing
    Dim circleObj1 As AcadCircle, circleObj2 As AcadCircle
    Set circleObj1 = ThisDrawing.ModelSpace.AddCircle _
                                        (centerPoint, radius1)

    Set circleObj2 = ThisDrawing.ModelSpace.AddCircle _
                                        (centerPoint, radius2)

    ' First put the objects to be copied into a form compatible
    ' with CopyObjects
    Dim objCollection(0 To 1) As Object
    Set objCollection(0) = circleObj1
    Set objCollection(1) = circleObj2

    ' Copy the objects into the model space. A
    ' collection of the new (copied) objects is returned.
    Dim retObjects As Variant
    retObjects = ThisDrawing.CopyObjects(objCollection)

    Dim ptFrom(0 To 2) As Double
    ptFrom(0) = 0: ptFrom(1) = 0: ptFrom(2) = 0

    Dim ptTo(0 To 2) As Double
    ptTo(0) = 15: ptTo(1) = 0: ptTo(2) = 0

    Dim nCount As Integer
    For nCount = 0 To UBound(retObjects)
        Dim entObj As AcadEntity
        Set entObj = retObjects(nCount)

        entObj.color = acRed
        entObj.Move ptFrom, ptTo
    Next
End Sub
```

Copy Objects between Databases

You can copy objects between two databases. The Clone function is used to copy objects within the same database, while the WblockCloneObjects method is used to copy objects from one database to another. The WblockCloneObjects method is a member of the Database object. The WblockCloneObjects method requires the following parameters:

- *ObjectIdCollection* - List of objects to be cloned.
- *ObjectId* - ObjectId of the new parent object for the objects being cloned.

- **IdMapping** - Data structure of the current and new ObjectIds for the objects being cloned.
- **DuplicateRecordCloning** - Determines how duplicate objects should be handled.
- **Defer Translation** - Controls if object ids should be translated.

Copy an object from one database to another

This example creates two Circle objects, then uses the WblockCloneObjects method to copy the circles into a new drawing. The example also creates a new drawing using the *acad.dwt* file before the circles are copied.

VB.NET

```
Imports Autodesk.AutoCAD.Runtime
Imports Autodesk.AutoCAD.ApplicationServices
Imports Autodesk.AutoCAD.DatabaseServices
Imports Autodesk.AutoCAD.Geometry

<CommandMethod("CopyObjectsBetweenDatabases", CommandFlags.Session)> _
Public Sub CopyObjectsBetweenDatabases()
  Dim acObjIdColl As ObjectIdCollection = New ObjectIdCollection()

  '' Get the current document and database
  Dim acDoc As Document = Application.DocumentManager.MdiActiveDocument
  Dim acCurDb As Database = acDoc.Database

  '' Lock the current document
  Using acLckDocCur As DocumentLock = acDoc.LockDocument()

      '' Start a transaction
      Using acTrans As Transaction = acCurDb.TransactionManager.StartTransaction()

          '' Open the Block table for read
          Dim acBlkTbl As BlockTable
          acBlkTbl = acTrans.GetObject(acCurDb.BlockTableId, OpenMode.ForRead)

          '' Open the Block table record Model space for write
          Dim acBlkTblRec As BlockTableRecord
          acBlkTblRec = acTrans.GetObject(acBlkTbl(BlockTableRecord.ModelSpace), _
                                  OpenMode.ForWrite)

          '' Create a circle that is at (0,0,0) with a radius of 5
          Dim acCirc1 As Circle = New Circle()
          acCirc1.SetDatabaseDefaults()
          acCirc1.Center = New Point3d(0, 0, 0)
          acCirc1.Radius = 5

          '' Add the new object to the block table record and the transaction
          acBlkTblRec.AppendEntity(acCirc1)
          acTrans.AddNewlyCreatedDBObject(acCirc1, True)

          '' Create a circle that is at (0,0,0) with a radius of 7
          Dim acCirc2 As Circle = New Circle()
          acCirc2.SetDatabaseDefaults()
          acCirc2.Center = New Point3d(0, 0, 0)
          acCirc2.Radius = 7

          '' Add the new object to the block table record and the transaction
          acBlkTblRec.AppendEntity(acCirc2)
          acTrans.AddNewlyCreatedDBObject(acCirc2, True)

          '' Add all the objects to copy to the new document
```

```
            acObjIdColl = New ObjectIdCollection()
            acObjIdColl.Add(acCirc1.ObjectId)
            acObjIdColl.Add(acCirc2.ObjectId)

            '' Save the new objects to the database
            acTrans.Commit()
        End Using

        '' Unlock the document
    End Using

    '' Change the file and path to match a drawing template on your workstation
    Dim sLocalRoot As String = Application.GetSystemVariable("LOCALROOTPREFIX")
    Dim sTemplatePath As String = sLocalRoot + "Template\acad.dwt"

    '' Create a new drawing to copy the objects to
    Dim acDocMgr As DocumentCollection = Application.DocumentManager
    Dim acNewDoc As Document = acDocMgr.Add(sTemplatePath)
    Dim acDbNewDoc As Database = acNewDoc.Database

    '' Lock the new document
    Using acLckDoc As DocumentLock = acNewDoc.LockDocument()

        '' Start a transaction in the new database
        Using acTrans = acDbNewDoc.TransactionManager.StartTransaction()

            '' Open the Block table for read
            Dim acBlkTblNewDoc As BlockTable
            acBlkTblNewDoc = acTrans.GetObject(acDbNewDoc.BlockTableId, _
                                               OpenMode.ForRead)

            '' Open the Block table record Model space for read
            Dim acBlkTblRecNewDoc As BlockTableRecord
            acBlkTblRecNewDoc =
acTrans.GetObject(acBlkTblNewDoc(BlockTableRecord.ModelSpace), _
                                               OpenMode.ForRead)

            '' Clone the objects to the new database
            Dim acIdMap As IdMapping = New IdMapping()
            acCurDb.WblockCloneObjects(acObjIdColl, acBlkTblRecNewDoc.ObjectId,
acIdMap, _
                                       DuplicateRecordCloning.Ignore, False)

            '' Save the copied objects to the database
            acTrans.Commit()
        End Using

        '' Unlock the document
    End Using

    '' Set the new document current
    acDocMgr.MdiActiveDocument = acNewDoc
End Sub
```

C#

```
using Autodesk.AutoCAD.Runtime;
using Autodesk.AutoCAD.ApplicationServices;
using Autodesk.AutoCAD.DatabaseServices;
using Autodesk.AutoCAD.Geometry;

[CommandMethod("CopyObjectsBetweenDatabases", CommandFlags.Session)]
public static void CopyObjectsBetweenDatabases()
{
```

```csharp
ObjectIdCollection acObjIdColl = new ObjectIdCollection();

// Get the current document and database
Document acDoc = Application.DocumentManager.MdiActiveDocument;
Database acCurDb = acDoc.Database;

// Lock the current document
using (DocumentLock acLckDocCur = acDoc.LockDocument())
{
    // Start a transaction
    using (Transaction acTrans = acCurDb.TransactionManager.StartTransaction())
    {
        // Open the Block table record for read
        BlockTable acBlkTbl;
        acBlkTbl = acTrans.GetObject(acCurDb.BlockTableId,
                                     OpenMode.ForRead) as BlockTable;

        // Open the Block table record Model space for write
        BlockTableRecord acBlkTblRec;
        acBlkTblRec = acTrans.GetObject(acBlkTbl[BlockTableRecord.ModelSpace],
                                        OpenMode.ForWrite) as BlockTableRecord;

        // Create a circle that is at (0,0,0) with a radius of 5
        Circle acCirc1 = new Circle();
        acCirc1.SetDatabaseDefaults();
        acCirc1.Center = new Point3d(0, 0, 0);
        acCirc1.Radius = 5;

        // Add the new object to the block table record and the transaction
        acBlkTblRec.AppendEntity(acCirc1);
        acTrans.AddNewlyCreatedDBObject(acCirc1, true);

        // Create a circle that is at (0,0,0) with a radius of 7
        Circle acCirc2 = new Circle();
        acCirc2.SetDatabaseDefaults();
        acCirc2.Center = new Point3d(0, 0, 0);
        acCirc2.Radius = 7;

        // Add the new object to the block table record and the transaction
        acBlkTblRec.AppendEntity(acCirc2);
        acTrans.AddNewlyCreatedDBObject(acCirc2, true);

        // Add all the objects to copy to the new document
        acObjIdColl = new ObjectIdCollection();
        acObjIdColl.Add(acCirc1.ObjectId);
        acObjIdColl.Add(acCirc2.ObjectId);

        // Save the new objects to the database
        acTrans.Commit();
    }

    // Unlock the document
}

// Change the file and path to match a drawing template on your workstation
string sLocalRoot = Application.GetSystemVariable("LOCALROOTPREFIX") as string;
string sTemplatePath = sLocalRoot + "Template\\acad.dwt";

// Create a new drawing to copy the objects to
DocumentCollection acDocMgr = Application.DocumentManager;
Document acNewDoc = acDocMgr.Add(sTemplatePath);
Database acDbNewDoc = acNewDoc.Database;

// Lock the new document
using (DocumentLock acLckDoc = acNewDoc.LockDocument())
```

```
    {
        // Start a transaction in the new database
        using (Transaction acTrans =
acDbNewDoc.TransactionManager.StartTransaction())
        {
            // Open the Block table for read
            BlockTable acBlkTblNewDoc;
            acBlkTblNewDoc = acTrans.GetObject(acDbNewDoc.BlockTableId,
                                               OpenMode.ForRead) as BlockTable;

            // Open the Block table record Model space for read
            BlockTableRecord acBlkTblRecNewDoc;
            acBlkTblRecNewDoc =
acTrans.GetObject(acBlkTblNewDoc[BlockTableRecord.ModelSpace],
                                                OpenMode.ForRead) as
BlockTableRecord;

            // Clone the objects to the new database
            IdMapping acIdMap = new IdMapping();
            acCurDb.WblockCloneObjects(acObjIdColl, acBlkTblRecNewDoc.ObjectId,
acIdMap,
                                       DuplicateRecordCloning.Ignore, false);

            // Save the copied objects to the database
            acTrans.Commit();
        }

        // Unlock the document
    }

    // Set the new document current
    acDocMgr.MdiActiveDocument = acNewDoc;
}
```

⊟ *VBA/ActiveX Code Reference*

```
Sub CopyObjectsBetweenDatabases()
    Dim DOC0 As AcadDocument
    Dim circleObj1 As AcadCircle, circleObj2 As AcadCircle
    Dim centerPoint(0 To 2) As Double
    Dim radius1 As Double, radius2 As Double
    Dim objCollection(0 To 1) As Object
    Dim retObjects As Variant

    ' Define the Circle object
    centerPoint(0) = 0: centerPoint(1) = 0: centerPoint(2) = 0
    radius1 = 5#: radius2 = 7#

    ' Add two circles to the current drawing
    Set circleObj1 = ThisDrawing.ModelSpace.AddCircle _
                    (centerPoint, radius1)
    Set circleObj2 = ThisDrawing.ModelSpace.AddCircle _
                    (centerPoint, radius2)

    ' Save pointer to the current drawing
    Set DOC0 = ThisDrawing.Application.ActiveDocument

    ' Copy objects
    '
    ' First put the objects to be copied into a form compatible
    ' with CopyObjects
    Set objCollection(0) = circleObj1
    Set objCollection(1) = circleObj2
```

```
    ' Create a new drawing and point to its model space
    Dim Doc1MSpace As AcadModelSpace
    Dim DOC1 As AcadDocument

    Set DOC1 = Documents.Add
    Set Doc1MSpace = DOC1.ModelSpace

    ' Copy the objects into the model space of the new drawing. A
    ' collection of the new (copied) objects is returned.
    retObjects = DOC0.CopyObjects(objCollection, Doc1MSpace)
End Sub
```

Offset Objects

Offsetting an object creates a new object at a specified offset distance from the original object. You can offset arcs, circles, ellipses, lines, lightweight polylines, polylines, splines, and xlines.

To offset an object, use the GetOffsetCurves method provided for that object. The function requires a positive or negative numeric value for the distance to offset the object. If the distance is negative, it is interpreted by AutoCAD as being an offset to make a "smaller" curve (that is, for an arc it would offset to a radius that is the given distance less than the starting curve's radius). If "smaller" has no meaning, then AutoCAD would offset in the direction of smaller X,Y,Z WCS coordinates.

objects offset

For many objects, the result of this operation will be a single new curve (which may not be of the same type as the original curve). For example, offsetting an ellipse will result in a spline because the result does fit the equation of an ellipse. In some cases it may be necessary for the offset result to be several curves. Because of this, the function returns a DBObjectCollection object, which contains all the objects that are created by offsetting the curve. The returned DBObjectCollection object needs to be iterated for each object created and then be appended to the drawing database.

For more information about offsetting objects, see "Copy, Offset, or Mirror Objects" in the *AutoCAD User's Guide*.

Offset a polyline

This example creates a lightweight polyline and then offsets it.

VB.NET

```vbnet
Imports Autodesk.AutoCAD.Runtime
Imports Autodesk.AutoCAD.ApplicationServices
Imports Autodesk.AutoCAD.DatabaseServices

<CommandMethod("OffsetObject")> _
  Public Sub OffsetObject()
    '' Get the current document and database
    Dim acDoc As Document = Application.DocumentManager.MdiActiveDocument
    Dim acCurDb As Database = acDoc.Database

    '' Start a transaction
    Using acTrans As Transaction = acCurDb.TransactionManager.StartTransaction()

        '' Open the Block table for read
        Dim acBlkTbl As BlockTable
        acBlkTbl = acTrans.GetObject(acCurDb.BlockTableId, _
                                     OpenMode.ForRead)

        '' Open the Block table record Model space for write
        Dim acBlkTblRec As BlockTableRecord
        acBlkTblRec = acTrans.GetObject(acBlkTbl(BlockTableRecord.ModelSpace), _
                                        OpenMode.ForWrite)

        '' Create a lightweight polyline
        Dim acPoly As Polyline = New Polyline()
        acPoly.SetDatabaseDefaults()
        acPoly.AddVertexAt(0, New Point2d(1, 1), 0, 0, 0)
        acPoly.AddVertexAt(1, New Point2d(1, 2), 0, 0, 0)
        acPoly.AddVertexAt(2, New Point2d(2, 2), 0, 0, 0)
        acPoly.AddVertexAt(3, New Point2d(3, 2), 0, 0, 0)
        acPoly.AddVertexAt(4, New Point2d(4, 4), 0, 0, 0)
        acPoly.AddVertexAt(5, New Point2d(4, 1), 0, 0, 0)

        '' Add the new object to the block table record and the transaction
        acBlkTblRec.AppendEntity(acPoly)
        acTrans.AddNewlyCreatedDBObject(acPoly, True)

        '' Offset the polyline a given distance
        Dim acDbObjColl As DBObjectCollection = acPoly.GetOffsetCurves(0.25)

        '' Step through the new objects created
        For Each acEnt As Entity In acDbObjColl
            '' Add each offset object
            acBlkTblRec.AppendEntity(acEnt)
            acTrans.AddNewlyCreatedDBObject(acEnt, True)
        Next

        '' Save the new objects to the database
        acTrans.Commit()
    End Using
End Sub
```

C#

```csharp
using Autodesk.AutoCAD.Runtime;
using Autodesk.AutoCAD.ApplicationServices;
using Autodesk.AutoCAD.DatabaseServices;

[CommandMethod("OffsetObject")]
public static void OffsetObject()
{
  // Get the current document and database
  Document acDoc = Application.DocumentManager.MdiActiveDocument;
```

```
    Database acCurDb = acDoc.Database;

    // Start a transaction
    using (Transaction acTrans = acCurDb.TransactionManager.StartTransaction())
    {
        // Open the Block table for read
        BlockTable acBlkTbl;
        acBlkTbl = acTrans.GetObject(acCurDb.BlockTableId,
                                     OpenMode.ForRead) as BlockTable;

        // Open the Block table record Model space for write
        BlockTableRecord acBlkTblRec;
        acBlkTblRec = acTrans.GetObject(acBlkTbl[BlockTableRecord.ModelSpace],
                                        OpenMode.ForWrite) as BlockTableRecord;

        // Create a lightweight polyline
        Polyline acPoly = new Polyline();
        acPoly.SetDatabaseDefaults();
        acPoly.AddVertexAt(0, new Point2d(1, 1), 0, 0, 0);
        acPoly.AddVertexAt(1, new Point2d(1, 2), 0, 0, 0);
        acPoly.AddVertexAt(2, new Point2d(2, 2), 0, 0, 0);
        acPoly.AddVertexAt(3, new Point2d(3, 2), 0, 0, 0);
        acPoly.AddVertexAt(4, new Point2d(4, 4), 0, 0, 0);
        acPoly.AddVertexAt(5, new Point2d(4, 1), 0, 0, 0);

        // Add the new object to the block table record and the transaction
        acBlkTblRec.AppendEntity(acPoly);
        acTrans.AddNewlyCreatedDBObject(acPoly, true);

        // Offset the polyline a given distance
        DBObjectCollection acDbObjColl = acPoly.GetOffsetCurves(0.25);

        // Step through the new objects created
        foreach (Entity acEnt in acDbObjColl)
        {
            // Add each offset object
            acBlkTblRec.AppendEntity(acEnt);
            acTrans.AddNewlyCreatedDBObject(acEnt, true);
        }

        // Save the new objects to the database
        acTrans.Commit();
    }
}
```

☐ *VBA/ActiveX Code Reference*

```
Sub OffsetObject()
    ' Create the polyline
    Dim plineObj As AcadLWPolyline
    Dim points(0 To 11) As Double
    points(0) = 1: points(1) = 1
    points(2) = 1: points(3) = 2
    points(4) = 2: points(5) = 2
    points(6) = 3: points(7) = 2
    points(8) = 4: points(9) = 4
    points(10) = 4: points(11) = 1
    Set plineObj = ThisDrawing.ModelSpace. _
                                    AddLightWeightPolyline(points)
    plineObj.Closed = True
    ZoomAll

    ' Offset the polyline
```

```
    Dim offsetObj As Variant
    offsetObj = plineObj.Offset(0.25)

    ZoomAll
End Su
```

Transform Objects

You move, scale, rotate and mirror an object using a 4 by 4 transformation matrix represented by a Matrix3d object and the TransformBy method. You can also use the GetTransformedCopy method to create a copy of an entity and then apply the transformation to the copy. The Matrix3d object is part of the Geometry namespace.

The first three columns of the matrix specify scale and rotation. The fourth column of the matrix is a translation vector. The following table demonstrates the transformation matrix configuration, where R = Rotation and T = Translation:

Transformation matrix configuration

R00	R01	R02	T0
R10	R11	R12	T1
R20	R21	R22	T2
0	0	0	1

To transform an object, first initialize a Matrix3d object. You can initialize the transformation matrix using an array of doubles or starting with a matrix in which represents the World coordinate system or a user coordinate system. Once initialized, you then use the functions of the Matrix3d object to modify the scaling, rotation, or displacement transformation of the matrix.

After the transformation matrix is complete, apply the matrix to the object using the TransformBy method. The following line of code demonstrates applying a matrix (dMatrix) to an object (acObj):

VB.NET

```
acObj.TransformBy(dMatrix)
```

C#

```
acObj.TransformBy(dMatrix);
```

Example of a rotation matrix

The following shows a single data array to define a transformation matrix, assigned to the variable dMatrix, which will rotate an entity by 90 degrees about the point (0, 0, 0).

Rotation Matrix: 90 degrees about point (0, 0, 0)

0.0	-1.0	0.0	0.0
1.0	0.0	0.0	0.0
0.0	0.0	1.0	0.0
0.0	0.0	0.0	1.0

VB.NET

Initialize a transformation matrix with a data array in which contains the information to rotate an object 90 degrees.

```
Dim dMatrix(0 To 15) As Double

dMatrix(0) = 0.0
dMatrix(1) = -1.0
dMatrix(2) = 0.0
dMatrix(3) = 0.0
dMatrix(4) = 1.0
dMatrix(5) = 0.0
dMatrix(6) = 0.0
dMatrix(7) = 0.0
dMatrix(8) = 0.0
dMatrix(9) = 0.0
dMatrix(10) = 1.0
dMatrix(11) = 0.0
dMatrix(12) = 0.0
dMatrix(13) = 0.0
dMatrix(14) = 0.0
dMatrix(15) = 1.0


Dim acMat3d As Matrix3d = New Matrix3d(dMatrix)
```

Initialize a transformation matrix without a data array and use the Rotation function to return a transformation matrix that rotates an object 90 degrees.

```
Dim acMat3d As Matrix3d = New Matrix3d()

Matrix3d.Rotation(Math.PI / 2, _
                  curUCS.Zaxis, _
                  New Point3d(0, 0, 0))
```

C#

Initialize a transformation matrix with a data array in which contains the information to rotate an object 90 degrees.

```
double[] dMatrix = new double[16];

dMatrix[0] = 0.0;
dMatrix[1] = -1.0;
dMatrix[2] = 0.0;
dMatrix[3] = 0.0;
dMatrix[4] = 1.0;
dMatrix[5] = 0.0;
dMatrix[6] = 0.0;
dMatrix[7] = 0.0;
dMatrix[8] = 0.0;
dMatrix[9] = 0.0;
dMatrix[10] = 1.0;
dMatrix[11] = 0.0;
dMatrix[12] = 0.0;
dMatrix[13] = 0.0;
dMatrix[14] = 0.0;
dMatrix[15] = 1.0;


Matrix3d acMat3d = new Matrix3d(dMatrix);
```

Initialize a transformation matrix without a data array and use the Rotation function to return a transformation matrix that rotates an object 90 degrees.

```
Matrix3d acMat3d = new Matrix3d();

acMat3d = Matrix3d.Rotation(Math.PI / 2,
                            curUCS.Zaxis,
                            new Point3d(0, 0, 0));
```

Additional examples of transformation matrices

The following are more examples of transformation matrices:

Rotation Matrix: 45 degrees about point (5, 5, 0)

0.707107	-0.707107	0.0	5.0
0.707107	0.707107	0.0	-2.071068
0.0	0.0	1.0	0.0
0.0	0.0	0.0	1.0

Translation Matrix: move an entity by (10, 10, 0)

1.0	0.0	0.0	10.0
0.0	1.0	0.0	10.0
0.0	0.0	1.0	0.0
0.0	0.0	0.0	1.0

Scaling Matrix: scale by 10,10 at point (0, 0, 0)

10.0	0.0	0.0	0.0
0.0	10.0	0.0	0.0
0.0	0.0	10.0	0.0
0.0	0.0	0.0	1.0

Scaling Matrix: scale by 10,10 at point (2, 2, 0)

10.0	0.0	0.0	-18.0
0.0	10.0	0.0	-18.0
0.0	0.0	10.0	0.0
0.0	0.0	0.0	1.0

Topics in this section

Move Objects

You can move all drawing objects and attribute reference objects along a specified vector.

To move an object, use the Displacement function of a transformation matrix. This function requires a Vector3d object as input. If you do not know the vector that you need, you can create a Point3d object and then use the GetVectorTo method to return the vector between

two points. The displacement vector indicates how far the given object is to be moved and in what direction.

object selected   object moved
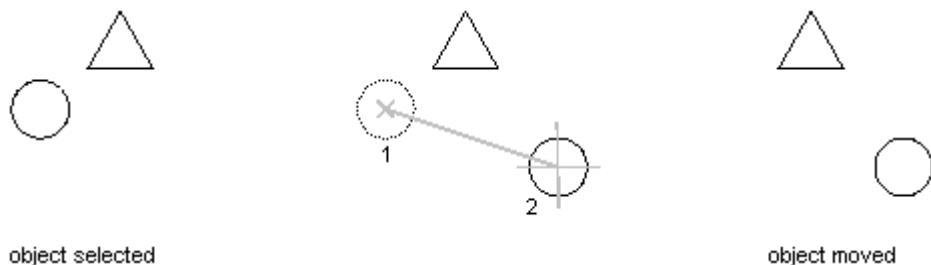
For more information about moving objects, see "Move Objects" in the *AutoCAD User's Guide.*

Move a circle along a vector

This example creates a circle and then moves that circle two units along the *X* axis.

VB.NET

```
Imports Autodesk.AutoCAD.Runtime
Imports Autodesk.AutoCAD.ApplicationServices
Imports Autodesk.AutoCAD.DatabaseServices
Imports Autodesk.AutoCAD.Geometry

<CommandMethod("MoveObject")> _
Public Sub MoveObject()
  '' Get the current document and database
  Dim acDoc As Document = Application.DocumentManager.MdiActiveDocument
  Dim acCurDb As Database = acDoc.Database

  '' Start a transaction
  Using acTrans As Transaction = acCurDb.TransactionManager.StartTransaction()

      '' Open the Block table for read
      Dim acBlkTbl As BlockTable
      acBlkTbl = acTrans.GetObject(acCurDb.BlockTableId, _
                                   OpenMode.ForRead)

      '' Open the Block table record Model space for write
      Dim acBlkTblRec As BlockTableRecord
      acBlkTblRec = acTrans.GetObject(acBlkTbl(BlockTableRecord.ModelSpace), _
                                      OpenMode.ForWrite)

      '' Create a circle that is at 2,2 with a radius of 0.5
      Dim acCirc As Circle = New Circle()
      acCirc.SetDatabaseDefaults()
      acCirc.Center = New Point3d(2, 2, 0)
      acCirc.Radius = 0.5

      '' Create a matrix and move the circle using a vector from (0,0,0) to
(2,0,0)
      Dim acPt3d As Point3d = New Point3d(0, 0, 0)
      Dim acVec3d As Vector3d = acPt3d.GetVectorTo(New Point3d(2, 0, 0))

      acCirc.TransformBy(Matrix3d.Displacement(acVec3d))

      '' Add the new object to the block table record and the transaction
      acBlkTblRec.AppendEntity(acCirc)
      acTrans.AddNewlyCreatedDBObject(acCirc, True)
```

```
      '' Save the new objects to the database
      acTrans.Commit()
  End Using
End Sub
```

C#

```csharp
using Autodesk.AutoCAD.Runtime;
using Autodesk.AutoCAD.ApplicationServices;
using Autodesk.AutoCAD.DatabaseServices;
using Autodesk.AutoCAD.Geometry;

[CommandMethod("MoveObject")]
public static void MoveObject()
{
  // Get the current document and database
  Document acDoc = Application.DocumentManager.MdiActiveDocument;
  Database acCurDb = acDoc.Database;

  // Start a transaction
  using (Transaction acTrans = acCurDb.TransactionManager.StartTransaction())
  {
      // Open the Block table for read
      BlockTable acBlkTbl;
      acBlkTbl = acTrans.GetObject(acCurDb.BlockTableId,
                                   OpenMode.ForRead) as BlockTable;

      // Open the Block table record Model space for write
      BlockTableRecord acBlkTblRec;
      acBlkTblRec = acTrans.GetObject(acBlkTbl[BlockTableRecord.ModelSpace],
                                      OpenMode.ForWrite) as BlockTableRecord;

      // Create a circle that is at 2,2 with a radius of 0.5
      Circle acCirc = new Circle();
      acCirc.SetDatabaseDefaults();
      acCirc.Center = new Point3d(2, 2, 0);
      acCirc.Radius = 0.5;

      // Create a matrix and move the circle using a vector from (0,0,0) to
(2,0,0)
      Point3d acPt3d = new Point3d(0, 0, 0);
      Vector3d acVec3d = acPt3d.GetVectorTo(new Point3d(2, 0, 0));

      acCirc.TransformBy(Matrix3d.Displacement(acVec3d));

      // Add the new object to the block table record and the transaction
      acBlkTblRec.AppendEntity(acCirc);
      acTrans.AddNewlyCreatedDBObject(acCirc, true);

      // Save the new objects to the database
      acTrans.Commit();
  }
}
```

⊟ *VBA/ActiveX Code Reference*

```
Sub MoveObject()
    ' Create the circle
    Dim circleObj As AcadCircle
    Dim center(0 To 2) As Double
    Dim radius As Double
```

```
    center(0) = 2#: center(1) = 2#: center(2) = 0#
    radius = 0.5
    Set circleObj = ThisDrawing.ModelSpace. _
                                AddCircle(center, radius)
    ZoomAll

    ' Define the points that make up the move vector.
    ' The move vector will move the circle 2 units
    ' along the x axis.
    Dim point1(0 To 2) As Double
    Dim point2(0 To 2) As Double
    point1(0) = 0: point1(1) = 0: point1(2) = 0
    point2(0) = 2: point2(1) = 0: point2(2) = 0

    ' Move the circle
    circleObj.Move point1, point2
    circleObj.Update
End Sub
```

Rotate Objects

You can rotate all drawing objects and attribute reference objects.

To rotate an object, use the Rotation function of a transformation matrix. This function requires a rotation angle represented in radians, an axis of rotation, and a base point. The axis of rotation must be expressed as a Vector3d object and the base point as a Point3d object. This angle determines how far an object rotates around the base point relative to its current location.

selected object   rotation angle   rotated object
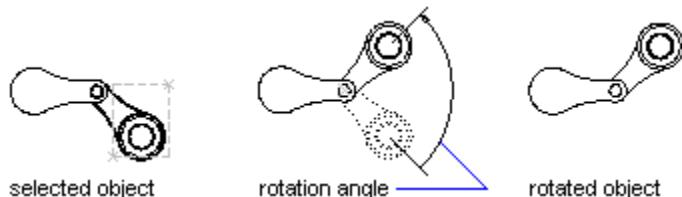
For more information about rotating objects, see "Rotate Objects" in the *User's Guide*.

Rotate a polyline about a base point

This example creates a closed lightweight polyline, and then rotates the polyline 45 degrees about the base point (4, 4.25, 0).

VB.NET

```
Imports Autodesk.AutoCAD.Runtime
Imports Autodesk.AutoCAD.ApplicationServices
Imports Autodesk.AutoCAD.DatabaseServices
Imports Autodesk.AutoCAD.Geometry

<CommandMethod("RotateObject")> _
Public Sub RotateObject()
    '' Get the current document and database
    Dim acDoc As Document = Application.DocumentManager.MdiActiveDocument
```

```vb
    Dim acCurDb As Database = acDoc.Database

    '' Start a transaction
    Using acTrans As Transaction = acCurDb.TransactionManager.StartTransaction()

        '' Open the Block table for read
        Dim acBlkTbl As BlockTable
        acBlkTbl = acTrans.GetObject(acCurDb.BlockTableId, _
                                     OpenMode.ForRead)

        '' Open the Block table record Model space for write
        Dim acBlkTblRec As BlockTableRecord
        acBlkTblRec = acTrans.GetObject(acBlkTbl(BlockTableRecord.ModelSpace), _
                                        OpenMode.ForWrite)

        '' Create a lightweight polyline
        Dim acPoly As Polyline = New Polyline()
        acPoly.SetDatabaseDefaults()
        acPoly.AddVertexAt(0, New Point2d(1, 2), 0, 0, 0)
        acPoly.AddVertexAt(1, New Point2d(1, 3), 0, 0, 0)
        acPoly.AddVertexAt(2, New Point2d(2, 3), 0, 0, 0)
        acPoly.AddVertexAt(3, New Point2d(3, 3), 0, 0, 0)
        acPoly.AddVertexAt(4, New Point2d(4, 4), 0, 0, 0)
        acPoly.AddVertexAt(5, New Point2d(4, 2), 0, 0, 0)

        '' Close the polyline
        acPoly.Closed = True

        Dim curUCSMatrix As Matrix3d = acDoc.Editor.CurrentUserCoordinateSystem
        Dim curUCS As CoordinateSystem3d = curUCSMatrix.CoordinateSystem3d

        '' Rotate the polyline 45 degrees, around the Z-axis of the current UCS
        '' using a base point of (4,4.25,0)
        acPoly.TransformBy(Matrix3d.Rotation(0.7854, _
                                             curUCS.Zaxis, _
                                             New Point3d(4, 4.25, 0)))

        '' Add the new object to the block table record and the transaction
        acBlkTblRec.AppendEntity(acPoly)
        acTrans.AddNewlyCreatedDBObject(acPoly, True)

        '' Save the new objects to the database
        acTrans.Commit()
    End Using
End Sub
```

C#

```csharp
using Autodesk.AutoCAD.Runtime;
using Autodesk.AutoCAD.ApplicationServices;
using Autodesk.AutoCAD.DatabaseServices;
using Autodesk.AutoCAD.Geometry;

[CommandMethod("RotateObject")]
public static void RotateObject()
{
  // Get the current document and database
  Document acDoc = Application.DocumentManager.MdiActiveDocument;
  Database acCurDb = acDoc.Database;

  // Start a transaction
  using (Transaction acTrans = acCurDb.TransactionManager.StartTransaction())
  {
      // Open the Block table for read
```

```
        BlockTable acBlkTbl;
        acBlkTbl = acTrans.GetObject(acCurDb.BlockTableId,
                                     OpenMode.ForRead) as BlockTable;

        // Open the Block table record Model space for write
        BlockTableRecord acBlkTblRec;
        acBlkTblRec = acTrans.GetObject(acBlkTbl[BlockTableRecord.ModelSpace],
                                        OpenMode.ForWrite) as BlockTableRecord;

        // Create a lightweight polyline
        Polyline acPoly = new Polyline();
        acPoly.SetDatabaseDefaults();
        acPoly.AddVertexAt(0, new Point2d(1, 2), 0, 0, 0);
        acPoly.AddVertexAt(1, new Point2d(1, 3), 0, 0, 0);
        acPoly.AddVertexAt(2, new Point2d(2, 3), 0, 0, 0);
        acPoly.AddVertexAt(3, new Point2d(3, 3), 0, 0, 0);
        acPoly.AddVertexAt(4, new Point2d(4, 4), 0, 0, 0);
        acPoly.AddVertexAt(5, new Point2d(4, 2), 0, 0, 0);

        // Close the polyline
        acPoly.Closed = true;

        Matrix3d curUCSMatrix = acDoc.Editor.CurrentUserCoordinateSystem;
        CoordinateSystem3d curUCS = curUCSMatrix.CoordinateSystem3d;

        // Rotate the polyline 45 degrees, around the Z-axis of the current UCS
        // using a base point of (4,4.25,0)
        acPoly.TransformBy(Matrix3d.Rotation(0.7854,
                                             curUCS.Zaxis,
                                             new Point3d(4, 4.25, 0)));

        // Add the new object to the block table record and the transaction
        acBlkTblRec.AppendEntity(acPoly);
        acTrans.AddNewlyCreatedDBObject(acPoly, true);

        // Save the new objects to the database
        acTrans.Commit();
    }
}
```

VBA/ActiveX Code Reference

```
Sub RotateObject()
    ' Create the polyline
    Dim plineObj As AcadLWPolyline
    Dim points(0 To 11) As Double
    points(0) = 1: points(1) = 2
    points(2) = 1: points(3) = 3
    points(4) = 2: points(5) = 3
    points(6) = 3: points(7) = 3
    points(8) = 4: points(9) = 4
    points(10) = 4: points(11) = 2
    Set plineObj = ThisDrawing.ModelSpace. _
                                AddLightWeightPolyline(points)
    plineObj.Closed = True
    ZoomAll

    ' Define the rotation of 45 degrees about a
    ' base point of (4, 4.25, 0)
    Dim basePoint(0 To 2) As Double
    Dim rotationAngle As Double
    basePoint(0) = 4: basePoint(1) = 4.25: basePoint(2) = 0
    rotationAngle = 0.7853981    ' 45 degrees
```

```
    ' Rotate the polyline
    plineObj.Rotate basePoint, rotationAngle
    plineObj.Update
End Sub
```

Mirror Objects

Mirroring flips an object along an axis or mirror line. You can mirror all drawing objects.

To mirror an object, use the Mirroring function of a transformation matrix. This function requires a Point3d, Plane, or Line3d object to define the mirror line. Since mirroring is done with a transformation matrix, a new object is not created. If you want to maintain the original object, you will need to create a copy of the object first and then mirror it.
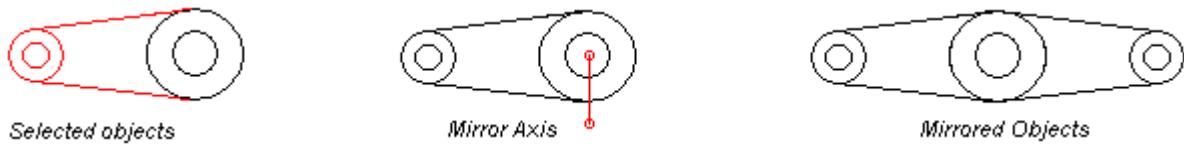
To manage the reflection properties of Text objects, use the MIRRTEXT system variable. The default setting of MIRRTEXT is On (1), which causes Text objects to be mirrored just as any other object. When MIRRTEXT is Off (0), text is not mirrored. Use the GetSystemVariable and SetSystemVariable methods to query and set the MIRRTEXT setting.
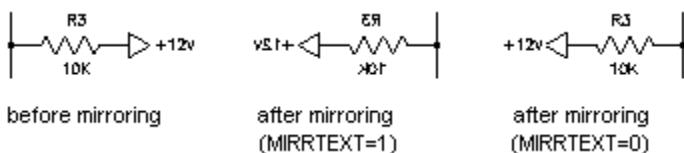
You can mirror a Viewport object in paper space, although doing so has no effect on its model space view or on model space objects.

For more information about mirroring objects, see "Copy, Offset, or Mirror Objects" in the *AutoCAD User's Guide*.

Mirror a polyline about an axis

This example creates a lightweight polyline and mirrors that polyline about an axis. The newly created polyline is colored blue.

VB.NET

```
Imports Autodesk.AutoCAD.Runtime
Imports Autodesk.AutoCAD.ApplicationServices
Imports Autodesk.AutoCAD.DatabaseServices
Imports Autodesk.AutoCAD.Geometry

<CommandMethod("MirrorObject")> _
```

```vbnet
Public Sub MirrorObject()
    '' Get the current document and database
    Dim acDoc As Document = Application.DocumentManager.MdiActiveDocument
    Dim acCurDb As Database = acDoc.Database

    '' Start a transaction
    Using acTrans As Transaction = acCurDb.TransactionManager.StartTransaction()

        '' Open the Block table for read
        Dim acBlkTbl As BlockTable
        acBlkTbl = acTrans.GetObject(acCurDb.BlockTableId, _
                                     OpenMode.ForRead)

        '' Open the Block table record Model space for write
        Dim acBlkTblRec As BlockTableRecord
        acBlkTblRec = acTrans.GetObject(acBlkTbl(BlockTableRecord.ModelSpace), _
                                        OpenMode.ForWrite)

        '' Create a lightweight polyline
        Dim acPoly As Polyline = New Polyline()
        acPoly.SetDatabaseDefaults()
        acPoly.AddVertexAt(0, New Point2d(1, 1), 0, 0, 0)
        acPoly.AddVertexAt(1, New Point2d(1, 2), 0, 0, 0)
        acPoly.AddVertexAt(2, New Point2d(2, 2), 0, 0, 0)
        acPoly.AddVertexAt(3, New Point2d(3, 2), 0, 0, 0)
        acPoly.AddVertexAt(4, New Point2d(4, 4), 0, 0, 0)
        acPoly.AddVertexAt(5, New Point2d(4, 1), 0, 0, 0)

        '' Create a bulge of -2 at vertex 1
        acPoly.SetBulgeAt(1, -2)

        '' Close the polyline
        acPoly.Closed = True

        '' Add the new object to the block table record and the transaction
        acBlkTblRec.AppendEntity(acPoly)
        acTrans.AddNewlyCreatedDBObject(acPoly, True)

        '' Create a copy of the original polyline
        Dim acPolyMirCopy As Polyline = acPoly.Clone()
        acPolyMirCopy.ColorIndex = 5

        '' Define the mirror line
        Dim acPtFrom As Point3d = New Point3d(0, 4.25, 0)
        Dim acPtTo As Point3d = New Point3d(4, 4.25, 0)
        Dim acLine3d As Line3d = New Line3d(acPtFrom, acPtTo)

        '' Mirror the polyline across the X axis
        acPolyMirCopy.TransformBy(Matrix3d.Mirroring(acLine3d))

        '' Add the new object to the block table record and the transaction
        acBlkTblRec.AppendEntity(acPolyMirCopy)
        acTrans.AddNewlyCreatedDBObject(acPolyMirCopy, True)

        '' Save the new objects to the database
        acTrans.Commit()
    End Using
End Sub
```

C#

```csharp
using Autodesk.AutoCAD.Runtime;
using Autodesk.AutoCAD.ApplicationServices;
using Autodesk.AutoCAD.DatabaseServices;
```

```csharp
using Autodesk.AutoCAD.Geometry;

[CommandMethod("MirrorObject")]
public static void MirrorObject()
{
  // Get the current document and database
  Document acDoc = Application.DocumentManager.MdiActiveDocument;
  Database acCurDb = acDoc.Database;

  // Start a transaction
  using (Transaction acTrans = acCurDb.TransactionManager.StartTransaction())
  {
      // Open the Block table for read
      BlockTable acBlkTbl;
      acBlkTbl = acTrans.GetObject(acCurDb.BlockTableId,
                                   OpenMode.ForRead) as BlockTable;

      // Open the Block table record Model space for write
      BlockTableRecord acBlkTblRec;
      acBlkTblRec = acTrans.GetObject(acBlkTbl[BlockTableRecord.ModelSpace],
                                      OpenMode.ForWrite) as BlockTableRecord;

      // Create a lightweight polyline
      Polyline acPoly = new Polyline();
      acPoly.SetDatabaseDefaults();
      acPoly.AddVertexAt(0, new Point2d(1, 1), 0, 0, 0);
      acPoly.AddVertexAt(1, new Point2d(1, 2), 0, 0, 0);
      acPoly.AddVertexAt(2, new Point2d(2, 2), 0, 0, 0);
      acPoly.AddVertexAt(3, new Point2d(3, 2), 0, 0, 0);
      acPoly.AddVertexAt(4, new Point2d(4, 4), 0, 0, 0);
      acPoly.AddVertexAt(5, new Point2d(4, 1), 0, 0, 0);

      // Create a bulge of -2 at vertex 1
      acPoly.SetBulgeAt(1, -2);

      // Close the polyline
      acPoly.Closed = true;

      // Add the new object to the block table record and the transaction
      acBlkTblRec.AppendEntity(acPoly);
      acTrans.AddNewlyCreatedDBObject(acPoly, true);

      // Create a copy of the original polyline
      Polyline acPolyMirCopy = acPoly.Clone() as Polyline;
      acPolyMirCopy.ColorIndex = 5;

      // Define the mirror line
      Point3d acPtFrom = new Point3d(0, 4.25, 0);
      Point3d acPtTo = new Point3d(4, 4.25, 0);
      Line3d acLine3d = new Line3d(acPtFrom, acPtTo);

      // Mirror the polyline across the X axis
      acPolyMirCopy.TransformBy(Matrix3d.Mirroring(acLine3d));

      // Add the new object to the block table record and the transaction
      acBlkTblRec.AppendEntity(acPolyMirCopy);
      acTrans.AddNewlyCreatedDBObject(acPolyMirCopy, true);

      // Save the new objects to the database
      acTrans.Commit();
  }
}
```

```
Sub MirrorObject()
    ' Create the polyline
    Dim plineObj As AcadLWPolyline
    Dim points(0 To 11) As Double
    points(0) = 1: points(1) = 1
    points(2) = 1: points(3) = 2
    points(4) = 2: points(5) = 2
    points(6) = 3: points(7) = 2
    points(8) = 4: points(9) = 4
    points(10) = 4: points(11) = 1
    Set plineObj = ThisDrawing.ModelSpace. _
                        AddLightWeightPolyline(points)

    plineObj.SetBulge 1, -2

    plineObj.Closed = True
    ZoomAll

    ' Define the mirror axis
    Dim point1(0 To 2) As Double
    Dim point2(0 To 2) As Double
    point1(0) = 0: point1(1) = 4.25: point1(2) = 0
    point2(0) = 4: point2(1) = 4.25: point2(2) = 0

    ' Mirror the polyline
    Dim mirrorObj As AcadLWPolyline
    Set mirrorObj = plineObj.Mirror(point1, point2)

    mirrorObj.color = acBlue

    ZoomAll
End Sub
```

Scale Objects

You scale an object by specifying a base point and scale factor based on the current drawing units. You can scale all drawing objects, as well as attribute reference objects.

To scale an object, use the Scaling function of a transformation matrix. This function requires a numeric value for the scale factor of the object and a Point3d object for the base point of the scaling operation. The Scaling function scales the object equally in the *X*, *Y*, and *Z* directions. The dimensions of the object are multiplied by the scale factor. A scale factor greater than 1 enlarges the object. A scale factor between 0 and 1 reduces the object.

NoteIf you need to scale an object non-uniformly, you need to initialize a transformation matrix using the appropriate data array and then use the TransformBy method of the object. For more information on initializing a transformation matrix with a data array, see Transform Objects.
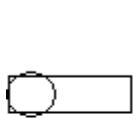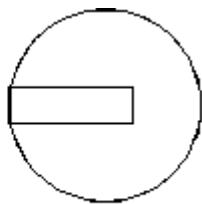
scale factor=.5   scale factor=2

For more information about scaling, see "Resize or Reshape Objects" in the *AutoCAD User's Guide*.

Scale a polyline

This example creates a closed lightweight polyline and then scales the polyline by 0.5 from the base point (4,4.25,0).

VB.NET

```
Imports Autodesk.AutoCAD.Runtime
Imports Autodesk.AutoCAD.ApplicationServices
Imports Autodesk.AutoCAD.DatabaseServices
Imports Autodesk.AutoCAD.Geometry

<CommandMethod("ScaleObject")> _
Public Sub ScaleObject()
  '' Get the current document and database
  Dim acDoc As Document = Application.DocumentManager.MdiActiveDocument
  Dim acCurDb As Database = acDoc.Database

  '' Start a transaction
  Using acTrans As Transaction = acCurDb.TransactionManager.StartTransaction()

      '' Open the Block table for read
      Dim acBlkTbl As BlockTable
      acBlkTbl = acTrans.GetObject(acCurDb.BlockTableId, _
                                   OpenMode.ForRead)

      '' Open the Block table record Model space for write
      Dim acBlkTblRec As BlockTableRecord
      acBlkTblRec = acTrans.GetObject(acBlkTbl(BlockTableRecord.ModelSpace), _
                                      OpenMode.ForWrite)

      '' Create a lightweight polyline
      Dim acPoly As Polyline = New Polyline()
      acPoly.SetDatabaseDefaults()
      acPoly.AddVertexAt(0, New Point2d(1, 2), 0, 0, 0)
      acPoly.AddVertexAt(1, New Point2d(1, 3), 0, 0, 0)
      acPoly.AddVertexAt(2, New Point2d(2, 3), 0, 0, 0)
      acPoly.AddVertexAt(3, New Point2d(3, 3), 0, 0, 0)
      acPoly.AddVertexAt(4, New Point2d(4, 4), 0, 0, 0)
      acPoly.AddVertexAt(5, New Point2d(4, 2), 0, 0, 0)

      '' Close the polyline
      acPoly.Closed = True

      '' Reduce the object by a factor of 0.5
      '' using a base point of (4,4.25,0)
      acPoly.TransformBy(Matrix3d.Scaling(0.5, New Point3d(4, 4.25, 0)))

      '' Add the new object to the block table record and the transaction
      acBlkTblRec.AppendEntity(acPoly)
```

```
      acTrans.AddNewlyCreatedDBObject(acPoly, True)

      '' Save the new objects to the database
      acTrans.Commit()
   End Using
End Sub
```

C#

```csharp
using Autodesk.AutoCAD.Runtime;
using Autodesk.AutoCAD.ApplicationServices;
using Autodesk.AutoCAD.DatabaseServices;
using Autodesk.AutoCAD.Geometry;

[CommandMethod("ScaleObject")]
public static void ScaleObject()
{
  // Get the current document and database
  Document acDoc = Application.DocumentManager.MdiActiveDocument;
  Database acCurDb = acDoc.Database;

  // Start a transaction
  using (Transaction acTrans = acCurDb.TransactionManager.StartTransaction())
  {
      // Open the Block table for read
      BlockTable acBlkTbl;
      acBlkTbl = acTrans.GetObject(acCurDb.BlockTableId,
                                   OpenMode.ForRead) as BlockTable;

      // Open the Block table record Model space for write
      BlockTableRecord acBlkTblRec;
      acBlkTblRec = acTrans.GetObject(acBlkTbl[BlockTableRecord.ModelSpace],
                                      OpenMode.ForWrite) as BlockTableRecord;

      // Create a lightweight polyline
      Polyline acPoly = new Polyline();
      acPoly.SetDatabaseDefaults();
      acPoly.AddVertexAt(0, new Point2d(1, 2), 0, 0, 0);
      acPoly.AddVertexAt(1, new Point2d(1, 3), 0, 0, 0);
      acPoly.AddVertexAt(2, new Point2d(2, 3), 0, 0, 0);
      acPoly.AddVertexAt(3, new Point2d(3, 3), 0, 0, 0);
      acPoly.AddVertexAt(4, new Point2d(4, 4), 0, 0, 0);
      acPoly.AddVertexAt(5, new Point2d(4, 2), 0, 0, 0);

      // Close the polyline
      acPoly.Closed = true;

      // Reduce the object by a factor of 0.5
      // using a base point of (4,4.25,0)
      acPoly.TransformBy(Matrix3d.Scaling(0.5, new Point3d(4, 4.25, 0)));

      // Add the new object to the block table record and the transaction
      acBlkTblRec.AppendEntity(acPoly);
      acTrans.AddNewlyCreatedDBObject(acPoly, true);

      // Save the new objects to the database
      acTrans.Commit();
  }
}
```

```
Sub ScaleObject()
    ' Create the polyline
    Dim plineObj As AcadLWPolyline
    Dim points(0 To 11) As Double
    points(0) = 1: points(1) = 2
    points(2) = 1: points(3) = 3
    points(4) = 2: points(5) = 3
    points(6) = 3: points(7) = 3
    points(8) = 4: points(9) = 4
    points(10) = 4: points(11) = 2
    Set plineObj = ThisDrawing.ModelSpace. _
                            AddLightWeightPolyline(points)
    plineObj.Closed = True
    ZoomAll

    ' Define the scale
    Dim basePoint(0 To 2) As Double
    Dim scalefactor As Double
    basePoint(0) = 4: basePoint(1) = 4.25: basePoint(2) = 0
    scalefactor = 0.5

    ' Scale the polyline
    plineObj.ScaleEntity basePoint, scalefactor
    plineObj.Update
End Sub
```

Array Objects

You can create a polar or rectangular array of an object. Arrays of objects are not created using a dedicated set of functions, but are created through a combination of copying objects, and then using a transformation matrix to rotate and move the copied object. The following outlines the basic logic for each type of array:

- *Polar array*. Copy the object to be arrayed and move it based on an angle around a the base point. The distance from the object to the base point of the array is used to calculate the placement of each copy that is created. Once the copied object is moved, you can then rotate the object based on its angle from the base point. Once each copy is created, it needs to be appended to the block table record.
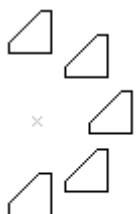
polar array to
fill=180; objects
not rotated

- *Rectangular array*. Copy the object to array based on the number of desired rows and columns. The distance that the copied objects are copied is based on a specified distance between the rows and columns. You first want to create the number of copies of the original to

complete the first row or column. Once the first row or column is created, you can then create the number of objects for the remaining rows or columns based on the first row or column you created. Once each copy is created, it needs to be appended to the block table record.
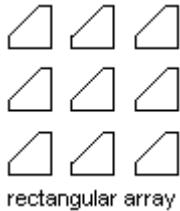
rectangular array

For more information about arrays, see "Create an Array of Objects" in the *AutoCAD User's Guide.*

Topics in this section

- [Create Polar Arrays](#)
- [Create Rectangular Arrays](#)

Create Polar Arrays

This example creates a circle, and then performs a polar array of the circle. This creates four circles filling 180 degrees around a base point of (4, 4, 0).

VB.NET

```
Imports Autodesk.AutoCAD.Runtime
Imports Autodesk.AutoCAD.ApplicationServices
Imports Autodesk.AutoCAD.DatabaseServices
Imports Autodesk.AutoCAD.Geometry

Public Shared Function PolarPoints(ByVal pPt As Point2d, _
                                   ByVal dAng As Double, _
                                   ByVal dDist As Double)

  Return New Point2d(pPt.X + dDist * Math.Cos(dAng), _
                     pPt.Y + dDist * Math.Sin(dAng))
End Function

<CommandMethod("PolarArrayObject")> _
Public Sub PolarArrayObject()
  '' Get the current document and database
  Dim acDoc As Document = Application.DocumentManager.MdiActiveDocument
  Dim acCurDb As Database = acDoc.Database

  '' Start a transaction
  Using acTrans As Transaction = acCurDb.TransactionManager.StartTransaction()

      '' Open the Block table record for read
      Dim acBlkTbl As BlockTable
      acBlkTbl = acTrans.GetObject(acCurDb.BlockTableId, _
                                   OpenMode.ForRead)

      '' Open the Block table record Model space for write
```

```vbnet
Dim acBlkTblRec As BlockTableRecord
acBlkTblRec = acTrans.GetObject(acBlkTbl(BlockTableRecord.ModelSpace), _
                                OpenMode.ForWrite)

'' Create a circle that is at 2,2 with a radius of 1
Dim acCirc As Circle = New Circle()
acCirc.SetDatabaseDefaults()
acCirc.Center = New Point3d(2, 2, 0)
acCirc.Radius = 1

'' Add the new object to the block table record and the transaction
acBlkTblRec.AppendEntity(acCirc)
acTrans.AddNewlyCreatedDBObject(acCirc, True)

'' Create a 4 object polar array that goes a 180
Dim nCount As Integer = 1

'' Set a value in radians for 60 degrees
Dim dAng As Double = 1.0472

'' Use (4,4,0) as the base point for the array
Dim acPt2dArrayBase As Point2d = New Point2d(4, 4)

While (nCount < 4)
    Dim acEntClone As Entity = acCirc.Clone()

    Dim acExts As Extents3d
    Dim acPtObjBase As Point2d

    '' Typically the upper-left corner of an object's extents is used
    '' for the point on the object to be arrayed unless it is
    '' an object like a circle.
    Dim acCircArrObj As Circle = acEntClone

    If IsDBNull(acCircArrObj) = False Then
        acPtObjBase = New Point2d(acCircArrObj.Center.X, _
                                  acCircArrObj.Center.Y)
    Else
        acExts = acEntClone.Bounds.GetValueOrDefault()
        acPtObjBase = New Point2d(acExts.MinPoint.X, _
                                  acExts.MaxPoint.Y)
    End If

    Dim dDist As Double = acPt2dArrayBase.GetDistanceTo(acPtObjBase)
    Dim dAngFromX As Double = acPt2dArrayBase.GetVectorTo(acPtObjBase).Angle

    Dim acPt2dTo As Point2d = PolarPoints(acPt2dArrayBase, _
                                          (nCount * dAng) + dAngFromX, _
                                          dDist)

    Dim acVec2d As Vector2d = acPtObjBase.GetVectorTo(acPt2dTo)
    Dim acVec3d As Vector3d = New Vector3d(acVec2d.X, acVec2d.Y, 0)
    acEntClone.TransformBy(Matrix3d.Displacement(acVec3d))

    '' The following code demonstrates how to rotate each object like
    '' the ARRAY command does.
    'acExts = acEntClone.Bounds.GetValueOrDefault()
    'acPtObjBase = New Point2d(acExts.MinPoint.X, _
    ' acExts.MaxPoint.Y)
    '
    '' Rotate the cloned entity and around its upper-left extents point
    'Dim curUCSMatrix As Matrix3d = acDoc.Editor.CurrentUserCoordinateSystem
    'Dim curUCS As CoordinateSystem3d = curUCSMatrix.CoordinateSystem3d
    'acEntClone.TransformBy(Matrix3d.Rotation(nCount * dAng, _
    ' curUCS.Zaxis, _
```

```
            ' New Point3d(acPtObjBase.X, _
            ' acPtObjBase.Y, 0)))

            acBlkTblRec.AppendEntity(acEntClone)
            acTrans.AddNewlyCreatedDBObject(acEntClone, True)

            nCount = nCount + 1
        End While

        '' Save the new objects to the database
        acTrans.Commit()
    End Using
End Sub
```

C#

```csharp
using Autodesk.AutoCAD.Runtime;
using Autodesk.AutoCAD.ApplicationServices;
using Autodesk.AutoCAD.DatabaseServices;
using Autodesk.AutoCAD.Geometry;

static Point2d PolarPoints(Point2d pPt, double dAng, double dDist)
{
  return new Point2d(pPt.X + dDist * Math.Cos(dAng),
                     pPt.Y + dDist * Math.Sin(dAng));
}

[CommandMethod("PolarArrayObject")]
public static void PolarArrayObject()
{
  // Get the current document and database
  Document acDoc = Application.DocumentManager.MdiActiveDocument;
  Database acCurDb = acDoc.Database;

  // Start a transaction
  using (Transaction acTrans = acCurDb.TransactionManager.StartTransaction())
  {
      // Open the Block table record for read
      BlockTable acBlkTbl;
      acBlkTbl = acTrans.GetObject(acCurDb.BlockTableId,
                                   OpenMode.ForRead) as BlockTable;

      // Open the Block table record Model space for write
      BlockTableRecord acBlkTblRec;
      acBlkTblRec = acTrans.GetObject(acBlkTbl[BlockTableRecord.ModelSpace],
                                      OpenMode.ForWrite) as BlockTableRecord;

      // Create a circle that is at 2,2 with a radius of 1
      Circle acCirc = new Circle();
      acCirc.SetDatabaseDefaults();
      acCirc.Center = new Point3d(2, 2, 0);
      acCirc.Radius = 1;

      // Add the new object to the block table record and the transaction
      acBlkTblRec.AppendEntity(acCirc);
      acTrans.AddNewlyCreatedDBObject(acCirc, true);

      // Create a 4 object polar array that goes a 180
      int nCount = 1;

      // Set a value in radians for 60 degrees
      double dAng = 1.0472;

      // Use (4,4,0) as the base point for the array
```

```
        Point2d acPt2dArrayBase = new Point2d(4, 4);

    while (nCount < 4)
    {
        Entity acEntClone = acCirc.Clone() as Entity;

        Extents3d acExts;
        Point2d acPtObjBase;

        // Typically the upper-left corner of an object's extents is used
        // for the point on the object to be arrayed unless it is
        // an object like a circle.
        Circle acCircArrObj = acEntClone as Circle;

        if (acCircArrObj != null)
        {
            acPtObjBase = new Point2d(acCircArrObj.Center.X,
                                      acCircArrObj.Center.Y);
        }
        else
        {
            acExts = acEntClone.Bounds.GetValueOrDefault();
            acPtObjBase = new Point2d(acExts.MinPoint.X,
                                      acExts.MaxPoint.Y);
        }

        double dDist = acPt2dArrayBase.GetDistanceTo(acPtObjBase);
        double dAngFromX = acPt2dArrayBase.GetVectorTo(acPtObjBase).Angle;

        Point2d acPt2dTo = PolarPoints(acPt2dArrayBase,
                                       (nCount * dAng) + dAngFromX,
                                       dDist);

        Vector2d acVec2d = acPtObjBase.GetVectorTo(acPt2dTo);
        Vector3d acVec3d = new Vector3d(acVec2d.X, acVec2d.Y, 0);
        acEntClone.TransformBy(Matrix3d.Displacement(acVec3d));

        /*
        // The following code demonstrates how to rotate each object like
        // the ARRAY command does.
        acExts = acEntClone.Bounds.GetValueOrDefault();
        acPtObjBase = new Point2d(acExts.MinPoint.X,
                                  acExts.MaxPoint.Y);

        // Rotate the cloned entity around its upper-left extents point
        Matrix3d curUCSMatrix = acDoc.Editor.CurrentUserCoordinateSystem;
        CoordinateSystem3d curUCS = curUCSMatrix.CoordinateSystem3d;
        acEntClone.TransformBy(Matrix3d.Rotation(nCount * dAng,
                                                 curUCS.Zaxis,
                                                 new Point3d(acPtObjBase.X,
                                                             acPtObjBase.Y,
0)));
        */

        acBlkTblRec.AppendEntity(acEntClone);
        acTrans.AddNewlyCreatedDBObject(acEntClone, true);

        nCount = nCount + 1;
    }

    // Save the new objects to the database
    acTrans.Commit();
    }
}
```

```
Sub PolarArrayObject()
    ' Create the circle
    Dim circleObj As AcadCircle
    Dim center(0 To 2) As Double
    Dim radius As Double
    center(0) = 2#: center(1) = 2#: center(2) = 0#
    radius = 1
    Set circleObj = ThisDrawing.ModelSpace. _
                                 AddCircle(center, radius)
    ZoomAll

    ' Define the polar array
    Dim noOfObjects As Integer
    Dim angleToFill As Double
    Dim basePnt(0 To 2) As Double
    noOfObjects = 4
    angleToFill = 3.14          ' 180 degrees
    basePnt(0) = 4#: basePnt(1) = 4#: basePnt(2) = 0#

    ' The following example will create 4 copies
    ' of an object by rotating and copying it about
    ' the point (4,4,0).
    Dim retObj As Variant
    retObj = circleObj.ArrayPolar _
                        (noOfObjects, angleToFill, basePnt)

    ZoomAll
End Sub
```

Create Rectangular Arrays

This example creates a circle and then performs a rectangular array of the circle, creating five rows and five columns of circles.

VB.NET

```
Imports Autodesk.AutoCAD.Runtime
Imports Autodesk.AutoCAD.ApplicationServices
Imports Autodesk.AutoCAD.DatabaseServices
Imports Autodesk.AutoCAD.Geometry

Public Shared Function PolarPoints(ByVal pPt As Point2d, _
                                   ByVal dAng As Double, _
                                   ByVal dDist As Double)

  Return New Point2d(pPt.X + dDist * Math.Cos(dAng), _
                     pPt.Y + dDist * Math.Sin(dAng))
End Function

<CommandMethod("RectangularArrayObject")> _
Public Sub RectangularArrayObject()
  '' Get the current document and database
  Dim acDoc As Document = Application.DocumentManager.MdiActiveDocument
  Dim acCurDb As Database = acDoc.Database
```

```vb
'' Start a transaction
Using acTrans As Transaction = acCurDb.TransactionManager.StartTransaction()

    '' Open the Block table record for read
    Dim acBlkTbl As BlockTable
    acBlkTbl = acTrans.GetObject(acCurDb.BlockTableId, _
                                 OpenMode.ForRead)

    '' Open the Block table record Model space for write
    Dim acBlkTblRec As BlockTableRecord
    acBlkTblRec = acTrans.GetObject(acBlkTbl(BlockTableRecord.ModelSpace), _
                                    OpenMode.ForWrite)

    '' Create a circle that is at 2,2 with a radius of 0.5
    Dim acCirc As Circle = New Circle()
    acCirc.SetDatabaseDefaults()
    acCirc.Center = New Point3d(2, 2, 0)
    acCirc.Radius = 0.5

    '' Add the new object to the block table record and the transaction
    acBlkTblRec.AppendEntity(acCirc)
    acTrans.AddNewlyCreatedDBObject(acCirc, True)

    '' Create a rectangular array with 5 rows and 5 columns
    Dim nRows As Integer = 5
    Dim nColumns As Integer = 5

    '' Set the row and column offsets along with the base array angle
    Dim dRowOffset As Double = 1
    Dim dColumnOffset As Double = 1
    Dim dArrayAng As Double = 0

    '' Get the angle from X for the current UCS
    Dim curUCSMatrix As Matrix3d = acDoc.Editor.CurrentUserCoordinateSystem
    Dim curUCS As CoordinateSystem3d = curUCSMatrix.CoordinateSystem3d
    Dim acVec2dAng As Vector2d = New Vector2d(curUCS.Xaxis.X, _
                                              curUCS.Xaxis.Y)

    '' If the UCS is rotated, adjust the array angle accordingly
    dArrayAng = dArrayAng + acVec2dAng.Angle

    '' Use the upper-left corner of the objects extents for the array base point
    Dim acExts As Extents3d = acCirc.Bounds.GetValueOrDefault()
    Dim acPt2dArrayBase As Point2d = New Point2d(acExts.MinPoint.X, _
                                                 acExts.MaxPoint.Y)

    '' Track the objects created for each column
    Dim acDBObjCollCols As DBObjectCollection = New DBObjectCollection()
    acDBObjCollCols.Add(acCirc)

    '' Create the number of objects for the first column
    Dim nColumnsCount As Integer = 1
    While (nColumns > nColumnsCount)
        Dim acEntClone As Entity = acCirc.Clone()
        acDBObjCollCols.Add(acEntClone)

        '' Caclucate the new point for the copied object (move)
        Dim acPt2dTo As Point2d = PolarPoints(acPt2dArrayBase, _
                                              dArrayAng, _
                                              dColumnOffset * nColumnsCount)

        Dim acVec2d As Vector2d = acPt2dArrayBase.GetVectorTo(acPt2dTo)
        Dim acVec3d As Vector3d = New Vector3d(acVec2d.X, acVec2d.Y, 0)
        acEntClone.TransformBy(Matrix3d.Displacement(acVec3d))
```

```
            acBlkTblRec.AppendEntity(acEntClone)
            acTrans.AddNewlyCreatedDBObject(acEntClone, True)

            nColumnsCount = nColumnsCount + 1
        End While

        '' Set a value in radians for 90 degrees
        Dim dAng As Double = Math.PI / 2

        '' Track the objects created for each row and column
        Dim acDBObjCollLvls As DBObjectCollection = New DBObjectCollection()

        For Each acObj As DBObject In acDBObjCollCols
            acDBObjCollLvls.Add(acObj)
        Next

        '' Create the number of objects for each row
        For Each acEnt As Entity In acDBObjCollCols
            Dim nRowsCount As Integer = 1

            While (nRows > nRowsCount)
                Dim acEntClone As Entity = acEnt.Clone()
                acDBObjCollLvls.Add(acEntClone)

                '' Caclucate the new point for the copied object (move)
                Dim acPt2dTo As Point2d = PolarPoints(acPt2dArrayBase, _
                                                      dArrayAng + dAng, _
                                                      dRowOffset * nRowsCount)

                Dim acVec2d As Vector2d = acPt2dArrayBase.GetVectorTo(acPt2dTo)
                Dim acVec3d As Vector3d = New Vector3d(acVec2d.X, acVec2d.Y, 0)
                acEntClone.TransformBy(Matrix3d.Displacement(acVec3d))

                acBlkTblRec.AppendEntity(acEntClone)
                acTrans.AddNewlyCreatedDBObject(acEntClone, True)

                nRowsCount = nRowsCount + 1
            End While
        Next

        '' Save the new objects to the database
        acTrans.Commit()
    End Using
End Sub
```

C#

```csharp
using Autodesk.AutoCAD.Runtime;
using Autodesk.AutoCAD.ApplicationServices;
using Autodesk.AutoCAD.DatabaseServices;
using Autodesk.AutoCAD.Geometry;

static Point2d PolarPoints(Point2d pPt, double dAng, double dDist)
{
    return new Point2d(pPt.X + dDist * Math.Cos(dAng),
                       pPt.Y + dDist * Math.Sin(dAng));
}

[CommandMethod("RectangularArrayObject")]
public static void RectangularArrayObject()
{
    // Get the current document and database
    Document acDoc = Application.DocumentManager.MdiActiveDocument;
```

```
Database acCurDb = acDoc.Database;

// Start a transaction
using (Transaction acTrans = acCurDb.TransactionManager.StartTransaction())
{
    // Open the Block table record for read
    BlockTable acBlkTbl;
    acBlkTbl = acTrans.GetObject(acCurDb.BlockTableId,
                                 OpenMode.ForRead) as BlockTable;

    // Open the Block table record Model space for write
    BlockTableRecord acBlkTblRec;
    acBlkTblRec = acTrans.GetObject(acBlkTbl[BlockTableRecord.ModelSpace],
                                    OpenMode.ForWrite) as BlockTableRecord;

    // Create a circle that is at 2,2 with a radius of 0.5
    Circle acCirc = new Circle();
    acCirc.SetDatabaseDefaults();
    acCirc.Center = new Point3d(2, 2, 0);
    acCirc.Radius = 0.5;

    // Add the new object to the block table record and the transaction
    acBlkTblRec.AppendEntity(acCirc);
    acTrans.AddNewlyCreatedDBObject(acCirc, true);

    // Create a rectangular array with 5 rows and 5 columns
    int nRows = 5;
    int nColumns = 5;

    // Set the row and column offsets along with the base array angle
    double dRowOffset = 1;
    double dColumnOffset = 1;
    double dArrayAng = 0;

    // Get the angle from X for the current UCS
    Matrix3d curUCSMatrix = acDoc.Editor.CurrentUserCoordinateSystem;
    CoordinateSystem3d curUCS = curUCSMatrix.CoordinateSystem3d;
    Vector2d acVec2dAng = new Vector2d(curUCS.Xaxis.X,
                                       curUCS.Xaxis.Y);

    // If the UCS is rotated, adjust the array angle accordingly
    dArrayAng = dArrayAng + acVec2dAng.Angle;

    // Use the upper-left corner of the objects extents for the array base point
    Extents3d acExts = acCirc.Bounds.GetValueOrDefault();
    Point2d acPt2dArrayBase = new Point2d(acExts.MinPoint.X,
                                          acExts.MaxPoint.Y);

    // Track the objects created for each column
    DBObjectCollection acDBObjCollCols = new DBObjectCollection();
    acDBObjCollCols.Add(acCirc);

    // Create the number of objects for the first column
    int nColumnsCount = 1;
    while (nColumns > nColumnsCount)
    {
        Entity acEntClone = acCirc.Clone() as Entity;
        acDBObjCollCols.Add(acEntClone);

        // Caclucate the new point for the copied object (move)
        Point2d acPt2dTo = PolarPoints(acPt2dArrayBase,
                                       dArrayAng,
                                       dColumnOffset * nColumnsCount);

        Vector2d acVec2d = acPt2dArrayBase.GetVectorTo(acPt2dTo);
```

```
        Vector3d acVec3d = new Vector3d(acVec2d.X, acVec2d.Y, 0);
        acEntClone.TransformBy(Matrix3d.Displacement(acVec3d));

        acBlkTblRec.AppendEntity(acEntClone);
        acTrans.AddNewlyCreatedDBObject(acEntClone, true);

        nColumnsCount = nColumnsCount + 1;
    }

    // Set a value in radians for 90 degrees
    double dAng = Math.PI / 2;

    // Track the objects created for each row and column
    DBObjectCollection acDBObjCollLvls = new DBObjectCollection();

    foreach (DBObject acObj in acDBObjCollCols)
    {
        acDBObjCollLvls.Add(acObj);
    }

    // Create the number of objects for each row
    foreach (Entity acEnt in acDBObjCollCols)
    {
        int nRowsCount = 1;

        while (nRows > nRowsCount)
        {
            Entity acEntClone = acEnt.Clone() as Entity;
            acDBObjCollLvls.Add(acEntClone);

            // Caclucate the new point for the copied object (move)
            Point2d acPt2dTo = PolarPoints(acPt2dArrayBase,
                                           dArrayAng + dAng,
                                           dRowOffset * nRowsCount);

            Vector2d acVec2d = acPt2dArrayBase.GetVectorTo(acPt2dTo);
            Vector3d acVec3d = new Vector3d(acVec2d.X, acVec2d.Y, 0);
            acEntClone.TransformBy(Matrix3d.Displacement(acVec3d));

            acBlkTblRec.AppendEntity(acEntClone);
            acTrans.AddNewlyCreatedDBObject(acEntClone, true);

            nRowsCount = nRowsCount + 1;
        }
    }

    // Save the new objects to the database
    acTrans.Commit();
    }
}
```

VBA/ActiveX Code Reference

```
Sub RectangularArrayObject()
    ' Create the circle
    Dim circleObj As AcadCircle
    Dim center(0 To 2) As Double
    Dim radius As Double
    center(0) = 2#: center(1) = 2#: center(2) = 0#
    radius = 0.5
    Set circleObj = ThisDrawing.ModelSpace. _
                              AddCircle(center, radius)
    ZoomAll
```

```
    ' Define the rectangular array
    Dim numberOfRows As Long
    Dim numberOfColumns As Long
    Dim numberOfLevels As Long
    Dim distanceBwtnRows As Double
    Dim distanceBwtnColumns As Double
    Dim distanceBwtnLevels As Double
    numberOfRows = 5
    numberOfColumns = 5
    numberOfLevels = 0
    distanceBwtnRows = 1
    distanceBwtnColumns = 1
    distanceBwtnLevels = 0

    ' Create the array of objects
    Dim retObj As Variant
    retObj = circleObj.ArrayRectangular _
                (numberOfRows, numberOfColumns, numberOfLevels, _
                 distanceBwtnRows, distanceBwtnColumns, distanceBwtnLevels)

    ZoomAll
End Sub
```

Extend and Trim Objects

You can change the angle of arcs and you can change the length of lines, open polylines, elliptical arcs, and open splines. The results are similar to both extending and trimming objects.

You can extend or trim an object by editing its properties. For example, to lengthen a line, simply change the coordinates of the StartPoint or EndPoint properties. To change the angle of an arc, change the StartAngle or EndAngle property of the arc. Once you have altered an object's property or properties, you might need to regenerate the display to see the changes in the drawing window.

For more information about extending and trimming objects, see "Resize or Reshape Objects" in the *AutoCAD User's Guide*.

Lengthen a line

This example creates a line and then changes the endpoint of that line, resulting in a longer line.

VB.NET

```
Imports Autodesk.AutoCAD.Runtime
Imports Autodesk.AutoCAD.ApplicationServices
Imports Autodesk.AutoCAD.DatabaseServices
Imports Autodesk.AutoCAD.Geometry

<CommandMethod("ExtendObject")> _
Public Sub ExtendObject()
  '' Get the current document and database
  Dim acDoc As Document = Application.DocumentManager.MdiActiveDocument
```

```vbnet
    Dim acCurDb As Database = acDoc.Database

    '' Start a transaction
    Using acTrans As Transaction = acCurDb.TransactionManager.StartTransaction()

        '' Open the Block table for read
        Dim acBlkTbl As BlockTable
        acBlkTbl = acTrans.GetObject(acCurDb.BlockTableId, _
                                     OpenMode.ForRead)

        '' Open the Block table record Model space for write
        Dim acBlkTblRec As BlockTableRecord
        acBlkTblRec = acTrans.GetObject(acBlkTbl(BlockTableRecord.ModelSpace), _
                                        OpenMode.ForWrite)

        '' Create a line that starts at (4,4,0) and ends at (7,7,0)
        Dim acLine As Line = New Line(New Point3d(4, 4, 0), _
                                      New Point3d(7, 7, 0))

        acLine.SetDatabaseDefaults()

        '' Add the new object to the block table record and the transaction
        acBlkTblRec.AppendEntity(acLine)
        acTrans.AddNewlyCreatedDBObject(acLine, True)

        '' Update the display and diaplay a message box
        acDoc.Editor.Regen()
        Application.ShowAlertDialog("Before extend")

        '' Double the length of the line
        acLine.EndPoint = acLine.EndPoint + acLine.Delta

        '' Save the new object to the database
        acTrans.Commit()
    End Using
End Sub
```

C#

```csharp
using Autodesk.AutoCAD.Runtime;
using Autodesk.AutoCAD.ApplicationServices;
using Autodesk.AutoCAD.DatabaseServices;
using Autodesk.AutoCAD.Geometry;

[CommandMethod("ExtendObject")]
public static void ExtendObject()
{
  // Get the current document and database
  Document acDoc = Application.DocumentManager.MdiActiveDocument;
  Database acCurDb = acDoc.Database;

  // Start a transaction
  using (Transaction acTrans = acCurDb.TransactionManager.StartTransaction())
  {
      // Open the Block table for read
      BlockTable acBlkTbl;
      acBlkTbl = acTrans.GetObject(acCurDb.BlockTableId,
                                   OpenMode.ForRead) as BlockTable;

      // Open the Block table record Model space for write
      BlockTableRecord acBlkTblRec;
      acBlkTblRec = acTrans.GetObject(acBlkTbl[BlockTableRecord.ModelSpace],
                                      OpenMode.ForWrite) as BlockTableRecord;
```

```
    // Create a line that starts at (4,4,0) and ends at (7,7,0)
    Line acLine = new Line(new Point3d(4, 4, 0),
                           new Point3d(7, 7, 0));

    acLine.SetDatabaseDefaults();

    // Add the new object to the block table record and the transaction
    acBlkTblRec.AppendEntity(acLine);
    acTrans.AddNewlyCreatedDBObject(acLine, true);

    // Update the display and diaplay a message box
    acDoc.Editor.Regen();
    Application.ShowAlertDialog("Before extend");

    // Double the length of the line
    acLine.EndPoint = acLine.EndPoint + acLine.Delta;

    // Save the new object to the database
    acTrans.Commit();
  }
}
```

⊟ *VBA/ActiveX Code Reference*

```
Sub ExtendObject()
    ' Define and create the line
    Dim lineObj As AcadLine
    Dim startPoint(0 To 2) As Double
    Dim endPoint(0 To 2)  As Double
    startPoint(0) = 4
    startPoint(1) = 4
    startPoint(2) = 0
    endPoint(0) = 7
    endPoint(1) = 7
    endPoint(2) = 0
    Set lineObj = ThisDrawing.ModelSpace. _
                               AddLine(startPoint, endPoint)
    lineObj.Update

    ' Double the length of the line
    endPoint(0) = lineObj.endPoint(0) + lineObj.Delta(0)
    endPoint(1) = lineObj.endPoint(1) + lineObj.Delta(1)
    endPoint(2) = lineObj.endPoint(2) + lineObj.Delta(2)
    lineObj.endPoint = endPoint
    lineObj.Update
End Sub
```

Explode Objects

Exploding an object converts the single object to its constituent parts. You use the Explode function to explode an object, and it requires a DBObjectCollection object in which is used to return the resulting objects. For example, exploding a polyline can result in the creation of an object collection that contains multiple lines and arcs.

If a block is exploded, the object collection returned holds the graphical objects in which define the block. After an object is exploded, the original object is left unaltered. If you want

the returned objects to replace the original object, the original object must be erased and then the returned objects must be added to a block table record.

For more information about exploding objects, see "Disassociate Compound Objects (Explode)" in the *AutoCAD User's Guide*.

Explode a polyline

This example creates a lightweight polyline object and then explodes the polyline into its simplest objects. After the polyline is exploded, it is disposed of and the returned objects are added to Model space.

VB.NET

```
Imports Autodesk.AutoCAD.Runtime
Imports Autodesk.AutoCAD.ApplicationServices
Imports Autodesk.AutoCAD.DatabaseServices
Imports Autodesk.AutoCAD.Geometry

<CommandMethod("ExplodeObject")> _
Public Sub ExplodeObject()
  '' Get the current document and database
  Dim acDoc As Document = Application.DocumentManager.MdiActiveDocument
  Dim acCurDb As Database = acDoc.Database

  '' Start a transaction
  Using acTrans As Transaction = acCurDb.TransactionManager.StartTransaction()

      '' Open the Block table for read
      Dim acBlkTbl As BlockTable
      acBlkTbl = acTrans.GetObject(acCurDb.BlockTableId, _
                                   OpenMode.ForRead)

      '' Open the Block table record Model space for write
      Dim acBlkTblRec As BlockTableRecord
      acBlkTblRec = acTrans.GetObject(acBlkTbl(BlockTableRecord.ModelSpace), _
                                      OpenMode.ForWrite)

      '' Create a lightweight polyline
      Using acPoly As Polyline = New Polyline()
          acPoly.SetDatabaseDefaults()
          acPoly.AddVertexAt(0, New Point2d(1, 1), 0, 0, 0)
          acPoly.AddVertexAt(1, New Point2d(1, 2), 0, 0, 0)
          acPoly.AddVertexAt(2, New Point2d(2, 2), 0, 0, 0)
          acPoly.AddVertexAt(3, New Point2d(3, 2), 0, 0, 0)
          acPoly.AddVertexAt(4, New Point2d(4, 4), 0, 0, 0)
          acPoly.AddVertexAt(5, New Point2d(4, 1), 0, 0, 0)

          '' Sets the bulge at index 3
          acPoly.SetBulgeAt(3, -0.5)

          '' Explodes the polyline
          Dim acDBObjColl As DBObjectCollection = New DBObjectCollection()
          acPoly.Explode(acDBObjColl)

          For Each acEnt As Entity In acDBObjColl
              '' Add the new object to the block table record and the transaction
              acBlkTblRec.AppendEntity(acEnt)
              acTrans.AddNewlyCreatedDBObject(acEnt, True)
          Next

          '' Dispose of the in memory polyline
```

```
        End Using

        '' Save the new objects to the database
        acTrans.Commit()
    End Using
End Sub
```

C#

```csharp
using Autodesk.AutoCAD.Runtime;
using Autodesk.AutoCAD.ApplicationServices;
using Autodesk.AutoCAD.DatabaseServices;
using Autodesk.AutoCAD.Geometry;

[CommandMethod("ExplodeObject")]
public static void ExplodeObject()
{
    // Get the current document and database
    Document acDoc = Application.DocumentManager.MdiActiveDocument;
    Database acCurDb = acDoc.Database;

    // Start a transaction
    using (Transaction acTrans = acCurDb.TransactionManager.StartTransaction())
    {
        // Open the Block table for read
        BlockTable acBlkTbl;
        acBlkTbl = acTrans.GetObject(acCurDb.BlockTableId,
                                     OpenMode.ForRead) as BlockTable;

        // Open the Block table record Model space for write
        BlockTableRecord acBlkTblRec;
        acBlkTblRec = acTrans.GetObject(acBlkTbl[BlockTableRecord.ModelSpace],
                                        OpenMode.ForWrite) as BlockTableRecord;

        // Create a lightweight polyline
        using (Polyline acPoly = new Polyline())
        {
            acPoly.SetDatabaseDefaults();
            acPoly.AddVertexAt(0, new Point2d(1, 1), 0, 0, 0);
            acPoly.AddVertexAt(1, new Point2d(1, 2), 0, 0, 0);
            acPoly.AddVertexAt(2, new Point2d(2, 2), 0, 0, 0);
            acPoly.AddVertexAt(3, new Point2d(3, 2), 0, 0, 0);
            acPoly.AddVertexAt(4, new Point2d(4, 4), 0, 0, 0);
            acPoly.AddVertexAt(5, new Point2d(4, 1), 0, 0, 0);

            // Sets the bulge at index 3
            acPoly.SetBulgeAt(3, -0.5);

            // Explodes the polyline
            DBObjectCollection acDBObjColl = new DBObjectCollection();
            acPoly.Explode(acDBObjColl);

            foreach (Entity acEnt in acDBObjColl)
            {
                // Add the new object to the block table record and the transaction
                acBlkTblRec.AppendEntity(acEnt);
                acTrans.AddNewlyCreatedDBObject(acEnt, true);
            }

            // Dispose of the in memory polyline
        }

        // Save the new objects to the database
        acTrans.Commit();
```

```
  }
}
```

```
Sub ExplodeObject()
    Dim plineObj As AcadLWPolyline
    Dim points(0 To 11) As Double

    ' Define the 2D polyline points
    points(0) = 1: points(1) = 1
    points(2) = 1: points(3) = 2
    points(4) = 2: points(5) = 2
    points(6) = 3: points(7) = 2
    points(8) = 4: points(9) = 4
    points(10) = 4: points(11) = 1

    ' Create a light weight Polyline object
    Set plineObj = ThisDrawing.ModelSpace. _
                             AddLightWeightPolyline(points)

    ' Set the bulge on one segment to vary the
    ' type of objects in the polyline
    plineObj.SetBulge 3, -0.5

    ' Explode the polyline
    Dim explodedObjects As Variant
    explodedObjects = plineObj.Explode

    ' Erase the polyline
    plineObj.Erase
End Sub
```

Edit Polylines

2D and 3D polylines, rectangles, polygons, donuts, and 3D polygon meshes are all polyline variants and are edited in the same way.

AutoCAD recognizes both fit polylines and spline-fit polylines. A spline-fit polyline uses a curve fit, similar to a B-spline. There are two kinds of spline-fit polylines: quadratic and cubic. Both polylines are controlled by the SPLINETYPE system variable. A fit polyline uses standard curves for curve fit and utilizes any tangent directions set on any given vertex.

To edit a polyline, use the properties and methods of the Polyline, Polyline2d, or Polyline3d object. Use the following properties and methods to open or close a polyline, change the coordinates of a polyline vertex, or add a vertex:

Closed property

Opens or closes the polyline.

ConstantWidth property

Sets the constant width for a lightweight and 2D polyline.

AppendVertex method

Adds a vertex to a 2D or 3D polyline.

AddVertexAt method

Adds a vertex to a lightweight polyline.

ReverseCurve

Reverses the direction of the polyline.

Use the following methods to update the bulge or width of a polyline:

SetBulgeAt

Sets the bulge of a light polyline, given the segment index.

SetStartWidthAt

Sets the start width of a lightweight polyline, given the segment index.

Straighten

Straightens a 2D or 3D polyline.

For more information about editing polylines, see "Modify or Join Polyline" in the *User's Guide*.

Edit a polyline

This example creates a lightweight polyline. It then adds a bulge to the third segment of the polyline, appends a vertex to the polyline, changes the width of the last segment, and finally closes the polyline.

VB.NET

```
Imports Autodesk.AutoCAD.Runtime
Imports Autodesk.AutoCAD.ApplicationServices
Imports Autodesk.AutoCAD.DatabaseServices
Imports Autodesk.AutoCAD.Geometry

<CommandMethod("EditPolyline")> _
Public Sub EditPolyline()
  '' Get the current document and database
  Dim acDoc As Document = Application.DocumentManager.MdiActiveDocument
  Dim acCurDb As Database = acDoc.Database

  '' Start a transaction
  Using acTrans As Transaction = acCurDb.TransactionManager.StartTransaction()

    '' Open the Block table for read
    Dim acBlkTbl As BlockTable
    acBlkTbl = acTrans.GetObject(acCurDb.BlockTableId, _
```

```
                                      OpenMode.ForRead)

      '' Open the Block table record Model space for write
      Dim acBlkTblRec As BlockTableRecord
      acBlkTblRec = acTrans.GetObject(acBlkTbl(BlockTableRecord.ModelSpace), _
                                      OpenMode.ForWrite)

      '' Create a lightweight polyline
      Dim acPoly As Polyline = New Polyline()
      acPoly.SetDatabaseDefaults()
      acPoly.AddVertexAt(0, New Point2d(1, 1), 0, 0, 0)
      acPoly.AddVertexAt(1, New Point2d(1, 2), 0, 0, 0)
      acPoly.AddVertexAt(2, New Point2d(2, 2), 0, 0, 0)
      acPoly.AddVertexAt(3, New Point2d(3, 2), 0, 0, 0)
      acPoly.AddVertexAt(4, New Point2d(4, 4), 0, 0, 0)

      '' Add the new object to the block table record and the transaction
      acBlkTblRec.AppendEntity(acPoly)
      acTrans.AddNewlyCreatedDBObject(acPoly, True)

      '' Sets the bulge at index 3
      acPoly.SetBulgeAt(3, -0.5)

      '' Add a new vertex
      acPoly.AddVertexAt(5, New Point2d(4, 1), 0, 0, 0)

      '' Sets the start and end width at index 4
      acPoly.SetStartWidthAt(4, 0.1)
      acPoly.SetEndWidthAt(4, 0.5)

      '' Close the polyline
      acPoly.Closed = True

      '' Save the new objects to the database
      acTrans.Commit()
   End Using
End Sub
```

C#

```csharp
using Autodesk.AutoCAD.Runtime;
using Autodesk.AutoCAD.ApplicationServices;
using Autodesk.AutoCAD.DatabaseServices;
using Autodesk.AutoCAD.Geometry;

[CommandMethod("EditPolyline")]
public static void EditPolyline()
{
  // Get the current document and database
  Document acDoc = Application.DocumentManager.MdiActiveDocument;
  Database acCurDb = acDoc.Database;

  // Start a transaction
  using (Transaction acTrans = acCurDb.TransactionManager.StartTransaction())
  {
      // Open the Block table for read
      BlockTable acBlkTbl;
      acBlkTbl = acTrans.GetObject(acCurDb.BlockTableId,
                                   OpenMode.ForRead) as BlockTable;

      // Open the Block table record Model space for write
      BlockTableRecord acBlkTblRec;
      acBlkTblRec = acTrans.GetObject(acBlkTbl[BlockTableRecord.ModelSpace],
                                      OpenMode.ForWrite) as BlockTableRecord;
```

```
        // Create a lightweight polyline
        Polyline acPoly = new Polyline();
        acPoly.SetDatabaseDefaults();
        acPoly.AddVertexAt(0, new Point2d(1, 1), 0, 0, 0);
        acPoly.AddVertexAt(1, new Point2d(1, 2), 0, 0, 0);
        acPoly.AddVertexAt(2, new Point2d(2, 2), 0, 0, 0);
        acPoly.AddVertexAt(3, new Point2d(3, 2), 0, 0, 0);
        acPoly.AddVertexAt(4, new Point2d(4, 4), 0, 0, 0);

        // Add the new object to the block table record and the transaction
        acBlkTblRec.AppendEntity(acPoly);
        acTrans.AddNewlyCreatedDBObject(acPoly, true);

        // Sets the bulge at index 3
        acPoly.SetBulgeAt(3, -0.5);

        // Add a new vertex
        acPoly.AddVertexAt(5, new Point2d(4, 1), 0, 0, 0);

        // Sets the start and end width at index 4
        acPoly.SetStartWidthAt(4, 0.1);
        acPoly.SetEndWidthAt(4, 0.5);

        // Close the polyline
        acPoly.Closed = true;

        // Save the new objects to the database
        acTrans.Commit();
    }
}
```

VBA/ActiveX Code Reference

```
Sub EditPolyline()
    Dim plineObj As AcadLWPolyline
    Dim points(0 To 9) As Double

    ' Define the 2D polyline points
    points(0) = 1: points(1) = 1
    points(2) = 1: points(3) = 2
    points(4) = 2: points(5) = 2
    points(6) = 3: points(7) = 2
    points(8) = 4: points(9) = 4

    ' Create a light weight Polyline object
    Set plineObj = ThisDrawing.ModelSpace. _
                              AddLightWeightPolyline(points)

    ' Add a bulge to segment 3
    plineObj.SetBulge 3, -0.5

    ' Define the new vertex
    Dim newVertex(0 To 1) As Double
    newVertex(0) = 4: newVertex(1) = 1

    ' Add the vertex to the polyline
    plineObj.AddVertex 5, newVertex

    ' Set the width of the new segment
    plineObj.SetWidth 4, 0.1, 0.5

    ' Close the polyline
```

```
    plineObj.Closed = True
    plineObj.Update
End Sub
```

Edit Hatches

You can edit both hatch boundaries and hatch patterns. If you edit the boundary of an associative hatch, the pattern is updated as long as the editing results in a valid boundary. Associative hatches are updated even if they're on layers that are turned off. You can modify hatch patterns or choose a new pattern for an existing hatch, but associativity can only be set when a hatch is created. You can check to see if a Hatch object is associative by using the Associative property.

You must re-evaluate a hatch using the EvaluateHatch method to see any edits to the hatch.

For more information about editing hatches, see "Modify Hatches and Solid-Filled Areas" in the *AutoCAD User's Guide.*

Topics in this section

- [Edit Hatch Boundaries](#)
- [Edit Hatch Patterns](#)

Edit Hatch Boundaries

You can append, insert, or remove loops from the boundaries of a Hatch object. Associative hatches are updated to match any changes made to their boundaries. Non-associative hatches are not updated.

To edit a hatch boundary, use one of the following methods:

AppendLoop

> Appends a loop to the hatch. You define the type of loop being appended with first parameter of the AppendLoop method and the constants defined by the HatchLoopTypes enum.

GetLoopAt

> Gets the loop at a given index of a hatch.

InsertLoopAt

> Inserts a loop at a given index of a hatch.

RemoveLoopAt

Deletes a loop at a given index of a hatch.

To query a hatch boundary, use one of the following methods:

LoopTypeAt

Gets the type of loop at a given index of a hatch.

NumberOfLoops

Gets the number of loops of a hatch.

Append an inner loop to a hatch

This example creates an associative hatch. It then creates a circle and appends the circle as an inner loop to the hatch.

VB.NET

```
Imports Autodesk.AutoCAD.Runtime
Imports Autodesk.AutoCAD.ApplicationServices
Imports Autodesk.AutoCAD.DatabaseServices
Imports Autodesk.AutoCAD.Geometry

<CommandMethod("EditHatchAppendLoop")> _
Public Sub EditHatchAppendLoop()
  '' Get the current document and database
  Dim acDoc As Document = Application.DocumentManager.MdiActiveDocument
  Dim acCurDb As Database = acDoc.Database

  '' Start a transaction
  Using acTrans As Transaction = acCurDb.TransactionManager.StartTransaction()

      '' Open the Block table for read
      Dim acBlkTbl As BlockTable
      acBlkTbl = acTrans.GetObject(acCurDb.BlockTableId, _
                                   OpenMode.ForRead)

      '' Open the Block table record Model space for write
      Dim acBlkTblRec As BlockTableRecord
      acBlkTblRec = acTrans.GetObject(acBlkTbl(BlockTableRecord.ModelSpace), _
                                      OpenMode.ForWrite)

      '' Create an arc object for the closed boundary to hatch
      Dim acArc As Arc = New Arc(New Point3d(5, 3, 0), 3, 0, 3.141592)
      acArc.SetDatabaseDefaults()

      acBlkTblRec.AppendEntity(acArc)
      acTrans.AddNewlyCreatedDBObject(acArc, True)

      '' Create an line object for the closed boundary to hatch
      Dim acLine As Line = New Line(acArc.StartPoint, acArc.EndPoint)
      acLine.SetDatabaseDefaults()

      acBlkTblRec.AppendEntity(acLine)
      acTrans.AddNewlyCreatedDBObject(acLine, True)

      '' Adds the arc and line to an object id collection
```

```
        Dim acObjIdColl As ObjectIdCollection = New ObjectIdCollection()
        acObjIdColl.Add(acArc.ObjectId)
        acObjIdColl.Add(acLine.ObjectId)

        '' Create the hatch object and append it to the block table record
        Dim acHatch As Hatch = New Hatch()
        acBlkTblRec.AppendEntity(acHatch)
        acTrans.AddNewlyCreatedDBObject(acHatch, True)

        '' Set the properties of the hatch object
        '' Associative must be set after the hatch object is appended to the
        '' block table record and before AppendLoop
        acHatch.SetDatabaseDefaults()
        acHatch.SetHatchPattern(HatchPatternType.PreDefined, "ANSI31")
        acHatch.Associative = True
        acHatch.AppendLoop(HatchLoopTypes.Outermost, acObjIdColl)

        '' Create a circle object for the inner boundary of the hatch
        Dim acCirc As Circle = New Circle()
        acCirc.SetDatabaseDefaults()
        acCirc.Center = New Point3d(5, 4.5, 0)
        acCirc.Radius = 1

        acBlkTblRec.AppendEntity(acCirc)
        acTrans.AddNewlyCreatedDBObject(acCirc, True)

        '' Adds the circle to an object id collection
        acObjIdColl.Clear()
        acObjIdColl.Add(acCirc.ObjectId)

        '' Append the circle as the inner loop of the hatch and evaluate it
        acHatch.AppendLoop(HatchLoopTypes.Default, acObjIdColl)
        acHatch.EvaluateHatch(True)

        '' Save the new object to the database
        acTrans.Commit()
    End Using
End Sub
```

C#

```
using Autodesk.AutoCAD.Runtime;
using Autodesk.AutoCAD.ApplicationServices;
using Autodesk.AutoCAD.DatabaseServices;
using Autodesk.AutoCAD.Geometry;

[CommandMethod("EditHatchAppendLoop")]
public static void EditHatchAppendLoop()
{
    // Get the current document and database
    Document acDoc = Application.DocumentManager.MdiActiveDocument;
    Database acCurDb = acDoc.Database;

    // Start a transaction
    using (Transaction acTrans = acCurDb.TransactionManager.StartTransaction())
    {
        // Open the Block table for read
        BlockTable acBlkTbl;
        acBlkTbl = acTrans.GetObject(acCurDb.BlockTableId,
                                     OpenMode.ForRead) as BlockTable;

        // Open the Block table record Model space for write
        BlockTableRecord acBlkTblRec;
        acBlkTblRec = acTrans.GetObject(acBlkTbl[BlockTableRecord.ModelSpace],
```

```
                                    OpenMode.ForWrite) as BlockTableRecord;

    // Create an arc object for the closed boundary to hatch
    Arc acArc = new Arc(new Point3d(5, 3, 0), 3, 0, 3.141592);
    acArc.SetDatabaseDefaults();

    acBlkTblRec.AppendEntity(acArc);
    acTrans.AddNewlyCreatedDBObject(acArc, true);

    // Create an line object for the closed boundary to hatch
    Line acLine = new Line(acArc.StartPoint, acArc.EndPoint);
    acLine.SetDatabaseDefaults();

    acBlkTblRec.AppendEntity(acLine);
    acTrans.AddNewlyCreatedDBObject(acLine, true);

    // Adds the arc and line to an object id collection
    ObjectIdCollection acObjIdColl = new ObjectIdCollection();
    acObjIdColl.Add(acArc.ObjectId);
    acObjIdColl.Add(acLine.ObjectId);

    // Create the hatch object and append it to the block table record
    Hatch acHatch = new Hatch();
    acBlkTblRec.AppendEntity(acHatch);
    acTrans.AddNewlyCreatedDBObject(acHatch, true);

    // Set the properties of the hatch object
    // Associative must be set after the hatch object is appended to the
    // block table record and before AppendLoop
    acHatch.SetDatabaseDefaults();
    acHatch.SetHatchPattern(HatchPatternType.PreDefined, "ANSI31");
    acHatch.Associative = true;
    acHatch.AppendLoop(HatchLoopTypes.Outermost, acObjIdColl);

    // Create a circle object for the inner boundary of the hatch
    Circle acCirc = new Circle();
    acCirc.SetDatabaseDefaults();
    acCirc.Center = new Point3d(5, 4.5, 0);
    acCirc.Radius = 1;

    acBlkTblRec.AppendEntity(acCirc);
    acTrans.AddNewlyCreatedDBObject(acCirc, true);

    // Adds the circle to an object id collection
    acObjIdColl.Clear();
    acObjIdColl.Add(acCirc.ObjectId);

    // Append the circle as the inner loop of the hatch and evaluate it
    acHatch.AppendLoop(HatchLoopTypes.Default, acObjIdColl);
    acHatch.EvaluateHatch(true);

    // Save the new object to the database
    acTrans.Commit();
  }
}
```

⊟ *VBA/ActiveX Code Reference*

```
Sub EditHatchAppendLoop()
    Dim hatchObj As AcadHatch
    Dim patternName As String
    Dim PatternType As Long
    Dim bAssociativity As Boolean
```

```
    ' Define and create the hatch
    patternName = "ANSI31"
    PatternType = 0
    bAssociativity = True
    Set hatchObj = ThisDrawing.ModelSpace. _
                        AddHatch(PatternType, patternName, bAssociativity)

    ' Create the outer loop for the hatch.
    Dim outerLoop(0 To 1) As AcadEntity
    Dim center(0 To 2) As Double
    Dim radius As Double
    Dim startAngle As Double
    Dim endAngle As Double
    center(0) = 5: center(1) = 3: center(2) = 0
    radius = 3
    startAngle = 0
    endAngle = 3.141592
    Set outerLoop(0) = ThisDrawing.ModelSpace. _
                            AddArc(center, radius, startAngle, endAngle)
    Set outerLoop(1) = ThisDrawing.ModelSpace. _
                            AddLine(outerLoop(0).startPoint, outerLoop(0).endPoint)

    ' Append the outer loop to the hatch object
    hatchObj.AppendOuterLoop (outerLoop)

    ' Create a circle as the inner loop for the hatch.
    Dim innerLoop(0) As AcadEntity
    center(0) = 5: center(1) = 4.5: center(2) = 0
    radius = 1
    Set innerLoop(0) = ThisDrawing.ModelSpace. _
                            AddCircle(center, radius)

    ' Append the circle as an inner loop to the hatch
    hatchObj.AppendInnerLoop (innerLoop)

    ' Evaluate and display the hatch
    hatchObj.Evaluate
    ThisDrawing.Regen True
End Sub
```

Edit Hatch Patterns

You can change the angle or spacing of an existing hatch pattern or replace it with a solid-fill, gradient fill, or one of the predefined patterns that AutoCAD offers. The Pattern option in the Boundary Hatch dialog box displays a list of these patterns. To reduce file size, the hatch is defined in the drawing as a single graphic object.

Use the following properties and methods to edit the hatch patterns:

GradientAngle

Specifies the gradient angle of the hatch.

GradientName

Returns the gradient name of the hatch.

GradientShift

Specifies the gradient shift of the hatch.

GradientType

Returns the gradient type of the hatch.

PatternAngle

Specifies the angle of the hatch pattern.

PatternDouble

Specifies if the user-defined hatch is double-hatched.

PatternName

Returns the hatch pattern name of the hatch. (Use the SetHatchPattern method to set the hatch pattern name and type of the hatch.)

PatternScale

Specifies the hatch pattern scale.

PatternSpace

Specifies the user-defined hatch pattern spacing.

PatternType

Returns the hatch pattern type of the hatch. (Use the SetHatchPattern method to set the hatch pattern name and type of the hatch.)

SetGradient

Sets the gradient type and name for the hatch.

SetHatchPattern

Sets the pattern type and name for the hatch.

Change the pattern spacing of a hatch

This example creates a hatch. It then adds two to the current pattern spacing for the hatch.

VB.NET

```
Imports Autodesk.AutoCAD.Runtime
Imports Autodesk.AutoCAD.ApplicationServices
Imports Autodesk.AutoCAD.DatabaseServices
Imports Autodesk.AutoCAD.Geometry
```

```vbnet
<CommandMethod("EditHatchPatternScale")> _
Public Sub EditHatchPatternScale()
  '' Get the current document and database
  Dim acDoc As Document = Application.DocumentManager.MdiActiveDocument
  Dim acCurDb As Database = acDoc.Database

  '' Start a transaction
  Using acTrans As Transaction = acCurDb.TransactionManager.StartTransaction()

      '' Open the Block table for read
      Dim acBlkTbl As BlockTable
      acBlkTbl = acTrans.GetObject(acCurDb.BlockTableId, _
                                   OpenMode.ForRead)

      '' Open the Block table record Model space for write
      Dim acBlkTblRec As BlockTableRecord
      acBlkTblRec = acTrans.GetObject(acBlkTbl(BlockTableRecord.ModelSpace), _
                                      OpenMode.ForWrite)

      '' Create a circle object for the boundary of the hatch
      Dim acCirc As Circle = New Circle()
      acCirc.SetDatabaseDefaults()
      acCirc.Center = New Point3d(5, 3, 0)
      acCirc.Radius = 3

      acBlkTblRec.AppendEntity(acCirc)
      acTrans.AddNewlyCreatedDBObject(acCirc, True)

      '' Adds the arc and line to an object id collection
      Dim acObjIdColl As ObjectIdCollection = New ObjectIdCollection()
      acObjIdColl.Add(acCirc.ObjectId)

      '' Create the hatch object and append it to the block table record
      Dim acHatch As Hatch = New Hatch()
      acBlkTblRec.AppendEntity(acHatch)
      acTrans.AddNewlyCreatedDBObject(acHatch, True)

      '' Set the properties of the hatch object
      '' Associative must be set after the hatch object is appended to the
      '' block table record and before AppendLoop
      acHatch.SetDatabaseDefaults()
      acHatch.SetHatchPattern(HatchPatternType.PreDefined, "ANSI31")
      acHatch.Associative = True
      acHatch.AppendLoop(HatchLoopTypes.Outermost, acObjIdColl)

      '' Evaluate the hatch
      acHatch.EvaluateHatch(True)

      '' Increase the pattern scale by 2 and re-evaluate the hatch
      acHatch.PatternScale = acHatch.PatternScale + 2
      acHatch.SetHatchPattern(acHatch.PatternType, acHatch.PatternName)
      acHatch.EvaluateHatch(True)

      '' Save the new object to the database
      acTrans.Commit()
  End Using
End Sub
```

C#

```csharp
using Autodesk.AutoCAD.Runtime;
using Autodesk.AutoCAD.ApplicationServices;
using Autodesk.AutoCAD.DatabaseServices;
```

```csharp
using Autodesk.AutoCAD.Geometry;

[CommandMethod("EditHatchPatternScale")]
public static void EditHatchPatternScale()
{
  // Get the current document and database
  Document acDoc = Application.DocumentManager.MdiActiveDocument;
  Database acCurDb = acDoc.Database;

  // Start a transaction
  using (Transaction acTrans = acCurDb.TransactionManager.StartTransaction())
  {
      // Open the Block table for read
      BlockTable acBlkTbl;
      acBlkTbl = acTrans.GetObject(acCurDb.BlockTableId,
                                   OpenMode.ForRead) as BlockTable;

      // Open the Block table record Model space for write
      BlockTableRecord acBlkTblRec;
      acBlkTblRec = acTrans.GetObject(acBlkTbl[BlockTableRecord.ModelSpace],
                                      OpenMode.ForWrite) as BlockTableRecord;

      // Create a circle object for the boundary of the hatch
      Circle acCirc = new Circle();
      acCirc.SetDatabaseDefaults();
      acCirc.Center = new Point3d(5, 3, 0);
      acCirc.Radius = 3;

      acBlkTblRec.AppendEntity(acCirc);
      acTrans.AddNewlyCreatedDBObject(acCirc, true);

      // Adds the arc and line to an object id collection
      ObjectIdCollection acObjIdColl = new ObjectIdCollection();
      acObjIdColl.Add(acCirc.ObjectId);

      // Create the hatch object and append it to the block table record
      Hatch acHatch = new Hatch();
      acBlkTblRec.AppendEntity(acHatch);
      acTrans.AddNewlyCreatedDBObject(acHatch, true);

      // Set the properties of the hatch object
      // Associative must be set after the hatch object is appended to the
      // block table record and before AppendLoop
      acHatch.SetDatabaseDefaults();
      acHatch.SetHatchPattern(HatchPatternType.PreDefined, "ANSI31");
      acHatch.Associative = true;
      acHatch.AppendLoop(HatchLoopTypes.Outermost, acObjIdColl);

      // Evaluate the hatch
      acHatch.EvaluateHatch(true);

      // Increase the pattern scale by 2 and re-evaluate the hatch
      acHatch.PatternScale = acHatch.PatternScale + 2;
      acHatch.SetHatchPattern(acHatch.PatternType, acHatch.PatternName);
      acHatch.EvaluateHatch(true);

      // Save the new object to the database
      acTrans.Commit();
  }
}
```

```
Sub EditHatchPatternScale()
    Dim hatchObj As AcadHatch
    Dim patternName As String
    Dim PatternType As Long
    Dim bAssociativity As Boolean

    ' Define the hatch
    patternName = "ANSI31"
    PatternType = 0
    bAssociativity = True

    ' Create the associative Hatch object
    Set hatchObj = ThisDrawing.ModelSpace. _
                        AddHatch(PatternType, patternName, bAssociativity)

    ' Create the outer loop for the hatch.
    Dim outerLoop(0 To 0) As AcadEntity
    Dim center(0 To 2) As Double
    Dim radius As Double
    center(0) = 5
    center(1) = 3
    center(2) = 0
    radius = 3
    Set outerLoop(0) = ThisDrawing.ModelSpace. _
                            AddCircle(center, radius)
    hatchObj.AppendOuterLoop (outerLoop)
    hatchObj.Evaluate

    ' Change the scale of the hatch pattern by
    ' adding 2 to the current scale
    hatchObj.patternScale = hatchObj.patternScale + 2
    hatchObj.Evaluate
    ThisDrawing.Regen True
End Sub
```

Use Layers, Colors, and Linetypes

Layers are like transparent overlays on which you organize and group different kinds of drawing information. The objects you create have properties including layers, colors, and linetypes. Color helps you distinguish similar elements in your drawings, and linetypes help you differentiate easily between different drafting elements, such as centerlines or hidden lines. Organizing layers and objects on layers makes it easier to manage the information in your drawings.

For more information about this topic, see "Control the Properties of Objects" in the *AutoCAD User's Guide*.

Topics in this section

- [Work with Layers](#)
- [Work with Colors](#)
- [Work with Linetypes](#)

Work with Layers

You are always drawing on a layer. It may be the default layer or a layer you create and name yourself. Each layer has an associated color and linetype among other properties. For example, you can create a layer on which you draw only centerlines and assign the color blue and the linetype CENTER to that layer. Then, whenever you want to draw centerlines you can switch to that layer and start drawing.

All layers and linetypes are stored in separate symbol tables. Layers are kept within the Layers table, and linetypes are kept within the Linetypes table.

For more information about working with layers, see "Work with Layers" in the *AutoCAD User's Guide*.

Topics in this section

- Sort Layers and Linetypes
- Create and Name Layers
- Make a Layer Current
- Turn Layers On and Off
- Freeze and Thaw Layers
- Lock and Unlock Layers
- Assign Color to a Layer
- Assign a Linetype to a Layer
- Erase Layers

Sort Layers and Linetypes

You can iterate through the Layers and Linetypes tables to find all the layers and linetypes in a drawing.

Iterate through the Layers table

The following code iterates through the Layers table to gather the names of all the layers in the drawing. The names are then displayed in a message box.

VB.NET

```
Imports Autodesk.AutoCAD.Runtime
Imports Autodesk.AutoCAD.ApplicationServices
Imports Autodesk.AutoCAD.DatabaseServices

<CommandMethod("DisplayLayerNames")> _
Public Sub DisplayLayerNames()
  '' Get the current document and database
  Dim acDoc As Document = Application.DocumentManager.MdiActiveDocument
  Dim acCurDb As Database = acDoc.Database

  '' Start a transaction
  Using acTrans As Transaction = acCurDb.TransactionManager.StartTransaction()
```

```
        '' Open the Layer table for read
        Dim acLyrTbl As LayerTable
        acLyrTbl = acTrans.GetObject(acCurDb.LayerTableId, _
                                     OpenMode.ForRead)

        Dim sLayerNames As String = ""

        For Each acObjId As ObjectId In acLyrTbl
            Dim acLyrTblRec As LayerTableRecord
            acLyrTblRec = acTrans.GetObject(acObjId, _
                                            OpenMode.ForRead)

            sLayerNames = sLayerNames & vbLf & acLyrTblRec.Name
        Next

        Application.ShowAlertDialog("The layers in this drawing are: " & _
                                    sLayerNames)

        '' Dispose of the transaction
    End Using
End Sub
```

C#

```csharp
using Autodesk.AutoCAD.Runtime;
using Autodesk.AutoCAD.ApplicationServices;
using Autodesk.AutoCAD.DatabaseServices;

[CommandMethod("DisplayLayerNames")]
public static void DisplayLayerNames()
{
  // Get the current document and database
  Document acDoc = Application.DocumentManager.MdiActiveDocument;
  Database acCurDb = acDoc.Database;

  // Start a transaction
  using (Transaction acTrans = acCurDb.TransactionManager.StartTransaction())
  {
      // Open the Layer table for read
      LayerTable acLyrTbl;
      acLyrTbl = acTrans.GetObject(acCurDb.LayerTableId,
                                   OpenMode.ForRead) as LayerTable;

      string sLayerNames = "";

      foreach (ObjectId acObjId in acLyrTbl)
      {
          LayerTableRecord acLyrTblRec;
          acLyrTblRec = acTrans.GetObject(acObjId,
                                          OpenMode.ForRead) as LayerTableRecord;

          sLayerNames = sLayerNames + "\n" + acLyrTblRec.Name;
      }

      Application.ShowAlertDialog("The layers in this drawing are: " +
                                  sLayerNames);

      // Dispose of the transaction
  }
}
```

```
Sub DisplayLayerNames()
    Dim layerNames As String
    Dim entry As AcadLayer
    layerNames = ""

    For Each entry In ThisDrawing.Layers
        layerNames = layerNames + entry.Name + vbCrLf
    Next

    MsgBox "The layers in this drawing are: " + _
            vbCrLf + layerNames
End Sub
```

Create and Name Layers

You can create new layers and assign color and linetype properties to those layers. Each individual layer is part of the Layers table. Use the Add function to create a new layer and add it to the Layers table.

You can assign a name to a layer when it is created. To change the name of a layer after it has been created, use the Name property. Layer names can include up to 255 characters and contain letters, digits, and the special characters dollar sign ($), hyphen (-), and underscore (_).

For more information about creating layers, see "Create and Name Layers" in the *AutoCAD User's Guide*.

Create a new layer, assign it the color red, and add an object to the layer

The following code creates a new layer and circle object. The new layer is assigned the color red. The circle is assigned to the layer, and the color of the circle changes accordingly.

VB.NET

```
Imports Autodesk.AutoCAD.Runtime
Imports Autodesk.AutoCAD.ApplicationServices
Imports Autodesk.AutoCAD.DatabaseServices
Imports Autodesk.AutoCAD.Geometry
Imports Autodesk.AutoCAD.Colors

<CommandMethod("CreateAndAssignALayer")> _
Public Sub CreateAndAssignALayer()
  '' Get the current document and database
  Dim acDoc As Document = Application.DocumentManager.MdiActiveDocument
  Dim acCurDb As Database = acDoc.Database

  '' Start a transaction
  Using acTrans As Transaction = acCurDb.TransactionManager.StartTransaction()

      '' Open the Layer table for read
```

```vb
        Dim acLyrTbl As LayerTable
        acLyrTbl = acTrans.GetObject(acCurDb.LayerTableId, _
                                     OpenMode.ForRead)

        Dim sLayerName As String = "Center"

        If acLyrTbl.Has(sLayerName) = False Then
            Dim acLyrTblRec As LayerTableRecord = New LayerTableRecord()

            '' Assign the layer the ACI color 1 and a name
            acLyrTblRec.Color = Color.FromColorIndex(ColorMethod.ByAci, 1)
            acLyrTblRec.Name = sLayerName

            '' Upgrade the Layer table for write
            acLyrTbl.UpgradeOpen()

            '' Append the new layer to the Layer table and the transaction
            acLyrTbl.Add(acLyrTblRec)
            acTrans.AddNewlyCreatedDBObject(acLyrTblRec, True)
        End If

        '' Open the Block table for read
        Dim acBlkTbl As BlockTable
        acBlkTbl = acTrans.GetObject(acCurDb.BlockTableId, _
                                     OpenMode.ForRead)

        '' Open the Block table record Model space for write
        Dim acBlkTblRec As BlockTableRecord
        acBlkTblRec = acTrans.GetObject(acBlkTbl(BlockTableRecord.ModelSpace), _
                                        OpenMode.ForWrite)

        '' Create a circle object
        Dim acCirc As Circle = New Circle()
        acCirc.SetDatabaseDefaults()
        acCirc.Center = New Point3d(2, 2, 0)
        acCirc.Radius = 1
        acCirc.Layer = sLayerName

        acBlkTblRec.AppendEntity(acCirc)
        acTrans.AddNewlyCreatedDBObject(acCirc, True)

        '' Save the changes and dispose of the transaction
        acTrans.Commit()
    End Using
End Sub
```

C#

```csharp
using Autodesk.AutoCAD.Runtime;
using Autodesk.AutoCAD.ApplicationServices;
using Autodesk.AutoCAD.DatabaseServices;
using Autodesk.AutoCAD.Geometry;
using Autodesk.AutoCAD.Colors;

[CommandMethod("CreateAndAssignALayer")]
public static void CreateAndAssignALayer()
{
    // Get the current document and database
    Document acDoc = Application.DocumentManager.MdiActiveDocument;
    Database acCurDb = acDoc.Database;

    // Start a transaction
    using (Transaction acTrans = acCurDb.TransactionManager.StartTransaction())
    {
```

```
    // Open the Layer table for read
    LayerTable acLyrTbl;
    acLyrTbl = acTrans.GetObject(acCurDb.LayerTableId,
                                 OpenMode.ForRead) as LayerTable;

    string sLayerName = "Center";

    if (acLyrTbl.Has(sLayerName) == false)
    {
        LayerTableRecord acLyrTblRec = new LayerTableRecord();

        // Assign the layer the ACI color 1 and a name
        acLyrTblRec.Color = Color.FromColorIndex(ColorMethod.ByAci, 1);
        acLyrTblRec.Name = sLayerName;

        // Upgrade the Layer table for write
        acLyrTbl.UpgradeOpen();

        // Append the new layer to the Layer table and the transaction
        acLyrTbl.Add(acLyrTblRec);
        acTrans.AddNewlyCreatedDBObject(acLyrTblRec, true);
    }

    // Open the Block table for read
    BlockTable acBlkTbl;
    acBlkTbl = acTrans.GetObject(acCurDb.BlockTableId,
                                 OpenMode.ForRead) as BlockTable;

    // Open the Block table record Model space for write
    BlockTableRecord acBlkTblRec;
    acBlkTblRec = acTrans.GetObject(acBlkTbl[BlockTableRecord.ModelSpace],
                                    OpenMode.ForWrite) as BlockTableRecord;

    // Create a circle object
    Circle acCirc = new Circle();
    acCirc.SetDatabaseDefaults();
    acCirc.Center = new Point3d(2, 2, 0);
    acCirc.Radius = 1;
    acCirc.Layer = sLayerName;

    acBlkTblRec.AppendEntity(acCirc);
    acTrans.AddNewlyCreatedDBObject(acCirc, true);

    // Save the changes and dispose of the transaction
    acTrans.Commit();
  }
}
```

⊟ *VBA/ActiveX Code Reference*

```
Sub CreateAssignALayer()
    ' Create a new layer and assign it the color red
    Dim layerObj As AcadLayer
    Set layerObj = ThisDrawing.Layers.Add("Center")
    layerObj.color = acRed

    ' Create a circle
    Dim circleObj As AcadCircle
    Dim center(0 To 2) As Double
    Dim radius As Double
    center(0) = 2: center(1) = 2: center(2) = 0
    radius = 1
    Set circleObj = ThisDrawing.ModelSpace. _
```

```
                    AddCircle(center, radius)

    ' Place the circle on the Center layer
    circleObj.Layer = "Center"

    circleObj.Update
End Sub
```

Make a Layer Current

You are always drawing on the active layer. When you make a layer active, you create new objects on that layer. If you make a different layer active, any new objects you create is assigned that new active layer and uses its color and linetype. You cannot make a layer active if it is frozen.

To make a layer active, use the Clayer property of the Database object or the CLAYER system variable. For example:

Make a layer current through the database

This example sets a layer current through the Database object with the Clayer property.

VB.NET

```
Imports Autodesk.AutoCAD.Runtime
Imports Autodesk.AutoCAD.ApplicationServices
Imports Autodesk.AutoCAD.DatabaseServices

<CommandMethod("SetLayerCurrent")> _
Public Sub SetLayerCurrent()
  '' Get the current document and database
  Dim acDoc As Document = Application.DocumentManager.MdiActiveDocument
  Dim acCurDb As Database = acDoc.Database

  '' Start a transaction
  Using acTrans As Transaction = acCurDb.TransactionManager.StartTransaction()

      '' Open the Layer table for read
      Dim acLyrTbl As LayerTable
      acLyrTbl = acTrans.GetObject(acCurDb.LayerTableId, _
                                   OpenMode.ForRead)

      Dim sLayerName As String = "Center"

      If acLyrTbl.Has(sLayerName) = True Then
          '' Set the layer Center current
          acCurDb.Clayer = acLyrTbl(sLayerName)

          '' Save the changes
          acTrans.Commit()
      End If

      '' Dispose of the transaction
  End Using
End Sub
```

C#

```
using Autodesk.AutoCAD.Runtime;
using Autodesk.AutoCAD.ApplicationServices;
using Autodesk.AutoCAD.DatabaseServices;

[CommandMethod("SetLayerCurrent")]
public static void SetLayerCurrent()
{
  // Get the current document and database
  Document acDoc = Application.DocumentManager.MdiActiveDocument;
  Database acCurDb = acDoc.Database;

  // Start a transaction
  using (Transaction acTrans = acCurDb.TransactionManager.StartTransaction())
  {
      // Open the Layer table for read
      LayerTable acLyrTbl;
      acLyrTbl = acTrans.GetObject(acCurDb.LayerTableId,
                                   OpenMode.ForRead) as LayerTable;

      string sLayerName = "Center";

      if (acLyrTbl.Has(sLayerName) == true)
      {
          // Set the layer Center current
          acCurDb.Clayer = acLyrTbl[sLayerName];

          // Save the changes
          acTrans.Commit();
      }

      // Dispose of the transaction
  }
}
```

▢ *VBA/ActiveX Code Reference*

```
ThisDrawing.ActiveLayer = ThisDrawing.Layers("Center")
```

Make a layer current with the CLAYER system variable

This example sets a layer current with the CLAYER system variable.

VB.NET

```
Application.SetSystemVariable("CLAYER", "Center")
```

C#

```
Application.SetSystemVariable("CLAYER", "Center");
```

▢ *VBA/ActiveX Code Reference*

```
ThisDrawing.SetVariable "CLAYER", "Center"
```

Turn Layers On and Off

Layers in which are turned off are regenerated with the drawing but are not displayed or plotted. By turning layers off, you avoid regenerating the drawing every time you thaw a layer. When you turn a layer on that has been turned off, AutoCAD redraws the objects on that layer.

Use the IsOff property on the Layer Table Record object that represents the layer you want to turn on or off. If you input a value of TRUE, the layer is turned off. If you input a value of FALSE, the layer is turned on.

Turn off a layer

This example creates a new layer and turns it off, and then adds a circle to the layer so that the circle is no longer visible.

VB.NET

```
Imports Autodesk.AutoCAD.Runtime
Imports Autodesk.AutoCAD.ApplicationServices
Imports Autodesk.AutoCAD.DatabaseServices
Imports Autodesk.AutoCAD.Geometry

<CommandMethod("TurnLayerOff")> _
Public Sub TurnLayerOff()
  '' Get the current document and database
  Dim acDoc As Document = Application.DocumentManager.MdiActiveDocument
  Dim acCurDb As Database = acDoc.Database

  '' Start a transaction
  Using acTrans As Transaction = acCurDb.TransactionManager.StartTransaction()

      '' Open the Layer table for read
      Dim acLyrTbl As LayerTable
      acLyrTbl = acTrans.GetObject(acCurDb.LayerTableId, _
                                   OpenMode.ForRead)

      Dim sLayerName As String = "ABC"
      Dim acLyrTblRec As LayerTableRecord

      If acLyrTbl.Has(sLayerName) = False Then
          acLyrTblRec = New LayerTableRecord()

          '' Assign the layer a name
          acLyrTblRec.Name = sLayerName

          '' Upgrade the Layer table for write
          acLyrTbl.UpgradeOpen()

          '' Append the new layer to the Layer table and the transaction
          acLyrTbl.Add(acLyrTblRec)
          acTrans.AddNewlyCreatedDBObject(acLyrTblRec, True)
      Else
          acLyrTblRec = acTrans.GetObject(acLyrTbl(sLayerName), _
                                          OpenMode.ForWrite)
      End If
```

```vb
        '' Turn the layer off
        acLyrTblRec.IsOff = True

        '' Open the Block table for read
        Dim acBlkTbl As BlockTable
        acBlkTbl = acTrans.GetObject(acCurDb.BlockTableId, _
                                     OpenMode.ForRead)

        '' Open the Block table record Model space for write
        Dim acBlkTblRec As BlockTableRecord
        acBlkTblRec = acTrans.GetObject(acBlkTbl(BlockTableRecord.ModelSpace), _
                                        OpenMode.ForWrite)

        '' Create a circle object
        Dim acCirc As Circle = New Circle()
        acCirc.SetDatabaseDefaults()
        acCirc.Center = New Point3d(2, 2, 0)
        acCirc.Radius = 1
        acCirc.Layer = sLayerName

        acBlkTblRec.AppendEntity(acCirc)
        acTrans.AddNewlyCreatedDBObject(acCirc, True)

        '' Save the changes and dispose of the transaction
        acTrans.Commit()
    End Using
End Sub
```

C#

```csharp
using Autodesk.AutoCAD.Runtime;
using Autodesk.AutoCAD.ApplicationServices;
using Autodesk.AutoCAD.DatabaseServices;
using Autodesk.AutoCAD.Geometry;

[CommandMethod("TurnLayerOff")]
public static void TurnLayerOff()
{
    // Get the current document and database
    Document acDoc = Application.DocumentManager.MdiActiveDocument;
    Database acCurDb = acDoc.Database;

    // Start a transaction
    using (Transaction acTrans = acCurDb.TransactionManager.StartTransaction())
    {
        // Open the Layer table for read
        LayerTable acLyrTbl;
        acLyrTbl = acTrans.GetObject(acCurDb.LayerTableId,
                                     OpenMode.ForRead) as LayerTable;

        string sLayerName = "ABC";
        LayerTableRecord acLyrTblRec;

        if (acLyrTbl.Has(sLayerName) == false)
        {
            acLyrTblRec = new LayerTableRecord();

            // Assign the layer a name
            acLyrTblRec.Name = sLayerName;

            // Upgrade the Layer table for write
            acLyrTbl.UpgradeOpen();

            // Append the new layer to the Layer table and the transaction
```

```
            acLyrTbl.Add(acLyrTblRec);
            acTrans.AddNewlyCreatedDBObject(acLyrTblRec, true);
        }
        else
        {
            acLyrTblRec = acTrans.GetObject(acLyrTbl[sLayerName],
                                            OpenMode.ForWrite) as LayerTableRecord;
        }

        // Turn the layer off
        acLyrTblRec.IsOff = true;

        // Open the Block table for read
        BlockTable acBlkTbl;
        acBlkTbl = acTrans.GetObject(acCurDb.BlockTableId,
                                     OpenMode.ForRead) as BlockTable;

        // Open the Block table record Model space for write
        BlockTableRecord acBlkTblRec;
        acBlkTblRec = acTrans.GetObject(acBlkTbl[BlockTableRecord.ModelSpace],
                                        OpenMode.ForWrite) as BlockTableRecord;

        // Create a circle object
        Circle acCirc = new Circle();
        acCirc.SetDatabaseDefaults();
        acCirc.Center = new Point3d(2, 2, 0);
        acCirc.Radius = 1;
        acCirc.Layer = sLayerName;

        acBlkTblRec.AppendEntity(acCirc);
        acTrans.AddNewlyCreatedDBObject(acCirc, true);

        // Save the changes and dispose of the transaction
        acTrans.Commit();
    }
}
```

VBA/ActiveX Code Reference

```
Sub TurnLayerOff()
    ' Create a new layer called "ABC"
    Dim layerObj As AcadLayer
    Set layerObj = ThisDrawing.Layers.Add("ABC")

    ' Turn off layer "ABC"
    layerObj.LayerOn = False

    ' Create a circle
    Dim circleObj As AcadCircle
    Dim center(0 To 2) As Double
    Dim radius As Double
    center(0) = 2: center(1) = 2: center(2) = 0
    radius = 1
    Set circleObj = ThisDrawing.ModelSpace. _
                                AddCircle(center, radius)

    ' Assign the circle to the "ABC" layer
    circleObj.Layer = "ABC"
    circleObj.Update

    ThisDrawing.Regen acActiveViewport
End Sub
```

Freeze and Thaw Layers

You can freeze layers to speed up display changes, improve object selection performance, and reduce regeneration time for complex drawings. AutoCAD does not display, plot, or regenerate objects on frozen layers. Freeze the layers that you will not be working with for long periods of time. When you "thaw" a frozen layer, AutoCAD regenerates and displays the objects on that layer.

Use the IsFrozen property to freeze or thaw a layer. If you input a value of TRUE, the layer is frozen. If you input a value of FALSE, the layer is thawed.

Freeze a layer

This example creates a new layer called "ABC" and then freezes the layer.

VB.NET

```vbnet
Imports Autodesk.AutoCAD.Runtime
Imports Autodesk.AutoCAD.ApplicationServices
Imports Autodesk.AutoCAD.DatabaseServices

<CommandMethod("FreezeLayer")> _
Public Sub FreezeLayer()
  '' Get the current document and database
  Dim acDoc As Document = Application.DocumentManager.MdiActiveDocument
  Dim acCurDb As Database = acDoc.Database

  '' Start a transaction
  Using acTrans As Transaction = acCurDb.TransactionManager.StartTransaction()

      '' Open the Layer table for read
      Dim acLyrTbl As LayerTable
      acLyrTbl = acTrans.GetObject(acCurDb.LayerTableId, _
                                   OpenMode.ForRead)

      Dim sLayerName As String = "ABC"
      Dim acLyrTblRec As LayerTableRecord

      If acLyrTbl.Has(sLayerName) = False Then
          acLyrTblRec = New LayerTableRecord()

          '' Assign the layer a name
          acLyrTblRec.Name = sLayerName

          '' Upgrade the Layer table for write
          acLyrTbl.UpgradeOpen()

          '' Append the new layer to the Layer table and the transaction
          acLyrTbl.Add(acLyrTblRec)
          acTrans.AddNewlyCreatedDBObject(acLyrTblRec, True)
      Else
          acLyrTblRec = acTrans.GetObject(acLyrTbl(sLayerName), _
                                          OpenMode.ForWrite)
      End If

      '' Freeze the layer
      acLyrTblRec.IsFrozen = True
```

```
        '' Save the changes and dispose of the transaction
        acTrans.Commit()
    End Using
End Sub
```

C#

```csharp
using Autodesk.AutoCAD.Runtime;
using Autodesk.AutoCAD.ApplicationServices;
using Autodesk.AutoCAD.DatabaseServices;

[CommandMethod("FreezeLayer")]
public static void FreezeLayer()
{
    // Get the current document and database
    Document acDoc = Application.DocumentManager.MdiActiveDocument;
    Database acCurDb = acDoc.Database;

    // Start a transaction
    using (Transaction acTrans = acCurDb.TransactionManager.StartTransaction())
    {
        // Open the Layer table for read
        LayerTable acLyrTbl;
        acLyrTbl = acTrans.GetObject(acCurDb.LayerTableId,
                                     OpenMode.ForRead) as LayerTable;

        string sLayerName = "ABC";
        LayerTableRecord acLyrTblRec;

        if (acLyrTbl.Has(sLayerName) == false)
        {
            acLyrTblRec = new LayerTableRecord();

            // Assign the layer a name
            acLyrTblRec.Name = sLayerName;

            // Upgrade the Layer table for write
            acLyrTbl.UpgradeOpen();

            // Append the new layer to the Layer table and the transaction
            acLyrTbl.Add(acLyrTblRec);
            acTrans.AddNewlyCreatedDBObject(acLyrTblRec, true);
        }
        else
        {
            acLyrTblRec = acTrans.GetObject(acLyrTbl[sLayerName],
                                            OpenMode.ForWrite) as LayerTableRecord;
        }

        // Freeze the layer
        acLyrTblRec.IsFrozen = true;

        // Save the changes and dispose of the transaction
        acTrans.Commit();
    }
}
```

```
Sub FreezeLayer()
    ' Create a new layer called "ABC"
    Dim layerObj As AcadLayer
    Set layerObj = ThisDrawing.Layers.Add("ABC")

    ' Freeze layer "ABC"
    layerObj.Freeze = True
End Sub
```

Lock and Unlock Layers

You cannot edit objects on a locked layer; however, they are still visible if the layer is on and thawed. You can make a locked layer current and you can add objects to it. You can freeze and turn off locked layers and change their associated colors and linetypes.

Use the IsLocked property to lock or unlock a layer. If you input a value of TRUE, the layer is locked. If you input a value of FALSE, the layer is unlocked.

Lock a layer

This example creates a new layer called "ABC" and then locks the layer.

VB.NET

```
Imports Autodesk.AutoCAD.Runtime
Imports Autodesk.AutoCAD.ApplicationServices
Imports Autodesk.AutoCAD.DatabaseServices

<CommandMethod("LockLayer")> _
Public Sub LockLayer()
  '' Get the current document and database
  Dim acDoc As Document = Application.DocumentManager.MdiActiveDocument
  Dim acCurDb As Database = acDoc.Database

  '' Start a transaction
  Using acTrans As Transaction = acCurDb.TransactionManager.StartTransaction()

      '' Open the Layer table for read
      Dim acLyrTbl As LayerTable
      acLyrTbl = acTrans.GetObject(acCurDb.LayerTableId, _
                                   OpenMode.ForRead)

      Dim sLayerName As String = "ABC"
      Dim acLyrTblRec As LayerTableRecord

      If acLyrTbl.Has(sLayerName) = False Then
          acLyrTblRec = New LayerTableRecord()

          '' Assign the layer a name
          acLyrTblRec.Name = sLayerName
```

```vb
        '' Upgrade the Layer table for write
        acLyrTbl.UpgradeOpen()

        '' Append the new layer to the Layer table and the transaction
        acLyrTbl.Add(acLyrTblRec)
        acTrans.AddNewlyCreatedDBObject(acLyrTblRec, True)
    Else
        acLyrTblRec = acTrans.GetObject(acLyrTbl(sLayerName), _
                                        OpenMode.ForWrite)
    End If

    '' Lock the layer
    acLyrTblRec.IsLocked = True

    '' Save the changes and dispose of the transaction
    acTrans.Commit()
  End Using
End Sub
```

C#

```csharp
using Autodesk.AutoCAD.Runtime;
using Autodesk.AutoCAD.ApplicationServices;
using Autodesk.AutoCAD.DatabaseServices;

[CommandMethod("LockLayer")]
public static void LockLayer()
{
  // Get the current document and database
  Document acDoc = Application.DocumentManager.MdiActiveDocument;
  Database acCurDb = acDoc.Database;

  // Start a transaction
  using (Transaction acTrans = acCurDb.TransactionManager.StartTransaction())
  {
      // Open the Layer table for read
      LayerTable acLyrTbl;
      acLyrTbl = acTrans.GetObject(acCurDb.LayerTableId,
                                   OpenMode.ForRead) as LayerTable;

      string sLayerName = "ABC";
      LayerTableRecord acLyrTblRec;

      if (acLyrTbl.Has(sLayerName) == false)
      {
          acLyrTblRec = new LayerTableRecord();

          // Assign the layer a name
          acLyrTblRec.Name = sLayerName;

          // Upgrade the Layer table for write
          acLyrTbl.UpgradeOpen();

          // Append the new layer to the Layer table and the transaction
          acLyrTbl.Add(acLyrTblRec);
          acTrans.AddNewlyCreatedDBObject(acLyrTblRec, true);
      }
      else
      {
          acLyrTblRec = acTrans.GetObject(acLyrTbl[sLayerName],
                                          OpenMode.ForWrite) as LayerTableRecord;
      }

      // Lock the layer
```

```
        acLyrTblRec.IsLocked = true;

        // Save the changes and dispose of the transaction
        acTrans.Commit();
    }
}
```

⊟ *VBA/ActiveX Code Reference*

```
Sub LockLayer()
    ' Create a new layer called "ABC"
    Dim layerObj As AcadLayer
    Set layerObj = ThisDrawing.Layers.Add("ABC")

    ' Lock layer "ABC"
    layerObj.Lock = True
End Sub
```

Assign Color to a Layer

Each layer can have its own color assigned to it. Colors for a layer are identified by the Color object which is part of the Colors namespace. This object can hold an RGB value, ACI number (an integer from 1 to 255), or a color book color.

To assign a color to a layer, use the Color property.

NoteObjects such as lines and circles support two different properties to control their current color. The Color property is used to assign an RGB value, ACI number, or a color book color, while the ColorIndex property only supports ACI numbers.

If you use the ACI color 0 or ByBlock, AutoCAD draws new objects in the default color (white or black, depending on your configuration) until they are grouped into a block. When the block is inserted, the objects in the block inherit the current property setting.

If you use the ACI color 256 or ByLayer, new objects inherit the color of the layer upon which they are drawn.

Set the color of a layer

The following example creates three new layers and each is assigned a different color using each of the three color methods.

VB.NET

```
Imports Autodesk.AutoCAD.Runtime
Imports Autodesk.AutoCAD.ApplicationServices
Imports Autodesk.AutoCAD.DatabaseServices
Imports Autodesk.AutoCAD.Colors

<CommandMethod("SetLayerColor")> _
Public Sub SetLayerColor()
```

```vb
    '' Get the current document and database
    Dim acDoc As Document = Application.DocumentManager.MdiActiveDocument
    Dim acCurDb As Database = acDoc.Database

    '' Start a transaction
    Using acTrans As Transaction = acCurDb.TransactionManager.StartTransaction()

        '' Open the Layer table for read
        Dim acLyrTbl As LayerTable
        acLyrTbl = acTrans.GetObject(acCurDb.LayerTableId, _
                                     OpenMode.ForRead)

        '' Define an array of layer names
        Dim sLayerNames(2) As String
        sLayerNames(0) = "ACIRed"
        sLayerNames(1) = "TrueBlue"
        sLayerNames(2) = "ColorBookYellow"

        '' Define an array of colors for the layers
        Dim acColors(2) As Color
        acColors(0) = Color.FromColorIndex(ColorMethod.ByAci, 1)
        acColors(1) = Color.FromRgb(23, 54, 232)
        acColors(2) = Color.FromNames("PANTONE Yellow 0131 C", _
                                      "PANTONE(R) pastel coated")

        Dim nCnt As Integer = 0

        '' Add or change each layer in the drawing
        For Each sLayerName As String In sLayerNames
            Dim acLyrTblRec As LayerTableRecord

            If acLyrTbl.Has(sLayerName) = False Then
                acLyrTblRec = New LayerTableRecord()

                '' Assign the layer a name
                acLyrTblRec.Name = sLayerName

                '' Upgrade the Layer table for write
                If acLyrTbl.IsWriteEnabled = False Then acLyrTbl.UpgradeOpen()

                '' Append the new layer to the Layer table and the transaction
                acLyrTbl.Add(acLyrTblRec)
                acTrans.AddNewlyCreatedDBObject(acLyrTblRec, True)
            Else
                '' Open the layer if it already exists for write
                acLyrTblRec = acTrans.GetObject(acLyrTbl(sLayerName), _
                                                OpenMode.ForWrite)
            End If

            '' Set the color of the layer
            acLyrTblRec.Color = acColors(nCnt)

            nCnt = nCnt + 1
        Next

        '' Save the changes and dispose of the transaction
        acTrans.Commit()
    End Using
End Sub
```

C#

```
using Autodesk.AutoCAD.Runtime;
using Autodesk.AutoCAD.ApplicationServices;
using Autodesk.AutoCAD.DatabaseServices;
using Autodesk.AutoCAD.Colors;

[CommandMethod("SetLayerColor")]
public static void SetLayerColor()
{
  // Get the current document and database
  Document acDoc = Application.DocumentManager.MdiActiveDocument;
  Database acCurDb = acDoc.Database;

  // Start a transaction
  using (Transaction acTrans = acCurDb.TransactionManager.StartTransaction())
  {
      // Open the Layer table for read
      LayerTable acLyrTbl;
      acLyrTbl = acTrans.GetObject(acCurDb.LayerTableId,
                                   OpenMode.ForRead) as LayerTable;

      // Define an array of layer names
      string[] sLayerNames = new string[3];
      sLayerNames[0] = "ACIRed";
      sLayerNames[1] = "TrueBlue";
      sLayerNames[2] = "ColorBookYellow";

      // Define an array of colors for the layers
      Color[] acColors = new Color[3];
      acColors[0] = Color.FromColorIndex(ColorMethod.ByAci, 1);
      acColors[1] = Color.FromRgb(23, 54, 232);
      acColors[2] = Color.FromNames("PANTONE Yellow 0131 C",
                                    "PANTONE(R) pastel coated");

      int nCnt = 0;

      // Add or change each layer in the drawing
      foreach (string sLayerName in sLayerNames)
      {
          LayerTableRecord acLyrTblRec;

          if (acLyrTbl.Has(sLayerName) == false)
          {
              acLyrTblRec = new LayerTableRecord();

              // Assign the layer a name
              acLyrTblRec.Name = sLayerName;

              // Upgrade the Layer table for write
              if (acLyrTbl.IsWriteEnabled == false) acLyrTbl.UpgradeOpen();

              // Append the new layer to the Layer table and the transaction
              acLyrTbl.Add(acLyrTblRec);
              acTrans.AddNewlyCreatedDBObject(acLyrTblRec, true);
          }
          else
          {
              // Open the layer if it already exists for write
              acLyrTblRec = acTrans.GetObject(acLyrTbl[sLayerName],
                                              OpenMode.ForWrite) as
LayerTableRecord;
          }
```

```
            // Set the color of the layer
            acLyrTblRec.Color = acColors[nCnt];

            nCnt = nCnt + 1;
        }

        // Save the changes and dispose of the transaction
        acTrans.Commit();
    }
}
```

⊟ *VBA/ActiveX Code Reference*

```
Sub SetLayerColor()
    Dim layerObj As AcadLayer

    ' Define an array of layer names
    Dim sLayerNames(2) As String
    sLayerNames(0) = "ACIRed"
    sLayerNames(1) = "TrueBlue"
    sLayerNames(2) = "ColorBookYelow"

    ' Define an array of colors
    Dim colorObj(2) As New AcadAcCmColor

    colorObj(0).ColorMethod = acColorMethodByACI
    colorObj(0).ColorIndex = acRed
    Call colorObj(1).SetRGB(23, 54, 232)
    Call colorObj(2).SetColorBookColor("PANTONE(R) pastel coated", _
                                        "PANTONE Yellow 0131 C")

    Dim nCnt As Integer

    ' Step through each layer name and create a new layer
    For Each sLayerName In sLayerNames
        Set layerObj = ThisDrawing.Layers.Add(sLayerName)
        layerObj.TrueColor = colorObj(nCnt)

        nCnt = nCnt + 1
    Next
End Sub
```

Assign a Linetype to a Layer

When you are defining layers, linetypes provide another way to convey visual information. A linetype is a repeating pattern of dashes, dots, and blank spaces you can use to distinguish the purpose of one line from another.

The linetype name and definition describe the particular dash-dot sequence, the relative lengths of dashes and blank spaces, and the characteristics of any included text or shapes.

Use the Linetype property to assign a linetype to a layer. This property takes the name of the linetype as input.

NoteBefore a linetype can be assigned to a layer it must be defined in the drawing first. For information on working with linetypes, see [Work with Linetypes](#).

Set the linetype for a layer

The following example creates a new layer named "ABC" and assigns it the "Center" linetype.

VB.NET

```
Imports Autodesk.AutoCAD.Runtime
Imports Autodesk.AutoCAD.ApplicationServices
Imports Autodesk.AutoCAD.DatabaseServices

<CommandMethod("SetLayerLinetype")> _
Public Sub SetLayerLinetype()
  '' Get the current document and database
  Dim acDoc As Document = Application.DocumentManager.MdiActiveDocument
  Dim acCurDb As Database = acDoc.Database

  '' Start a transaction
  Using acTrans As Transaction = acCurDb.TransactionManager.StartTransaction()

      '' Open the Layer table for read
      Dim acLyrTbl As LayerTable
      acLyrTbl = acTrans.GetObject(acCurDb.LayerTableId, _
                                   OpenMode.ForRead)

      Dim sLayerName As String = "ABC"
      Dim acLyrTblRec As LayerTableRecord

      If acLyrTbl.Has(sLayerName) = False Then
          acLyrTblRec = New LayerTableRecord()

          '' Assign the layer a name
          acLyrTblRec.Name = sLayerName

          '' Upgrade the Layer table for write
          acLyrTbl.UpgradeOpen()

          '' Append the new layer to the Layer table and the transaction
          acLyrTbl.Add(acLyrTblRec)
          acTrans.AddNewlyCreatedDBObject(acLyrTblRec, True)
      Else
          acLyrTblRec = acTrans.GetObject(acLyrTbl(sLayerName), _
                                          OpenMode.ForRead)
      End If

      '' Open the Layer table for read
      Dim acLinTbl As LinetypeTable
      acLinTbl = acTrans.GetObject(acCurDb.LinetypeTableId, _
                                   OpenMode.ForRead)

      If acLinTbl.Has("Center") = True Then
          '' Upgrade the Layer Table Record for write
          acLyrTblRec.UpgradeOpen()

          '' Set the linetype for the layer
          acLyrTblRec.LinetypeObjectId = acLinTbl("Center")
      End If

      '' Save the changes and dispose of the transaction
      acTrans.Commit()
```

```
   End Using
End Sub
```

C#

```csharp
using Autodesk.AutoCAD.Runtime;
using Autodesk.AutoCAD.ApplicationServices;
using Autodesk.AutoCAD.DatabaseServices;

[CommandMethod("SetLayerLinetype")]
public static void SetLayerLinetype()
{
  // Get the current document and database
  Document acDoc = Application.DocumentManager.MdiActiveDocument;
  Database acCurDb = acDoc.Database;

  // Start a transaction
  using (Transaction acTrans = acCurDb.TransactionManager.StartTransaction())
  {
      // Open the Layer table for read
      LayerTable acLyrTbl;
      acLyrTbl = acTrans.GetObject(acCurDb.LayerTableId,
                                   OpenMode.ForRead) as LayerTable;

      string sLayerName = "ABC";
      LayerTableRecord acLyrTblRec;

      if (acLyrTbl.Has(sLayerName) == false)
      {
          acLyrTblRec = new LayerTableRecord();

          // Assign the layer a name
          acLyrTblRec.Name = sLayerName;

          // Upgrade the Layer table for write
          acLyrTbl.UpgradeOpen();

          // Append the new layer to the Layer table and the transaction
          acLyrTbl.Add(acLyrTblRec);
          acTrans.AddNewlyCreatedDBObject(acLyrTblRec, true);
      }
      else
      {
          acLyrTblRec = acTrans.GetObject(acLyrTbl[sLayerName],
                                          OpenMode.ForRead) as LayerTableRecord;
      }

      // Open the Layer table for read
      LinetypeTable acLinTbl;
      acLinTbl = acTrans.GetObject(acCurDb.LinetypeTableId,
                                   OpenMode.ForRead) as LinetypeTable;

      if (acLinTbl.Has("Center") == true)
      {
          // Upgrade the Layer Table Record for write
          acLyrTblRec.UpgradeOpen();

          // Set the linetype for the layer
          acLyrTblRec.LinetypeObjectId = acLinTbl["Center"];
      }

      // Save the changes and dispose of the transaction
      acTrans.Commit();
  }
```

```
}
```

```
Sub SetLayerLinetype()
    On Error Resume Next
    Dim layerObj As AcadLayer

    Set layerObj = ThisDrawing.Layers.Add("ABC")
    layerObj.Linetype = "Center"
End Sub
```

Erase Layers

You can erase a layer at any time during a drawing session. You cannot erase the current layer, layer 0, an xref-dependent layer, or a layer that contains objects.

To erase a layer, use the Erase method. It is recommended to use the Purge function to verify that the layer can be purged, along with verifying that it is not layer 0, Defpoints, or the current layer.

NoteLayers referenced by block definitions, along with the special layer named DEFPOINTS, cannot be deleted even if they do not contain visible objects.

VB.NET

```
Imports Autodesk.AutoCAD.Runtime
Imports Autodesk.AutoCAD.ApplicationServices
Imports Autodesk.AutoCAD.DatabaseServices

<CommandMethod("EraseLayer")> _
Public Sub EraseLayer()
  '' Get the current document and database
  Dim acDoc As Document = Application.DocumentManager.MdiActiveDocument
  Dim acCurDb As Database = acDoc.Database

  '' Start a transaction
  Using acTrans As Transaction = acCurDb.TransactionManager.StartTransaction()

      '' Open the Layer table for read
      Dim acLyrTbl As LayerTable
      acLyrTbl = acTrans.GetObject(acCurDb.LayerTableId, _
                                   OpenMode.ForRead)

      Dim sLayerName As String = "ABC"

      If acLyrTbl.Has(sLayerName) = True Then
          '' Check to see if it is safe to erase layer
          Dim acObjIdColl As ObjectIdCollection = New ObjectIdCollection()
          acObjIdColl.Add(acLyrTbl(sLayerName))
          acCurDb.Purge(acObjIdColl)

          If acObjIdColl.Count > 0 Then
              Dim acLyrTblRec As LayerTableRecord
```

```
                acLyrTblRec = acTrans.GetObject(acObjIdColl(0), OpenMode.ForWrite)

                Try
                    '' Erase the unreferenced layer
                    acLyrTblRec.Erase(True)

                    '' Save the changes and dispose of the transaction
                    acTrans.Commit()
                Catch Ex As Autodesk.AutoCAD.Runtime.Exception
                    '' Layer could not be deleted
                    Application.ShowAlertDialog("Error:\n" + Ex.Message)
                End Try
            End If
        End If
    End Using
End Sub
```

C#

```csharp
using Autodesk.AutoCAD.Runtime;
using Autodesk.AutoCAD.ApplicationServices;
using Autodesk.AutoCAD.DatabaseServices;

[CommandMethod("EraseLayer")]
public static void EraseLayer()
{
    // Get the current document and database
    Document acDoc = Application.DocumentManager.MdiActiveDocument;
    Database acCurDb = acDoc.Database;

    // Start a transaction
    using (Transaction acTrans = acCurDb.TransactionManager.StartTransaction())
    {
        // Open the Layer table for read
        LayerTable acLyrTbl;
        acLyrTbl = acTrans.GetObject(acCurDb.LayerTableId,
                                     OpenMode.ForRead) as LayerTable;

        string sLayerName = "ABC";

        if (acLyrTbl.Has(sLayerName) == true)
        {
            // Check to see if it is safe to erase layer
            ObjectIdCollection acObjIdColl = new ObjectIdCollection();
            acObjIdColl.Add(acLyrTbl[sLayerName]);
            acCurDb.Purge(acObjIdColl);

            if (acObjIdColl.Count > 0)
            {
                LayerTableRecord acLyrTblRec;
                acLyrTblRec = acTrans.GetObject(acObjIdColl[0],
                                                OpenMode.ForWrite) as
LayerTableRecord;

                try
                {
                    // Erase the unreferenced layer
                    acLyrTblRec.Erase(true);

                    // Save the changes and dispose of the transaction
                    acTrans.Commit();
                }
                catch (Autodesk.AutoCAD.Runtime.Exception Ex)
                {
```

```
                // Layer could not be deleted
                Application.ShowAlertDialog("Error:\n" + Ex.Message);
            }
        }
    }
  }
}
```

⊟ *VBA/ActiveX Code Reference*

```
Sub EraseLayer()
    On Error Resume Next

    Dim layerObj As AcadLayer
    Set layerObj = ThisDrawing.Layers("ABC")

    layerObj.Delete
End Sub
```

Work with Colors

You can assign a color to an individual object in a drawing using its Color or ColorIndex property. The ColorIndex property accepts an AutoCAD Color Index (ACI) value as a numeric value of 0 - 256. The Color property is used to assign an ACI number, true color, or color book color to an object. To change the value of the Color property, you use the Color object which is under the Colors namespace.

The Color object has a SetRGB method which allows you to choose from millions of color combinations based on mixing a red, green and blue color value together. The Color object also contains methods and properties for specifying color names, color books, color indexes, and color values.

You can also assign colors to layers. If you want an object to inherit the color of the layer it is on, set the object's color to ByLayer by setting its ACI value to 256. Any number of objects and layers can have the same color number. You can assign each color number to a different pen on a pen plotter or use the color numbers to identify certain objects in the drawing, even though you cannot see the colors on your screen.

For more information about working with colors, see "Work with Colors" in the *AutoCAD User's Guide*.

Topics in this section

- [Assign a color value to an object](#)
- [Make a color current through the database](#)
- [Make a color current with the CECOLOR system variable](#)

Assign a color value to an object

The following example creates 4 circles and assigns a different color to each circle using four different methods.

VB.NET

```vbnet
Imports Autodesk.AutoCAD.Runtime
Imports Autodesk.AutoCAD.ApplicationServices
Imports Autodesk.AutoCAD.DatabaseServices
Imports Autodesk.AutoCAD.Geometry
Imports Autodesk.AutoCAD.Colors

<CommandMethod("SetObjectColor")> _
Public Sub SetObjectColor()
  '' Get the current document and database
  Dim acDoc As Document = Application.DocumentManager.MdiActiveDocument
  Dim acCurDb As Database = acDoc.Database

  '' Start a transaction
  Using acTrans As Transaction = acCurDb.TransactionManager.StartTransaction()

      '' Define an array of colors for the layers
      Dim acColors(2) As Color
      acColors(0) = Color.FromColorIndex(ColorMethod.ByAci, 1)
      acColors(1) = Color.FromRgb(23, 54, 232)
      acColors(2) = Color.FromNames("PANTONE Yellow 0131 C", _
                                    "PANTONE(R) pastel coated")

      '' Open the Block table for read
      Dim acBlkTbl As BlockTable
      acBlkTbl = acTrans.GetObject(acCurDb.BlockTableId, _
                                   OpenMode.ForRead)

      '' Open the Block table record Model space for write
      Dim acBlkTblRec As BlockTableRecord
      acBlkTblRec = acTrans.GetObject(acBlkTbl(BlockTableRecord.ModelSpace), _
                                      OpenMode.ForWrite)

      '' Create a circle object and assign it the ACI value of 4
      Dim acPt As Point3d = New Point3d(0, 3, 0)
      Dim acCirc As Circle = New Circle()
      acCirc.SetDatabaseDefaults()
      acCirc.Center = acPt
      acCirc.Radius = 1
      acCirc.ColorIndex = 4

      acBlkTblRec.AppendEntity(acCirc)
      acTrans.AddNewlyCreatedDBObject(acCirc, True)

      Dim nCnt As Integer = 0

      While (nCnt < 3)
          '' Create a copy of the circle
          Dim acCircCopy As Circle
          acCircCopy = acCirc.Clone()

          '' Shift the copy along the Y-axis
          acPt = New Point3d(acPt.X, acPt.Y + 3, acPt.Z)
          acCircCopy.Center = acPt
```

```
        '' Assign the new color to the circle
        acCircCopy.Color = acColors(nCnt)

        acBlkTblRec.AppendEntity(acCircCopy)
        acTrans.AddNewlyCreatedDBObject(acCircCopy, True)

        nCnt = nCnt + 1
    End While

    '' Save the changes and dispose of the transaction
    acTrans.Commit()
  End Using
End Sub
```

C#

```csharp
using Autodesk.AutoCAD.Runtime;
using Autodesk.AutoCAD.ApplicationServices;
using Autodesk.AutoCAD.DatabaseServices;
using Autodesk.AutoCAD.Geometry;
using Autodesk.AutoCAD.Colors;

[CommandMethod("SetObjectColor")]
public static void SetObjectColor()
{
  // Get the current document and database
  Document acDoc = Application.DocumentManager.MdiActiveDocument;
  Database acCurDb = acDoc.Database;

  // Start a transaction
  using (Transaction acTrans = acCurDb.TransactionManager.StartTransaction())
  {
      // Define an array of colors for the layers
      Color[] acColors = new Color[3];
      acColors[0] = Color.FromColorIndex(ColorMethod.ByAci, 1);
      acColors[1] = Color.FromRgb(23, 54, 232);
      acColors[2] = Color.FromNames("PANTONE Yellow 0131 C",
                                    "PANTONE(R) pastel coated");

      // Open the Block table for read
      BlockTable acBlkTbl;
      acBlkTbl = acTrans.GetObject(acCurDb.BlockTableId,
                                   OpenMode.ForRead) as BlockTable;

      // Open the Block table record Model space for write
      BlockTableRecord acBlkTblRec;
      acBlkTblRec = acTrans.GetObject(acBlkTbl[BlockTableRecord.ModelSpace],
                                      OpenMode.ForWrite) as BlockTableRecord;

      // Create a circle object and assign it the ACI value of 4
      Point3d acPt = new Point3d(0, 3, 0);
      Circle acCirc = new Circle();
      acCirc.SetDatabaseDefaults();
      acCirc.Center = acPt;
      acCirc.Radius = 1;
      acCirc.ColorIndex = 4;

      acBlkTblRec.AppendEntity(acCirc);
      acTrans.AddNewlyCreatedDBObject(acCirc, true);

      int nCnt = 0;

      while (nCnt < 3)
```

```
    {
        // Create a copy of the circle
        Circle acCircCopy;
        acCircCopy = acCirc.Clone() as Circle;

        // Shift the copy along the Y-axis
        acPt = new Point3d(acPt.X, acPt.Y + 3, acPt.Z);
        acCircCopy.Center = acPt;

        // Assign the new color to the circle
        acCircCopy.Color = acColors[nCnt];

        acBlkTblRec.AppendEntity(acCircCopy);
        acTrans.AddNewlyCreatedDBObject(acCircCopy, true);

        nCnt = nCnt + 1;
    }

    // Save the changes and dispose of the transaction
    acTrans.Commit();
  }
}
```

⊟ *VBA/ActiveX Code Reference*

```
Sub SetObjectColor()
    ' Define an array of colors
    Dim colorObj(2) As New AcadAcCmColor

    colorObj(0).ColorMethod = acColorMethodByACI
    colorObj(0).ColorIndex = acRed
    Call colorObj(1).SetRGB(23, 54, 232)
    Call colorObj(2).SetColorBookColor("PANTONE(R) pastel coated", _
                                       "PANTONE Yellow 0131 C")

    ' Define the center point of the circle
    Dim centerPt(0 To 2) As Double
    centerPt(0) = 0: centerPt(1) = 3: centerPt(2) = 0

    ' Create a new circle and assign it the ACI value of 4
    Dim circleObj As AcadCircle
    Set circleObj = ThisDrawing.ModelSpace.AddCircle(centerPt, 1)
    circleObj.color = acCyan

    Dim nCnt As Integer

    ' Create 3 more circles
    While (nCnt < 3)
       ' Create a copy of the circle
       Dim circleObjCopy As AcadCircle
       Set circleObjCopy = circleObj.Copy

       ' Shift the copy along the Y-axis
       centerPt(1) = centerPt(1) + 3
       circleObjCopy.Center = centerPt

       ' Assign the new color to the circle
       circleObjCopy.TrueColor = colorObj(nCnt)

       nCnt = nCnt + 1
    Wend
End Sub
```

Make a color current through the database

This example sets a color current through the Database object with the Cecolor property.

VB.NET

```
Imports Autodesk.AutoCAD.Runtime
Imports Autodesk.AutoCAD.ApplicationServices
Imports Autodesk.AutoCAD.DatabaseServices
Imports Autodesk.AutoCAD.Colors

<CommandMethod("SetColorCurrent")> _
Public Sub SetColorCurrent()
  '' Get the current document
  Dim acDoc As Document = Application.DocumentManager.MdiActiveDocument

  '' Set the current color
  acDoc.Database.Cecolor = Color.FromColorIndex(ColorMethod.ByLayer, 256)
End Sub
```

C#

```
using Autodesk.AutoCAD.Runtime;
using Autodesk.AutoCAD.ApplicationServices;
using Autodesk.AutoCAD.DatabaseServices;
using Autodesk.AutoCAD.Colors;

[CommandMethod("SetColorCurrent")]
public static void SetColorCurrent()
{
  // Get the current document
  Document acDoc = Application.DocumentManager.MdiActiveDocument;

  // Set the current color
  acDoc.Database.Cecolor = Color.FromColorIndex(ColorMethod.ByLayer, 256);
}
```

Make a color current with the CECOLOR system variable

This example sets the color Red current with the CECOLOR system variable.

VB.NET

```
Application.SetSystemVariable("CECOLOR", "1")
```

C#

```
Application.SetSystemVariable("CECOLOR", "1");
```

⊟ *VBA/ActiveX Code Reference*

```
ThisDrawing.SetVariable "CECOLOR", "1"
```

Work with Linetypes

A linetype is a repeating pattern of dashes, dots, and blank spaces. A complex linetype is a repeating pattern of symbols. To use a linetype you must first load it into your drawing. A linetype definition must exist in a LIN library file before a linetype can be loaded into a drawing. To load a linetype into your drawing, use the member method LoadLineTypeFile of a Database object.

For more information about working with linetypes, see "Overview of Linetypes" in the *AutoCAD User's Guide*.

NoteThe linetypes used internally by AutoCAD should not be confused with the hardware linetypes provided by some plotters. The two types of dashed lines produce similar results. Do *not* use both types at the same time, however, because the results can be unpredictable.

Load a linetype into AutoCAD

This example attempts to load the linetype "CENTER" from the *acad.lin* file. If the linetype already exists, or the file does not exist, then a message is displayed.

VB.NET

```
Imports Autodesk.AutoCAD.Runtime
Imports Autodesk.AutoCAD.ApplicationServices
Imports Autodesk.AutoCAD.DatabaseServices

<CommandMethod("LoadLinetype")> _
Public Sub LoadLinetype()
  '' Get the current document and database
  Dim acDoc As Document = Application.DocumentManager.MdiActiveDocument
  Dim acCurDb As Database = acDoc.Database

  '' Start a transaction
  Using acTrans As Transaction = acCurDb.TransactionManager.StartTransaction()

      '' Open the Linetype table for read
      Dim acLineTypTbl As LinetypeTable
      acLineTypTbl = acTrans.GetObject(acCurDb.LinetypeTableId, _
                                       OpenMode.ForRead)

      Dim sLineTypName As String = "Center"

      If acLineTypTbl.Has(sLineTypName) = False Then
          '' Load the Center Linetype
          acCurDb.LoadLineTypeFile(sLineTypName, "acad.lin")
      End If

      '' Save the changes and dispose of the transaction
      acTrans.Commit()
  End Using
End Sub
```

C#

```
using Autodesk.AutoCAD.Runtime;
using Autodesk.AutoCAD.ApplicationServices;
using Autodesk.AutoCAD.DatabaseServices;

[CommandMethod("LoadLinetype")]
```

```
public static void LoadLinetype()
{
  // Get the current document and database
  Document acDoc = Application.DocumentManager.MdiActiveDocument;
  Database acCurDb = acDoc.Database;

  // Start a transaction
  using (Transaction acTrans = acCurDb.TransactionManager.StartTransaction())
  {
      // Open the Linetype table for read
      LinetypeTable acLineTypTbl;
      acLineTypTbl = acTrans.GetObject(acCurDb.LinetypeTableId,
                                       OpenMode.ForRead) as LinetypeTable;

      string sLineTypName = "Center";

      if (acLineTypTbl.Has(sLineTypName) == false)
      {
          // Load the Center Linetype
          acCurDb.LoadLineTypeFile(sLineTypName, "acad.lin");
      }

      // Save the changes and dispose of the transaction
      acTrans.Commit();
  }
}
```

⊟ *VBA/ActiveX Code Reference*

```
Sub LoadLinetype()
    On Error GoTo ERRORHANDLER

    Dim linetypeName As String
    linetypeName = "CENTER"

    ' Load "CENTER" line type from acad.lin file
    ThisDrawing.Linetypes.Load linetypeName, "acad.lin"
    Exit Sub

ERRORHANDLER:
    MsgBox Err.Description

End Sub
```

Topics in this section

- [Make a Linetype Active](#)
- [Rename Linetypes](#)
- [Delete Linetypes](#)
- [Change Linetype Descriptions](#)
- [Specify Linetype Scale](#)

Make a Linetype Active

To use a linetype, you must make it active. All newly created objects are drawn using the active linetype. There are two different methods for applying a linetype to an object: direct or inherited. You can directly assign a linetype to an object which overrides the linetype assigned to the layer the object is on. Otherwise, an object inherits the linetype of the layer it is on by having its Linetype or LinetypeId property set to represent the ByLayer linetype.

NoteXref-dependent linetypes cannot be made active.

There are three linetypes that exist in each drawing: BYBLOCK, BYLAYER and CONTINUOUS. Each of these linetypes can be accessed from the Linetype table object or using methods from the SymbolUtilityServices object. The following methods allow you to obtain the object id for these default linetypes:

- *GetLinetypeByBlockId* - Returns the object id for the BYBLOCK linetype.
- *GetLinetypeByLayerId* - Returns the object id for the BYLAYER linetype.
- *GetLinetypeContinuousId* - Returns the object id for the CONTINUOUS linetype.

For more information about activating a linetype, see "Set the Current Linetype" in the *AutoCAD User's Guide*.

Topics in this section

- [Assign a linetype to an object](#)
- [Make a linetype current through the database](#)
- [Make a linetype current with the CELTYPE system variable](#)

Assign a linetype to an object

The following example creates a circle and assigns the "Center" linetype to it.

VB.NET

```
Imports Autodesk.AutoCAD.Runtime
Imports Autodesk.AutoCAD.ApplicationServices
Imports Autodesk.AutoCAD.DatabaseServices
Imports Autodesk.AutoCAD.Geometry

<CommandMethod("SetObjectLinetype")> _
Public Sub SetObjectLinetype()
  '' Get the current document and database
  Dim acDoc As Document = Application.DocumentManager.MdiActiveDocument
  Dim acCurDb As Database = acDoc.Database

  '' Start a transaction
  Using acTrans As Transaction = acCurDb.TransactionManager.StartTransaction()

      '' Open the Linetype table for read
```

```vb
        Dim acLineTypTbl As LinetypeTable
        acLineTypTbl = acTrans.GetObject(acCurDb.LinetypeTableId, _
                                         OpenMode.ForRead)

        Dim sLineTypName As String = "Center"

        If acLineTypTbl.Has(sLineTypName) = False Then
            acCurDb.LoadLineTypeFile(sLineTypName, "acad.lin")
        End If

        '' Open the Block table for read
        Dim acBlkTbl As BlockTable
        acBlkTbl = acTrans.GetObject(acCurDb.BlockTableId, _
                                     OpenMode.ForRead)

        '' Open the Block table record Model space for write
        Dim acBlkTblRec As BlockTableRecord
        acBlkTblRec = acTrans.GetObject(acBlkTbl(BlockTableRecord.ModelSpace), _
                                        OpenMode.ForWrite)

        '' Create a circle object
        Dim acCirc As Circle = New Circle()
        acCirc.SetDatabaseDefaults()
        acCirc.Center = New Point3d(2, 2, 0)
        acCirc.Radius = 1
        acCirc.Linetype = sLineTypName

        acBlkTblRec.AppendEntity(acCirc)
        acTrans.AddNewlyCreatedDBObject(acCirc, True)

        '' Save the changes and dispose of the transaction
        acTrans.Commit()
    End Using
End Sub
```

C#

```csharp
using Autodesk.AutoCAD.Runtime;
using Autodesk.AutoCAD.ApplicationServices;
using Autodesk.AutoCAD.DatabaseServices;
using Autodesk.AutoCAD.Geometry;

[CommandMethod("SetObjectLinetype")]
public static void SetObjectLinetype()
{
    // Get the current document and database
    Document acDoc = Application.DocumentManager.MdiActiveDocument;
    Database acCurDb = acDoc.Database;

    // Start a transaction
    using (Transaction acTrans = acCurDb.TransactionManager.StartTransaction())
    {
        // Open the Linetype table for read
        LinetypeTable acLineTypTbl;
        acLineTypTbl = acTrans.GetObject(acCurDb.LinetypeTableId,
                                         OpenMode.ForRead) as LinetypeTable;

        string sLineTypName = "Center";

        if (acLineTypTbl.Has(sLineTypName) == false)
        {
            acCurDb.LoadLineTypeFile(sLineTypName, "acad.lin");
        }
```

```
    // Open the Block table for read
    BlockTable acBlkTbl;
    acBlkTbl = acTrans.GetObject(acCurDb.BlockTableId,
                                 OpenMode.ForRead) as BlockTable;

    // Open the Block table record Model space for write
    BlockTableRecord acBlkTblRec;
    acBlkTblRec = acTrans.GetObject(acBlkTbl[BlockTableRecord.ModelSpace],
                                    OpenMode.ForWrite) as BlockTableRecord;

    // Create a circle object
    Circle acCirc = new Circle();
    acCirc.SetDatabaseDefaults();
    acCirc.Center = new Point3d(2, 2, 0);
    acCirc.Radius = 1;
    acCirc.Linetype = sLineTypName;

    acBlkTblRec.AppendEntity(acCirc);
    acTrans.AddNewlyCreatedDBObject(acCirc, true);

    // Save the changes and dispose of the transaction
    acTrans.Commit();
  }
}
```

⊟ *VBA/ActiveX Code Reference*

```
Sub SetObjectLinetype()
    ' Load the Center linetype
    Dim linetypeName As String
    linetypeName = "Center"

    On Error Resume Next
    ThisDrawing.Linetypes.Load linetypeName, "acad.lin"

    ' Define the center point of the circle
    Dim centerPt(0 To 2) As Double
    centerPt(0) = 0: centerPt(1) = 3: centerPt(2) = 0

    ' Create a new circle and assign it the ACI value of 4
    Dim circleObj As AcadCircle
    Set circleObj = ThisDrawing.ModelSpace.AddCircle(centerPt, 1)
    circleObj.Linetype = linetypeName

    circleObj.Update
End Sub
```

Make a linetype current through the database

This example sets a linetype current through the Database object with the Celtype property.

VB.NET

```
Imports Autodesk.AutoCAD.Runtime
Imports Autodesk.AutoCAD.ApplicationServices
```

```
Imports Autodesk.AutoCAD.DatabaseServices

<CommandMethod("SetLinetypeCurrent")> _
Public Sub SetLinetypeCurrent()
  '' Get the current document and database
  Dim acDoc As Document = Application.DocumentManager.MdiActiveDocument
  Dim acCurDb As Database = acDoc.Database

  '' Start a transaction
  Using acTrans As Transaction = acCurDb.TransactionManager.StartTransaction()

      '' Open the Linetype table for read
      Dim acLineTypTbl As LinetypeTable
      acLineTypTbl = acTrans.GetObject(acCurDb.LinetypeTableId, _
                                       OpenMode.ForRead)

      Dim sLineTypName As String = "Center"

      If acLineTypTbl.Has(sLineTypName) = True Then
          '' Set the linetype Center current
          acCurDb.Celtype = acLineTypTbl(sLineTypName)

          '' Save the changes
          acTrans.Commit()
      End If

      '' Dispose of the transaction
  End Using
End Sub
```

C#

```
using Autodesk.AutoCAD.Runtime;
using Autodesk.AutoCAD.ApplicationServices;
using Autodesk.AutoCAD.DatabaseServices;

[CommandMethod("SetLinetypeCurrent")]
public static void SetLinetypeCurrent()
{
  // Get the current document and database
  Document acDoc = Application.DocumentManager.MdiActiveDocument;
  Database acCurDb = acDoc.Database;

  // Start a transaction
  using (Transaction acTrans = acCurDb.TransactionManager.StartTransaction())
  {
      // Open the Linetype table for read
      LinetypeTable acLineTypTbl;
      acLineTypTbl = acTrans.GetObject(acCurDb.LinetypeTableId,
                                       OpenMode.ForRead) as LinetypeTable;

      string sLineTypName = "Center";

      if (acLineTypTbl.Has(sLineTypName) == true)
      {
          // Set the linetype Center current
          acCurDb.Celtype = acLineTypTbl[sLineTypName];

          // Save the changes
          acTrans.Commit();
      }

      // Dispose of the transaction
  }
```

```
}
```

VBA/ActiveX Code Reference

```
ThisDrawing.ActiveLinetype = ThisDrawing.Linetypes.Item("Center")
```

Make a linetype current with the CELTYPE system variable

This example sets a linetype current with the CELTYPE system variable.

VB.NET

```
Application.SetSystemVariable("CELTYPE", "Center")
```

C#

```
Application.SetSystemVariable("CELTYPE", "Center");
```

VBA/ActiveX Code Reference

```
ThisDrawing.SetVariable "CELTYPE", "Center"
```

Rename Linetypes

To rename a linetype, use the Name property. When you rename a linetype, you are renaming only the linetype definition in your drawing. The name in the LIN library file is not updated to reflect the new name.

Delete Linetypes

To delete a linetype, use the Erase method. You can delete a linetype at any time during a drawing session; however, linetypes that cannot be deleted include BYLAYER, BYBLOCK, CONTINUOUS, the current linetype, a linetype in use, and xref-dependent linetypes. Also, linetypes referenced by block definitions cannot be deleted, even if they are not used by any objects.

For more information about deleting linetypes, see "Set the Current Linetype" in the *AutoCAD User's Guide.*

Change Linetype Descriptions

Linetypes can have a description associated with them. The description provides an ASCII representation of the linetype. You can assign or change a linetype description by using the AsciiDescription property.

A linetype description can have up to 47 characters. The description can be a comment or a series of underscores, dots, dashes, and spaces to show a simple representation of the linetype pattern.

Change the description of a linetype

The following example changes the description of the current linetype.

VB.NET

```
Imports Autodesk.AutoCAD.Runtime
Imports Autodesk.AutoCAD.ApplicationServices
Imports Autodesk.AutoCAD.DatabaseServices

<CommandMethod("ChangeLinetypeDescription")> _
Public Sub ChangeLinetypeDescription()
  '' Get the current document and database
  Dim acDoc As Document = Application.DocumentManager.MdiActiveDocument
  Dim acCurDb As Database = acDoc.Database

  '' Start a transaction
  Using acTrans As Transaction = acCurDb.TransactionManager.StartTransaction()

      '' Open the Linetype table record of the current linetype for write
      Dim acLineTypTblRec As LinetypeTableRecord
      acLineTypTblRec = acTrans.GetObject(acCurDb.Celtype, _
                                          OpenMode.ForWrite)

      '' Change the description of the current linetype
      acLineTypTblRec.AsciiDescription = "Exterior Wall"

      '' Save the changes and dispose of the transaction
      acTrans.Commit()
  End Using
End Sub
```

C#

```
using Autodesk.AutoCAD.Runtime;
using Autodesk.AutoCAD.ApplicationServices;
using Autodesk.AutoCAD.DatabaseServices;

[CommandMethod("ChangeLinetypeDescription")]
public static void ChangeLinetypeDescription()
```

```
{
  // Get the current document and database
  Document acDoc = Application.DocumentManager.MdiActiveDocument;
  Database acCurDb = acDoc.Database;

  // Start a transaction
  using (Transaction acTrans = acCurDb.TransactionManager.StartTransaction())
  {
      // Open the Linetype table record of the current linetype for write
      LinetypeTableRecord acLineTypTblRec;
      acLineTypTblRec = acTrans.GetObject(acCurDb.Celtype,
                                          OpenMode.ForWrite) as
LinetypeTableRecord;

      // Change the description of the current linetype
      acLineTypTblRec.AsciiDescription = "Exterior Wall";

      // Save the changes and dispose of the transaction
      acTrans.Commit();
  }
}
```

⊟ *VBA/ActiveX Code Reference*

```
ThisDrawing.ActiveLinetype.Description = "Exterior Wall"
```

Specify Linetype Scale

You can specify the linetype scale for objects you create. The smaller the scale, the more repetitions of the pattern are generated per drawing unit. By default, AutoCAD uses a global linetype scale of 1.0, which is equal to one drawing unit. You can change the linetype scale for all drawing objects and attribute references.

The CELTSCALE system variable sets the linetype scale for newly created objects. LTSCALE system variable changes the global linetype scale of existing objects as well as new objects. The LinetypeScale property on an object is used to change the linetype scale for an object. The linetype scale at which an object is displayed at is based on the an individual object's linetype scale multiplied by the global linetype scale.

For more information about linetype scales, see "Control Linetype Scale" in the *AutoCAD User's Guide*.

Change the linetype scale for an object

VB.NET

```
Imports Autodesk.AutoCAD.Runtime
Imports Autodesk.AutoCAD.ApplicationServices
Imports Autodesk.AutoCAD.DatabaseServices
Imports Autodesk.AutoCAD.Geometry

<CommandMethod("SetObjectLinetypeScale")> _
```

```vbnet
Public Sub SetObjectLinetypeScale()
    '' Get the current document and database
    Dim acDoc As Document = Application.DocumentManager.MdiActiveDocument
    Dim acCurDb As Database = acDoc.Database

    '' Start a transaction
    Using acTrans As Transaction = acCurDb.TransactionManager.StartTransaction()

        '' Save the current linetype
        Dim acObjId As ObjectId = acCurDb.Celtype

        '' Set the global linetype scale
        acCurDb.Ltscale = 3

        '' Open the Linetype table for read
        Dim acLineTypTbl As LinetypeTable
        acLineTypTbl = acTrans.GetObject(acCurDb.LinetypeTableId, _
                                         OpenMode.ForRead)

        Dim sLineTypName As String = "Border"

        If acLineTypTbl.Has(sLineTypName) = False Then
            acCurDb.LoadLineTypeFile(sLineTypName, "acad.lin")
        End If

        '' Set the Border linetype current
        acCurDb.Celtype = acLineTypTbl(sLineTypName)

        '' Open the Block table for read
        Dim acBlkTbl As BlockTable
        acBlkTbl = acTrans.GetObject(acCurDb.BlockTableId, _
                                     OpenMode.ForRead)

        '' Open the Block table record Model space for write
        Dim acBlkTblRec As BlockTableRecord
        acBlkTblRec = acTrans.GetObject(acBlkTbl(BlockTableRecord.ModelSpace), _
                                        OpenMode.ForWrite)

        '' Create a circle object and set its linetype
        '' scale to half of full size
        Dim acCirc1 As Circle = New Circle()
        acCirc1.SetDatabaseDefaults()
        acCirc1.Center = New Point3d(2, 2, 0)
        acCirc1.Radius = 4
        acCirc1.LinetypeScale = 0.5

        acBlkTblRec.AppendEntity(acCirc1)
        acTrans.AddNewlyCreatedDBObject(acCirc1, True)

        '' Create a second circle object
        Dim acCirc2 As Circle = New Circle()
        acCirc2.SetDatabaseDefaults()
        acCirc2.Center = New Point3d(12, 2, 0)
        acCirc2.Radius = 4

        acBlkTblRec.AppendEntity(acCirc2)
        acTrans.AddNewlyCreatedDBObject(acCirc2, True)

        '' Restore the original active linetype
        acCurDb.Celtype = acObjId

        '' Save the changes and dispose of the transaction
        acTrans.Commit()
    End Using
End Sub
```

C#

```csharp
using Autodesk.AutoCAD.Runtime;
using Autodesk.AutoCAD.ApplicationServices;
using Autodesk.AutoCAD.DatabaseServices;
using Autodesk.AutoCAD.Geometry;

[CommandMethod("SetObjectLinetypeScale")]
public static void SetObjectLinetypeScale()
{
    // Get the current document and database
    Document acDoc = Application.DocumentManager.MdiActiveDocument;
    Database acCurDb = acDoc.Database;

    // Start a transaction
    using (Transaction acTrans = acCurDb.TransactionManager.StartTransaction())
    {
        // Save the current linetype
        ObjectId acObjId = acCurDb.Celtype;

        // Set the global linetype scale
        acCurDb.Ltscale = 3;

        // Open the Linetype table for read
        LinetypeTable acLineTypTbl;
        acLineTypTbl = acTrans.GetObject(acCurDb.LinetypeTableId,
                                         OpenMode.ForRead) as LinetypeTable;

        string sLineTypName = "Border";

        if (acLineTypTbl.Has(sLineTypName) == false)
        {
            acCurDb.LoadLineTypeFile(sLineTypName, "acad.lin");
        }

        // Set the Border linetype current
        acCurDb.Celtype = acLineTypTbl[sLineTypName];

        // Open the Block table for read
        BlockTable acBlkTbl;
        acBlkTbl = acTrans.GetObject(acCurDb.BlockTableId,
                                     OpenMode.ForRead) as BlockTable;

        // Open the Block table record Model space for write
        BlockTableRecord acBlkTblRec;
        acBlkTblRec = acTrans.GetObject(acBlkTbl[BlockTableRecord.ModelSpace],
                                        OpenMode.ForWrite) as BlockTableRecord;

        // Create a circle object and set its linetype
        // scale to half of full size
        Circle acCirc1 = new Circle();
        acCirc1.SetDatabaseDefaults();
        acCirc1.Center = new Point3d(2, 2, 0);
        acCirc1.Radius = 4;
        acCirc1.LinetypeScale = 0.5;

        acBlkTblRec.AppendEntity(acCirc1);
        acTrans.AddNewlyCreatedDBObject(acCirc1, true);

        // Create a second circle object
        Circle acCirc2 = new Circle();
        acCirc2.SetDatabaseDefaults();
        acCirc2.Center = new Point3d(12, 2, 0);
        acCirc2.Radius = 4;
```

```
        acBlkTblRec.AppendEntity(acCirc2);
        acTrans.AddNewlyCreatedDBObject(acCirc2, true);

        // Restore the original active linetype
        acCurDb.Celtype = acObjId;

        // Save the changes and dispose of the transaction
        acTrans.Commit();
    }
}
```

⊟ *VBA/ActiveX Code Reference*

```
Sub SetObjectLinetypeScale()
    ' Save the current linetype
    Dim currLineType As AcadLineType
    Set currLineType = ThisDrawing.ActiveLinetype

    ' Set global linetype scale
    ThisDrawing.SetVariable "LTSCALE", 3

    ' Load the Border linetype
    On Error Resume Next
    If Not IsObject(ThisDrawing.Linetypes.Item("Border")) Then
      ThisDrawing.Linetypes.Load "Border", "acad.lin"
    End If

    ThisDrawing.ActiveLinetype = ThisDrawing.Linetypes.Item("BORDER")

    ' Create a circle object in model space
    Dim center(0 To 2) As Double
    center(0) = 2: center(1) = 2: center(2) = 0

    Dim circleObj1 As AcadCircle
    Set circleObj1 = ThisDrawing.ModelSpace.AddCircle(center, 4)

    ' Set the linetype scale of the circle to half of full size
    circleObj1.LinetypeScale = 0.5
    circleObj1.Update

    Dim circleObj2 As AcadCircle
    center(0) = center(0) + 10
    Set circleObj2 = ThisDrawing.ModelSpace.AddCircle(center, 4)
    circleObj2.Update

    ' Restore original active linetype
    ThisDrawing.ActiveLinetype = currLineType
End Sub
```

Save and Restore Layer States

You can save layer states in a drawing and restore them later. This makes it easy to return to specified settings for all layers during different stages when completing a drawing or when plotting a drawing.

Layer states include whether or not a layer is turned on, frozen, locked, plotted, and automatically frozen in new viewports, and the layer's color, linetype, lineweight, and plot style. You can specify which settings you want to save, and you can save different groups of settings for a drawing.

The LayerStateManager is used to save and restore layer states.

For more information about saving layer settings, see "Save and Restore Layer Settings" in the *AutoCAD User's Guide.*

Topics in this section

- Understand How AutoCAD Saves Layer States
- Use the LayerStateManager to Manage Layer States

Understand How AutoCAD Saves Layer States

AutoCAD saves layer setting information in an extension dictionary of the Layers table object. When you first save a layer state, AutoCAD does the following:

- Creates an extension dictionary on the Layers table.
- Creates a Dictionary object named ACAD_LAYERSTATE in the extension dictionary.
- Stores the properties of each layer in the drawing in an XRecord object in the ACAD_LAYERSTATE dictionary. AutoCAD stores all layer settings in the XRecord, but identifies the specific settings you chose to save. When you restore the layer settings, AutoCAD restores only the settings you chose to save.

Each time you save another layer setting in the drawing, AutoCAD creates another XRecord object describing the saved settings and stores the XRecord in the ACAD_LAYERSTATE dictionary. The following diagram illustrates the process.
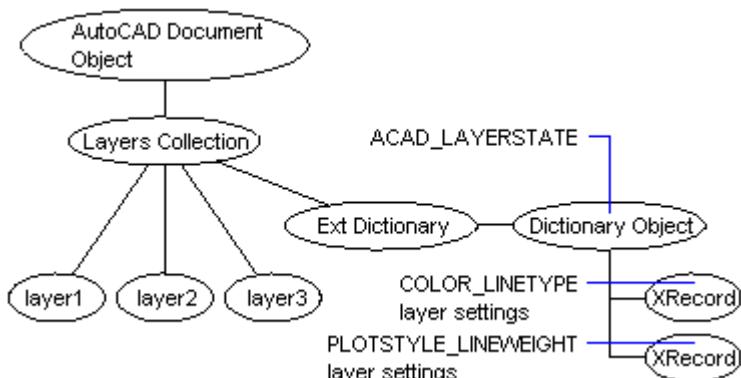
You do not need (and should not try) to directly manipulate the entries when working with layer states. Use the functions of the LayerStateManager object to access the dictionary. Once you have a reference to the dictionary, you can step through each of the entries which are represented as DBDictionaryEntry objects.

List the saved layer states in a drawing

If layer states have been saved in the current drawing, the following code lists the names of all saved layer states:

VB.NET

```
Imports Autodesk.AutoCAD.Runtime
Imports Autodesk.AutoCAD.ApplicationServices
Imports Autodesk.AutoCAD.DatabaseServices

<CommandMethod("ListLayerStates")> _
Public Sub ListLayerStates()
  '' Get the current document and database
  Dim acDoc As Document = Application.DocumentManager.MdiActiveDocument
  Dim acCurDb As Database = acDoc.Database

  '' Start a transaction
  Using acTrans As Transaction = acCurDb.TransactionManager.StartTransaction()

      Dim acLyrStMan As LayerStateManager
      acLyrStMan = acCurDb.LayerStateManager

      Dim acDbDict As DBDictionary
      acDbDict = acTrans.GetObject(acLyrStMan.LayerStatesDictionaryId(True), _
                                   OpenMode.ForRead)

      Dim sLayerStateNames As String = ""

      For Each acDbDictEnt As DBDictionaryEntry In acDbDict
          sLayerStateNames = sLayerStateNames & vbLf & acDbDictEnt.Key
      Next

      Application.ShowAlertDialog("The saved layer settings in this drawing are:" _
& _
                                  sLayerStateNames)

      '' Dispose of the transaction
  End Using
End Sub
```

C#

```
using Autodesk.AutoCAD.Runtime;
using Autodesk.AutoCAD.ApplicationServices;
using Autodesk.AutoCAD.DatabaseServices;

[CommandMethod("ListLayerStates")]
public static void ListLayerStates()
{
  // Get the current document and database
  Document acDoc = Application.DocumentManager.MdiActiveDocument;
  Database acCurDb = acDoc.Database;

  // Start a transaction
  using (Transaction acTrans = acCurDb.TransactionManager.StartTransaction())
  {
```

```
        LayerStateManager acLyrStMan;
        acLyrStMan = acCurDb.LayerStateManager;

        DBDictionary acDbDict;
        acDbDict = acTrans.GetObject(acLyrStMan.LayerStatesDictionaryId(true),
                                     OpenMode.ForRead) as DBDictionary;

        string sLayerStateNames = "";

        foreach (DBDictionaryEntry acDbDictEnt in acDbDict)
        {
            sLayerStateNames = sLayerStateNames + "\n" + acDbDictEnt.Key;
        }

        Application.ShowAlertDialog("The saved layer settings in this drawing are:"
+
                                    sLayerStateNames);

        // Dispose of the transaction
    }
}
```

⊟ *VBA/ActiveX Code Reference*

```
Sub ListLayerStates()
     On Error Resume Next
    Dim oLSMDict As AcadDictionary
    Dim XRec As Object
    Dim layerstateNames As String

    layerstateNames = ""
    ' Get the ACAD_LAYERSTATES dictionary, which is in the
    ' extension dictionary in the Layers object.
    Set oLSMDict = ThisDrawing.Layers. _
                              GetExtensionDictionary.Item("ACAD_LAYERSTATES")

    ' List the name of each saved layer setting. Settings are
    ' stored as XRecords in the dictionary.
    For Each XRec In oLSMDict
        layerstateNames = layerstateNames + XRec.Name + vbCrLf
    Next XRec

    MsgBox "The saved layer settings in this drawing are: " + _
           vbCrLf + layerstateNames
End Sub
```

Use the LayerStateManager to Manage Layer States

The LayerStateManager object provides a set of functions for creating and manipulating saved layer states. Use the following LayerStateManager functions for working with layer states:

DeleteLayerState

Deletes a saved layer state.

ExportLayerState

Exports the specified saved layer state to a LAS file.

ImportLayerState

Imports a layer state from the specified LAS file.

ImportLayerStateFromDb

Imports a layer state from another database.

RenameLayerState

Renames a saved layer state.

RestoreLayerState

Restores the specified layer state in the current drawing.

SaveLayerState

Saves the specified layer state and its properties.

The LayerStateManager object for a database can be accessed by using the LayerManagerState property of a Database object.

VB.NET

```
Dim acDoc As Document = Application.DocumentManager.MdiActiveDocument
Dim acCurDb As Database = acDoc.Database

Dim acLyrStMan As LayerStateManager
acLyrStMan = acCurDb.LayerStateManager
```

C#

```
Document acDoc = Application.DocumentManager.MdiActiveDocument;
Database acCurDb = acDoc.Database;

LayerStateManager acLyrStMan;
acLyrStMan = acCurDb.LayerStateManager;
```

⊟ *VBA/ActiveX Code Reference*

After you retrieve the LayerStateManager object, you must associate a database with it before you can access the object's methods. Use the SetDatabase method to associate a database with the LayerStateManager.

```
Dim oLSM As AcadLayerStateManager
Set oLSM = ThisDrawing.Application. _
              GetInterfaceObject("AutoCAD.AcadLayerStateManager.18")

oLSM.SetDatabase ThisDrawing.Database
```

Topics in this section

Save Layer States

Use the SaveLayerState method to save a set of layer settings in a drawing. The SaveLayerState method requires three parameters. The first parameter is a string naming the layer state you are saving. The second parameter identifies the layer properties you want to save. Use the constants of the LayerStateMasks enum to identify the layer settings you want to save. The following table lists the constants that are part of the LayerStateMasks enum.

Constants for layer state mask

Constant name	Layer property
Color	Color
CurrentViewport	Current viewport layers frozen or thawed
Frozen	Frozen or thawed
LastRestored	Last restored layer
LineType	Linetype
LineWeight	Lineweight
Locked	Locked or unlocked
NewViewport	New viewport layers frozen or thawed
None	No layer settings
On	On or off
Plot	Plotting on or off
PlotStyle	Plot style

Add the constants together to specify multiple properties.

The third parameter required is the object id of the viewport whose layer settings you want to save. Use ObjectId.Null to not specify a viewport. If you try to save a layer state under a name that already exists, an error is returned. You must rename or delete the existing layer state before you can reuse the name.

Save a layer's color and linetype settings

The following code saves the color and linetype settings of the current layers in the drawing under the name ColorLinetype.

VB.NET

```vbnet
Imports Autodesk.AutoCAD.Runtime
Imports Autodesk.AutoCAD.ApplicationServices
Imports Autodesk.AutoCAD.DatabaseServices

<CommandMethod("SaveLayerColorAndLinetype")> _
Public Sub SaveLayerColorAndLinetype()
  '' Get the current document
  Dim acDoc As Document = Application.DocumentManager.MdiActiveDocument

  Dim acLyrStMan As LayerStateManager
  acLyrStMan = acDoc.Database.LayerStateManager

  Dim sLyrStName As String = "ColorLinetype"

  If acLyrStMan.HasLayerState(sLyrStName) = False Then
      acLyrStMan.SaveLayerState(sLyrStName, _
                                LayerStateMasks.Color + _
                                LayerStateMasks.LineType, _
                                ObjectId.Null)
  End If
End Sub
```

C#

```csharp
using Autodesk.AutoCAD.Runtime;
using Autodesk.AutoCAD.ApplicationServices;
using Autodesk.AutoCAD.DatabaseServices;

[CommandMethod("SaveLayerColorAndLinetype")]
public static void SaveLayerColorAndLinetype()
{
  // Get the current document
  Document acDoc = Application.DocumentManager.MdiActiveDocument;

  LayerStateManager acLyrStMan;
  acLyrStMan = acDoc.Database.LayerStateManager;

  string sLyrStName = "ColorLinetype";

  if (acLyrStMan.HasLayerState(sLyrStName) == false)
  {
      acLyrStMan.SaveLayerState(sLyrStName,
                                LayerStateMasks.Color |
                                LayerStateMasks.LineType,
                                ObjectId.Null);
  }
}
```

⊟ VBA/ActiveX Code Reference

```vba
Sub SaveLayerColorAndLinetype()
    Dim oLSM As AcadLayerStateManager

    ' Access the LayerStateManager object
    Set oLSM = ThisDrawing.Application. _
                    GetInterfaceObject("AutoCAD.AcadLayerStateManager.18")

    ' Associate the current drawing database with LayerStateManager
    oLSM.SetDatabase ThisDrawing.Database
    oLSM.Save "ColorLinetype", acLsColor + acLsLineType
End Sub
```

Rename Layer States

The RenameLayerState method renames a saved layer state from one name to another in a drawing. The following code renames the ColorLinetype layer settings to OldColorLinetype.

VB.NET Imports Autodesk.AutoCAD.Runtime

```vbnet
Imports Autodesk.AutoCAD.ApplicationServices
Imports Autodesk.AutoCAD.DatabaseServices

<CommandMethod("RenameLayerState")> _
Public Sub RenameLayerState()
  '' Get the current document
  Dim acDoc As Document = Application.DocumentManager.MdiActiveDocument

  Dim acLyrStMan As LayerStateManager
  acLyrStMan = acDoc.Database.LayerStateManager

  Dim sLyrStName As String = "ColorLinetype"
  Dim sLyrStNewName As String = "OldColorLinetype"

  If acLyrStMan.HasLayerState(sLyrStName) = True And _
     acLyrStMan.HasLayerState(sLyrStNewName) = False Then
       acLyrStMan.RenameLayerState(sLyrStName, sLyrStNewName)
  End If
End Sub
```

C#

```csharp
using Autodesk.AutoCAD.Runtime;
using Autodesk.AutoCAD.ApplicationServices;
using Autodesk.AutoCAD.DatabaseServices;

[CommandMethod("RenameLayerState")]
public static void RenameLayerState()
{
  // Get the current document
  Document acDoc = Application.DocumentManager.MdiActiveDocument;

  LayerStateManager acLyrStMan;
  acLyrStMan = acDoc.Database.LayerStateManager;

  string sLyrStName = "ColorLinetype";
  string sLyrStNewName = "OldColorLinetype";

  if (acLyrStMan.HasLayerState(sLyrStName) == true &&
      acLyrStMan.HasLayerState(sLyrStNewName) == false)
  {
      acLyrStMan.RenameLayerState(sLyrStName, sLyrStNewName);
  }
}
```

```
Sub RenameLayerState()
    Dim oLSM As AcadLayerStateManager
    Set oLSM = ThisDrawing.Application. _
                GetInterfaceObject("AutoCAD.AcadLayerStateManager.18")

    oLSM.SetDatabase ThisDrawing.Database
    oLSM.Rename "ColorLinetype", "OldColorLinetype"
End Sub
```

Delete Layer States

The DeleteLayerState method removes a saved layer state from a drawing. The following code deletes the layer state saved under the name ColorLinetype.

VB.NET

```
Imports Autodesk.AutoCAD.Runtime
Imports Autodesk.AutoCAD.ApplicationServices
Imports Autodesk.AutoCAD.DatabaseServices

<CommandMethod("RemoveLayerState")> _
Public Sub RemoveLayerState()
  '' Get the current document
  Dim acDoc As Document = Application.DocumentManager.MdiActiveDocument

  Dim acLyrStMan As LayerStateManager
  acLyrStMan = acDoc.Database.LayerStateManager

  Dim sLyrStName As String = "ColorLinetype"

  If acLyrStMan.HasLayerState(sLyrStName) = True Then
      acLyrStMan.DeleteLayerState(sLyrStName)
  End If
End Sub
```

C#

```
using Autodesk.AutoCAD.Runtime;
using Autodesk.AutoCAD.ApplicationServices;
using Autodesk.AutoCAD.DatabaseServices;

[CommandMethod("RemoveLayerState")]
public static void RemoveLayerState()
{
  // Get the current document
  Document acDoc = Application.DocumentManager.MdiActiveDocument;

  LayerStateManager acLyrStMan;
  acLyrStMan = acDoc.Database.LayerStateManager;

  string sLyrStName = "ColorLinetype";

  if (acLyrStMan.HasLayerState(sLyrStName) == true)
```

```
    {
        acLyrStMan.DeleteLayerState(sLyrStName);
    }
}
```

⊟ *VBA/ActiveX Code Reference*

```
Sub RemoveLayerState()
    Dim oLSM As AcadLayerStateManager
    Set oLSM = ThisDrawing.Application. _
                GetInterfaceObject("AutoCAD.AcadLayerStateManager.18")

    oLSM.SetDatabase ThisDrawing.Database
    oLSM.Delete "ColorLinetype"
End Sub
```

Restore Layer States

The RestoreLayerState method resets the layer settings in a layer state and requires four values. The first value is the name of the layer state to restore, and the second value is the object id of the viewport whose layer settings you want to restore. The third value is an integer that defines how layers not in the layer state are handled. The fourth value determines which layer settings are restored.

The following values determine how layers not in a layer state are handled:

- 0 - Layers not in the layer state are left unchanged
- 1 - Layers not in the layer state are turned Off
- 2 - Layers not in the layer state are frozen in the current viewport
- 4 - Layer settings are restored as viewport overrides

NoteYou can use the sum of multiple values previous listed to define the restore behavior of layers not in a layer state. For example, you can turn off and freeze the layers that are not saved with a layer state.

For example, if you save the color and linetype settings under the name "ColorLinetype" and subsequently change those settings, restoring "ColorLinetype" resets the layers to the colors and linetypes they had when "ColorLinetype" was saved. If you add new layers to the drawing after saving "ColorLinetype," those new layers are not affected when you restore "ColorLinetype."

Restore the color and linetype settings of a drawing's layers

Assuming that the color and linetype settings of the layers in the current drawing were previously saved under the name "ColorLinetype," the following code restores the color and linetype settings of each layer in the drawing to the value they had when "ColorLinetype" was saved.

VB.NET

```
Imports Autodesk.AutoCAD.Runtime
Imports Autodesk.AutoCAD.ApplicationServices
Imports Autodesk.AutoCAD.DatabaseServices

<CommandMethod("RestoreLayerState")> _
Public Sub RestoreLayerState()
  '' Get the current document
  Dim acDoc As Document = Application.DocumentManager.MdiActiveDocument

  Dim acLyrStMan As LayerStateManager
  acLyrStMan = acDoc.Database.LayerStateManager

  Dim sLyrStName As String = "ColorLinetype"

  If acLyrStMan.HasLayerState(sLyrStName) = True Then
      acLyrStMan.RestoreLayerState(sLyrStName, _
                                   ObjectId.Null, _
                                   1, _
                                   LayerStateMasks.Color + _
                                   LayerStateMasks.LineType)
  End If
End Sub
```

C#

```
using Autodesk.AutoCAD.Runtime;
using Autodesk.AutoCAD.ApplicationServices;
using Autodesk.AutoCAD.DatabaseServices;

[CommandMethod("RestoreLayerState")]
public static void RestoreLayerState()
{
  // Get the current document
  Document acDoc = Application.DocumentManager.MdiActiveDocument;

  LayerStateManager acLyrStMan;
  acLyrStMan = acDoc.Database.LayerStateManager;

  string sLyrStName = "ColorLinetype";

  if (acLyrStMan.HasLayerState(sLyrStName) == true)
  {
      acLyrStMan.RestoreLayerState(sLyrStName,
                                   ObjectId.Null,
                                   1,
                                   LayerStateMasks.Color |
                                   LayerStateMasks.LineType);
  }
}
```

⊟ *VBA/ActiveX Code Reference*

```
Sub RestoreLayerState()
    Dim oLSM As AcadLayerStateManager
    Set oLSM = ThisDrawing.Application. _
                   GetInterfaceObject("AutoCAD.AcadLayerStateManager.18")

    oLSM.SetDatabase ThisDrawing.Database
    oLSM.Restore "ColorLinetype"
End Sub
```

Export and Import Saved Layer States

You can export and import saved layer states to use the same layer settings in other drawings. Use the ExportLayerState method to export a saved layer state to an LAS file; use the ImportLayerState method to import a LAS file into a drawing.

NoteImporting layer states does not restore them; you must use the RestoreLayerState method to restore the layer state after it is imported.

The ExportLayerState method accepts two parameters. The first parameter is a string identifying the saved layer state to export. The second parameter is the name of the file you are exporting the layer state to. If you do not specify a path for the file, it is saved in the same directory in which the drawing was opened from. If the file name you specified already exists, the existing file is overwritten. Use a *.las* extension when naming files; this is the extension AutoCAD recognizes for exported layer state files.

The ImportLayerState method accepts one parameter: a string naming the file that contains the layer states you are importing. If the layer state you want to import does not exist in a LAS file, but a drawing file. You can open the drawing file and then use the ImportLayerStateFromDb method to import a layer state from the Database object of the other drawing.

When you import layer states, an error condition is raised if any properties referenced in the saved settings are not available in the drawing you are importing to. The import is completed, however, and default properties are used. For example, if an exported layer is set to a linetype that is not loaded in the drawing it is being imported into, an error condition is raised and the drawing's default linetype is substituted. Your code should account for this error condition and continue processing if it is raised.

If the imported file defines settings for layers that do not exist in the current drawing, those layers are created in the current drawing. When you use the RestoreLayerState method, the properties specified when the settings were saved are assigned to the new layers; all other properties of the new layers are assigned default settings.

Export saved layer settings

The following example exports a saved layer state to a file named *ColorLinetype.las*.

VB.NET

```
Imports Autodesk.AutoCAD.Runtime
Imports Autodesk.AutoCAD.ApplicationServices
Imports Autodesk.AutoCAD.DatabaseServices

<CommandMethod("ExportLayerState")> _
Public Sub ExportLayerState()
  '' Get the current document
  Dim acDoc As Document = Application.DocumentManager.MdiActiveDocument

  Dim acLyrStMan As LayerStateManager
  acLyrStMan = acDoc.Database.LayerStateManager

  Dim sLyrStName As String = "ColorLinetype"
```

```
    If acLyrStMan.HasLayerState(sLyrStName) = True Then
        acLyrStMan.ExportLayerState(sLyrStName, "c:\my documents\" & _
                                                sLyrStName & ".las")
    End If
End Sub
```

C#

```
using Autodesk.AutoCAD.Runtime;
using Autodesk.AutoCAD.ApplicationServices;
using Autodesk.AutoCAD.DatabaseServices;

[CommandMethod("ExportLayerState")]
public static void ExportLayerState()
{
  // Get the current document
  Document acDoc = Application.DocumentManager.MdiActiveDocument;
  LayerStateManager acLyrStMan;
  acLyrStMan = acDoc.Database.LayerStateManager;

  string sLyrStName = "ColorLinetype";

  if (acLyrStMan.HasLayerState(sLyrStName) == true)
  {
      acLyrStMan.ExportLayerState(sLyrStName, "c:\\my documents\\" +
                                              sLyrStName + ".las");
  }
}
```

⊟ *VBA/ActiveX Code Reference*

```
Sub ExportLayerStates()
    Dim oLSM As AcadLayerStateManager
    Set oLSM = ThisDrawing.Application. _
       GetInterfaceObject("AutoCAD.AcadLayerStateManager.18")

    oLSM.SetDatabase ThisDrawing.Database
    oLSM.Export "ColorLinetype", "c:\my documents\ColorLinetype.las"
End Sub
```

Import saved layer settings

The following example imports the layer state from a file named *ColorLinetype.las*.

VB.NET

```
Imports Autodesk.AutoCAD.Runtime
Imports Autodesk.AutoCAD.ApplicationServices
Imports Autodesk.AutoCAD.DatabaseServices

<CommandMethod("ImportLayerState")> _
Public Sub ImportLayerState()
  '' Get the current document
  Dim acDoc As Document = Application.DocumentManager.MdiActiveDocument

  Dim acLyrStMan As LayerStateManager
  acLyrStMan = acDoc.Database.LayerStateManager

  Dim sLyrStFileName As String = "c:\my documents\ColorLinetype.las"

  If System.IO.File.Exists(sLyrStFileName) Then
```

```
      Try
          acLyrStMan.ImportLayerState(sLyrStFileName)
      Catch ex As Autodesk.AutoCAD.Runtime.Exception
          Application.ShowAlertDialog(ex.Message)
      End Try
   End If
End Sub
```

C#

```csharp
using Autodesk.AutoCAD.Runtime;
using Autodesk.AutoCAD.ApplicationServices;
using Autodesk.AutoCAD.DatabaseServices;

[CommandMethod("ImportLayerState")]
public static void ImportLayerState()
{
  // Get the current document
  Document acDoc = Application.DocumentManager.MdiActiveDocument;

  LayerStateManager acLyrStMan;
  acLyrStMan = acDoc.Database.LayerStateManager;

  string sLyrStFileName = "c:\\my documents\\ColorLinetype.las";

  if (System.IO.File.Exists(sLyrStFileName))
  {
      try
      {
          acLyrStMan.ImportLayerState(sLyrStFileName);
      }
      catch (Autodesk.AutoCAD.Runtime.Exception ex)
      {
          Application.ShowAlertDialog(ex.Message);
      }
  }
}
```

☐ VBA/ActiveX Code Reference

```vb
Sub ImportLayerStates()
    Dim oLSM As AcadLayerStateManager
    Set oLSM = ThisDrawing.Application. _
                   GetInterfaceObject("AutoCAD.AcadLayerStateManager.18")
    oLSM.SetDatabase ThisDrawing.Database

    ' If the drawing you're importing to does not contain
    ' all the linetypes referenced in the saved settings,
    ' an error is returned. The import is completed, though,
    ' and the default linetype is used.
    On Error Resume Next
    oLSM.Import "c:\my documents\ColorLType.las"

    If Err.Number = -2145386359 Then
        ' Error indicates a linetype is not defined
        MsgBox ("One or more linetypes specified in the imported " + _
                "settings is not defined in your drawing")
    End If

    On Error GoTo 0
End Sub
```

Add Text to Drawings

Text conveys important information in your drawing. Use text objects for title blocks, to label parts of the drawing, to give specifications, or to make annotations.

AutoCAD provides various ways to create text. For short, simple entries, use single line text. For longer entries with internal formatting, use multiline text (MText). Although all entered text uses the current text style, which establishes the default font and format settings, you can use several methods to customize the text appearance.

For more information about working with text, see "Create Text" in the *AutoCAD User's Guide*.

Topics in this section

- Work with Text Styles
- Use Single-Line Text (Text)
- Use Multiline Text (MText)
- Use Unicode Characters, Control Codes, and Special Characters
- Substitute Fonts
- Check Spelling

Work with Text Styles

All text in an AutoCAD drawing has a style associated with it. When you enter text, AutoCAD uses the current text style, which sets the font, size, angle, orientation, and other text characteristics. You can use or modify the default style or create and load a new style. Once you've created a style, you can modify its attributes or delete it when you no longer need it.

Topics in this section

- Create and Modify Text Styles
- Assign Fonts
- Use TrueType Fonts
- Use Unicode and Big Fonts
- Set Text Height
- Set Obliquing Angle
- Set Text Generation Flag

Create and Modify Text Styles

New text inherits height, width factor, obliquing angle, and text generation properties from the current text style. To create a text style, create a new instance of a TextStyleTableRecord

object. Assign the new text style a name using the Name property. Then open the TextStyleTable object for write and use the Add method to create the new text style.

Style names can contain letters, numbers, and the special characters dollar sign ($), underscore (_), and hyphen (-). AutoCAD converts the characters to uppercase. If you do not enter a style name, the new style will not have a name.

You can modify an existing style by changing the properties of the TextStyleTableRecord object. If you want to work with the current text style, use the TextStyle property of the Database object which holds the object id of the current text style.

You can also update existing text of that style type to reflect the changes. Use the following properties to modify a TextStyleTableRecord object:

BigFontFileName

> Specifies the special shape definition file used for a non-ASCII character set.

FileName

> Specifies the file associated with a font (character style).

FlagBits

> Specifies backward text, upside-down text, or both.

Font

> Specifies the typeface, bold, italic, character set, and pitch and family settings of the text style.

IsVertical

> Specifies vertical or horizontal text.

ObliquingAngle

> Specifies the slant of the characters.

TextSize

> Specifies the character height.

XScale

> Specifies the expansion or compression of the characters.

If you change an existing style's font or orientation, all text using that style is changed to use the new font or orientation. Changing text height, width factor, and oblique angle does not change existing text but does change subsequently created text objects.

NoteThe drawing must be regenerated to see any changes to the above properties.

Assign Fonts

Fonts define the shapes of the text characters that make up each character set. A single font can be used by more than one style. Use the FileName property to set the font file for the text style. You can assign TrueType or AutoCAD-compiled SHX fonts to a text style.

Set text fonts

The following example gets the current font values using the Font property for the active text style and then changes the typeface for the font to "PlayBill." To see the effects of changing the typeface, add some multiline or single-line text to your current drawing before running the example. Note that, if you don't have the PlayBill font on your system, you need to substitute a font you do have in order for this example to work.

VB.NET

```
Imports Autodesk.AutoCAD.Runtime
Imports Autodesk.AutoCAD.ApplicationServices
Imports Autodesk.AutoCAD.DatabaseServices

<CommandMethod("UpdateTextFont")> _
Public Sub UpdateTextFont()
  '' Get the current document and database
  Dim acDoc As Document = Application.DocumentManager.MdiActiveDocument
  Dim acCurDb As Database = acDoc.Database

  '' Start a transaction
  Using acTrans As Transaction = acCurDb.TransactionManager.StartTransaction()

      '' Open the current text style for write
      Dim acTextStyleTblRec As TextStyleTableRecord
      acTextStyleTblRec = acTrans.GetObject(acCurDb.Textstyle, _
                                            OpenMode.ForWrite)

      '' Get the current font settings
      Dim acFont As Autodesk.AutoCAD.GraphicsInterface.FontDescriptor
      acFont = acTextStyleTblRec.Font

      '' Update the text style's typeface with "PlayBill"
      Dim acNewFont As Autodesk.AutoCAD.GraphicsInterface.FontDescriptor
      acNewFont = New _
        Autodesk.AutoCAD.GraphicsInterface.FontDescriptor("PlayBill", _
                                                acFont.Bold, _
                                                acFont.Italic, _
                                                acFont.CharacterSet, _
                                                acFont.PitchAndFamily)

      acTextStyleTblRec.Font = acNewFont

      acDoc.Editor.Regen()

      '' Save the changes and dispose of the transaction
      acTrans.Commit()
  End Using
End Sub
```

C#

```
using Autodesk.AutoCAD.Runtime;
using Autodesk.AutoCAD.ApplicationServices;
using Autodesk.AutoCAD.DatabaseServices;

[CommandMethod("UpdateTextFont")]
public static void UpdateTextFont()
{
  // Get the current document and database
  Document acDoc = Application.DocumentManager.MdiActiveDocument;
  Database acCurDb = acDoc.Database;

  // Start a transaction
  using (Transaction acTrans = acCurDb.TransactionManager.StartTransaction())
  {
      // Open the current text style for write
      TextStyleTableRecord acTextStyleTblRec;
      acTextStyleTblRec = acTrans.GetObject(acCurDb.Textstyle,
                                            OpenMode.ForWrite) as
TextStyleTableRecord;

      // Get the current font settings
      Autodesk.AutoCAD.GraphicsInterface.FontDescriptor acFont;
      acFont = acTextStyleTblRec.Font;

      // Update the text style's typeface with "PlayBill"
      Autodesk.AutoCAD.GraphicsInterface.FontDescriptor acNewFont;
      acNewFont = new
        Autodesk.AutoCAD.GraphicsInterface.FontDescriptor("PlayBill",
                                                acFont.Bold,
                                                acFont.Italic,
                                                acFont.CharacterSet,
                                                acFont.PitchAndFamily);

      acTextStyleTblRec.Font = acNewFont;

      acDoc.Editor.Regen();

      // Save the changes and dispose of the transaction
      acTrans.Commit();
  }
}
```

⊟ *VBA/ActiveX Code Reference*

```
Sub UpdateTextFont()

    MsgBox ("Look at the text now...")

    Dim typeFace As String
    Dim SavetypeFace As String
    Dim Bold As Boolean
    Dim Italic As Boolean
    Dim charSet As Long
    Dim PitchandFamily As Long

    ' Get the current settings to fill in the
    ' default values for the SetFont method
    ThisDrawing.ActiveTextStyle.GetFont typeFace, _
                Bold, Italic, charSet, PitchandFamily
```

```
    ' Change the typeface for the font
    SavetypeFace = typeFace
    typeFace = "PlayBill"
    ThisDrawing.ActiveTextStyle.SetFont typeFace, _
                Bold, Italic, charSet, PitchandFamily
    ThisDrawing.Regen acActiveViewport
End Sub
```

Use TrueType Fonts

TrueType fonts always appear filled in your drawing; however, when you plot, the TEXTFILL system variable controls whether the fonts are filled. By default TEXTFILL is set to 1 to plot the filled-in fonts. When you export a drawing to PostScript®format   and print it on a PostScript device, the font is plotted as designed.

Use Unicode and Big Fonts

AutoCAD supports the Unicode character-encoding standard. A Unicode font can contain 65,535 characters, with shapes for many languages. All of the AutoCAD SHX shape fonts that are shipped with the product support Unicode fonts.

The text files for some alphabets contain thousands of non-ASCII characters. To accommodate such text, AutoCAD supports a special type of shape definition known as a Big Font file. You can set a style to use both regular and Big Font files. Specify normal fonts using the FileName property. Specify Big Fonts using the BigFontFileName property.

NoteFont file names cannot contain commas.

AutoCAD allows you to specify an alternate font to use when a specified font file cannot be located. Use the FONTALT system variable and the SetSystemVariable member method of the Application to change the alternate font used.

Change font files

The following example code changes the FileName and BigFontFileName properties. You need to replace the path information listed in this example with path and file names appropriate for your system.

VB.NET

```
Imports Autodesk.AutoCAD.Runtime
Imports Autodesk.AutoCAD.ApplicationServices
Imports Autodesk.AutoCAD.DatabaseServices
```

```vba
<CommandMethod("ChangeFontFiles")> _
Public Sub ChangeFontFiles()
  '' Get the current document and database
  Dim acDoc As Document = Application.DocumentManager.MdiActiveDocument
  Dim acCurDb As Database = acDoc.Database

  '' Start a transaction
  Using acTrans As Transaction = acCurDb.TransactionManager.StartTransaction()

      '' Open the current text style for write
      Dim acTextStyleTblRec As TextStyleTableRecord
      acTextStyleTblRec = acTrans.GetObject(acCurDb.Textstyle, _
                                            OpenMode.ForWrite)

      '' Change the font files used for both Big and Regular fonts
      acTextStyleTblRec.BigFontFileName = "C:\AutoCAD\Fonts\bigfont.shx"
      acTextStyleTblRec.FileName = "C:\AutoCAD\Fonts\italic.shx"

      '' Save the changes and dispose of the transaction
      acTrans.Commit()
  End Using
End Sub
```

C#

```csharp
using Autodesk.AutoCAD.Runtime;
using Autodesk.AutoCAD.ApplicationServices;
using Autodesk.AutoCAD.DatabaseServices;

[CommandMethod("ChangeFontFiles")]
public static void ChangeFontFiles()
{
  // Get the current document and database
  Document acDoc = Application.DocumentManager.MdiActiveDocument;
  Database acCurDb = acDoc.Database;

  // Start a transaction
  using (Transaction acTrans = acCurDb.TransactionManager.StartTransaction())
  {
      // Open the current text style for write
      TextStyleTableRecord acTextStyleTblRec;
      acTextStyleTblRec = acTrans.GetObject(acCurDb.Textstyle,
                                            OpenMode.ForWrite) as
TextStyleTableRecord;

      // Change the font files used for both Big and Regular fonts
      acTextStyleTblRec.BigFontFileName = "C:/AutoCAD/Fonts/bigfont.shx";
      acTextStyleTblRec.FileName = "C:/AutoCAD/Fonts/italic.shx";

      // Save the changes and dispose of the transaction
      acTrans.Commit();
  }
}
```

⊟ *VBA/ActiveX Code Reference*

```vba
Sub ChangeFontFiles()
    ThisDrawing.ActiveTextStyle.BigFontFile = _
                "C:/AutoCAD/Fonts/bigfont.shx"

    ThisDrawing.ActiveTextStyle.fontFile = _
```

```
                        "C:/AutoCAD/Fonts/italic.shx"
End Sub
```

Set Text Height

Text height determines the size in drawing units of the letters in the font you are using. The value usually represents the size of the uppercase letters, with the exception of TrueType fonts.

For TrueType fonts, the value specified for text height might not represent the height of uppercase letters. The height specified represents the height of a capital letter plus an accent area reserved for accent marks and other marks used in non-English languages. The relative portion of areas assigned to capital letters and accent characters is determined by the font designer at the time the font is designed, and, consequently, will vary from font to font.

In addition to the height of a capital letter and the ascent area that make up the height specified by the user, TrueType fonts have a descent area for portions of characters that extend below the text insertion line. Examples of such characters are y, j, p, g, and q.

You specify the text height using the TextSize property of the text style or the Height property of a text object. This property accepts positive numbers only.

Set Obliquing Angle

The obliquing angle determines the forward or backward slant of the text. The angle represents the offset from its vertical axis (90 degrees). To set the obliquing angle, use the ObliquingAngle property to change a text style or the Oblique property of a text object. The obliquing angle must be provided in radians. A positive angle denotes a lean to the right, a negative value will have 2*PI added to it to convert it to its positive equivalent.

Create oblique text

This example creates a single-line text object then slants it 45 degrees.

VB.NET

```
Imports Autodesk.AutoCAD.Runtime
Imports Autodesk.AutoCAD.ApplicationServices
Imports Autodesk.AutoCAD.DatabaseServices
Imports Autodesk.AutoCAD.Geometry

<CommandMethod("ObliqueText")> _
Public Sub ObliqueText()
  '' Get the current document and database
```

```vb
    Dim acDoc As Document = Application.DocumentManager.MdiActiveDocument
    Dim acCurDb As Database = acDoc.Database

    '' Start a transaction
    Using acTrans As Transaction = acCurDb.TransactionManager.StartTransaction()

        '' Open the Block table for read
        Dim acBlkTbl As BlockTable
        acBlkTbl = acTrans.GetObject(acCurDb.BlockTableId, _
                                     OpenMode.ForRead)

        '' Open the Block table record Model space for write
        Dim acBlkTblRec As BlockTableRecord
        acBlkTblRec = acTrans.GetObject(acBlkTbl(BlockTableRecord.ModelSpace), _
                                        OpenMode.ForWrite)

        '' Create a single-line text object
        Dim acText As DBText = New DBText()
        acText.SetDatabaseDefaults()
        acText.Position = New Point3d(3, 3, 0)
        acText.Height = 0.5
        acText.TextString = "Hello, World."

        '' Change the oblique angle of the text object to 45 degrees(0.707 in
radians)
        acText.Oblique = 0.707

        acBlkTblRec.AppendEntity(acText)
        acTrans.AddNewlyCreatedDBObject(acText, True)

        '' Save the changes and dispose of the transaction
        acTrans.Commit()
    End Using
End Sub
```

C#

```csharp
using Autodesk.AutoCAD.Runtime;
using Autodesk.AutoCAD.ApplicationServices;
using Autodesk.AutoCAD.DatabaseServices;
using Autodesk.AutoCAD.Geometry;

[CommandMethod("ObliqueText")]
public static void ObliqueText()
{
  // Get the current document and database
  Document acDoc = Application.DocumentManager.MdiActiveDocument;
  Database acCurDb = acDoc.Database;

  // Start a transaction
  using (Transaction acTrans = acCurDb.TransactionManager.StartTransaction())
  {
      // Open the Block table for read
      BlockTable acBlkTbl;
      acBlkTbl = acTrans.GetObject(acCurDb.BlockTableId,
                                   OpenMode.ForRead) as BlockTable;

      // Open the Block table record Model space for write
      BlockTableRecord acBlkTblRec;
      acBlkTblRec = acTrans.GetObject(acBlkTbl[BlockTableRecord.ModelSpace],
                                      OpenMode.ForWrite) as BlockTableRecord;

      // Create a single-line text object
      DBText acText = new DBText();
```

```
        acText.SetDatabaseDefaults();
        acText.Position = new Point3d(3, 3, 0);
        acText.Height = 0.5;
        acText.TextString = "Hello, World.";

        // Change the oblique angle of the text object to 45 degrees(0.707 in
radians)
        acText.Oblique = 0.707;

        acBlkTblRec.AppendEntity(acText);
        acTrans.AddNewlyCreatedDBObject(acText, true);

        // Save the changes and dispose of the transaction
        acTrans.Commit();
    }
}
```

```
Sub ObliqueText()
    Dim textObj As AcadText
    Dim textString As String
    Dim insertionPoint(0 To 2) As Double
    Dim height As Double

    ' Define the text object
    textString = "Hello, World."
    insertionPoint(0) = 3
    insertionPoint(1) = 3
    insertionPoint(2) = 0
    height = 0.5

    ' Create the text object in model space
    Set textObj = ThisDrawing.ModelSpace. _
                     AddText(textString, insertionPoint, height)

    ' Change the value of the ObliqueAngle
    ' to 45 degrees (.707 radians)
    textObj.ObliqueAngle = 0.707
    textObj.Update
End Sub
```

Set Text Generation Flag

The text generation flag specifies if text is displayed backwards or upside-down. Use the FlagBits property to define if a text style controls the display of text to be displayed backwards or upside-down, or use the IsMirroredInX and IsMirroredInY properties of a text object to control individually control a text object.

Set FlagBits to 2 if you want text to be displayed backwards and 4 if it should be displayed upside-down. Use a value of 6 to display text both backwards and upside-down. If you are modifying a text object, set IsMirroredInX to TRUE if you want the text to appear backwards and set IsMirroredInY to TRUE if you want it to be displayed upside-down.

Display text backwards

The following example creates a single-line text object, then sets it to be displayed backwards using the IsMirroredInX property.

VB.NET

```
Imports Autodesk.AutoCAD.Runtime
Imports Autodesk.AutoCAD.ApplicationServices
Imports Autodesk.AutoCAD.DatabaseServices
Imports Autodesk.AutoCAD.Geometry

<CommandMethod("BackwardsText")> _
Public Sub BackwardsText()
  '' Get the current document and database
  Dim acDoc As Document = Application.DocumentManager.MdiActiveDocument
  Dim acCurDb As Database = acDoc.Database

  '' Start a transaction
  Using acTrans As Transaction = acCurDb.TransactionManager.StartTransaction()

      '' Open the Block table for read
      Dim acBlkTbl As BlockTable
      acBlkTbl = acTrans.GetObject(acCurDb.BlockTableId, _
                                   OpenMode.ForRead)

      '' Open the Block table record Model space for write
      Dim acBlkTblRec As BlockTableRecord
      acBlkTblRec = acTrans.GetObject(acBlkTbl(BlockTableRecord.ModelSpace), _
                                      OpenMode.ForWrite)

      '' Create a single-line text object
      Dim acText As DBText = New DBText()
      acText.SetDatabaseDefaults()
      acText.Position = New Point3d(3, 3, 0)
      acText.Height = 0.5
      acText.TextString = "Hello, World."

      '' Display the text backwards
      acText.IsMirroredInX = True

      acBlkTblRec.AppendEntity(acText)
      acTrans.AddNewlyCreatedDBObject(acText, True)

      '' Save the changes and dispose of the transaction
      acTrans.Commit()
  End Using
End Sub
```

C#

```
using Autodesk.AutoCAD.Runtime;
using Autodesk.AutoCAD.ApplicationServices;
using Autodesk.AutoCAD.DatabaseServices;
using Autodesk.AutoCAD.Geometry;

[CommandMethod("BackwardsText")]
public static void BackwardsText()
{
  // Get the current document and database
  Document acDoc = Application.DocumentManager.MdiActiveDocument;
  Database acCurDb = acDoc.Database;
```

```
    // Start a transaction
    using (Transaction acTrans = acCurDb.TransactionManager.StartTransaction())
    {
        // Open the Block table for read
        BlockTable acBlkTbl;
        acBlkTbl = acTrans.GetObject(acCurDb.BlockTableId,
                                     OpenMode.ForRead) as BlockTable;

        // Open the Block table record Model space for write
        BlockTableRecord acBlkTblRec;
        acBlkTblRec = acTrans.GetObject(acBlkTbl[BlockTableRecord.ModelSpace],
                                        OpenMode.ForWrite) as BlockTableRecord;

        // Create a single-line text object
        DBText acText = new DBText();
        acText.SetDatabaseDefaults();
        acText.Position = new Point3d(3, 3, 0);
        acText.Height = 0.5;
        acText.TextString = "Hello, World.";

        // Display the text backwards
        acText.IsMirroredInX = true;

        acBlkTblRec.AppendEntity(acText);
        acTrans.AddNewlyCreatedDBObject(acText, true);

        // Save the changes and dispose of the transaction
        acTrans.Commit();
    }
}
```

VBA/ActiveX Code Reference

```
Sub BackwardsText()
    Dim textObj As AcadText
    Dim textString As String
    Dim insertionPoint(0 To 2) As Double
    Dim height As Double

    ' Create the text object
    textString = "Hello, World."
    insertionPoint(0) = 3
    insertionPoint(1) = 3
    insertionPoint(2) = 0
    height = 0.5
    Set textObj = ThisDrawing.ModelSpace. _
                    AddText(textString, insertionPoint, height)

    ' Change the value of the TextGenerationFlag
    textObj.TextGenerationFlag = acTextFlagBackward
    textObj.Update
End Sub
```

Use Single-Line Text (Text)

The text you add to your drawings conveys a variety of information. It may be a complex specification, title block information, a label, or even part of the drawing. For shorter entries that do not require multiple fonts or lines, create an instance of a DBText object which represents a single-line of text and is convenient for labels.

Topics in this section

- Create Single-Line Text
- Format Single-Line Text
- Align Single-Line Text
- Change Single-Line Text

Create Single-Line Text

Each individual line of text is a distinct object when using single-line text. To create a single-line text object, you create an instance of a DBText object and then add it to either the block table record that represents Model or Paper space. When you create a new instance of a DBText object, you do not pass the constructor any parameters.

To Create Line Text

The following example creates a single-line text object in Model space at the coordinate (2, 2, 0) with a height of 0.5 and the text string "Hello, World.".

VB.NET

```
Imports Autodesk.AutoCAD.Runtime
Imports Autodesk.AutoCAD.ApplicationServices
Imports Autodesk.AutoCAD.DatabaseServices
Imports Autodesk.AutoCAD.Geometry

<CommandMethod("CreateText")> _
Public Sub CreateText()
  '' Get the current document and database
  Dim acDoc As Document = Application.DocumentManager.MdiActiveDocument
  Dim acCurDb As Database = acDoc.Database

  '' Start a transaction
  Using acTrans As Transaction = acCurDb.TransactionManager.StartTransaction()

      '' Open the Block table for read
      Dim acBlkTbl As BlockTable
      acBlkTbl = acTrans.GetObject(acCurDb.BlockTableId, _
                                  OpenMode.ForRead)

      '' Open the Block table record Model space for write
      Dim acBlkTblRec As BlockTableRecord
      acBlkTblRec = acTrans.GetObject(acBlkTbl(BlockTableRecord.ModelSpace), _
```

```
                                          OpenMode.ForWrite)

      '' Create a single-line text object
      Dim acText As DBText = New DBText()
      acText.SetDatabaseDefaults()
      acText.Position = New Point3d(2, 2, 0)
      acText.Height = 0.5
      acText.TextString = "Hello, World."

      acBlkTblRec.AppendEntity(acText)
      acTrans.AddNewlyCreatedDBObject(acText, True)

      '' Save the changes and dispose of the transaction
      acTrans.Commit()
  End Using
End Sub
```

C#

```
using Autodesk.AutoCAD.Runtime;
using Autodesk.AutoCAD.ApplicationServices;
using Autodesk.AutoCAD.DatabaseServices;
using Autodesk.AutoCAD.Geometry;

[CommandMethod("CreateText")]
public static void CreateText()
{
  // Get the current document and database
  Document acDoc = Application.DocumentManager.MdiActiveDocument;
  Database acCurDb = acDoc.Database;

  // Start a transaction
  using (Transaction acTrans = acCurDb.TransactionManager.StartTransaction())
  {
      // Open the Block table for read
      BlockTable acBlkTbl;
      acBlkTbl = acTrans.GetObject(acCurDb.BlockTableId,
                                   OpenMode.ForRead) as BlockTable;

      // Open the Block table record Model space for write
      BlockTableRecord acBlkTblRec;
      acBlkTblRec = acTrans.GetObject(acBlkTbl[BlockTableRecord.ModelSpace],
                                      OpenMode.ForWrite) as BlockTableRecord;

      // Create a single-line text object
      DBText acText = new DBText();
      acText.SetDatabaseDefaults();
      acText.Position = new Point3d(2, 2, 0);
      acText.Height = 0.5;
      acText.TextString = "Hello, World.";

      acBlkTblRec.AppendEntity(acText);
      acTrans.AddNewlyCreatedDBObject(acText, true);

      // Save the changes and dispose of the transaction
      acTrans.Commit();
  }
}
```

☐ *VBA/ActiveX Code Reference*

```
Sub CreateText()
```

```
    Dim textObj As AcadText
    Dim textString As String
    Dim insertionPoint(0 To 2) As Double
    Dim height As Double

    ' Create the text object
    textString = "Hello, World."
    insertionPoint(0) = 2
    insertionPoint(1) = 2
    insertionPoint(2) = 0
    height = 0.5

    Set textObj = ThisDrawing.ModelSpace. _
                    AddText(textString, insertionPoint, height)
    textObj.Update
End Sub
```

Format Single-Line Text

A single-text object is created using the active text style. You can change the formatting of a single-line text object by changing the text style associated with it, or by editing the properties of the single-line text object directly. You *cannot* apply formats to individual words and characters with single-line text objects.

To change a text style associated with an individual single-line text object, set the TextStyleId property to a new text style. Once you have changed the text style, you must regenerate the drawing or update the object to see the changes in your drawing.

In addition to the standard editable properties for entities (color, layer, linetype, and so forth), other properties that you can change on a single-line text object include the following:

HorizontalMode

> Specifies the horizontal alignment for the text.

VerticalMode

> Specifies the vertical alignment for the text.

Position

> Specifies the insertion point for the text.

Oblique

> Specifies the oblique angle of the individual text object.

Rotation

> Specifies the rotation angle in radians for the text.

WidthFactor

Specifies the scale factor for the text.

AlignmentPoint

Specifies the alignment point for the text.

IsMirroredInX

Specifies whether the text is displayed backwards.

IsMirroredInY

Specifies whether the text is displayed upside-down.

TextString

Specifies the actual text string displayed.

Once you have changed a property, regenerate the drawing or update the object to see the changes made.

Align Single-Line Text

You can justify single-line text horizontally and vertically. Left alignment is the default. To set the horizontal and vertical alignment options, use the HorizontalMode and VerticalMode properties.

Normally when a text object is closed, the position and alignment points of the text object are adjusted according to its justification and text style. However, the alignment of an in memory text object will not automatically be updated. Call the AdjustAlignment method to update the alignment of the text object based on its current property values.

Realign text

The following example creates a single-line text (DBText) object and a point (DBPoint) object. The point object is set to the text alignment point, and is changed to a red crosshair so that it is visible. The text alignment is changed and a message box is displayed so that the macro execution is halted. This allows you to see the impact of changing the text alignment.

VB.NET

```
Imports Autodesk.AutoCAD.Runtime
Imports Autodesk.AutoCAD.ApplicationServices
Imports Autodesk.AutoCAD.DatabaseServices
Imports Autodesk.AutoCAD.Geometry
```

```vb
<CommandMethod("TextAlignment")> _
Public Sub TextAlignment()
  '' Get the current document and database
  Dim acDoc As Document = Application.DocumentManager.MdiActiveDocument
  Dim acCurDb As Database = acDoc.Database

  '' Start a transaction
  Using acTrans As Transaction = acCurDb.TransactionManager.StartTransaction()

      '' Open the Block table for read
      Dim acBlkTbl As BlockTable
      acBlkTbl = acTrans.GetObject(acCurDb.BlockTableId, _
                                   OpenMode.ForRead)

      '' Open the Block table record Model space for write
      Dim acBlkTblRec As BlockTableRecord
      acBlkTblRec = acTrans.GetObject(acBlkTbl(BlockTableRecord.ModelSpace), _
                                   OpenMode.ForWrite)

      Dim textString(0 To 2) As String
      textString(0) = "Left"
      textString(1) = "Center"
      textString(2) = "Right"

      Dim textAlign(0 To 2) As Integer
      textAlign(0) = TextHorizontalMode.TextLeft
      textAlign(1) = TextHorizontalMode.TextCenter
      textAlign(2) = TextHorizontalMode.TextRight

      Dim acPtIns As Point3d = New Point3d(3, 3, 0)
      Dim acPtAlign As Point3d = New Point3d(3, 3, 0)

      Dim nCnt As Integer = 0

      For Each strVal As String In textString
          '' Create a single-line text object
          Dim acText As DBText = New DBText()
          acText.SetDatabaseDefaults()
          acText.Position = acPtIns
          acText.Height = 0.5
          acText.TextString = strVal

          '' Set the alignment for the text
          acText.HorizontalMode = textAlign(nCnt)

          If acText.HorizontalMode <> TextHorizontalMode.TextLeft Then
              acText.AlignmentPoint = acPtAlign
          End If

          acBlkTblRec.AppendEntity(acText)
          acTrans.AddNewlyCreatedDBObject(acText, True)

          '' Create a point over the alignment point of the text
          Dim acPoint As DBPoint = New DBPoint(acPtAlign)
          acPoint.SetDatabaseDefaults()
          acPoint.ColorIndex = 1

          acBlkTblRec.AppendEntity(acPoint)
          acTrans.AddNewlyCreatedDBObject(acPoint, True)

          '' Adjust the insertion and alignment points
          acPtIns = New Point3d(acPtIns.X, acPtIns.Y + 3, 0)
          acPtAlign = acPtIns

          nCnt = nCnt + 1
```

```
      Next

      '' Set the point style to crosshair
      Application.SetSystemVariable("PDMODE", 2)

      '' Save the changes and dispose of the transaction
      acTrans.Commit()
  End Using
End Sub
```

C#

```
using Autodesk.AutoCAD.Runtime;
using Autodesk.AutoCAD.ApplicationServices;
using Autodesk.AutoCAD.DatabaseServices;
using Autodesk.AutoCAD.Geometry;

[CommandMethod("TextAlignment")]
public static void TextAlignment()
{
  // Get the current document and database
  Document acDoc = Application.DocumentManager.MdiActiveDocument;
  Database acCurDb = acDoc.Database;

  // Start a transaction
  using (Transaction acTrans = acCurDb.TransactionManager.StartTransaction())
  {
      // Open the Block table for read
      BlockTable acBlkTbl;
      acBlkTbl = acTrans.GetObject(acCurDb.BlockTableId,
                                   OpenMode.ForRead) as BlockTable;

      // Open the Block table record Model space for write
      BlockTableRecord acBlkTblRec;
      acBlkTblRec = acTrans.GetObject(acBlkTbl[BlockTableRecord.ModelSpace],
                                      OpenMode.ForWrite) as BlockTableRecord;

      string[] textString = new string[3];
      textString[0] = "Left";
      textString[1] = "Center";
      textString[2] = "Right";

      int[] textAlign = new int[3];
      textAlign[0] = (int)TextHorizontalMode.TextLeft;
      textAlign[1] = (int)TextHorizontalMode.TextCenter;
      textAlign[2] = (int)TextHorizontalMode.TextRight;

      Point3d acPtIns = new Point3d(3, 3, 0);
      Point3d acPtAlign = new Point3d(3, 3, 0);

      int nCnt = 0;

      foreach (string strVal in textString)
      {
          // Create a single-line text object
          DBText acText = new DBText();
          acText.SetDatabaseDefaults();
          acText.Position = acPtIns;
          acText.Height = 0.5;
          acText.TextString = strVal;

          // Set the alignment for the text
          acText.HorizontalMode = (TextHorizontalMode)textAlign[nCnt];
```

```
            if (acText.HorizontalMode != TextHorizontalMode.TextLeft)
            {
                acText.AlignmentPoint = acPtAlign;
            }

            acBlkTblRec.AppendEntity(acText);
            acTrans.AddNewlyCreatedDBObject(acText, true);

            // Create a point over the alignment point of the text
            DBPoint acPoint = new DBPoint(acPtAlign);
            acPoint.SetDatabaseDefaults();
            acPoint.ColorIndex = 1;

            acBlkTblRec.AppendEntity(acPoint);
            acTrans.AddNewlyCreatedDBObject(acPoint, true);

            // Adjust the insertion and alignment points
            acPtIns = new Point3d(acPtIns.X, acPtIns.Y + 3, 0);
            acPtAlign = acPtIns;

            nCnt = nCnt + 1;
        }

        // Set the point style to crosshair
        Application.SetSystemVariable("PDMODE", 2);

        // Save the changes and dispose of the transaction
        acTrans.Commit();
    }
}
```

⊟ *VBA/ActiveX Code Reference*

```
Sub TextAlignment()
    ' Set the point style to crosshair
    ThisDrawing.SetVariable "PDMODE", 2

    Dim textObj As AcadText
    Dim pointObj As AcadPoint

    ' Define the text strings and insertion point for the text objects
    Dim textString(0 To 2) As String
    textString(0) = "Left"
    textString(1) = "Center"
    textString(2) = "Right"

    Dim textAlign(0 To 2) As Integer
    textAlign(0) = acAlignmentLeft
    textAlign(1) = acAlignmentCenter
    textAlign(2) = acAlignmentRight

    Dim insertionPoint(0 To 2) As Double
    insertionPoint(0) = 3: insertionPoint(1) = 0: insertionPoint(2) = 0

    Dim alignmentPoint(0 To 2) As Double
    alignmentPoint(0) = 3: alignmentPoint(1) = 0: alignmentPoint(2) = 0

    Dim nCnt As Integer
    For Each strVal In textString
        ' Create the Text object in model space
        Set textObj = ThisDrawing.ModelSpace. _
                      AddText(strVal, insertionPoint, 0.5)
```

```
        ' Set the alignment for the text
        textObj.Alignment = textAlign(nCnt)

        On Error Resume Next
        textObj.TextAlignmentPoint = alignmentPoint

        ' Create a point over the alignment point of the text
        Set pointObj = ThisDrawing.ModelSpace.AddPoint(alignmentPoint)
        pointObj.color = acRed

        ' Adjust the insertion and alignment points
        insertionPoint(1) = insertionPoint(1) + 3
        alignmentPoint(1) = insertionPoint(1)

        nCnt = nCnt + 1
    Next
End Sub
```

Change Single-Line Text

Like any other object, text objects can be moved, rotated, erased, and copied. You also can mirror text. If you do not want the text to be reversed when you mirror it, you can set the MIRRTEXT system variable to 0. You move, rotate and copy objects using the TransformBy and Clone methods. For more information on the TransformBy and Clone methods, see [Edit Named and 2D Objects](#).

Use Multiline Text (MText)

For long, complex entries, create multiline text (MText). Multiline text fits a specified width but can extend vertically to an indefinite length. You can format individual words or characters within the MText.

Topics in this section

- [Create Multiline Text](#)
- [Format Multiline Text](#)

Create Multiline Text

You can create a multiline text object by first creating an instance of a MText object and then adding it to a block table record that represents Model or Paper space. The MText object constructor does not take any parameters. After an instance of an MText object is created, you can then assign it a text string, insertion point, and width among other values using its properties. Other properties that you can change affect the object's text height, justification, rotation angle, and text style, or apply character formatting to selected characters

Create a multiline text object

The following example creates an MText object in Model space, at the coordinate (2, 2, 0).

VB.NET

```
Imports Autodesk.AutoCAD.Runtime
Imports Autodesk.AutoCAD.ApplicationServices
Imports Autodesk.AutoCAD.DatabaseServices
Imports Autodesk.AutoCAD.Geometry

<CommandMethod("CreateMText")> _
Public Sub CreateMText()
  '' Get the current document and database
  Dim acDoc As Document = Application.DocumentManager.MdiActiveDocument
  Dim acCurDb As Database = acDoc.Database

  '' Start a transaction
  Using acTrans As Transaction = acCurDb.TransactionManager.StartTransaction()

      '' Open the Block table for read
      Dim acBlkTbl As BlockTable
      acBlkTbl = acTrans.GetObject(acCurDb.BlockTableId, _
                                   OpenMode.ForRead)

      '' Open the Block table record Model space for write
      Dim acBlkTblRec As BlockTableRecord
      acBlkTblRec = acTrans.GetObject(acBlkTbl(BlockTableRecord.ModelSpace), _
                                      OpenMode.ForWrite)

      '' Create a multiline text object
      Dim acMText As MText = New MText()
      acMText.SetDatabaseDefaults()
      acMText.Location = New Point3d(2, 2, 0)
      acMText.Width = 4
      acMText.Contents = "This is a text string for the MText object."

      acBlkTblRec.AppendEntity(acMText)
      acTrans.AddNewlyCreatedDBObject(acMText, True)

      '' Save the changes and dispose of the transaction
      acTrans.Commit()
  End Using
End Sub
```

C#

```csharp
using Autodesk.AutoCAD.Runtime;
using Autodesk.AutoCAD.ApplicationServices;
using Autodesk.AutoCAD.DatabaseServices;
using Autodesk.AutoCAD.Geometry;

[CommandMethod("CreateMText")]
public static void CreateMText()
{
  // Get the current document and database
  Document acDoc = Application.DocumentManager.MdiActiveDocument;
  Database acCurDb = acDoc.Database;

  // Start a transaction
  using (Transaction acTrans = acCurDb.TransactionManager.StartTransaction())
  {
      // Open the Block table for read
      BlockTable acBlkTbl;
      acBlkTbl = acTrans.GetObject(acCurDb.BlockTableId,
                                   OpenMode.ForRead) as BlockTable;

      // Open the Block table record Model space for write
      BlockTableRecord acBlkTblRec;
      acBlkTblRec = acTrans.GetObject(acBlkTbl[BlockTableRecord.ModelSpace],
                                      OpenMode.ForWrite) as BlockTableRecord;

      // Create a multiline text object
      MText acMText = new MText();
      acMText.SetDatabaseDefaults();
      acMText.Location = new Point3d(2, 2, 0);
      acMText.Width = 4;
      acMText.Contents = "This is a text string for the MText object.";

      acBlkTblRec.AppendEntity(acMText);
      acTrans.AddNewlyCreatedDBObject(acMText, true);

      // Save the changes and dispose of the transaction
      acTrans.Commit();
  }
}
```

⊟ *VBA/ActiveX Code Reference*

```vba
Sub Ch4_CreateMText()
    Dim mtextObj As AcadMText
    Dim insertPoint(0 To 2) As Double
    Dim width As Double
    Dim textString As String

    insertPoint(0) = 2
    insertPoint(1) = 2
    insertPoint(2) = 0
    width = 4
    textString = "This is a text string for the mtext object."

    ' Create a text Object in model space
    Set mtextObj = ThisDrawing.ModelSpace. _
                        AddMText(insertPoint, width, textString)
    ZoomAll
End Sub
```

Format Multiline Text

Newly created multiline text automatically assumes the characteristics of the current text style. The default text style is STANDARD. You can override the default text style by applying formatting to individual characters and applying properties to the multiline text object. You also can indicate formatting or special characters using the methods described in this section.

Orientation options such as style, justification, width, and rotation affect all text within the multiline text boundary, not specific words or characters. Use the Attachment property to change the justification of a multiline text object, and the Rotation property to control the angle of rotation.

The TextStyleId property sets the font and formatting characteristics for a multiline text object. As you create multiline text, you can select which style you want to use from a list of existing styles. When you change the style of a multiline text object that has character formatting applied to any portion of the text, the style is applied to the entire object, and some formatting of characters might not be retained. For instance, changing from a TrueType style to a style using an SHX font or to another TrueType font causes the multiline text to use the new font for the entire object, and any character formatting is lost.

Formatting options such as underlining, stacked text, or fonts can be applied to individual words or characters within a paragraph. You also can change color, font, and text height. You can change the spaces between text characters or increase the width of the characters.

Use curly braces ({ }) to apply a format change only to the text within the braces. You can nest braces up to eight levels deep.

You also can enter the ASCII equivalent for control codes within lines or paragraphs to indicate formatting or special characters, such as tolerance or dimensioning symbols.

The following control characters can be used to create the text in the illustration. (For the ASCII equivalent of this string see the example following the illustration.)

{{\H1.5x; Big text} \A2; over text\A1;/\A0; under text}

Big text over text/under text

For more information about formatting multiline text, see "Format Characters Within Multiline Text" in the *AutoCAD User's Guide*.

Use control characters to format text

The following example creates and formats a multiline text object.

VB.NET

```
Imports Autodesk.AutoCAD.Runtime
Imports Autodesk.AutoCAD.ApplicationServices
Imports Autodesk.AutoCAD.DatabaseServices
```

```vb
Imports Autodesk.AutoCAD.Geometry

<CommandMethod("FormatMText")> _
Public Sub FormatMText()
  '' Get the current document and database
  Dim acDoc As Document = Application.DocumentManager.MdiActiveDocument
  Dim acCurDb As Database = acDoc.Database

  '' Start a transaction
  Using acTrans As Transaction = acCurDb.TransactionManager.StartTransaction()

      '' Open the Block table for read
      Dim acBlkTbl As BlockTable
      acBlkTbl = acTrans.GetObject(acCurDb.BlockTableId, _
                                   OpenMode.ForRead)

      '' Open the Block table record Model space for write
      Dim acBlkTblRec As BlockTableRecord
      acBlkTblRec = acTrans.GetObject(acBlkTbl(BlockTableRecord.ModelSpace), _
                                      OpenMode.ForWrite)

      '' Create a multiline text object
      Dim acMText As MText = New MText()
      acMText.SetDatabaseDefaults()
      acMText.Location = New Point3d(2, 2, 0)
      acMText.Width = 4.5
      acMText.Contents = "{{\H1.5x; Big text}\A2; over text\A1;/\A0;under text}"

      acBlkTblRec.AppendEntity(acMText)
      acTrans.AddNewlyCreatedDBObject(acMText, True)

      '' Save the changes and dispose of the transaction
      acTrans.Commit()
  End Using
End Sub
```

C#

```csharp
using Autodesk.AutoCAD.Runtime;
using Autodesk.AutoCAD.ApplicationServices;
using Autodesk.AutoCAD.DatabaseServices;
using Autodesk.AutoCAD.Geometry;

[CommandMethod("FormatMText")]
public static void FormatMText()
{
  // Get the current document and database
  Document acDoc = Application.DocumentManager.MdiActiveDocument;
  Database acCurDb = acDoc.Database;

  // Start a transaction
  using (Transaction acTrans = acCurDb.TransactionManager.StartTransaction())
  {
      // Open the Block table for read
      BlockTable acBlkTbl;
      acBlkTbl = acTrans.GetObject(acCurDb.BlockTableId,
                                   OpenMode.ForRead) as BlockTable;

      // Open the Block table record Model space for write
      BlockTableRecord acBlkTblRec;
      acBlkTblRec = acTrans.GetObject(acBlkTbl[BlockTableRecord.ModelSpace],
                                      OpenMode.ForWrite) as BlockTableRecord;

      // Create a multiline text object
```

```
      MText acMText = new MText();
      acMText.SetDatabaseDefaults();
      acMText.Location = new Point3d(2, 2, 0);
      acMText.Width = 4.5;
      acMText.Contents = "{{\\H1.5x; Big text}\\A2; over text\\A1;/\\A0;under
text}";

      acBlkTblRec.AppendEntity(acMText);
      acTrans.AddNewlyCreatedDBObject(acMText, true);

      // Save the changes and dispose of the transaction
      acTrans.Commit();
   }
}
```

⊟ *VBA/ActiveX Code Reference*

```
Sub FormatMText()
    Dim mtextObj As AcadMText
    Dim insertPoint(0 To 2) As Double
    Dim width As Double
    Dim textString As String

    insertPoint(0) = 2
    insertPoint(1) = 2
    insertPoint(2) = 0
    width = 4.5

    ' Define the ASCII characters for the control characters
    Dim OB As Long  ' Open Bracket  {
    Dim CB As Long  ' Close Bracket }
    Dim BS As Long  ' Back Slash    \
    Dim FS As Long  ' Forward Slash /
    Dim SC As Long  ' Semicolon     ;
    OB = Asc("{")
    CB = Asc("}")
    BS = Asc("\")
    FS = Asc("/")
    SC = Asc(";")

    ' Assign the text string the following line of control
    ' characters and text characters:
    ' {{\H1.5x; Big text}\A2; over text\A1;/\A0;under text}

    textString = Chr(OB) + Chr(OB) + Chr(BS) + "H1.5x" _
    + Chr(SC) + "Big text" + Chr(CB) + Chr(BS) + "A2" _
    + Chr(SC) + "over text" + Chr(BS) + "A1" + Chr(SC) _
    + Chr(FS) + Chr(BS) + "A0" + Chr(SC) + "under text" _
    + Chr(CB)

    ' Create a text Object in model space
    Set mtextObj = ThisDrawing.ModelSpace. _
                      AddMText(insertPoint, width, textString)
    ZoomAll
End Sub
```

Use Unicode Characters, Control Codes, and Special Characters

You can use Unicode characters, control codes, and special characters in a text string to represent symbols. (All nontext characters must be entered as their ASCII equivalent.)

You can create special characters by entering the following Unicode character strings:

Unicode character descriptions

Unicode character

Description

\U+00B0

Degree symbol

\U+00B1

Plus/minus tolerance symbol

\U+2205

Diameter dimensioning symbol

In addition to using Unicode characters for special characters, you can specify a special character by including control information in the text string. Use a pair of percent signs (%%) to introduce each control sequence. For example, the following control code works with standard AutoCAD text and PostScript fonts to draw character number nnn:

%%nnn

These control codes work with standard AutoCAD text fonts only:

Control code descriptions

Control code

Description

%%o

Toggles overscore mode on and off

%%u

Toggles underscore mode on and off

%%d

Draws degree symbol

%%p

Draws plus and minus tolerance symbol

%%c

Draws diameter dimensioning symbol

%%%

Draws single percent sign

Substitute Fonts

You can designate fonts to be substituted for other fonts or as defaults when AutoCAD cannot find a font specified in a drawing.

The fonts used for the text in your drawing are determined by the text style and, for MText, by individual font formats applied to sections of text.

You can use font mapping tables to ensure that your drawing uses only certain fonts, or to convert the fonts you used to other fonts. You can use these font mapping tables to enforce corporate font standards, or to facilitate offline printing. AutoCAD comes with a default font mapping table. You can edit this file using any ASCII text editor. You also can specify a different font mapping table file by using the FONTMAP system variable.

For more information about font mapping tables and substituting fonts, see "Substitute Fonts" in the [AutoCAD User's Guide](#).

Specify an Alternative Default Font

If your drawing specifies a font that is not currently on your system, AutoCAD automatically substitutes the font designated as your alternate font. By default, AutoCAD uses the [simplex.shx](#) file. However, you can specify a different font if necessary. Use the FONTALT system variable to set the alternative font file name.

If you use a text style that uses a Big Font, you can map it to another font using the FONTALT system variable. The font mapping must be done in pairs of font files: [txt.shx](#), [bigfont.shx](#).
If AutoCAD cannot find a font file when a drawing is opened, it applies a default set of font substitution rules.

Check Spelling

During a spelling check, AutoCAD matches the words in the drawing to the words in the current main dictionary. Any words you add are stored in the custom dictionary that is current at the time of the spelling check. For example, you can add proper names so that AutoCAD no longer identifies them as misspelled words.

To check spelling in another language, you can change to a different main dictionary.

There is no method for checking spelling provided in the AutoCAD .NET API. However, you can specify a different main dictionary using the DCTMAIN system variable, or a different custom dictionary using the DCTCUST system variable.

For more information about spellings checks, see "Check Spelling" in the *AutoCAD User's Guide*.

6   Dimensions and Tolerances

Dimensions add measurements to a drawing. Tolerances specify by how much a dimension can vary. With .NET API, dimensions can be managed with dimension styles and overrides.

Topics in this section

- Dimensioning Concepts
- Create Dimensions
- Edit Dimensions
- Work with Dimension Styles
- Dimension in Model Space and Paper Space
- Create Leaders and Annotation
- Use Geometric Tolerances

Dimensioning Concepts

Dimensions show the geometric measurements of objects, the distances or angles between objects, or the *X* and *Y* coordinates of a feature. AutoCAD® provides three basic types of dimensioning: linear, radial, and angular. Linear dimensions include aligned, rotated, and ordinate dimensions.
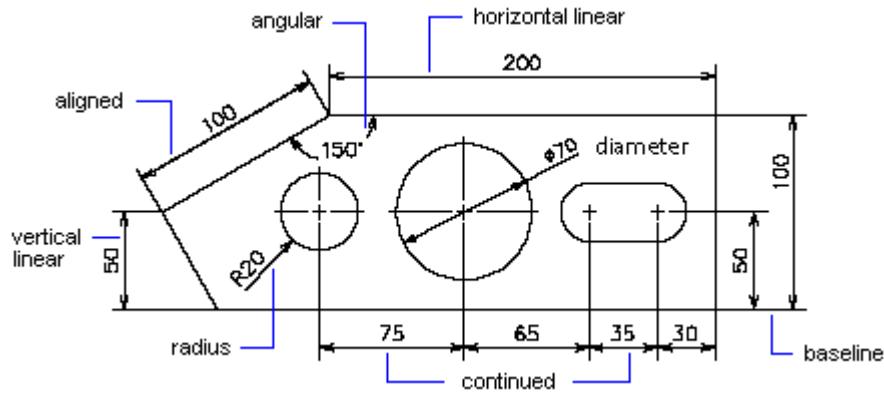
You can create dimensions for lines, multilines, arcs, circles, and polyline segments, or you can create dimensions that stand alone.

AutoCAD draws dimensions on the current layer. Every dimension has a dimension style associated with it, whether it is the default style or one you define. The style controls characteristics such as color, text style, arrowheads, and the scale of the elements in a dimension. Dimensions do not support object thickness. Style families allow for subtle modifications to a base style for different types of dimensions. Overrides allow for style modifications to a specific dimension.

For more information about dimensions, see "Change Existing Objects" in the *AutoCAD User's Guide*.

Topics in this section

Parts of a Dimension

Dimensions are made up of many different objects, such as lines, text, solid fills and blocks. While each dimension type might appear slightly different from one and another, they do have several parts in common.
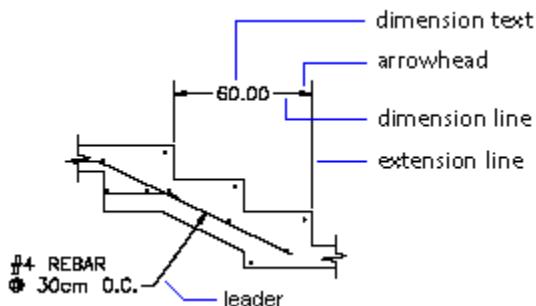
- *Dimension line.* A line that indicates the direction and extent of a dimension. For an angular dimension, the dimension line is an arc.
- *Extension line.* A line in which extends from the feature being dimensioned to the dimension line. Extension lines are also refered to as projection lines or witness lines.
- *Arrowhead.* A symbol that is used to indicate the ends of the dimension line. Arrowheads are also refered to as symbols of termination or just termination.
- *Dimension text.* A ext string that usually indicates the actual measurement of the distance or andle being measured. The text may also include prefixes, suffixes, and tolerances.
- *Leader.* A solid line leading from some annotation to the referenced feature.
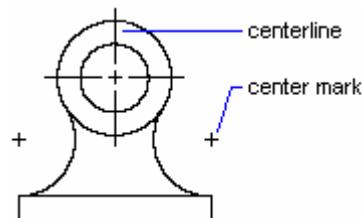
- *Center mark.* A small cross that marks the center of a circle or arc.
- *Centerline.* A set of broken lines that mark the center of a circle or arc.

For more information about the parts of a dimension, see "Parts of a Dimension"in the *AutoCAD User's Guide*.

Define the Dimension System Variables

The dimension system variables control the appearance of dimensions. The dimension system variables include DIMAUNIT, DIMUPT, DIMTOFL, DIMFIT, DIMTIH, DIMTOH, DIMJUST, and DIMTAD. You can set these variables by using the SetSystemVariable method which can be accessed from the Application object. For example, the following line of code sets the DIMAUNIT system variable (the units format for angular dimensions) to radians (3):

VB.NET

```
Application.SetSystemVariable("DIMAUNIT", 3)
```

C#

```
Application.SetSystemVariable("DIMAUNIT", 3);
```

⊟ *VBA/ActiveX Code Reference*

```
ThisDrawing.SetVariable "DIMAUNIT", 3
```

For more information about the dimensioning system variables, see "Use Dimension Styles" in the *AutoCAD User's Guide*.

Set Dimension Text Styles

Dimension text refers to any kind of text associated with dimensions, including measurements, tolerances (both lateral and geometric), prefixes, suffixes, and textual notes in single-line or paragraph form. You can use the default measurement computed by AutoCAD as the text, supply your own text, or suppress the text entirely. You can use dimension text to add information, such as special manufacturing procedures or assembly instructions. Dimension text uses the text style specified by the DIMTXSTY system variable.
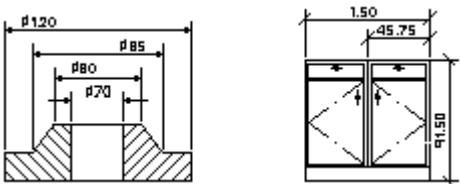
For more information about dimension text, see "Control Dimension Text" in the *AutoCAD User's Guide*.

Understand Leader Lines

A default leader line is a straight line with an arrowhead that points to a feature in a drawing. Usually, a leader's function is to connect annotation with the feature. Annotation in this case means paragraph text, blocks, or feature control frames. Such leader lines are different from the simple leader lines AutoCAD creates automatically for radial, diameter, and linear dimensions whose text does not fit between extension lines.
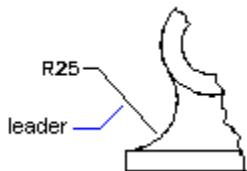
Leader objects are associated with the annotation, so when the annotation is edited, the leader is updated accordingly. You can copy annotation used elsewhere in a drawing and append it to a leader, or you can create new annotation. You can also create a leader with no annotation appended.

For more information about leaders, see "Overview of Creating Text and Leaders" in the *AutoCAD User's Guide*.

Understand Associative Dimensions

Associative dimensions automatically adjust their locations, orientations, and measurement values when the geometric objects associated with them are modified. The DIMASSOC system variable controls associative dimensioning. Set DIMASSOC to 2 to turn on associative dimensioning.

For more information about associative dimensions, see "Associative Dimensions" in the *AutoCAD User's Guide*.

Create Dimensions

You can create linear, radial, angular, and ordinate dimensions.

When creating dimensions, the active dimension style is used. Once created, you can modify the extension line origins, the dimension text location, and the dimension text content and its angle relative to the dimension line. You can also change the dimension style used by the dimension.

For more information about creating dimensions, see "Change Existing Objects" in the *AutoCAD User's Guide*.

Topics in this section

Create Linear Dimensions

Linear dimensions can be aligned or rotated. Aligned dimensions have the dimension line parallel to the line along which the extension line origins lie. Rotated dimensions have the dimension line placed at an angle to the extension line origins.

You create linear dimensions by creating instances of the AlignedDimension and RotatedDimension objects. After you create an instance of a linear dimensions, you can modify the text, the angle of the text, or the angle of the dimension line. The following illustrations show how the type of linear dimension and the placement of the extension line origins affect the angle of the dimension line and text.

horizontal

vertical

aligned

rotated 315 degrees
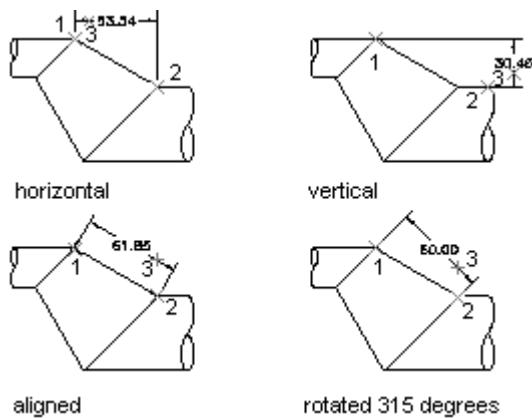
When you create an instance of an AlignedDimension object, you have the option to specify the extension line origins, the location of the dimension line, dimension text, and the dimension style to apply. If you do not pass any parameters into the AlignedDimension object constructor, the object is assigned a set of default property values.

The RotatedDimension object constructor offers the same options as the AlignedDimension object constructor, with one exception. The RotatedDimension object constructor takes an additional paramter that specifies the angle at which the dimension line is rotated.

For additional information about creating linear dimensions, see "Create Linear Dimensions" in the *AutoCAD User's Guide*.

Dimension joglines

Joglines on linear dimensions are not added through a set of properties, but extended data(Xdata). The application name responsible for dimension joglines is ACAD_DSTYLE_DIMJAG_POSITION. The following is an example of the Xdata structure that needs to be appended to a linear dimension.

VB.NET

```
'' Open the Registered Application table for read
Dim acRegAppTbl As RegAppTable
acRegAppTbl = <transaction>.GetObject(<current_database>.RegAppTableId, _
                                      OpenMode.ForRead)

'' Check to see if the app "ACAD_DSTYLE_DIMJAG_POSITION" is
'' registered and if not add it to the RegApp table
If acRegAppTbl.Has("ACAD_DSTYLE_DIMJAG_POSITION") = False Then
    Dim acRegAppTblRec As RegAppTableRecord = New RegAppTableRecord()

    acRegAppTblRec.Name = "ACAD_DSTYLE_DIMJAG_POSITION"

    acRegAppTbl.UpgradeOpen()

    acRegAppTbl.Add(acRegAppTblRec)
    <transaction>.AddNewlyCreatedDBObject(acRegAppTblRec, True)
End If

'' Create a result buffer to define the Xdata
Dim acResBuf As ResultBuffer = New ResultBuffer()
acResBuf.Add(New TypedValue(DxfCode.ExtendedDataRegAppName, _
                                   "ACAD_DSTYLE_DIMJAG_POSITION"))
acResBuf.Add(New TypedValue(DxfCode.ExtendedDataInteger16, 387))
acResBuf.Add(New TypedValue(DxfCode.ExtendedDataInteger16, 3))
acResBuf.Add(New TypedValue(DxfCode.ExtendedDataInteger16, 389))
acResBuf.Add(New TypedValue(DxfCode.ExtendedDataXCoordinate, _
                                   New Point3d(-1.26985, 3.91514, 0)))

'' Attach the Xdata to the dimension
<dimension_object>.XData = acResBuf
```

C#

```
// Open the Registered Application table for read
RegAppTable acRegAppTbl;
acRegAppTbl = <transaction>.GetObject(<current_database>.RegAppTableId,
                                      OpenMode.ForRead) as RegAppTable;

// Check to see if the app "ACAD_DSTYLE_DIMJAG_POSITION" is
// registered and if not add it to the RegApp table
if (acRegAppTbl.Has("ACAD_DSTYLE_DIMJAG_POSITION") == false)
{
    RegAppTableRecord acRegAppTblRec = new RegAppTableRecord();

    acRegAppTblRec.Name = "ACAD_DSTYLE_DIMJAG_POSITION";

    acRegAppTbl.UpgradeOpen();

    acRegAppTbl.Add(acRegAppTblRec);
    <transaction>.AddNewlyCreatedDBObject(acRegAppTblRec, true);
}

// Create a result buffer to define the Xdata
ResultBuffer acResBuf = new ResultBuffer();
acResBuf.Add(new TypedValue((int)DxfCode.ExtendedDataRegAppName,
                                   "ACAD_DSTYLE_DIMJAG_POSITION"));
```

```
acResBuf.Add(new TypedValue((int)DxfCode.ExtendedDataInteger16, 387));
acResBuf.Add(new TypedValue((int)DxfCode.ExtendedDataInteger16, 3));
acResBuf.Add(new TypedValue((int)DxfCode.ExtendedDataInteger16, 389));
acResBuf.Add(new TypedValue((int)DxfCode.ExtendedDataXCoordinate,
                                          new Point3d(-1.26985, 3.91514, 0)));

// Attach the Xdata to the dimension
<dimension_object>.XData = acResBuf;
```

⊟ *VBA/ActiveX Code Reference*

```
Dim DataType(0 To 4) As Integer
Dim Data(0 To 4) As Variant
Dim jogPoint(0 To 2) As Double

DataType(0) = 1001: Data(0) = "ACAD_DSTYLE_DIMJAG_POSITION"
DataType(1) = 1070: Data(1) = 387
DataType(2) = 1070: Data(2) = 3
DataType(3) = 1070: Data(3) = 389

jogPoint(0) = -1.26985: jogPoint(1) = 3.91514: jogPoint(2) = 0#
DataType(4) = 1010: Data(4) = jogPoint

' Attach the xdata to the dimension
<dimension_object>.SetXData DataType, Data
```

Create a rotated linear dimension

This example creates a rotated dimension in Model space.

VB.NET

```
Imports Autodesk.AutoCAD.Runtime
Imports Autodesk.AutoCAD.ApplicationServices
Imports Autodesk.AutoCAD.DatabaseServices
Imports Autodesk.AutoCAD.Geometry

<CommandMethod("CreateRotatedDimension")> _
Public Sub CreateRotatedDimension()
  '' Get the current database
  Dim acDoc As Document = Application.DocumentManager.MdiActiveDocument
  Dim acCurDb As Database = acDoc.Database

  '' Start a transaction
  Using acTrans As Transaction = acCurDb.TransactionManager.StartTransaction()

      '' Open the Block table for read
      Dim acBlkTbl As BlockTable
      acBlkTbl = acTrans.GetObject(acCurDb.BlockTableId, _
                                   OpenMode.ForRead)

      '' Open the Block table record Model space for write
      Dim acBlkTblRec As BlockTableRecord
      acBlkTblRec = acTrans.GetObject(acBlkTbl(BlockTableRecord.ModelSpace), _
                                      OpenMode.ForWrite)

      '' Create the rotated dimension
      Dim acRotDim As RotatedDimension = New RotatedDimension()
      acRotDim.SetDatabaseDefaults()
      acRotDim.XLine1Point = New Point3d(0, 0, 0)
      acRotDim.XLine2Point = New Point3d(6, 3, 0)
```

```
        acRotDim.Rotation = 0.707
        acRotDim.DimLinePoint = New Point3d(0, 5, 0)
        acRotDim.DimensionStyle = acCurDb.Dimstyle

        '' Add the new object to Model space and the transaction
        acBlkTblRec.AppendEntity(acRotDim)
        acTrans.AddNewlyCreatedDBObject(acRotDim, True)

        '' Commit the changes and dispose of the transaction
        acTrans.Commit()
    End Using
End Sub
```

C#

```csharp
using Autodesk.AutoCAD.Runtime;
using Autodesk.AutoCAD.ApplicationServices;
using Autodesk.AutoCAD.DatabaseServices;
using Autodesk.AutoCAD.Geometry;

[CommandMethod("CreateRotatedDimension")]
public static void CreateRotatedDimension()
{
  // Get the current database
  Document acDoc = Application.DocumentManager.MdiActiveDocument;
  Database acCurDb = acDoc.Database;

  // Start a transaction
  using (Transaction acTrans = acCurDb.TransactionManager.StartTransaction())
  {
      // Open the Block table for read
      BlockTable acBlkTbl;
      acBlkTbl = acTrans.GetObject(acCurDb.BlockTableId,
                                   OpenMode.ForRead) as BlockTable;

      // Open the Block table record Model space for write
      BlockTableRecord acBlkTblRec;
      acBlkTblRec = acTrans.GetObject(acBlkTbl[BlockTableRecord.ModelSpace],
                                      OpenMode.ForWrite) as BlockTableRecord;

      // Create the rotated dimension
      RotatedDimension acRotDim = new RotatedDimension();
      acRotDim.SetDatabaseDefaults();
      acRotDim.XLine1Point = new Point3d(0, 0, 0);
      acRotDim.XLine2Point = new Point3d(6, 3, 0);
      acRotDim.Rotation = 0.707;
      acRotDim.DimLinePoint = new Point3d(0, 5, 0);
      acRotDim.DimensionStyle = acCurDb.Dimstyle;

      // Add the new object to Model space and the transaction
      acBlkTblRec.AppendEntity(acRotDim);
      acTrans.AddNewlyCreatedDBObject(acRotDim, true);

      // Commit the changes and dispose of the transaction
      acTrans.Commit();
  }
}
```

⊟ *VBA/ActiveX Code Reference*

```vba
Sub CreateRotatedDimension()
    Dim dimObj As AcadDimRotated
```

```
    Dim rotationAngle As Double
    Dim startExtPoint(0 To 2) As Double
    Dim endExtPoint(0 To 2) As Double
    Dim dimLinePoint(0 To 2) As Double

    ' Define the dimension
    rotationAngle = 0.707
    startExtPoint(0) = 0: startExtPoint(1) = 0: startExtPoint(2) = 0
    endExtPoint(0) = 6: endExtPoint(1) = 3: endExtPoint(2) = 0
    dimLinePoint(0) = 0: dimLinePoint(1) = 5: dimLinePoint(2) = 0

    ' Create the rotated dimension in Model space
    Set dimObj = ThisDrawing.ModelSpace. _
                              AddDimRotated(startExtPoint, endExtPoint, _
                                            dimLinePoint, rotationAngle)

    ZoomAll
End Sub
```

Create Radial Dimensions

Radial dimensions measure the radii and diameters of arcs and circles. Radial and diametric dimensions are created by creaing instances of RadialDimension and DiametricDimension objects.

Different types of radial dimensions are created depending on the size of the circle or arc, the position of the dimension text, and the values in the DIMUPT, DIMTOFL, DIMFIT, DIMTIH, DIMTOH, DIMJUST, and DIMTAD dimension system variables. (System variables can be queried or set using the GetSystemVariable and SetSystemVariable methods.)

For horizontal dimension text, if the angle of the dimension line is more than 15 degrees from horizontal, and is outside the circle or arc, AutoCAD draws a hook line, also called a landing or dogleg. The hook line is placed next to or belowthe dimension text, as shown in the following illustrations:
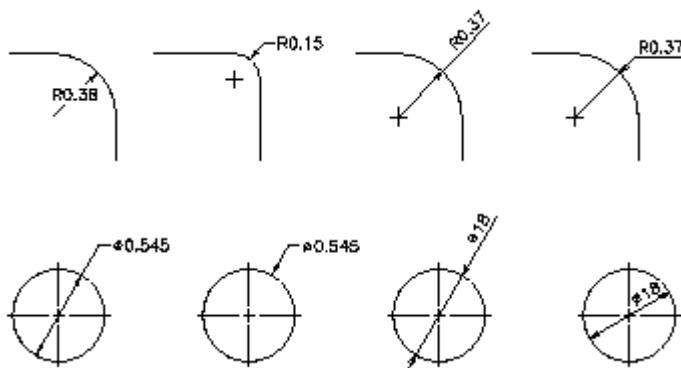
When you create an instance of a RadialDimension object, you have the option to specify the center and chord points, the length of the leader, dimension text, and the dimension style to apply. Creating a DiametricDimension object is similar to a RadialDimension object except you specify chord and far chord points instead of a center and chord point.

The LeaderLength property specifies the distance from the ChordPoint to the annotation text (or stop if no hook line is necessary).

For additional information about creating radial dimensions, see "Create Radial Dimensions" in the *AutoCAD User's Guide*.

Create a radial dimension

This example creates a radial dimension in Model space.

VB.NET

```
Imports Autodesk.AutoCAD.Runtime
Imports Autodesk.AutoCAD.ApplicationServices
Imports Autodesk.AutoCAD.DatabaseServices
Imports Autodesk.AutoCAD.Geometry

<CommandMethod("CreateRadialDimension")> _
Public Sub CreateRadialDimension()
  '' Get the current database
  Dim acDoc As Document = Application.DocumentManager.MdiActiveDocument
  Dim acCurDb As Database = acDoc.Database

  '' Start a transaction
  Using acTrans As Transaction = acCurDb.TransactionManager.StartTransaction()

      '' Open the Block table for read
      Dim acBlkTbl As BlockTable
      acBlkTbl = acTrans.GetObject(acCurDb.BlockTableId, _
                                   OpenMode.ForRead)

      '' Open the Block table record Model space for write
      Dim acBlkTblRec As BlockTableRecord
      acBlkTblRec = acTrans.GetObject(acBlkTbl(BlockTableRecord.ModelSpace), _
                                      OpenMode.ForWrite)

      '' Create the radial dimension
      Dim acRadDim As RadialDimension = New RadialDimension()
      acRadDim.SetDatabaseDefaults()
      acRadDim.Center = New Point3d(0, 0, 0)
      acRadDim.ChordPoint = New Point3d(5, 5, 0)
      acRadDim.LeaderLength = 5
      acRadDim.DimensionStyle = acCurDb.Dimstyle

      '' Add the new object to Model space and the transaction
      acBlkTblRec.AppendEntity(acRadDim)
      acTrans.AddNewlyCreatedDBObject(acRadDim, True)

      '' Commit the changes and dispose of the transaction
      acTrans.Commit()
  End Using
End Sub
```

C#

```
using Autodesk.AutoCAD.Runtime;
using Autodesk.AutoCAD.ApplicationServices;
using Autodesk.AutoCAD.DatabaseServices;
using Autodesk.AutoCAD.Geometry;

[CommandMethod("CreateRadialDimension")]
public static void CreateRadialDimension()
{
  // Get the current database
  Document acDoc = Application.DocumentManager.MdiActiveDocument;
```

```
    Database acCurDb = acDoc.Database;

    // Start a transaction
    using (Transaction acTrans = acCurDb.TransactionManager.StartTransaction())
    {
        // Open the Block table for read
        BlockTable acBlkTbl;
        acBlkTbl = acTrans.GetObject(acCurDb.BlockTableId,
                                     OpenMode.ForRead) as BlockTable;

        // Open the Block table record Model space for write
        BlockTableRecord acBlkTblRec;
        acBlkTblRec = acTrans.GetObject(acBlkTbl[BlockTableRecord.ModelSpace],
                                        OpenMode.ForWrite) as BlockTableRecord;

        // Create the radial dimension
        RadialDimension acRadDim = new RadialDimension();
        acRadDim.SetDatabaseDefaults();
        acRadDim.Center = new Point3d(0, 0, 0);
        acRadDim.ChordPoint = new Point3d(5, 5, 0);
        acRadDim.LeaderLength = 5;
        acRadDim.DimensionStyle = acCurDb.Dimstyle;

        // Add the new object to Model space and the transaction
        acBlkTblRec.AppendEntity(acRadDim);
        acTrans.AddNewlyCreatedDBObject(acRadDim, true);

        // Commit the changes and dispose of the transaction
        acTrans.Commit();
    }
}
```

⊟ *VBA/ActiveX Code Reference*

```
Sub CreateRadialDimension()
    Dim dimObj As AcadDimRadial
    Dim center(0 To 2) As Double
    Dim chordPoint(0 To 2) As Double
    Dim leaderLen As Integer

    ' Define the dimension
    center(0) = 0
    center(1) = 0
    center(2) = 0
    chordPoint(0) = 5
    chordPoint(1) = 5
    chordPoint(2) = 0
    leaderLen = 5

    ' Create the radial dimension in model space
    Set dimObj = ThisDrawing.ModelSpace. _
                             AddDimRadial(center, chordPoint, leaderLen)
    ZoomAll
End Sub
```

Create Angular Dimensions

Angular dimensions measure the angle between two lines or three points. For example, you can use them to measure the angle between two radii of a circle. The dimension line forms an arc. Angular dimensioms are created by creating instances of LineAngularDimension2 or Point3AngularDimension objects.

- **LineAngularDimension2**. Represents an angular dimension defined by two lines.
- **Point3AngularDimension**. Represents an angular dimension defined by three points.

When you create an instance of a LineAngularDimension2 or Point3AngularDimension object, the constructors can optionally accept a set parameters. The following parameters can be supplied when you create a new LineAngularDimension2 object:

- Extension line 1 start point (XLine1Start property)
- Extension line 1 end point (XLine1End property)
- Extension line 2 start point (XLine2Start property)
- Extension line 1 end point (XLine2End property)
- Arc point (ArcPoint property)
- Dimension text (DimensionText property)
- Dimension style (DimensionStyleName or DimensionStyle properties)

The following parameters can be supplied when you create a new Point3AngularDimension object:

- Center point (CenterPoint property)
- Extension line 1point (XLine1Point property)
- Extension line 2 point (XLine2Point property)
- Arc point (ArcPoint property)
- Dimension text (DimensionText property)
- Dimension style (DimensionStyleName or DimensionStyle properties)

For additional information about creating angular dimensions, see "Create Angular Dimensions" in the *AutoCAD User's Guide*.

Create an angular dimension

This example creates an angular dimension in model space.

VB.NET

```
Imports Autodesk.AutoCAD.Runtime
Imports Autodesk.AutoCAD.ApplicationServices
Imports Autodesk.AutoCAD.DatabaseServices
Imports Autodesk.AutoCAD.Geometry

<CommandMethod("CreateAngularDimension")> _
Public Sub CreateAngularDimension()
  '' Get the current database
  Dim acDoc As Document = Application.DocumentManager.MdiActiveDocument
  Dim acCurDb As Database = acDoc.Database

  '' Start a transaction
```

```
   Using acTrans As Transaction = acCurDb.TransactionManager.StartTransaction()

        '' Open the Block table for read
        Dim acBlkTbl As BlockTable
        acBlkTbl = acTrans.GetObject(acCurDb.BlockTableId, _
                                     OpenMode.ForRead)

        '' Open the Block table record Model space for write
        Dim acBlkTblRec As BlockTableRecord
        acBlkTblRec = acTrans.GetObject(acBlkTbl(BlockTableRecord.ModelSpace), _
                                        OpenMode.ForWrite)

        '' Create an angular dimension
        Dim acLinAngDim As LineAngularDimension2 = New LineAngularDimension2()
        acLinAngDim.SetDatabaseDefaults()
        acLinAngDim.XLine1Start = New Point3d(0, 5, 0)
        acLinAngDim.XLine1End = New Point3d(1, 7, 0)
        acLinAngDim.XLine2Start = New Point3d(0, 5, 0)
        acLinAngDim.XLine2End = New Point3d(1, 3, 0)
        acLinAngDim.ArcPoint = New Point3d(3, 5, 0)
        acLinAngDim.DimensionStyle = acCurDb.Dimstyle

        '' Add the new object to Model space and the transaction
        acBlkTblRec.AppendEntity(acLinAngDim)
        acTrans.AddNewlyCreatedDBObject(acLinAngDim, True)

        '' Commit the changes and dispose of the transaction
        acTrans.Commit()
    End Using
End Sub
```

C#

```csharp
using Autodesk.AutoCAD.Runtime;
using Autodesk.AutoCAD.ApplicationServices;
using Autodesk.AutoCAD.DatabaseServices;
using Autodesk.AutoCAD.Geometry;

[CommandMethod("CreateAngularDimension")]
public static void CreateAngularDimension()
{
  // Get the current database
  Document acDoc = Application.DocumentManager.MdiActiveDocument;
  Database acCurDb = acDoc.Database;

  // Start a transaction
  using (Transaction acTrans = acCurDb.TransactionManager.StartTransaction())
  {
      // Open the Block table for read
      BlockTable acBlkTbl;
      acBlkTbl = acTrans.GetObject(acCurDb.BlockTableId,
                                   OpenMode.ForRead) as BlockTable;

      // Open the Block table record Model space for write
      BlockTableRecord acBlkTblRec;
      acBlkTblRec = acTrans.GetObject(acBlkTbl[BlockTableRecord.ModelSpace],
                                      OpenMode.ForWrite) as BlockTableRecord;

      // Create an angular dimension
      LineAngularDimension2 acLinAngDim = new LineAngularDimension2();
      acLinAngDim.SetDatabaseDefaults();
      acLinAngDim.XLine1Start = new Point3d(0, 5, 0);
      acLinAngDim.XLine1End = new Point3d(1, 7, 0);
      acLinAngDim.XLine2Start = new Point3d(0, 5, 0);
```

```
            acLinAngDim.XLine2End = new Point3d(1, 3, 0);
            acLinAngDim.ArcPoint = new Point3d(3, 5, 0);
            acLinAngDim.DimensionStyle = acCurDb.Dimstyle;

            // Add the new object to Model space and the transaction
            acBlkTblRec.AppendEntity(acLinAngDim);
            acTrans.AddNewlyCreatedDBObject(acLinAngDim, true);

            // Commit the changes and dispose of the transaction
            acTrans.Commit();
    }
}
```

 VBA/ActiveX Code Reference

```
Sub CreateAngularDimension()
    Dim dimObj As AcadDimAngular
    Dim angVert(0 To 2) As Double
    Dim FirstPoint(0 To 2) As Double
    Dim SecondPoint(0 To 2) As Double
    Dim TextPoint(0 To 2) As Double

    ' Define the dimension
    angVert(0) = 0
    angVert(1) = 5
    angVert(2) = 0
    FirstPoint(0) = 1
    FirstPoint(1) = 7
    FirstPoint(2) = 0
    SecondPoint(0) = 1
    SecondPoint(1) = 3
    SecondPoint(2) = 0
    TextPoint(0) = 3
    TextPoint(1) = 5
    TextPoint(2) = 0

    ' Create the angular dimension in model space
    Set dimObj = ThisDrawing.ModelSpace. _
                    AddDimAngular(angVert, FirstPoint, SecondPoint, TextPoint)
    ZoomAll
End Sub
```

Create Jogged Radius Dimensions

Jogged radius dimensions measure the radius of an object and displays the dimension text with a radius symbol in front of it. You might use a jogged dimension over a radial dimension object when:

- An object's center point is located outside of the layout or is over an area of the model that does not have enough room for a radial dimension
- An object has a large radius

You create a jogged radius dimenion by creating an instance of a RadialDimensionLarge object. When you create an instance of a RadialDimensionLarge object, its constructors can

optionally accept a set parameters. The following parameters can be supplied when you create a new RadialDimensionLarge object:

- Center point (Center property)
- Chord point (ChordPoint property)
- Override center point (OverrideCenter property)
- Position of the jog lines (JogPoint property)
- Angle of the jog lines (JogAngle property)
- Dimension text (DimensionText property)
- Dimension style (DimensionStyleName or DimensionStyle properties)

For additional information about creating jogged radius dimensions, see "Create Radial Dimensions" in the *AutoCAD User's Guide*.

Create a jogged radius dimension

This example creates a jogged radius dimension in Model space.

VB.NET

```
Imports Autodesk.AutoCAD.Runtime
Imports Autodesk.AutoCAD.ApplicationServices
Imports Autodesk.AutoCAD.DatabaseServices
Imports Autodesk.AutoCAD.Geometry

<CommandMethod("CreateJoggedDimension")> _
Public Sub CreateJoggedDimension()
  '' Get the current database
  Dim acDoc As Document = Application.DocumentManager.MdiActiveDocument
  Dim acCurDb As Database = acDoc.Database

  '' Start a transaction
  Using acTrans As Transaction = acCurDb.TransactionManager.StartTransaction()

      '' Open the Block table for read
      Dim acBlkTbl As BlockTable
      acBlkTbl = acTrans.GetObject(acCurDb.BlockTableId, _
                                   OpenMode.ForRead)

      '' Open the Block table record Model space for write
      Dim acBlkTblRec As BlockTableRecord
      acBlkTblRec = acTrans.GetObject(acBlkTbl(BlockTableRecord.ModelSpace), _
                                      OpenMode.ForWrite)

      '' Create a large radius dimension
      Dim acRadDimLrg As RadialDimensionLarge = New RadialDimensionLarge()
      acRadDimLrg.SetDatabaseDefaults()
      acRadDimLrg.Center = New Point3d(-3, -4, 0)
      acRadDimLrg.ChordPoint = New Point3d(2, 7, 0)
      acRadDimLrg.OverrideCenter = New Point3d(0, 2, 0)
      acRadDimLrg.JogPoint = New Point3d(1, 4.5, 0)
      acRadDimLrg.JogAngle = 0.707
      acRadDimLrg.DimensionStyle = acCurDb.Dimstyle

      '' Add the new object to Model space and the transaction
      acBlkTblRec.AppendEntity(acRadDimLrg)
      acTrans.AddNewlyCreatedDBObject(acRadDimLrg, True)

      '' Commit the changes and dispose of the transaction
      acTrans.Commit()
```

```
  End Using
End Sub
```

C#

```csharp
using Autodesk.AutoCAD.Runtime;
using Autodesk.AutoCAD.ApplicationServices;
using Autodesk.AutoCAD.DatabaseServices;
using Autodesk.AutoCAD.Geometry;

[CommandMethod("CreateJoggedDimension")]
public static void CreateJoggedDimension()
{
  // Get the current database
  Document acDoc = Application.DocumentManager.MdiActiveDocument;
  Database acCurDb = acDoc.Database;

  // Start a transaction
  using (Transaction acTrans = acCurDb.TransactionManager.StartTransaction())
  {
      // Open the Block table for read
      BlockTable acBlkTbl;
      acBlkTbl = acTrans.GetObject(acCurDb.BlockTableId,
                                   OpenMode.ForRead) as BlockTable;

      // Open the Block table record Model space for write
      BlockTableRecord acBlkTblRec;
      acBlkTblRec = acTrans.GetObject(acBlkTbl[BlockTableRecord.ModelSpace],
                                      OpenMode.ForWrite) as BlockTableRecord;

      // Create a large radius dimension
      RadialDimensionLarge acRadDimLrg = new RadialDimensionLarge();
      acRadDimLrg.SetDatabaseDefaults();
      acRadDimLrg.Center = new Point3d(-3, -4, 0);
      acRadDimLrg.ChordPoint = new Point3d(2, 7, 0);
      acRadDimLrg.OverrideCenter = new Point3d(0, 2, 0);
      acRadDimLrg.JogPoint = new Point3d(1, 4.5, 0);
      acRadDimLrg.JogAngle = 0.707;
      acRadDimLrg.DimensionStyle = acCurDb.Dimstyle;

      // Add the new object to Model space and the transaction
      acBlkTblRec.AppendEntity(acRadDimLrg);
      acTrans.AddNewlyCreatedDBObject(acRadDimLrg, true);

      // Commit the changes and dispose of the transaction
      acTrans.Commit();
  }
}
```

⊟ *VBA/ActiveX Code Reference*

```vba
Sub CreateJoggedDimension()
    Dim dimObj As AcadDimRadialLarge
    Dim centerPoint(0 To 2) As Double
    Dim chordPoint(0 To 2) As Double
    Dim centerOverPoint(0 To 2) As Double
    Dim jogPoint(0 To 2) As Double
    Dim jogAngle As Double

    ' Define the dimension
    centerPoint(0) = -3: centerPoint(1) = -4: centerPoint(2) = 0
    chordPoint(0) = 2: chordPoint(1) = 7: chordPoint(2) = 0
```

```
        centerOverPoint(0) = 0: centerOverPoint(1) = 2: centerOverPoint(2) = 0
        jogPoint(0) = 1: jogPoint(1) = 4.5: jogPoint(2) = 0
        jogAngle = 0.707

        ' Create the jogged dimension in Model space
        Set dimObj = ThisDrawing.ModelSpace. _
                        AddDimRadialLarge(centerPoint, chordPoint, _
                                          centerOverPoint, jogPoint, _
                                          jogAngle)


        ZoomAll
End Sub
```

Create Arc Length Dimensions

Arc length dimensions measure the length along an arcand displays the dimension text with an arc symbol that is either above or proceeds it. You might use an arc length dimension when you need to dimension the actual length of an arc instead of just distance between its start and end points.

You create an arc length radius dimenion by creating an instance of an ArcDimension object. When you create an instance of an ArcDimension object, it requires a set of paramters that define the dimension. The following parameters must be supplied when you create a new ArcDimension object:

- Center point (Center property)
- Extension line 1 point (XLine1Point property)
- Extension line 2 point (XLine2Point property)
- Arc point (ArcPoint property)
- Dimension text (DimensionText property)
- Dimension style (DimensionStyleName or DimensionStyle properties)

Note The DIMARCSYM system variable controls if an arc symbol is displayed and where it is displayed in relation to the dimension text.

For additional information about creating arc length dimensions, see "Create Arc Length Dimensions" in the *AutoCAD User's Guide*.

Create an arc length dimension

This example creates an arc length dimension in Model space.

VB.NET

```
Imports Autodesk.AutoCAD.Runtime
Imports Autodesk.AutoCAD.ApplicationServices
Imports Autodesk.AutoCAD.DatabaseServices
Imports Autodesk.AutoCAD.Geometry

<CommandMethod("CreateArcLengthDimension")> _
Public Sub CreateArcLengthDimension()
  '' Get the current database
  Dim acDoc As Document = Application.DocumentManager.MdiActiveDocument
```

```
    Dim acCurDb As Database = acDoc.Database

    '' Start a transaction
    Using acTrans As Transaction = acCurDb.TransactionManager.StartTransaction()

        '' Open the Block table for read
        Dim acBlkTbl As BlockTable
        acBlkTbl = acTrans.GetObject(acCurDb.BlockTableId, _
                                     OpenMode.ForRead)

        '' Open the Block table record Model space for write
        Dim acBlkTblRec As BlockTableRecord
        acBlkTblRec = acTrans.GetObject(acBlkTbl(BlockTableRecord.ModelSpace), _
                                        OpenMode.ForWrite)

        '' Create an arc length dimension
        Dim acArcDim As ArcDimension = New ArcDimension(New Point3d(4.5, 1.5, 0), _
                                                        New Point3d(8, 4.25, 0), _
                                                        New Point3d(0, 2, 0), _
                                                        New Point3d(5, 7, 0), _
                                                        "<>", _
                                                        acCurDb.Dimstyle)

        acArcDim.SetDatabaseDefaults()

        '' Add the new object to Model space and the transaction
        acBlkTblRec.AppendEntity(acArcDim)
        acTrans.AddNewlyCreatedDBObject(acArcDim, True)

        '' Commit the changes and dispose of the transaction
        acTrans.Commit()
    End Using
End Sub
```

C#

```csharp
using Autodesk.AutoCAD.Runtime;
using Autodesk.AutoCAD.ApplicationServices;
using Autodesk.AutoCAD.DatabaseServices;
using Autodesk.AutoCAD.Geometry;

[CommandMethod("CreateArcLengthDimension")]
public static void CreateArcLengthDimension()
{
    // Get the current database
    Document acDoc = Application.DocumentManager.MdiActiveDocument;
    Database acCurDb = acDoc.Database;

    // Start a transaction
    using (Transaction acTrans = acCurDb.TransactionManager.StartTransaction())
    {
        // Open the Block table for read
        BlockTable acBlkTbl;
        acBlkTbl = acTrans.GetObject(acCurDb.BlockTableId,
                                     OpenMode.ForRead) as BlockTable;

        // Open the Block table record Model space for write
        BlockTableRecord acBlkTblRec;
        acBlkTblRec = acTrans.GetObject(acBlkTbl[BlockTableRecord.ModelSpace],
                                        OpenMode.ForWrite) as BlockTableRecord;

        // Create an arc length dimension
        ArcDimension acArcDim = new ArcDimension(new Point3d(4.5, 1.5, 0),
                                                 new Point3d(8, 4.25, 0),
```

```
                                                    new Point3d(0, 2, 0),
                                                    new Point3d(5, 7, 0),
                                                    "<>",
                                                    acCurDb.Dimstyle);

        acArcDim.SetDatabaseDefaults();

        // Add the new object to Model space and the transaction
        acBlkTblRec.AppendEntity(acArcDim);
        acTrans.AddNewlyCreatedDBObject(acArcDim, true);

        // Commit the changes and dispose of the transaction
        acTrans.Commit();
    }
}
```

⊟ *VBA/ActiveX Code Reference*

```
Sub CreateArcLengthDimension()
    Dim dimObj As AcadDimArcLength
    Dim arcCenterPoint(0 To 2) As Double
    Dim firstPoint(0 To 2) As Double
    Dim secondPoint(0 To 2) As Double
    Dim arcPoint(0 To 2) As Double

    ' Define the dimension
    arcCenterPoint(0) = 4.5: arcCenterPoint(1) = 1.5: arcCenterPoint(2) = 0
    firstPoint(0) = 8: firstPoint(1) = 4.25: firstPoint(2) = 0
    secondPoint(0) = 0: secondPoint(1) = 2: secondPoint(2) = 0
    arcPoint(0) = 5: arcPoint(1) = 7: arcPoint(2) = 0

    ' Create the arc length dimension in Model space
    Set dimObj = ThisDrawing.ModelSpace. _
                    AddDimArc(arcCenterPoint, firstPoint, _
                    secondPoint, arcPoint)

    ZoomAll
End Sub
```

Create Ordinate Dimensions

Ordinate, or datum, dimensions measure the perpendicular distance from an origin point, called the datum, to a dimensioned feature, such as a hole in a part. These dimensions prevent escalating errors by maintaining accurate offsets of the features from the datum.
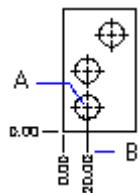
Ordinate dimensions consist of an *X* or *Y* ordinate with a leader line. *X*-datum ordinate dimensions measure the distance of a feature from the datum along the *X* axis. *Y*-datum ordinate dimensions measure the same distance along the *Y* axis. AutoCAD uses the origin

of the current user coordinate system (UCS) to determine the measured coordinates. The absolute value of the coordinate is used.

The dimension text is aligned with the ordinate leader line regardless of the orientation defined by the current dimension style. You can accept the default text or override it with your own.

You create an ordinate dimension by creating an instance of an OrdinateDimension object. When you create an instance of an OrdinateDimension object, its constructor can accept an optional set of parameters. The following parameters can be supplied when you create a new OrdinateDimension object:

- Use X axis (UsingXAxis property)
- Defining point (DefiningPoint property)
- Leader endpoint (LeaderEndPoint property)
- Dimension text (DimensionText property)
- Dimension style (DimensionStyleName or DimensionStyle properties)

When passing values into the OrdinateDimension object constructor, the first value is a boolean flag which specifies whether the dimension is an *X*-datum or *Y*-datum ordinate dimension. If you enter TRUE, an *X*-datum ordinate dimension is created. If you enter FALSE, a *Y*-datum ordinate dimension is created. The UsingXAxis property can also be used to specify if an ordinate dimension is an *X*-datum or *Y*-datum.

For additional information about creating ordinate dimensions, see "Create Ordinate Dimensions" in the *AutoCAD User's Guide*.

Create an ordinate dimension

This example creates an ordinate dimension in Model space.

VB.NET

```
Imports Autodesk.AutoCAD.Runtime
Imports Autodesk.AutoCAD.ApplicationServices
Imports Autodesk.AutoCAD.DatabaseServices
Imports Autodesk.AutoCAD.Geometry

<CommandMethod("CreateOrdinateDimension")> _
Public Sub CreateOrdinateDimension()
  '' Get the current database
  Dim acDoc As Document = Application.DocumentManager.MdiActiveDocument
  Dim acCurDb As Database = acDoc.Database

  '' Start a transaction
  Using acTrans As Transaction = acCurDb.TransactionManager.StartTransaction()

    '' Open the Block table for read
    Dim acBlkTbl As BlockTable
    acBlkTbl = acTrans.GetObject(acCurDb.BlockTableId, _
                                 OpenMode.ForRead)

    '' Open the Block table record Model space for write
    Dim acBlkTblRec As BlockTableRecord
    acBlkTblRec = acTrans.GetObject(acBlkTbl(BlockTableRecord.ModelSpace), _
                                    OpenMode.ForWrite)

    '' Create an ordinate dimension
    Dim acOrdDim As OrdinateDimension = New OrdinateDimension()
```

```
        acOrdDim.SetDatabaseDefaults()
        acOrdDim.UsingXAxis = True
        acOrdDim.DefiningPoint = New Point3d(5, 5, 0)
        acOrdDim.LeaderEndPoint = New Point3d(10, 5, 0)
        acOrdDim.DimensionStyle = acCurDb.Dimstyle

        '' Add the new object to Model space and the transaction
        acBlkTblRec.AppendEntity(acOrdDim)
        acTrans.AddNewlyCreatedDBObject(acOrdDim, True)

        '' Commit the changes and dispose of the transaction
        acTrans.Commit()
    End Using
End Sub
```

C#

```
using Autodesk.AutoCAD.Runtime;
using Autodesk.AutoCAD.ApplicationServices;
using Autodesk.AutoCAD.DatabaseServices;
using Autodesk.AutoCAD.Geometry;

[CommandMethod("CreateOrdinateDimension")]
public static void CreateOrdinateDimension()
{
    // Get the current database
    Document acDoc = Application.DocumentManager.MdiActiveDocument;
    Database acCurDb = acDoc.Database;

    // Start a transaction
    using (Transaction acTrans = acCurDb.TransactionManager.StartTransaction())
    {
        // Open the Block table for read
        BlockTable acBlkTbl;
        acBlkTbl = acTrans.GetObject(acCurDb.BlockTableId,
                                     OpenMode.ForRead) as BlockTable;

        // Open the Block table record Model space for write
        BlockTableRecord acBlkTblRec;
        acBlkTblRec = acTrans.GetObject(acBlkTbl[BlockTableRecord.ModelSpace],
                                        OpenMode.ForWrite) as BlockTableRecord;

        // Create an ordinate dimension
        OrdinateDimension acOrdDim = new OrdinateDimension();
        acOrdDim.SetDatabaseDefaults();
        acOrdDim.UsingXAxis = true;
        acOrdDim.DefiningPoint = new Point3d(5, 5, 0);
        acOrdDim.LeaderEndPoint = new Point3d(10, 5, 0);
        acOrdDim.DimensionStyle = acCurDb.Dimstyle;

        // Add the new object to Model space and the transaction
        acBlkTblRec.AppendEntity(acOrdDim);
        acTrans.AddNewlyCreatedDBObject(acOrdDim, true);

        // Commit the changes and dispose of the transaction
        acTrans.Commit();
    }
}
```

⊟ *VBA/ActiveX Code Reference*

```
Sub CreateOrdinateDimension()
```

```
        Dim dimObj As AcadDimOrdinate
        Dim definingPoint(0 To 2) As Double
        Dim leaderEndPoint(0 To 2) As Double
        Dim useXAxis As Boolean

        ' Define the dimension
        definingPoint(0) = 5
        definingPoint(1) = 5
        definingPoint(2) = 0
        leaderEndPoint(0) = 10
        leaderEndPoint(1) = 5
        leaderEndPoint(2) = 0
        useXAxis = True

        ' Create an ordinate dimension in model space
        Set dimObj = ThisDrawing.ModelSpace. _
                        AddDimOrdinate(definingPoint, _
                        leaderEndPoint, useXAxis)

        ZoomAll
End Sub
```

Edit Dimensions

As with other graphical objects in AutoCAD, you can edit dimensions using the methods and properties provided for the object.

The following properties are available for most dimension objects:

DimensionStyle

Specifies the object id of the dimension style.

DimensionStyleName

Specifies the name of the dimension style.

DimensionText

Specifies the user-defined text string for the dimension.

HorizontalRotation

Specifies the rotation angle in radians for the dimension.

Measurement

Specifies the actual measurement for the dimension.

TextPosition

Specifies the dimension text position.

TextRotation

Specifies the rotation angle of the dimension text.

For more information about editing dimensions, see "Modify Existing Dimensions" in the *AutoCAD User's Guide.*

In addition, to modifying a dimension object using its specific properties and methods you can also copy and transform dimension objects. For information on copying and transforming objects, see [Edit Named and 2D Objects](#).

Topics in this section

- [Override Dimension Text](#)

Override Dimension Text

The dimension value that is displayed can be replaced using the DimensionText property. Using this property you can completely replace the displayed value of the dimension, or you can append text to the value. To represent the measured value in the override dimension text, use the character string "<>" in the text.

Modify dimension text

This example appends some text to the value so that both the string and the dimension value are displayed.

VB.NET

```
Imports Autodesk.AutoCAD.Runtime
Imports Autodesk.AutoCAD.ApplicationServices
Imports Autodesk.AutoCAD.DatabaseServices
Imports Autodesk.AutoCAD.Geometry

<CommandMethod("OverrideDimensionText")> _
Public Sub OverrideDimensionText()
  '' Get the current database
  Dim acDoc As Document = Application.DocumentManager.MdiActiveDocument
  Dim acCurDb As Database = acDoc.Database

  '' Start a transaction
  Using acTrans As Transaction = acCurDb.TransactionManager.StartTransaction()

      '' Open the Block table for read
      Dim acBlkTbl As BlockTable
      acBlkTbl = acTrans.GetObject(acCurDb.BlockTableId, _
                                   OpenMode.ForRead)

      '' Open the Block table record Model space for write
      Dim acBlkTblRec As BlockTableRecord
      acBlkTblRec = acTrans.GetObject(acBlkTbl(BlockTableRecord.ModelSpace), _
                                      OpenMode.ForWrite)

      '' Create the aligned dimension
      Dim acAliDim As AlignedDimension = New AlignedDimension()
```

```vbnet
        acAliDim.SetDatabaseDefaults()
        acAliDim.XLine1Point = New Point3d(5, 3, 0)
        acAliDim.XLine2Point = New Point3d(10, 3, 0)
        acAliDim.DimLinePoint = New Point3d(7.5, 5, 0)
        acAliDim.DimensionStyle = acCurDb.Dimstyle

        '' Override the dimension text
        acAliDim.DimensionText = "The value is <>"

        '' Add the new object to Model space and the transaction
        acBlkTblRec.AppendEntity(acAliDim)
        acTrans.AddNewlyCreatedDBObject(acAliDim, True)

        '' Commit the changes and dispose of the transaction
        acTrans.Commit()
    End Using
End Sub
```

C#

```csharp
using Autodesk.AutoCAD.Runtime;
using Autodesk.AutoCAD.ApplicationServices;
using Autodesk.AutoCAD.DatabaseServices;
using Autodesk.AutoCAD.Geometry;

[CommandMethod("OverrideDimensionText")]
public static void OverrideDimensionText()
{
  // Get the current database
  Document acDoc = Application.DocumentManager.MdiActiveDocument;
  Database acCurDb = acDoc.Database;

  // Start a transaction
  using (Transaction acTrans = acCurDb.TransactionManager.StartTransaction())
  {
      // Open the Block table for read
      BlockTable acBlkTbl;
      acBlkTbl = acTrans.GetObject(acCurDb.BlockTableId,
                                   OpenMode.ForRead) as BlockTable;

      // Open the Block table record Model space for write
      BlockTableRecord acBlkTblRec;
      acBlkTblRec = acTrans.GetObject(acBlkTbl[BlockTableRecord.ModelSpace],
                                      OpenMode.ForWrite) as BlockTableRecord;

      // Create the aligned dimension
      AlignedDimension acAliDim = new AlignedDimension();
      acAliDim.SetDatabaseDefaults();
      acAliDim.XLine1Point = new Point3d(5, 3, 0);
      acAliDim.XLine2Point = new Point3d(10, 3, 0);
      acAliDim.DimLinePoint = new Point3d(7.5, 5, 0);
      acAliDim.DimensionStyle = acCurDb.Dimstyle;

      // Override the dimension text
      acAliDim.DimensionText = "The value is <>";

      // Add the new object to Model space and the transaction
      acBlkTblRec.AppendEntity(acAliDim);
      acTrans.AddNewlyCreatedDBObject(acAliDim, true);

      // Commit the changes and dispose of the transaction
      acTrans.Commit();
  }
}
```

```
Sub OverrideDimensionText()
    Dim dimObj As AcadDimAligned
    Dim point1(0 To 2) As Double
    Dim point2(0 To 2) As Double
    Dim location(0 To 2) As Double

    ' Define the dimension
    point1(0) = 5#: point1(1) = 3#: point1(2) = 0#
    point2(0) = 10#: point2(1) = 3#: point2(2) = 0#
    location(0) = 7.5: location(1) = 5#: location(2) = 0#

    ' Create an aligned dimension object in model space
    Set dimObj = ThisDrawing.ModelSpace. _
                    AddDimAligned(point1, point2, location)

    ' Change the text string for the dimension
    dimObj.TextOverride = "The value is <>"
    dimObj.Update
End Sub
```

Work with Dimension Styles

A named dimension style is a group of settings that determines the appearance of a dimension. Using named dimension styles, you can establish and enforce drafting standards for the dimensions in a drawing.

All dimensions are created using the active dimension style. If you do not define or apply a style before creating dimensions, AutoCAD applies the default style, STANDARD. To set the active dimension style, use the Dimstyle property of the current database.

To set up a dimension style, you begin by naming and saving a style. The new style is based on the current style and includes all the settings that define the parts of the dimensions, the positioning of text, and the appearance of annotation. Annotation in this case means primary and alternate units, tolerances, and text.

For more information about dimension styles, see "Use Dimension Styles" in the *AutoCAD User's Guide*.

Topics in this section

- Create, Modify, and Copy Dimension Styles
- Override the Dimension Style

Create, Modify, and Copy Dimension Styles

A new dimension style is created by creating an instance of a DimStyleTableRecord object and then adding it to the DimStyleTable with the Add method. Before the dimension style is added to the table, set the name of the new style with the Name property.

You can also copy an existing style or a style with overrides. Use the CopyFrom method to copy a dimension style from a source object to a dimension style. The source object can be another DimStyleTableRecord object, a Dimension, Tolerance, or Leader object, or even a Database object. If you copy the style settings from another dimension style, the current style is duplicated exactly. If you copy the style settings from aDimension, Tolerance, or Leader object, the current settings, including any object overrides, are copied to the style. If you copy the current style of a Database object, the dimension style plus any drawing overrides are copied to the new style.

Copy dimension styles and overrides

This example creates three new dimension styles and copies the current settings from the current Database, a given dimension style, and a given dimension to each new dimension style respectively. By following the appropriate setup before running this example, you will find that different dimension styles have been created.

1. Create a new drawing and make it the active drawing.
2. Create a linear dimension in the new drawing. This dimension should be the only object in the drawing.
3. Change the color of the dimension line to yellow.
4. Change the DIMCLRD system variable to 5 (blue).
5. Run the following example:

VB.NET

```
Imports Autodesk.AutoCAD.Runtime
Imports Autodesk.AutoCAD.ApplicationServices
Imports Autodesk.AutoCAD.DatabaseServices
Imports Autodesk.AutoCAD.Geometry

<CommandMethod("CopyDimStyles")> _
Public Sub CopyDimStyles()
  '' Get the current database
  Dim acDoc As Document = Application.DocumentManager.MdiActiveDocument
  Dim acCurDb As Database = acDoc.Database

  '' Start a transaction
  Using acTrans As Transaction = acCurDb.TransactionManager.StartTransaction()

      '' Open the Block table for read
      Dim acBlkTbl As BlockTable
      acBlkTbl = acTrans.GetObject(acCurDb.BlockTableId, _
                                   OpenMode.ForRead)

      '' Open the Block table record Model space for read
      Dim acBlkTblRec As BlockTableRecord
      acBlkTblRec = acTrans.GetObject(acBlkTbl(BlockTableRecord.ModelSpace), _
                                      OpenMode.ForRead)

      Dim acObj As Object = Nothing
      For Each acObjId As ObjectId In acBlkTblRec
```

```vbnet
            '' Get the first object in Model space
            acObj = acTrans.GetObject(acObjId, _
                                      OpenMode.ForRead)

            Exit For
        Next

        '' Open the DimStyle table for read
        Dim acDimStyleTbl As DimStyleTable
        acDimStyleTbl = acTrans.GetObject(acCurDb.DimStyleTableId, _
                                          OpenMode.ForRead)

        Dim strDimStyleNames(2) As String
        strDimStyleNames(0) = "Style 1 copied from a dim"
        strDimStyleNames(1) = "Style 2 copied from Style 1"
        strDimStyleNames(2) = "Style 3 copied from the running drawing values"

        Dim nCnt As Integer = 0

        '' Keep a reference of the first dimension style for later
        Dim acDimStyleTblRec1 As DimStyleTableRecord = Nothing

        '' Iterate the array of dimension style names
        For Each strDimStyleName As String In strDimStyleNames
            Dim acDimStyleTblRec As DimStyleTableRecord
            Dim acDimStyleTblRecCopy As DimStyleTableRecord = Nothing

            '' Check to see if the dimension style exists or not
            If acDimStyleTbl.Has(strDimStyleName) = False Then
                If acDimStyleTbl.IsWriteEnabled = False
ThenacDimStyleTbl.UpgradeOpen()

                acDimStyleTblRec = New DimStyleTableRecord()
                acDimStyleTblRec.Name = strDimStyleName

                acDimStyleTbl.Add(acDimStyleTblRec)
                acTrans.AddNewlyCreatedDBObject(acDimStyleTblRec, True)
            Else
                acDimStyleTblRec = acTrans.GetObject(acDimStyleTbl(strDimStyleName), _
                                                     OpenMode.ForWrite)
            End If

            '' Determine how the new dimension style is populated
            Select Case nCnt
                '' Assign the values of the dimension object to the new dimension
style
                Case 0
                    Try
                        '' Cast the object to a Dimension
                        Dim acDim As RotatedDimension = acObj

                        '' Copy the dimension style data from the dimension and
                        '' set the name of the dimension style as the copied
settings
                        '' are unnamed.
                        acDimStyleTblRecCopy = acDim.GetDimstyleData()
                        acDimStyleTblRec1 = acDimStyleTblRec
                    Catch
                        '' Object was not a dimension
                    End Try

                    '' Assign the values of the dimension style to the new dimension
style
                Case 1
```

```vb
            acDimStyleTblRecCopy = acDimStyleTblRec1

                '' Assign the values of the current drawing to the dimension
style
            Case 2
                acDimStyleTblRecCopy = acCurDb.GetDimstyleData()
        End Select

        '' Copy the dimension settings and set the name of the dimension style
        acDimStyleTblRec.CopyFrom(acDimStyleTblRecCopy)
        acDimStyleTblRec.Name = strDimStyleName

        '' Dispose of the copied dimension style
        acDimStyleTblRecCopy.Dispose()

        nCnt = nCnt + 1
    Next

    '' Commit the changes and dispose of the transaction
    acTrans.Commit()
  End Using
End Sub
```

C#

```csharp
using Autodesk.AutoCAD.Runtime;
using Autodesk.AutoCAD.ApplicationServices;
using Autodesk.AutoCAD.DatabaseServices;
using Autodesk.AutoCAD.Geometry;

[CommandMethod("CopyDimStyles")]
public static void CopyDimStyles()
{
  // Get the current database
  Document acDoc = Application.DocumentManager.MdiActiveDocument;
  Database acCurDb = acDoc.Database;

  // Start a transaction
  using (Transaction acTrans = acCurDb.TransactionManager.StartTransaction())
  {
      // Open the Block table for read
      BlockTable acBlkTbl;
      acBlkTbl = acTrans.GetObject(acCurDb.BlockTableId,
                                   OpenMode.ForRead) as BlockTable;

      // Open the Block table record Model space for read
      BlockTableRecord acBlkTblRec;
      acBlkTblRec = acTrans.GetObject(acBlkTbl[BlockTableRecord.ModelSpace],
                                      OpenMode.ForRead) as BlockTableRecord;

      object acObj = null;
      foreach (ObjectId acObjId in acBlkTblRec)
      {
          // Get the first object in Model space
          acObj = acTrans.GetObject(acObjId,
                                    OpenMode.ForRead);

          break;
      }

      // Open the DimStyle table for read
      DimStyleTable acDimStyleTbl;
      acDimStyleTbl = acTrans.GetObject(acCurDb.DimStyleTableId,
                                        OpenMode.ForRead) as DimStyleTable;
```

```csharp
        string[] strDimStyleNames = new string[3];
        strDimStyleNames[0] = "Style 1 copied from a dim";
        strDimStyleNames[1] = "Style 2 copied from Style 1";
        strDimStyleNames[2] = "Style 3 copied from the running drawing values";

        int nCnt = 0;

        // Keep a reference of the first dimension style for later
        DimStyleTableRecord acDimStyleTblRec1 = null;

        // Iterate the array of dimension style names
        foreach (string strDimStyleName in strDimStyleNames)
        {
            DimStyleTableRecord acDimStyleTblRec;
            DimStyleTableRecord acDimStyleTblRecCopy = null;

            // Check to see if the dimension style exists or not
            if (acDimStyleTbl.Has(strDimStyleName) == false)
            {
                if (acDimStyleTbl.IsWriteEnabled == false)
acDimStyleTbl.UpgradeOpen();

                acDimStyleTblRec = new DimStyleTableRecord();
                acDimStyleTblRec.Name = strDimStyleName;

                acDimStyleTbl.Add(acDimStyleTblRec);
                acTrans.AddNewlyCreatedDBObject(acDimStyleTblRec, true);
            }
            else
            {
                acDimStyleTblRec = acTrans.GetObject(acDimStyleTbl[strDimStyleName],
                                                     OpenMode.ForWrite) as
DimStyleTableRecord;
            }

            // Determine how the new dimension style is populated
            switch ((int)nCnt)
            {
                // Assign the values of the dimension object to the new dimension
style
                case 0:
                    try
                    {
                        // Cast the object to a Dimension
                        Dimension acDim = acObj as Dimension;

                        // Copy the dimension style data from the dimension and
                        // set the name of the dimension style as the copied
settings
                        // are unnamed.
                        acDimStyleTblRecCopy = acDim.GetDimstyleData();
                        acDimStyleTblRec1 = acDimStyleTblRec;
                    }
                    catch
                    {
                        // Object was not a dimension
                    }

                    break;
                // Assign the values of the dimension style to the new dimension
style
                case 1:
                    acDimStyleTblRecCopy = acDimStyleTblRec1;
                    break;
```

```
            // Assign the values of the current drawing to the dimension style
            case 2:
                acDimStyleTblRecCopy = acCurDb.GetDimstyleData();
                break;
        }

        // Copy the dimension settings and set the name of the dimension style
        acDimStyleTblRec.CopyFrom(acDimStyleTblRecCopy);
        acDimStyleTblRec.Name = strDimStyleName;

        // Dispose of the copied dimension style
        acDimStyleTblRecCopy.Dispose();

        nCnt = nCnt + 1;
    }

    // Commit the changes and dispose of the transaction
    acTrans.Commit();
    }
}
```

```
Sub CopyDimStyles()
    Dim newStyle1 As AcadDimStyle
    Dim newStyle2 As AcadDimStyle
    Dim newStyle3 As AcadDimStyle

    Set newStyle1 = ThisDrawing.DimStyles. _
                        Add("Style 1 copied from a dim")
    Call newStyle1.CopyFrom(ThisDrawing.ModelSpace(0))

    Set newStyle2 = ThisDrawing.DimStyles. _
                        Add("Style 2 copied from Style 1")
    Call newStyle2.CopyFrom(ThisDrawing.DimStyles. _
                                Item("Style 1 copied from a dim"))

    Set newStyle2 = ThisDrawing.DimStyles. _
                        Add("Style 3 copied from the running drawing values")
    Call newStyle2.CopyFrom(ThisDrawing)
End Sub
```

Open the Dimension Style Manager using the DIMSTYLE command. You should now have three dimension styles listed. Style 1 should have a yellow dimension line. Style 2 should be the same as Style 1. Style 3 should have a blue dimension line.

Override the Dimension Style

Each dimension has the capability of overriding the settings assigned to it by a dimension style. The following properties are available for most dimension objects:

Dimatfit

Specifies the display of dimension lines inside extension lines only, and forces dimension text and arrowheads inside or outside extension lines.

Dimaltrnd

Specifies the rounding of alternate units.

Dimasz

Specifies the size of dimension line arrowheads, leader line arrowheads, and hook lines.

Dimaunit

Specifies the unit format for angular dimensions.

Dimblk1, **Dimblk2**

Specifies the blocks to use for arrowheads of the dimension line.

Dimcen

Specifies the type and size of center mark for radial and diametric dimensions.

Dimclrd

Specifies the color of the dimension line for a dimension, leader, or tolerance object.

Dimclre

Specifies the color for dimension extension lines.

Dimclrt

Specifies the color of the text for dimension and tolerance objects.

Dimdec

Specifies the number of decimal places displayed for the primary units of a dimension or tolerance.

Dimdsep

Specifies the character to be used as the decimal separator in decimal dimension and tolerance values.

Dimexe

Specifies the distance the extension line extends beyond the dimension line.

Dimexo

Specifies the distance the extension lines are offset from the origin points.

Dimfrac

Specifies the format of fractional values in dimensions and tolerances.

Dimgap

Specifies the distance between the dimension text and the dimension line when you break the dimension line to accommodate dimension text.

Dimlfac

Specifies a global scale factor for linear dimension measurements.

Dimltex1, Dimltex2

Specifies the linetypes for the extension lines.

Dimlwd

Specifies the lineweight for the dimension line.

Dimlwe

Specifies the lineweight for the extension lines.

Dimjust

Specifies the horizontal justification for dimension text.

Dimrnd

Specifies the distance rounding for dimension measurements.

Dimsd1, Dimsd2

Specifies the suppression of the dimension lines.

Dimse1, Dimse2

Specifies the suppression of extension lines.

Dimtad

Specifies the vertical position of text in relation to the dimension line.

Dimtdec

Specifies the precision of tolerance values in primary dimensions.

Dimtfac

Specifies a scale factor for the text height of tolerance values relative to the dimension text height.

Dimlunit

Specifies the unit format for all dimensions except angular.

Dimtih

Specifies if the dimension text is to be drawn inside the extension lines.

Dimtm

Specifies the minimum tolerance limit for dimension text.

Dimtmove

Specifies how dimension text is drawn when text is moved.

Dimtofl

Specifies if a dimension line is drawn between the extension lines even when the text is placed outside the extension lines.

Dimtoh

Specifies the position of dimension text outside the extension lines for all dimension types except ordinate.

Dimtol

Specifies if tolerances are displayed with the dimension text.

Dimtolj

Specifies the vertical justification of tolerance values relative to the nominal dimension text.

Dimtp

Specifies the maximum tolerance limit for dimension text.

Dimtxt

Specifies the height for the dimension or tolerance text.

Dimzin

Specifies the suppression of leading and trailing zeros, and zero foot and inch measurements in dimension values.

Prefix

Specifies the dimension value prefix.

Suffix

Specifies the dimension value suffix.

TextPrecision

Specifies the precision of angular dimension text.

TextPosition

Specifies the dimension text position.

TextRotation

Specifies the rotation angle of the dimension text.

Enter a user-defined suffix for an aligned dimension

This example creates an aligned dimension in model space and uses the Suffix property to allow the user to change the text suffix for the dimension.

VB.NET

```
Imports Autodesk.AutoCAD.Runtime
Imports Autodesk.AutoCAD.ApplicationServices
Imports Autodesk.AutoCAD.DatabaseServices
Imports Autodesk.AutoCAD.EditorInput
Imports Autodesk.AutoCAD.Geometry

<CommandMethod("AddDimensionTextSuffix")> _
Public Sub AddDimensionTextSuffix()
  '' Get the current database
  Dim acDoc As Document = Application.DocumentManager.MdiActiveDocument
  Dim acCurDb As Database = acDoc.Database

  '' Start a transaction
  Using acTrans As Transaction = acCurDb.TransactionManager.StartTransaction()

      '' Open the Block table for read
      Dim acBlkTbl As BlockTable
      acBlkTbl = acTrans.GetObject(acCurDb.BlockTableId, _
                                   OpenMode.ForRead)

      '' Open the Block table record Model space for write
      Dim acBlkTblRec As BlockTableRecord
      acBlkTblRec = acTrans.GetObject(acBlkTbl(BlockTableRecord.ModelSpace), _
                                      OpenMode.ForWrite)

      '' Create the aligned dimension
      Dim acAliDim As AlignedDimension = New AlignedDimension()
      acAliDim.SetDatabaseDefaults()
      acAliDim.XLine1Point = New Point3d(0, 5, 0)
      acAliDim.XLine2Point = New Point3d(5, 5, 0)
      acAliDim.DimLinePoint = New Point3d(5, 7, 0)
      acAliDim.DimensionStyle = acCurDb.Dimstyle

      '' Add the new object to Model space and the transaction
```

```vb
        acBlkTblRec.AppendEntity(acAliDim)
        acTrans.AddNewlyCreatedDBObject(acAliDim, True)

        '' Append a suffix to the dimension text
        Dim pStrOpts As PromptStringOptions = New PromptStringOptions("")

        pStrOpts.Message = vbLf & "Enter a new text suffix for the dimension: "
        pStrOpts.AllowSpaces = True
        Dim pStrRes As PromptResult = acDoc.Editor.GetString(pStrOpts)

        If pStrRes.Status = PromptStatus.OK Then
            acAliDim.Suffix = pStrRes.StringResult
        End If

        '' Commit the changes and dispose of the transaction
        acTrans.Commit()
    End Using
End Sub
```

C#

```csharp
using Autodesk.AutoCAD.Runtime;
using Autodesk.AutoCAD.ApplicationServices;
using Autodesk.AutoCAD.DatabaseServices;
using Autodesk.AutoCAD.EditorInput;
using Autodesk.AutoCAD.Geometry;

[CommandMethod("AddDimensionTextSuffix")]
public static void AddDimensionTextSuffix()
{
    // Get the current database
    Document acDoc = Application.DocumentManager.MdiActiveDocument;
    Database acCurDb = acDoc.Database;

    // Start a transaction
    using (Transaction acTrans = acCurDb.TransactionManager.StartTransaction())
    {
        // Open the Block table for read
        BlockTable acBlkTbl;
        acBlkTbl = acTrans.GetObject(acCurDb.BlockTableId,
                                     OpenMode.ForRead) as BlockTable;

        // Open the Block table record Model space for write
        BlockTableRecord acBlkTblRec;
        acBlkTblRec = acTrans.GetObject(acBlkTbl[BlockTableRecord.ModelSpace],
                                        OpenMode.ForWrite) as BlockTableRecord;

        // Create the aligned dimension
        AlignedDimension acAliDim = new AlignedDimension();
        acAliDim.SetDatabaseDefaults();
        acAliDim.XLine1Point = new Point3d(0, 5, 0);
        acAliDim.XLine2Point = new Point3d(5, 5, 0);
        acAliDim.DimLinePoint = new Point3d(5, 7, 0);
        acAliDim.DimensionStyle = acCurDb.Dimstyle;

        // Add the new object to Model space and the transaction
        acBlkTblRec.AppendEntity(acAliDim);
        acTrans.AddNewlyCreatedDBObject(acAliDim, true);

        // Append a suffix to the dimension text
        PromptStringOptions pStrOpts = new PromptStringOptions("");

        pStrOpts.Message = "\nEnter a new text suffix for the dimension: ";
        pStrOpts.AllowSpaces = true;
```

```
        PromptResult pStrRes = acDoc.Editor.GetString(pStrOpts);

        if (pStrRes.Status == PromptStatus.OK)
        {
            acAliDim.Suffix = pStrRes.StringResult;
        }

        // Commit the changes and dispose of the transaction
        acTrans.Commit();
    }
}
```

```
Sub AddDimensionTextSuffix()
    Dim dimObj As AcadDimAligned
    Dim point1(0 To 2) As Double
    Dim point2(0 To 2) As Double
    Dim location(0 To 2) As Double
    Dim suffix As String

    ' Define the dimension
    point1(0) = 0: point1(1) = 5: point1(2) = 0
    point2(0) = 5: point2(1) = 5: point2(2) = 0
    location(0) = 5: location(1) = 7: location(2) = 0

    ' Create an aligned dimension object in model space
    Set dimObj = ThisDrawing.ModelSpace. _
                    AddDimAligned(point1, point2, location)

    ThisDrawing.Application.ZoomAll
    ' Allow the user to change the text suffix for the dimension
    suffix = ThisDrawing.Utility. _
                GetString(True, vbLf & "Enter a new text " & _
                                    "suffix for the dimension: ")

    ' Apply the change to the dimension
    dimObj.TextSuffix = suffix
    ThisDrawing.Regen acAllViewports
End Sub
```

Dimension in Model Space and Paper Space

You can draw dimensions in both Model space and Paper space. However, if the geometry you are dimensioning is in Model space, it is best to draw dimensions in Model space because AutoCAD places the definition points in the space where the geometry is drawn.

If you draw a dimension in Paper space that describes geometry in your model, the Paper space dimension does not change when you use editing commands or change the magnification of the display in the Model space viewport. The location of the Paper space dimensions also stays the same when you change a view from Paper space to Model space.

If you are dimensioning in Paper space and the global scale factor for linear dimensioning (the DIMLFAC system variable) is set at less than 0, the distance measured is multiplied by the absolute value of DIMLFAC. If you are dimensioning in Model space, the value of 1.0 is

used even if DIMLFAC is less than 0. AutoCAD computes a value for DIMLFAC if you change the variable at the Dim prompt and select the Viewport option. AutoCAD calculates the scaling of Model space to Paper space and assigns the negative of this value to DIMLFAC.

Create Leaders and Annotation

A leader is a line that connects some annotation to a feature in a drawing. Leaders and their annotation are associative, which means if you modify the annotation, the leader updates accordingly. Do not confuse the Leader object with the leader line AutoCAD automatically generates as part of a dimension line.

For more information about leaders, see "Create Text with Leaders" in the *AutoCAD User's Guide*.

Topics in this section

- Create Leader Lines
- Add the Annotation to a Leader
- Leader Associativity
- Edit Leader Associativity
- Edit Leaders

Create Leader Lines

You can create a leader line from any point or feature in a drawing and control its appearance. Leaders can be straight line segments or smooth spline curves. Leader color is controlled by the current dimension line color. Leader scale is controlled by the overall dimension scale set in the active dimension style. The type and size of the arrowhead, if one is present, is controlled by the arrowhead defined in the active style.

A small line known as a hook line usually connects the annotation to the leader. Hook lines appear with multiline text and feature control frames if the last leader line segment is at an angle greater than 15 degrees from horizontal. The hook line is the length of a single arrowhead. If the leader has no annotation, it has no hook line.

leader line

You create a leader by creating an instance of a Leader object. When you create an instance of a Leader object, its constructor does not accept any parameters. The AppendVertex method is used to define the position and length of the leader created.

Create a leader line

This example creates a leader line in model space. There is no annotation associated with the leader line.

VB.NET

```vbnet
Imports Autodesk.AutoCAD.Runtime
Imports Autodesk.AutoCAD.ApplicationServices
Imports Autodesk.AutoCAD.DatabaseServices
Imports Autodesk.AutoCAD.Geometry

<CommandMethod("CreateLeader")> _
Public Sub CreateLeader()
  '' Get the current database
  Dim acDoc As Document = Application.DocumentManager.MdiActiveDocument
  Dim acCurDb As Database = acDoc.Database

  '' Start a transaction
  Using acTrans As Transaction = acCurDb.TransactionManager.StartTransaction()

      '' Open the Block table for read
      Dim acBlkTbl As BlockTable
      acBlkTbl = acTrans.GetObject(acCurDb.BlockTableId, _
                                   OpenMode.ForRead)

      '' Open the Block table record Model space for write
      Dim acBlkTblRec As BlockTableRecord
      acBlkTblRec = acTrans.GetObject(acBlkTbl(BlockTableRecord.ModelSpace), _
                                      OpenMode.ForWrite)

      '' Create the leader
      Dim acLdr As Leader = New Leader()
      acLdr.SetDatabaseDefaults()
      acLdr.AppendVertex(New Point3d(0, 0, 0))
      acLdr.AppendVertex(New Point3d(4, 4, 0))
      acLdr.AppendVertex(New Point3d(4, 5, 0))
      acLdr.HasArrowHead = True

      '' Add the new object to Model space and the transaction
      acBlkTblRec.AppendEntity(acLdr)
      acTrans.AddNewlyCreatedDBObject(acLdr, True)

      '' Commit the changes and dispose of the transaction
      acTrans.Commit()
  End Using
End Sub
```

C#

```csharp
using Autodesk.AutoCAD.Runtime;
using Autodesk.AutoCAD.ApplicationServices;
using Autodesk.AutoCAD.DatabaseServices;
using Autodesk.AutoCAD.Geometry;

[CommandMethod("CreateLeader")]
public static void CreateLeader()
{
  // Get the current database
  Document acDoc = Application.DocumentManager.MdiActiveDocument;
  Database acCurDb = acDoc.Database;

  // Start a transaction
```

```
  using (Transaction acTrans = acCurDb.TransactionManager.StartTransaction())
  {
      // Open the Block table for read
      BlockTable acBlkTbl;
      acBlkTbl = acTrans.GetObject(acCurDb.BlockTableId,
                                   OpenMode.ForRead) as BlockTable;

      // Open the Block table record Model space for write
      BlockTableRecord acBlkTblRec;
      acBlkTblRec = acTrans.GetObject(acBlkTbl[BlockTableRecord.ModelSpace],
                                      OpenMode.ForWrite) as BlockTableRecord;

      // Create the leader
      Leader acLdr = new Leader();
      acLdr.SetDatabaseDefaults();
      acLdr.AppendVertex(new Point3d(0, 0, 0));
      acLdr.AppendVertex(new Point3d(4, 4, 0));
      acLdr.AppendVertex(new Point3d(4, 5, 0));
      acLdr.HasArrowHead = true;

      // Add the new object to Model space and the transaction
      acBlkTblRec.AppendEntity(acLdr);
      acTrans.AddNewlyCreatedDBObject(acLdr, true);

      // Commit the changes and dispose of the transaction
      acTrans.Commit();
  }
}
```

⊟ *VBA/ActiveX Code Reference*

```
Sub CreateLeader()
    Dim leaderObj As AcadLeader
    Dim points(0 To 8) As Double
    Dim leaderType As Integer
    Dim annotationObject As AcadObject

    points(0) = 0: points(1) = 0: points(2) = 0
    points(3) = 4: points(4) = 4: points(5) = 0
    points(6) = 4: points(7) = 5: points(8) = 0
    leaderType = acLineWithArrow
    Set annotationObject = Nothing

    ' Create the leader object in model space
    Set leaderObj = ThisDrawing.ModelSpace. _
                        AddLeader(points, annotationObject, leaderType)

    ZoomAll
End Sub
```

Add the Annotation to a Leader

Leader annotation can be a Tolerance, MText, or Block Reference object. You can create a new annotation, or you can append a copy of an existing annotation. Annotation is added to a leader by assigning the object id of the annotation object to the Annotation property.

Leader Associativity

Leaders are associated with their annotation so that when the annotation moves, the endpoint of the leader moves with it. As you move text and feature control frame annotation, the final leader line segment alternates between attaching to the left side and to the right side of the annotation according to the relation of the annotation to the penultimate (second to last) point of the leader. If the midpoint of the annotation is to the right of the penultimate leader point, then the leader attaches to the right; otherwise, it attaches to the left.

Removing either object from the drawing using either the Erase, Add (to add a block), or WBlock method will break associativity. If the leader and its annotation are copied together in a single operation, the new copy is associative. If they are copied separately, they will non-associative. If associativity is broken for any reason, for example, by copying only the Leader object or by erasing the annotation, the hook line will be removed from the leader.

Associate a leader to the annotation

This example creates an MText object. A leader line is then created using the MText object as its annotation.

VB.NET

```
Imports Autodesk.AutoCAD.Runtime
Imports Autodesk.AutoCAD.ApplicationServices
Imports Autodesk.AutoCAD.DatabaseServices
Imports Autodesk.AutoCAD.Geometry

<CommandMethod("AddLeaderAnnotation")> _
Public Sub AddLeaderAnnotation()
  '' Get the current database
  Dim acDoc As Document = Application.DocumentManager.MdiActiveDocument
  Dim acCurDb As Database = acDoc.Database

  '' Start a transaction
  Using acTrans As Transaction = acCurDb.TransactionManager.StartTransaction()

      '' Open the Block table for read
      Dim acBlkTbl As BlockTable
      acBlkTbl = acTrans.GetObject(acCurDb.BlockTableId, _
                                   OpenMode.ForRead)

      '' Open the Block table record Model space for write
      Dim acBlkTblRec As BlockTableRecord
      acBlkTblRec = acTrans.GetObject(acBlkTbl(BlockTableRecord.ModelSpace), _
                                      OpenMode.ForWrite)

      '' Create the MText annotation
      Dim acMText As MText = New MText()
      acMText.SetDatabaseDefaults()
      acMText.Contents = "Hello, World."
      acMText.Location = New Point3d(5, 5, 0)
      acMText.Width = 2

      '' Add the new object to Model space and the transaction
      acBlkTblRec.AppendEntity(acMText)
      acTrans.AddNewlyCreatedDBObject(acMText, True)
```

```
        '' Create the leader with annotation
        Dim acLdr As Leader = New Leader()
        acLdr.SetDatabaseDefaults()
        acLdr.AppendVertex(New Point3d(0, 0, 0))
        acLdr.AppendVertex(New Point3d(4, 4, 0))
        acLdr.AppendVertex(New Point3d(4, 5, 0))
        acLdr.HasArrowHead = True

        '' Add the new object to Model space and the transaction
        acBlkTblRec.AppendEntity(acLdr)
        acTrans.AddNewlyCreatedDBObject(acLdr, True)

        '' Attach the annotation after the leader object is added
        acLdr.Annotation = acMText.ObjectId
        acLdr.EvaluateLeader()

        '' Commit the changes and dispose of the transaction
        acTrans.Commit()
    End Using
End Sub
```

C#

```csharp
using Autodesk.AutoCAD.Runtime;
using Autodesk.AutoCAD.ApplicationServices;
using Autodesk.AutoCAD.DatabaseServices;
using Autodesk.AutoCAD.Geometry;

[CommandMethod("AddLeaderAnnotation")]
public static void AddLeaderAnnotation()
{
    // Get the current database
    Document acDoc = Application.DocumentManager.MdiActiveDocument;
    Database acCurDb = acDoc.Database;

    // Start a transaction
    using (Transaction acTrans = acCurDb.TransactionManager.StartTransaction())
    {
        // Open the Block table for read
        BlockTable acBlkTbl;
        acBlkTbl = acTrans.GetObject(acCurDb.BlockTableId,
                                     OpenMode.ForRead) as BlockTable;

        // Open the Block table record Model space for write
        BlockTableRecord acBlkTblRec;
        acBlkTblRec = acTrans.GetObject(acBlkTbl[BlockTableRecord.ModelSpace],
                                        OpenMode.ForWrite) as BlockTableRecord;

        // Create the MText annotation
        MText acMText = new MText();
        acMText.SetDatabaseDefaults();
        acMText.Contents = "Hello, World.";
        acMText.Location = new Point3d(5, 5, 0);
        acMText.Width = 2;

        // Add the new object to Model space and the transaction
        acBlkTblRec.AppendEntity(acMText);
        acTrans.AddNewlyCreatedDBObject(acMText, true);

        // Create the leader with annotation
        Leader acLdr = new Leader();
        acLdr.SetDatabaseDefaults();
        acLdr.AppendVertex(new Point3d(0, 0, 0));
```

```
        acLdr.AppendVertex(new Point3d(4, 4, 0));
        acLdr.AppendVertex(new Point3d(4, 5, 0));
        acLdr.HasArrowHead = true;

        // Add the new object to Model space and the transaction
        acBlkTblRec.AppendEntity(acLdr);
        acTrans.AddNewlyCreatedDBObject(acLdr, true);

        // Attach the annotation after the leader object is added
        acLdr.Annotation = acMText.ObjectId;
        acLdr.EvaluateLeader();

        // Commit the changes and dispose of the transaction
        acTrans.Commit();
    }
}
```

⊟ *VBA/ActiveX Code Reference*

```
Sub AddLeaderAnnotation()
    Dim leaderObj As AcadLeader
    Dim mtextObj As AcadMText
    Dim points(0 To 8) As Double
    Dim insertionPoint(0 To 2) As Double
    Dim width As Double
    Dim leaderType As Integer
    Dim annotationObject As Object
    Dim textString As String, msg As String

    ' Create the MText object in model space
    textString = "Hello, World."
    insertionPoint(0) = 5
    insertionPoint(1) = 5
    insertionPoint(2) = 0
    width = 2
    Set mtextObj = ThisDrawing.ModelSpace. _
                        AddMText(insertionPoint, width, textString)

    ' Data for Leader
    points(0) = 0: points(1) = 0: points(2) = 0
    points(3) = 4: points(4) = 4: points(5) = 0
    points(6) = 4: points(7) = 5: points(8) = 0
    leaderType = acLineWithArrow

    ' Create the Leader object in model space and associate
    ' the MText object with the leader
    Set annotationObject = mtextObj
    Set leaderObj = ThisDrawing.ModelSpace. _
                        AddLeader(points, annotationObject, leaderType)

    ZoomAll
End Sub
```

Edit Leader Associativity

Except for the associativity relation between the leader and annotation, the leader and its annotation are entirely separate objects in your drawing. Editing of the leader does not affect the annotation, and editing of the annotation does not affect the leader.

Although text annotation is created using the DIMCLRT, DIMTXT, and DIMTXSTY system variables to define its color, height, and style, it cannot be changed by these system variables because it is not a true dimension object. Text annotation must be edited the same way as any other MText object.

Use the Evaluate method to evaluate the relation of the leader to its associated annotation. This method will update the leader geometry if necessary.

Edit Leaders

Any modifications to leader annotation that change its position affect the position of the endpoint of the associated leader. Also, rotating the annotation causes the leader hook line (if any) to rotate.

Resize a leader by scaling it. If you scale the leader, the annotation stays in the same position relative to the leader endpoint but is not scaled. In addition to scaling, you can also move, mirror, and rotate a leader. Use the TransformBy method to edit the leader. Modify the associated annotation using its member properties and methods

Use Geometric Tolerances

Geometric tolerances show deviations of form, profile, orientation, location, and runout of a feature. You add geometric tolerances in feature control frames. These frames contain all the tolerance information for a single dimension.

For more information about using feature control frames and working with geometric tolerances, see "Add Geometric Tolerances" in the *AutoCAD User's Guide*.

Topics in this section

- Create Geometric Tolerances
- Edit Geometric Tolerances

Create Geometric Tolerances

You create a geometric tolerance by creating an instance of a FeatureControlFrame object. When you create an instance of a FeatureControlFrame object, its constructor can accept an optional set of parameters. The following parameters can be supplied when you create a new FeatureControlFrame object:

- Text string comprising the tolerance symbol (Text property)
- Insertion point (Location property)
- Normal vector (Normal property)
- Direction vector (Direction property)

Create a geometric tolerance

This example creates a simple geometric tolerance in Model space.

VB.NET

```
Imports Autodesk.AutoCAD.Runtime
Imports Autodesk.AutoCAD.ApplicationServices
Imports Autodesk.AutoCAD.DatabaseServices
Imports Autodesk.AutoCAD.Geometry

<CommandMethod("CreateGeometricTolerance")> _
Public Sub CreateGeometricTolerance()
  '' Get the current database
  Dim acDoc As Document = Application.DocumentManager.MdiActiveDocument
  Dim acCurDb As Database = acDoc.Database

  '' Start a transaction
  Using acTrans As Transaction = acCurDb.TransactionManager.StartTransaction()

      '' Open the Block table for read
      Dim acBlkTbl As BlockTable
      acBlkTbl = acTrans.GetObject(acCurDb.BlockTableId, _
                                   OpenMode.ForRead)

      '' Open the Block table record Model space for write
      Dim acBlkTblRec As BlockTableRecord
      acBlkTblRec = acTrans.GetObject(acBlkTbl(BlockTableRecord.ModelSpace), _
                                      OpenMode.ForWrite)

      '' Create the Geometric Tolerance (Feature Control Frame)
      Dim acFcf As FeatureControlFrame = New FeatureControlFrame()
      acFcf.SetDatabaseDefaults()
      acFcf.Text = "{\Fgdt;j}%%v{\Fgdt;n}0.001%%v%%v%%v%%v"
      acFcf.Location = New Point3d(5, 5, 0)

      '' Add the new object to Model space and the transaction
      acBlkTblRec.AppendEntity(acFcf)
      acTrans.AddNewlyCreatedDBObject(acFcf, True)

      '' Commit the changes and dispose of the transaction
      acTrans.Commit()
  End Using
End Sub
```

C#

```
using Autodesk.AutoCAD.Runtime;
using Autodesk.AutoCAD.ApplicationServices;
using Autodesk.AutoCAD.DatabaseServices;
using Autodesk.AutoCAD.Geometry;

[CommandMethod("CreateGeometricTolerance")]
public static void CreateGeometricTolerance()
{
  // Get the current database
  Document acDoc = Application.DocumentManager.MdiActiveDocument;
  Database acCurDb = acDoc.Database;

  // Start a transaction
  using (Transaction acTrans = acCurDb.TransactionManager.StartTransaction())
  {
      // Open the Block table for read
      BlockTable acBlkTbl;
      acBlkTbl = acTrans.GetObject(acCurDb.BlockTableId,
                                   OpenMode.ForRead) as BlockTable;

      // Open the Block table record Model space for write
      BlockTableRecord acBlkTblRec;
      acBlkTblRec = acTrans.GetObject(acBlkTbl[BlockTableRecord.ModelSpace],
                                      OpenMode.ForWrite) as BlockTableRecord;

      // Create the Geometric Tolerance (Feature Control Frame)
      FeatureControlFrame acFcf = new FeatureControlFrame();
      acFcf.SetDatabaseDefaults();
      acFcf.Text = "{\\Fgdt;j}%%v{\\Fgdt;n}0.001%%v%%v%%v%%v";
      acFcf.Location = new Point3d(5, 5, 0);

      // Add the new object to Model space and the transaction
      acBlkTblRec.AppendEntity(acFcf);
      acTrans.AddNewlyCreatedDBObject(acFcf, true);

      // Commit the changes and dispose of the transaction
      acTrans.Commit();
  }
}
```

⊟ VBA/ActiveX Code Reference

```
Sub CreateGeometricTolerance()
    Dim toleranceObj As AcadTolerance
    Dim textString As String
    Dim insertionPoint(0 To 2) As Double
    Dim direction(0 To 2) As Double

    ' Define the tolerance object
    textString = "{\Fgdt;j}%%v{\Fgdt;n}0.001%%v%%v%%v%%v"
    insertionPoint(0) = 5
    insertionPoint(1) = 5
    insertionPoint(2) = 0
    direction(0) = 1
    direction(1) = 0
    direction(2) = 0

    ' Create the tolerance object in model space
    Set toleranceObj = ThisDrawing.ModelSpace. _
```

```
                         AddTolerance(textString, insertionPoint, direction)

    ZoomAll
End Sub
```

Edit Geometric Tolerances

Geometric tolerances are influenced by several system variables and properties. The following system variables and properties affect the appearance of a geometric tolerance:

DIMCLRD

      Controls the color of the feature control frame.

DIMCLRT

      Controls the color of the tolerance text.

DIMGAP

      Controls the gap between the feature control frame and the text.

DIMTXT

      Controls the size of the tolerance text.

DIMTXTSTY

      Controls the style of the tolerance text.

7 Work in Three-Dimensional Space

Most drawings consist of two-dimensional (2D) views of objects that are three dimensional (3D). Though this method of drafting is widely used in the architectural and engineering communities, it is limited: the drawings are 2D representations of 3D objects and must be visually interpreted. Moreover, because the views are created independently, there are more possibilities for error and ambiguity. As a result, you may want to create true 3D models instead of 2D representations. You can use the AutoCAD drawing tools to create detailed, realistic 3D objects and manipulate them in various ways.

Topics in this section

- Specify 3D Coordinates
- Define a User Coordinate System
- Convert Coordinates
- Create 3D Objects
- Edit in 3D
- Edit 3D Solids

Specify 3D Coordinates

Entering 3D world coordinate system (WCS) coordinates is similar to entering 2D WCS coordinates. In addition to specifying *X* and *Y* values, you specify a *Z* value. 2D coordinates are represented by a Point2d object, while you use a Point3d object to represent 3D coordinates. Most properties and methods in the .NET API utilize 3D coordinates.

For more information about specifying 3D coordinates, see "Enter 3D Coordinates" in the *AutoCAD User's Guide*.

Define and query the coordinates for 2D and 3D polylines

This example creates two polylines, each with three coordinates. The first polyline is a 2D polyline, the second polyline is 3D. Notice that the length of the array containing the vertices is expanded to include the *Z* coordinates in the creation of the 3D polyline. The example concludes by querying the coordinates of the polylines and displaying the coordinates in a message box.

VB.NET

```
Imports Autodesk.AutoCAD.Runtime
Imports Autodesk.AutoCAD.ApplicationServices
Imports Autodesk.AutoCAD.DatabaseServices
Imports Autodesk.AutoCAD.Geometry

<CommandMethod("Polyline_2D_3D")> _
Public Sub Polyline_2D_3D()
  '' Get the current document dand database, and start a transaction
  Dim acDoc As Document = Application.DocumentManager.MdiActiveDocument
  Dim acCurDb As Database = acDoc.Database
```

```
Using acTrans As Transaction = acCurDb.TransactionManager.StartTransaction()
    '' Open the Block table for read
    Dim acBlkTbl As BlockTable
    acBlkTbl = acTrans.GetObject(acCurDb.BlockTableId, _
                                 OpenMode.ForRead)

    '' Open the Block table record Model space for write
    Dim acBlkTblRec As BlockTableRecord
    acBlkTblRec = acTrans.GetObject(acBlkTbl(BlockTableRecord.ModelSpace), _
                                    OpenMode.ForWrite)

    '' Create a polyline with two segments (3 points)
    Dim acPoly As Polyline = New Polyline()
    acPoly.SetDatabaseDefaults()
    acPoly.AddVertexAt(0, New Point2d(1, 1), 0, 0, 0)
    acPoly.AddVertexAt(1, New Point2d(1, 2), 0, 0, 0)
    acPoly.AddVertexAt(2, New Point2d(2, 2), 0, 0, 0)
    acPoly.ColorIndex = 1

    '' Add the new object to the block table record and the transaction
    acBlkTblRec.AppendEntity(acPoly)
    acTrans.AddNewlyCreatedDBObject(acPoly, True)

    '' Create a 3D polyline with two segments (3 points)
    Dim acPoly3d As Polyline3d = New Polyline3d()
    acPoly3d.SetDatabaseDefaults()
    acPoly3d.ColorIndex = 5

    '' Add the new object to the block table record and the transaction
    acBlkTblRec.AppendEntity(acPoly3d)
    acTrans.AddNewlyCreatedDBObject(acPoly3d, True)

    '' Before adding vertexes, the polyline must be in the drawing
    Dim acPts3dPoly As Point3dCollection = New Point3dCollection()
    acPts3dPoly.Add(New Point3d(1, 1, 0))
    acPts3dPoly.Add(New Point3d(2, 1, 0))
    acPts3dPoly.Add(New Point3d(2, 2, 0))

    For Each acPt3d As Point3d In acPts3dPoly
        Dim acPolVer3d As PolylineVertex3d = New PolylineVertex3d(acPt3d)
        acPoly3d.AppendVertex(acPolVer3d)
        acTrans.AddNewlyCreatedDBObject(acPolVer3d, True)
    Next

    '' Get the coordinates of the lightweight polyline
    Dim acPts2d As Point2dCollection = New Point2dCollection()
    For nCnt As Integer = 0 To acPoly.NumberOfVertices - 1
        acPts2d.Add(acPoly.GetPoint2dAt(nCnt))
    Next

    '' Get the coordinates of the 3D polyline
    Dim acPts3d As Point3dCollection = New Point3dCollection()
    For Each acObjIdVert As ObjectId In acPoly3d
        Dim acPolVer3d As PolylineVertex3d
        acPolVer3d = acTrans.GetObject(acObjIdVert, _
                                       OpenMode.ForRead)

        acPts3d.Add(acPolVer3d.Position)
    Next

    '' Display the Coordinates
    Application.ShowAlertDialog("2D polyline (red): " & vbLf & _
                                acPts2d(0).ToString() & vbLf & _
                                acPts2d(1).ToString() & vbLf & _
```

```
                                          acPts2d(2).ToString())

        Application.ShowAlertDialog("3D polyline (blue): " & vbLf & _
                                    acPts3d(0).ToString() & vbLf & _
                                    acPts3d(1).ToString() & vbLf & _
                                    acPts3d(2).ToString())

        '' Save the new objects to the database
        acTrans.Commit()
    End Using
End Sub
```

C#

```csharp
using Autodesk.AutoCAD.Runtime;
using Autodesk.AutoCAD.ApplicationServices;
using Autodesk.AutoCAD.DatabaseServices;
using Autodesk.AutoCAD.Geometry;

[CommandMethod("Polyline_2D_3D")]
public static void Polyline_2D_3D()
{
    // Get the current document and database, and start a transaction
    Document acDoc = Application.DocumentManager.MdiActiveDocument;
    Database acCurDb = acDoc.Database;

    using (Transaction acTrans = acCurDb.TransactionManager.StartTransaction())
    {
        // Open the Block table record for read
        BlockTable acBlkTbl;
        acBlkTbl = acTrans.GetObject(acCurDb.BlockTableId,
                                     OpenMode.ForRead) as BlockTable;

        // Open the Block table record Model space for write
        BlockTableRecord acBlkTblRec;
        acBlkTblRec = acTrans.GetObject(acBlkTbl[BlockTableRecord.ModelSpace],
                                        OpenMode.ForWrite) as BlockTableRecord;

        // Create a polyline with two segments (3 points)
        Polyline acPoly = new Polyline();
        acPoly.SetDatabaseDefaults();
        acPoly.AddVertexAt(0, new Point2d(1, 1), 0, 0, 0);
        acPoly.AddVertexAt(1, new Point2d(1, 2), 0, 0, 0);
        acPoly.AddVertexAt(2, new Point2d(2, 2), 0, 0, 0);
        acPoly.ColorIndex = 1;

        // Add the new object to the block table record and the transaction
        acBlkTblRec.AppendEntity(acPoly);
        acTrans.AddNewlyCreatedDBObject(acPoly, true);

        // Create a 3D polyline with two segments (3 points)
        Polyline3d acPoly3d = new Polyline3d();
        acPoly3d.SetDatabaseDefaults();
        acPoly3d.ColorIndex = 5;

        // Add the new object to the block table record and the transaction
        acBlkTblRec.AppendEntity(acPoly3d);
        acTrans.AddNewlyCreatedDBObject(acPoly3d, true);

        // Before adding vertexes, the polyline must be in the drawing
        Point3dCollection acPts3dPoly = new Point3dCollection();
        acPts3dPoly.Add(new Point3d(1, 1, 0));
        acPts3dPoly.Add(new Point3d(2, 1, 0));
        acPts3dPoly.Add(new Point3d(2, 2, 0));
```

```
        foreach (Point3d acPt3d in acPts3dPoly)
        {
            PolylineVertex3d acPolVer3d = new PolylineVertex3d(acPt3d);
            acPoly3d.AppendVertex(acPolVer3d);
            acTrans.AddNewlyCreatedDBObject(acPolVer3d, true);
        }

        // Get the coordinates of the lightweight polyline
        Point2dCollection acPts2d = new Point2dCollection();
        for (int nCnt = 0; nCnt < acPoly.NumberOfVertices; nCnt++)
        {
            acPts2d.Add(acPoly.GetPoint2dAt(nCnt));
        }

        // Get the coordinates of the 3D polyline
        Point3dCollection acPts3d = new Point3dCollection();
        foreach (ObjectId acObjIdVert in acPoly3d)
        {
            PolylineVertex3d acPolVer3d;
            acPolVer3d = acTrans.GetObject(acObjIdVert,
                                       OpenMode.ForRead) as PolylineVertex3d;

            acPts3d.Add(acPolVer3d.Position);
        }

        // Display the Coordinates
        Application.ShowAlertDialog("2D polyline (red): \n" +
                                    acPts2d[0].ToString() + "\n" +
                                    acPts2d[1].ToString() + "\n" +
                                    acPts2d[2].ToString());

        Application.ShowAlertDialog("3D polyline (blue): \n" +
                                    acPts3d[0].ToString() + "\n" +
                                    acPts3d[1].ToString() + "\n" +
                                    acPts3d[2].ToString());

        // Save the new object to the database
        acTrans.Commit();
    }
}
```

⊟ *VBA/ActiveX Code Reference*

```
Sub Polyline_2D_3D()
    Dim pline2DObj As AcadLWPolyline
    Dim pline3DObj As AcadPolyline

    Dim points2D(0 To 5) As Double
    Dim points3D(0 To 8) As Double

    ' Define three 2D polyline points
    points2D(0) = 1: points2D(1) = 1
    points2D(2) = 1: points2D(3) = 2
    points2D(4) = 2: points2D(5) = 2

    ' Define three 3D polyline points
    points3D(0) = 1: points3D(1) = 1: points3D(2) = 0
    points3D(3) = 2: points3D(4) = 1: points3D(5) = 0
    points3D(6) = 2: points3D(7) = 2: points3D(8) = 0

    ' Create the 2D light weight Polyline
    Set pline2DObj = ThisDrawing.ModelSpace. _
```

```
                                          AddLightWeightPolyline(points2D)
    pline2DObj.Color = acRed
    pline2DObj.Update

    ' Create the 3D polyline
    Set pline3DObj = ThisDrawing.ModelSpace. _
                                  AddPolyline(points3D)
    pline3DObj.Color = acBlue
    pline3DObj.Update

    ' Query the coordinates of the polylines
    Dim get2Dpts As Variant
    Dim get3Dpts As Variant

    get2Dpts = pline2DObj.Coordinates
    get3Dpts = pline3DObj.Coordinates

    ' Display the coordinates

    MsgBox ("2D polyline (red): " & vbCrLf & "(" & _
            get2Dpts(0) & ", " & get2Dpts(1) & ")" & vbCrLf & "(" & _
            get2Dpts(2) & ", " & get2Dpts(3) & ")"& vbCrLf & "(" & _
            get2Dpts(4) & ", " & get2Dpts(5) & ")")

    MsgBox ("3D polyline (blue): " & vbCrLf & "(" & _
            get3Dpts(0) & ", " & get3Dpts(1) & ", " & _
            get3Dpts(2) & ")" & vbCrLf & "(" & _
            get3Dpts(3) & ", " & get3Dpts(4) & ", " & _
            get3Dpts(5) & ")" & vbCrLf & "(" & _
            get3Dpts(6) & ", " & get3Dpts(7) & ", " & _
            get3Dpts(8) & ")")
End Sub
```

Define a User Coordinate System

You define a user coordinate system (UCS) object to change the location of the (0, 0, 0) origin point and the orientation of the *XY* plane and *Z* axis. You can locate and orient a UCS anywhere in 3D space, and you can define, save, and recall as many user coordinate systems as you require. Coordinate input and display are relative to the current UCS.

To indicate the origin and orientation of the UCS, you can display the UCS icon at the UCS origin point using the IconAtOrigin property of a Viewport object or the UCSICON system variable. If the UCS icon is turned on (IconVisible property) and is not displayed at the origin, it is displayed at the WCS coordinate defined by the UCSORG system variable.

You can create a new user coordinate system using the Add method of the UCSTable object. This method requires four values as input: the coordinate of the origin, a coordinate on the *X* and *Y* axes, and the name of the UCS.

All coordinates in the AutoCAD® ActiveX Automation are entered in the world coordinate system. Use the GetUCSMatrix method to return the transformation matrix of a given UCS. Use this transformation matrix to find the equivalent WCS coordinates.

To make a UCS active, use the ActiveUCS property on the Document object. If changes are made to the active UCS, the new UCS object must be reset as the active UCS for the

changes to appear. To reset the active UCS, simply call the ActiveUCS property again with the updated UCS object.

For more information about defining a UCS, see "Control the User Coordinate System in 3D" in the *User's Guide*.

Create a new UCS, make it active, and translate the coordinates of a point into the UCS coordinates

The following subroutine creates a new UCS and sets it as the active UCS for the drawing. It then asks the user to pick a point in the drawing, and returns both WCS and UCS coordinates for the point.

VB.NET

```
Imports Autodesk.AutoCAD.Runtime
Imports Autodesk.AutoCAD.ApplicationServices
Imports Autodesk.AutoCAD.DatabaseServices
Imports Autodesk.AutoCAD.EditorInput
Imports Autodesk.AutoCAD.Geometry

<CommandMethod("NewUCS")> _
Public Sub NewUCS()
  '' Get the current document and database, and start a transaction
  Dim acDoc As Document = Application.DocumentManager.MdiActiveDocument
  Dim acCurDb As Database = acDoc.Database

  Using acTrans As Transaction = acCurDb.TransactionManager.StartTransaction()
      '' Open the UCS table for read
      Dim acUCSTbl As UcsTable
      acUCSTbl = acTrans.GetObject(acCurDb.UcsTableId, _
                                   OpenMode.ForRead)

      Dim acUCSTblRec As UcsTableRecord

      '' Check to see if the "New_UCS" UCS table record exists
      If acUCSTbl.Has("New_UCS") = False Then
          acUCSTblRec = New UcsTableRecord()
          acUCSTblRec.Name = "New_UCS"

          '' Open the UCSTable for write
          acUCSTbl.UpgradeOpen()

          '' Add the new UCS table record
          acUCSTbl.Add(acUCSTblRec)
          acTrans.AddNewlyCreatedDBObject(acUCSTblRec, True)
      Else
          acUCSTblRec = acTrans.GetObject(acUCSTbl("New_UCS"), _
                                          OpenMode.ForWrite)
      End If

      acUCSTblRec.Origin = New Point3d(4, 5, 3)
      acUCSTblRec.XAxis = New Vector3d(1, 0, 0)
      acUCSTblRec.YAxis = New Vector3d(0, 1, 0)

      '' Open the active viewport
      Dim acVportTblRec As ViewportTableRecord
      acVportTblRec = acTrans.GetObject(acDoc.Editor.ActiveViewportId, _
                                        OpenMode.ForWrite)

      '' Display the UCS Icon at the origin of the current viewport
      acVportTblRec.IconAtOrigin = True
```

```
    acVportTblRec.IconEnabled = True

    '' Set the UCS current
    acVportTblRec.SetUcs(acUCSTblRec.ObjectId)
    acDoc.Editor.UpdateTiledViewportsFromDatabase()

    '' Display the name of the current UCS
    Dim acUCSTblRecActive As UcsTableRecord
    acUCSTblRecActive = acTrans.GetObject(acVportTblRec.UcsName, _
                                          OpenMode.ForRead)

    Application.ShowAlertDialog("The current UCS is: " & _
                               acUCSTblRecActive.Name)

    Dim pPtRes As PromptPointResult
    Dim pPtOpts As PromptPointOptions = New PromptPointOptions("")

    '' Prompt for a point
    pPtOpts.Message = vbLf & "Enter a point: "
    pPtRes = acDoc.Editor.GetPoint(pPtOpts)

    Dim pPt3dWCS As Point3d
    Dim pPt3dUCS As Point3d

    '' If a point was entered, then translate it to the current UCS
    If pPtRes.Status = PromptStatus.OK Then
        pPt3dWCS = pPtRes.Value
        pPt3dUCS = pPtRes.Value

        '' Translate the point from the current UCS to the WCS
        Dim newMatrix As Matrix3d = New Matrix3d()
        newMatrix = Matrix3d.AlignCoordinateSystem(Point3d.Origin, _
                                                   Vector3d.XAxis, _
                                                   Vector3d.YAxis, _
                                                   Vector3d.ZAxis, _
                                                   acVportTblRec.Ucs.Origin, _
                                                   acVportTblRec.Ucs.Xaxis, _
                                                   acVportTblRec.Ucs.Yaxis, _
                                                   acVportTblRec.Ucs.Zaxis)

        pPt3dWCS = pPt3dWCS.TransformBy(newMatrix)

        Application.ShowAlertDialog("The WCS coordinates are: " & vbLf & _
                                    pPt3dWCS.ToString() & vbLf & _
                                    "The UCS coordinates are: " & vbLf & _
                                    pPt3dUCS.ToString())
    End If

    '' Save the new objects to the database
    acTrans.Commit()
  End Using
End Sub
```

C#

```csharp
using Autodesk.AutoCAD.Runtime;
using Autodesk.AutoCAD.ApplicationServices;
using Autodesk.AutoCAD.DatabaseServices;
using Autodesk.AutoCAD.EditorInput;
using Autodesk.AutoCAD.Geometry;

[CommandMethod("NewUCS")]
public static void NewUCS()
{
```

```csharp
// Get the current document and database, and start a transaction
Document acDoc = Application.DocumentManager.MdiActiveDocument;
Database acCurDb = acDoc.Database;

using (Transaction acTrans = acCurDb.TransactionManager.StartTransaction())
{
    // Open the UCS table for read
    UcsTable acUCSTbl;
    acUCSTbl = acTrans.GetObject(acCurDb.UcsTableId,
                                 OpenMode.ForRead) as UcsTable;

    UcsTableRecord acUCSTblRec;

    // Check to see if the "New_UCS" UCS table record exists
    if (acUCSTbl.Has("New_UCS") == false)
    {
        acUCSTblRec = new UcsTableRecord();
        acUCSTblRec.Name = "New_UCS";

        // Open the UCSTable for write
        acUCSTbl.UpgradeOpen();

        // Add the new UCS table record
        acUCSTbl.Add(acUCSTblRec);
        acTrans.AddNewlyCreatedDBObject(acUCSTblRec, true);
    }
    else
    {
        acUCSTblRec = acTrans.GetObject(acUCSTbl["New_UCS"],
                                        OpenMode.ForWrite) as UcsTableRecord;
    }

    acUCSTblRec.Origin = new Point3d(4, 5, 3);
    acUCSTblRec.XAxis = new Vector3d(1, 0, 0);
    acUCSTblRec.YAxis = new Vector3d(0, 1, 0);

    // Open the active viewport
    ViewportTableRecord acVportTblRec;
    acVportTblRec = acTrans.GetObject(acDoc.Editor.ActiveViewportId,
                                      OpenMode.ForWrite) as ViewportTableRecord;

    // Display the UCS Icon at the origin of the current viewport
    acVportTblRec.IconAtOrigin = true;
    acVportTblRec.IconEnabled = true;

    // Set the UCS current
    acVportTblRec.SetUcs(acUCSTblRec.ObjectId);
    acDoc.Editor.UpdateTiledViewportsFromDatabase();

    // Display the name of the current UCS
    UcsTableRecord acUCSTblRecActive;
    acUCSTblRecActive = acTrans.GetObject(acVportTblRec.UcsName,
                                          OpenMode.ForRead) as UcsTableRecord;

    Application.ShowAlertDialog("The current UCS is: " +
                               acUCSTblRecActive.Name);

    PromptPointResult pPtRes;
    PromptPointOptions pPtOpts = new PromptPointOptions("");

    // Prompt for a point
    pPtOpts.Message = "\nEnter a point: ";
    pPtRes = acDoc.Editor.GetPoint(pPtOpts);

    Point3d pPt3dWCS;
```

```
        Point3d pPt3dUCS;

        // If a point was entered, then translate it to the current UCS
        if (pPtRes.Status == PromptStatus.OK)
        {
            pPt3dWCS = pPtRes.Value;
            pPt3dUCS = pPtRes.Value;

            // Translate the point from the current UCS to the WCS
            Matrix3d newMatrix = new Matrix3d();
            newMatrix = Matrix3d.AlignCoordinateSystem(Point3d.Origin,
                                                Vector3d.XAxis,
                                                Vector3d.YAxis,
                                                Vector3d.ZAxis,
                                                acVportTblRec.Ucs.Origin,
                                                acVportTblRec.Ucs.Xaxis,
                                                acVportTblRec.Ucs.Yaxis,
                                                acVportTblRec.Ucs.Zaxis);

            pPt3dWCS = pPt3dWCS.TransformBy(newMatrix);

            Application.ShowAlertDialog("The WCS coordinates are: \n" +
                                        pPt3dWCS.ToString() + "\n" +
                                        "The UCS coordinates are: \n" +
                                        pPt3dUCS.ToString());
        }

        // Save the new objects to the database
        acTrans.Commit();
    }
}
```

⊟ *VBA/ActiveX Code Reference*

```
Sub NewUCS()
    ' Define the variables we will need
    Dim ucsObj As AcadUCS
    Dim origin(0 To 2) As Double
    Dim xAxisPnt(0 To 2) As Double
    Dim yAxisPnt(0 To 2) As Double

    ' Define the UCS points
    origin(0) = 4: origin(1) = 5: origin(2) = 3
    xAxisPnt(0) = 5: xAxisPnt(1) = 5: xAxisPnt(2) = 3
    yAxisPnt(0) = 4: yAxisPnt(1) = 6: yAxisPnt(2) = 3

    ' Add the UCS to the
    ' UserCoordinatesSystems collection
    Set ucsObj = ThisDrawing.UserCoordinateSystems. _
                    Add(origin, xAxisPnt, yAxisPnt, "New_UCS")

    ' Display the UCS icon
    ThisDrawing.ActiveViewport.UCSIconAtOrigin = True
    ThisDrawing.ActiveViewport.UCSIconOn = True

    ' Make the new UCS the active UCS
    ThisDrawing.ActiveUCS = ucsObj
    MsgBox "The current UCS is : " & ThisDrawing.ActiveUCS.Name _
            & vbCrLf & " Pick a point in the drawing."

    ' Find the WCS and UCS coordinate of a point
    Dim WCSPnt As Variant
    Dim UCSPnt As Variant
```

```
    WCSPnt = ThisDrawing.Utility.GetPoint(, "Enter a point: ")
    UCSPnt = ThisDrawing.Utility.TranslateCoordinates _
                (WCSPnt, acWorld, acUCS, False)

    MsgBox "The WCS coordinates are: " & WCSPnt(0) & ", " _
            & WCSPnt(1) & ", " & WCSPnt(2) & vbCrLf & _
            "The UCS coordinates are: " & UCSPnt(0) & ", " _
            & UCSPnt(1) & ", " & UCSPnt(2)
End Sub
```

Convert Coordinates

The TransformBy method can translate a point or a displacement from one coordinate system to another. You use the AlignCoordinateSystem method to specify which coordinate system you are translating from and which coordinate system you are going to. The AlignCoordinateSystem method requires the following:

- Origin point of the coordinate system you are translating from
- Three 3D vectors that represent the X, Y and X axes of the coordinate system you are translating from
- Origin point of the coordinate system you are translating to
- Three 3D vectors that represent the X, Y and X axes of the coordinate system you are translating to

WCS

World coordinate system: The reference coordinate system. All other coordinate systems are defined relative to the WCS, which never changes. Values measured relative to the WCS are stable across changes to other coordinate systems. All points passed in and out of the methods and properties in the .NET API are expressed in the WCS unless otherwise specified.

UCS

User coordinate system (UCS): The working coordinate system. The user specifies a UCS to make drawing tasks easier. All points passed to AutoCAD commands, including those returned from AutoLISP routines and external functions, are points in the current UCS (unless the user precedes them with an * at the Command prompt). If you want your application to send coordinates in the WCS, OCS, or DCS to AutoCAD commands, you must first convert them to the UCS by calling the translating them and then transforming the Point3d or Point 2d object with the TransformBy method that represents the coordinate value.

OCS

Object coordinate system (also known as Entity coordinate system or ECS): Point values specified by certain methods and properties for the Polyline2d and Polyline objects are expressed in this coordinate system, relative to the object. These points are usually converted into the WCS, current UCS, or current DCS, according to the intended use of the object. Conversely, points in WCS, UCS, or DCS must be

translated into an OCS before they are written to the database by means of the same properties. See the *AutoCAD .NET Managed Class Guide* for the methods and properties that use this coordinate system.

When converting coordinates to or from the OCS you must consider the normal of the OCS.

DCS

Display coordinate system: The coordinate system where objects are transformed before they are displayed. The origin of the DCS is the point stored in the AutoCAD system variable TARGET, and its *Z* axis is the viewing direction. In other words, a viewport is always a plan view of its DCS. These coordinates can be used to determine where something will be displayed to the user.

PSDCS

Paper space DCS: This coordinate system can be transformed only to or from the DCS of a Model space viewport. This is essentially a 2D transformation, where the *X* and *Y* coordinates are always scaled. Therefore, it can be used to find the scale factor between the two coordinate systems. The PSDCS can be transformed only into a Model space viewport.

Translate OCS coordinates to WCS coordinates

This example creates a polyline in Model space. The first vertex for the polyline is then displayed in both the OCS and WCS coordinates.

VB.NET

```
Imports Autodesk.AutoCAD.Runtime
Imports Autodesk.AutoCAD.ApplicationServices
Imports Autodesk.AutoCAD.DatabaseServices
Imports Autodesk.AutoCAD.Geometry

<CommandMethod("TranslateCoordinates")> _
Public Sub TranslateCoordinates()
  '' Get the current document and database, and start a transaction
  Dim acDoc As Document = Application.DocumentManager.MdiActiveDocument
  Dim acCurDb As Database = acDoc.Database

  Using acTrans As Transaction = acCurDb.TransactionManager.StartTransaction()
      '' Open the Block table for read
      Dim acBlkTbl As BlockTable
      acBlkTbl = acTrans.GetObject(acCurDb.BlockTableId, _
                                   OpenMode.ForRead)

      '' Open the Block table record Model space for write
      Dim acBlkTblRec As BlockTableRecord
      acBlkTblRec = acTrans.GetObject(acBlkTbl(BlockTableRecord.ModelSpace), _
                                      OpenMode.ForWrite)

      '' Create a 2D polyline with two segments (3 points)
      Dim acPoly2d As Polyline2d = New Polyline2d()
      acPoly2d.SetDatabaseDefaults()

      '' Add the new object to the block table record and the transaction
      acBlkTblRec.AppendEntity(acPoly2d)
      acTrans.AddNewlyCreatedDBObject(acPoly2d, True)
```

```
      '' Before adding vertexes, the polyline must be in the drawing
      Dim acPts2dPoly As Point3dCollection = New Point3dCollection()
      acPts2dPoly.Add(New Point3d(1, 1, 0))
      acPts2dPoly.Add(New Point3d(1, 2, 0))
      acPts2dPoly.Add(New Point3d(2, 2, 0))
      acPts2dPoly.Add(New Point3d(3, 2, 0))
      acPts2dPoly.Add(New Point3d(4, 4, 0))

      For Each acPt3d As Point3d In acPts2dPoly
          Dim acVer2d As Vertex2d = New Vertex2d(acPt3d, 0, 0, 0, 0)
          acPoly2d.AppendVertex(acVer2d)
          acTrans.AddNewlyCreatedDBObject(acVer2d, True)
      Next

      '' Set the normal of the 2D polyline
      acPoly2d.Normal = New Vector3d(0, 1, 2)

      '' Get the first coordinate of the 2D polyline
      Dim acPts3d As Point3dCollection = New Point3dCollection()
      Dim acFirstVer As Vertex2d = Nothing
      For Each acObjIdVert As ObjectId In acPoly2d
          acFirstVer = acTrans.GetObject(acObjIdVert, _
                                         OpenMode.ForRead)

          acPts3d.Add(acFirstVer.Position)

          Exit For
      Next

      '' Get the first point of the polyline and
      '' use the eleveation for the Z value
      Dim pFirstVer As Point3d = New Point3d(acFirstVer.Position.X, _
                                             acFirstVer.Position.Y, _
                                             acPoly2d.Elevation)

      '' Translate the OCS to WCS
      Dim mWPlane As Matrix3d = Matrix3d.WorldToPlane(acPoly2d.Normal)
      Dim pWCSPt As Point3d = pFirstVer.TransformBy(mWPlane)

      Application.ShowAlertDialog("The first vertex has the following " & _
                                  "coordinates:" & _
                                  vbLf & "OCS: " + pFirstVer.ToString() & _
                                  vbLf & "WCS: " + pWCSPt.ToString())

      '' Save the new objects to the database
      acTrans.Commit()
  End Using
End Sub
```

C#

```
using Autodesk.AutoCAD.Runtime;
using Autodesk.AutoCAD.ApplicationServices;
using Autodesk.AutoCAD.DatabaseServices;
using Autodesk.AutoCAD.Geometry;

[CommandMethod("TranslateCoordinates")]
public static void TranslateCoordinates()
{
  // Get the current document and database, and start a transaction
  Document acDoc = Application.DocumentManager.MdiActiveDocument;
  Database acCurDb = acDoc.Database;
```

```
using (Transaction acTrans = acCurDb.TransactionManager.StartTransaction())
{
    // Open the Block table record for read
    BlockTable acBlkTbl;
    acBlkTbl = acTrans.GetObject(acCurDb.BlockTableId,
                                 OpenMode.ForRead) as BlockTable;

    // Open the Block table record Model space for write
    BlockTableRecord acBlkTblRec;
    acBlkTblRec = acTrans.GetObject(acBlkTbl[BlockTableRecord.ModelSpace],
                                    OpenMode.ForWrite) as BlockTableRecord;

    // Create a 2D polyline with two segments (3 points)
    Polyline2d acPoly2d = new Polyline2d();
    acPoly2d.SetDatabaseDefaults();

    // Add the new object to the block table record and the transaction
    acBlkTblRec.AppendEntity(acPoly2d);
    acTrans.AddNewlyCreatedDBObject(acPoly2d, true);

    // Before adding vertexes, the polyline must be in the drawing
    Point3dCollection acPts2dPoly = new Point3dCollection();
    acPts2dPoly.Add(new Point3d(1, 1, 0));
    acPts2dPoly.Add(new Point3d(1, 2, 0));
    acPts2dPoly.Add(new Point3d(2, 2, 0));
    acPts2dPoly.Add(new Point3d(3, 2, 0));
    acPts2dPoly.Add(new Point3d(4, 4, 0));

    foreach (Point3d acPt3d in acPts2dPoly)
    {
        Vertex2d acVer2d = new Vertex2d(acPt3d, 0, 0, 0, 0);
        acPoly2d.AppendVertex(acVer2d);
        acTrans.AddNewlyCreatedDBObject(acVer2d, true);
    }

    // Set the normal of the 2D polyline
    acPoly2d.Normal = new Vector3d(0, 1, 2);

    // Get the first coordinate of the 2D polyline
    Point3dCollection acPts3d = new Point3dCollection();
    Vertex2d acFirstVer = null;
    foreach (ObjectId acObjIdVert in acPoly2d)
    {
        acFirstVer = acTrans.GetObject(acObjIdVert,
                                       OpenMode.ForRead) as Vertex2d;

        acPts3d.Add(acFirstVer.Position);

        break;
    }

    // Get the first point of the polyline and
    // use the eleveation for the Z value
    Point3d pFirstVer = new Point3d(acFirstVer.Position.X,
                                    acFirstVer.Position.Y,
                                    acPoly2d.Elevation);

    // Translate the OCS to WCS
    Matrix3d mWPlane = Matrix3d.WorldToPlane(acPoly2d.Normal);
    Point3d pWCSPt = pFirstVer.TransformBy(mWPlane);

    Application.ShowAlertDialog("The first vertex has the following " +
                               "coordinates:" +
                               "\nOCS: " + pFirstVer.ToString() +
                               "\nWCS: " + pWCSPt.ToString());
```

```
        // Save the new objects to the database
        acTrans.Commit();
    }
}
```

⊟ *VBA/ActiveX Code Reference*

```
Sub TranslateCoordinates()
    ' Create a polyline in model space.
    Dim plineObj As AcadPolyline
    Dim points(0 To 14) As Double

    ' Define the 2D polyline points
    points(0) = 1: points(1) = 1: points(2) = 0
    points(3) = 1: points(4) = 2: points(5) = 0
    points(6) = 2: points(7) = 2: points(8) = 0
    points(9) = 3: points(10) = 2: points(11) = 0
    points(12) = 4: points(13) = 4: points(14) = 0

    ' Create a light weight Polyline object in model space
    Set plineObj = ThisDrawing.ModelSpace.AddPolyline(points)

    ' Find the X and Y coordinates of the
    ' first vertex of the polyline
    Dim firstVertex As Variant
    firstVertex = plineObj.Coordinate(0)

    ' Find the Z coordinate for the polyline
    ' using the elevation property
    firstVertex(2) = plineObj.Elevation

    ' Change the normal for the pline so that the
    ' difference between the coordinate systems
    ' is obvious.
    Dim plineNormal(0 To 2) As Double
    plineNormal(0) = 0#
    plineNormal(1) = 1#
    plineNormal(2) = 2#
    plineObj.Normal = plineNormal

    ' Translate the OCS coordinate into WCS
    Dim coordinateWCS As Variant
    coordinateWCS = ThisDrawing.Utility.TranslateCoordinates _
                       (firstVertex, acOCS, acWorld, False, plineNormal)

    ' Display the coordinates of the point
    MsgBox "The first vertex has the following coordinates:" _
          & vbCrLf & "OCS: (" & firstVertex(0) & "," & _
          firstVertex(1) & "," & firstVertex(2) & ")" & vbCrLf & _
          "WCS: (" & coordinateWCS(0) & "," & _
          coordinateWCS(1) & "," & coordinateWCS(2) & ")"
End Sub
```

Create 3D Objects

AutoCAD supports three types of 3D modeling: wireframe, surface, and solid. Each type has its own creation and editing techniques.

For more information about creating 3D objects, see "Create 3D Objects" in the *AutoCAD User's Guide*.

Topics in this section

- [Create Wireframes](#)
- [Create Meshes](#)
- [Create Polyface Meshes](#)
- [Create Solids](#)

Create Wireframes

With AutoCAD you can create wireframe models by positioning any 2D planar object anywhere in 3D space. You can position 2D objects in 3D space using several methods:

- Create the object by entering 3D points. You enter a coordinate that defines the *X*, *Y*, and *Z* location of the point.
- Set the default construction plane (*XY* plane) on which you will draw the object by defining a UCS.
- Move the object to its proper orientation in 3D space after you create it.

Also, you can create some wireframe objects, such as lines and 3D polylines, that can exist in all three dimensions.

For more information on creating wireframes, see "Create Wireframe Models" in the *AutoCAD User's Guide*.

Create Meshes

A rectangular mesh (PolygonMesh object) represents the surface of an object using planar facets. The density or number of facets for a mesh is defined in terms of a matrix of *M* and *N* vertices, similar to a grid consisting of columns and rows. *M* and *N* specify the column and row position, respectively, of any given vertex. You can create meshes in both 2D and 3D, but they are used primarily for 3D.

Create an instance of a PolygonMesh object and then specify the density and placement of the vertices for the new mesh. This method optionally takes six values: the type of Polygon

Mesh to create, two integers that define the number of vertices in the *M* and *N* directions, a collection of points containing the coordinates for all the vertices in the mesh, and two booleans that define if the mesh is closed in the *M* or *N* directions.Is

Once the PolygonMesh is created, use the IsMClosed and NClosed properties to close the mesh.

For more information on creating meshes, see "Create Surfaces" in the *AutoCAD User's Guide*.

Create a polygon mesh

This example creates a 4×4 polygon mesh. The direction of the active viewport is then adjusted so that the three-dimensional nature of the mesh is more easily viewed.

VB.NET

```
Imports Autodesk.AutoCAD.Runtime
Imports Autodesk.AutoCAD.ApplicationServices
Imports Autodesk.AutoCAD.DatabaseServices
Imports Autodesk.AutoCAD.Geometry

<CommandMethod("Create3DMesh")> _
Public Sub Create3DMesh()
  '' Get the current document and database, and start a transaction
  Dim acDoc As Document = Application.DocumentManager.MdiActiveDocument
  Dim acCurDb As Database = acDoc.Database

  Using acTrans As Transaction = acCurDb.TransactionManager.StartTransaction()
      '' Open the Block table for read
      Dim acBlkTbl As BlockTable
      acBlkTbl = acTrans.GetObject(acCurDb.BlockTableId, _
                                   OpenMode.ForRead)

      '' Open the Block table record Model space for write
      Dim acBlkTblRec As BlockTableRecord
      acBlkTblRec = acTrans.GetObject(acBlkTbl(BlockTableRecord.ModelSpace), _
                                      OpenMode.ForWrite)

      '' Create a polygon mesh
      Dim acPolyMesh As PolygonMesh = New PolygonMesh()
      acPolyMesh.SetDatabaseDefaults()
      acPolyMesh.MSize = 4
      acPolyMesh.NSize = 4

      '' Add the new object to the block table record and the transaction
      acBlkTblRec.AppendEntity(acPolyMesh)
      acTrans.AddNewlyCreatedDBObject(acPolyMesh, True)

      '' Before adding vertexes, the polyline must be in the drawing
      Dim acPts3dPMesh As Point3dCollection = New Point3dCollection()
      acPts3dPMesh.Add(New Point3d(0, 0, 0))
      acPts3dPMesh.Add(New Point3d(2, 0, 1))
      acPts3dPMesh.Add(New Point3d(4, 0, 0))
      acPts3dPMesh.Add(New Point3d(6, 0, 1))

      acPts3dPMesh.Add(New Point3d(0, 2, 0))
      acPts3dPMesh.Add(New Point3d(2, 2, 1))
      acPts3dPMesh.Add(New Point3d(4, 2, 0))
      acPts3dPMesh.Add(New Point3d(6, 2, 1))
```

```
        acPts3dPMesh.Add(New Point3d(0, 4, 0))
        acPts3dPMesh.Add(New Point3d(2, 4, 1))
        acPts3dPMesh.Add(New Point3d(4, 4, 0))
        acPts3dPMesh.Add(New Point3d(6, 4, 0))

        acPts3dPMesh.Add(New Point3d(0, 6, 0))
        acPts3dPMesh.Add(New Point3d(2, 6, 1))
        acPts3dPMesh.Add(New Point3d(4, 6, 0))
        acPts3dPMesh.Add(New Point3d(6, 6, 0))

        For Each acPt3d As Point3d In acPts3dPMesh
            Dim acPMeshVer As PolygonMeshVertex = New PolygonMeshVertex(acPt3d)
            acPolyMesh.AppendVertex(acPMeshVer)
            acTrans.AddNewlyCreatedDBObject(acPMeshVer, True)
        Next

        '' Open the active viewport
        Dim acVportTblRec As ViewportTableRecord
        acVportTblRec = acTrans.GetObject(acDoc.Editor.ActiveViewportId, _
                                          OpenMode.ForWrite)

        '' Rotate the view direction of the current viewport
        acVportTblRec.ViewDirection = New Vector3d(-1, -1, 1)
        acDoc.Editor.UpdateTiledViewportsFromDatabase()

        '' Save the new objects to the database
        acTrans.Commit()
    End Using
End Sub
```

C#

```
using Autodesk.AutoCAD.Runtime;
using Autodesk.AutoCAD.ApplicationServices;
using Autodesk.AutoCAD.DatabaseServices;
using Autodesk.AutoCAD.Geometry;

[CommandMethod("Create3DMesh")]
public static void Create3DMesh()
{
    // Get the current document and database, and start a transaction
    Document acDoc = Application.DocumentManager.MdiActiveDocument;
    Database acCurDb = acDoc.Database;

    using (Transaction acTrans = acCurDb.TransactionManager.StartTransaction())
    {
        // Open the Block table record for read
        BlockTable acBlkTbl;
        acBlkTbl = acTrans.GetObject(acCurDb.BlockTableId,
                                     OpenMode.ForRead) as BlockTable;

        // Open the Block table record Model space for write
        BlockTableRecord acBlkTblRec;
        acBlkTblRec = acTrans.GetObject(acBlkTbl[BlockTableRecord.ModelSpace],
                                        OpenMode.ForWrite) as BlockTableRecord;

        // Create a polygon mesh
        PolygonMesh acPolyMesh = new PolygonMesh();
        acPolyMesh.SetDatabaseDefaults();
        acPolyMesh.MSize = 4;
        acPolyMesh.NSize = 4;

        // Add the new object to the block table record and the transaction
        acBlkTblRec.AppendEntity(acPolyMesh);
```

```
        acTrans.AddNewlyCreatedDBObject(acPolyMesh, true);

        // Before adding vertexes, the polyline must be in the drawing
        Point3dCollection acPts3dPMesh = new Point3dCollection();
        acPts3dPMesh.Add(new Point3d(0, 0, 0));
        acPts3dPMesh.Add(new Point3d(2, 0, 1));
        acPts3dPMesh.Add(new Point3d(4, 0, 0));
        acPts3dPMesh.Add(new Point3d(6, 0, 1));

        acPts3dPMesh.Add(new Point3d(0, 2, 0));
        acPts3dPMesh.Add(new Point3d(2, 2, 1));
        acPts3dPMesh.Add(new Point3d(4, 2, 0));
        acPts3dPMesh.Add(new Point3d(6, 2, 1));

        acPts3dPMesh.Add(new Point3d(0, 4, 0));
        acPts3dPMesh.Add(new Point3d(2, 4, 1));
        acPts3dPMesh.Add(new Point3d(4, 4, 0));
        acPts3dPMesh.Add(new Point3d(6, 4, 0));

        acPts3dPMesh.Add(new Point3d(0, 6, 0));
        acPts3dPMesh.Add(new Point3d(2, 6, 1));
        acPts3dPMesh.Add(new Point3d(4, 6, 0));
        acPts3dPMesh.Add(new Point3d(6, 6, 0));

        foreach (Point3d acPt3d in acPts3dPMesh)
        {
            PolygonMeshVertex acPMeshVer = new PolygonMeshVertex(acPt3d);
            acPolyMesh.AppendVertex(acPMeshVer);
            acTrans.AddNewlyCreatedDBObject(acPMeshVer, true);
        }

        // Open the active viewport
        ViewportTableRecord acVportTblRec;
        acVportTblRec = acTrans.GetObject(acDoc.Editor.ActiveViewportId,
                                          OpenMode.ForWrite) as ViewportTableRecord;

        // Rotate the view direction of the current viewport
        acVportTblRec.ViewDirection = new Vector3d(-1, -1, 1);
        acDoc.Editor.UpdateTiledViewportsFromDatabase();

        // Save the new objects to the database
        acTrans.Commit();
    }
}
```

⊟ *VBA/ActiveX Code Reference*

```
Sub Create3DMesh()
    Dim meshObj As AcadPolygonMesh
    Dim mSize, nSize, Count As Integer

    ' create the matrix of points
    Dim points(0 To 47) As Double
    points(0) = 0: points(1) = 0: points(2) = 0
    points(3) = 2: points(4) = 0: points(5) = 1
    points(6) = 4: points(7) = 0: points(8) = 0
    points(9) = 6: points(10) = 0: points(11) = 1
    points(12) = 0: points(13) = 2: points(14) = 0
    points(15) = 2: points(16) = 2: points(17) = 1
    points(18) = 4: points(19) = 2: points(20) = 0
    points(21) = 6: points(22) = 2: points(23) = 1
    points(24) = 0: points(25) = 4: points(26) = 0
    points(27) = 2: points(28) = 4: points(29) = 1
```

```
    points(30) = 4: points(31) = 4: points(32) = 0
    points(33) = 6: points(34) = 4: points(35) = 0
    points(36) = 0: points(37) = 6: points(38) = 0
    points(39) = 2: points(40) = 6: points(41) = 1
    points(42) = 4: points(43) = 6: points(44) = 0
    points(45) = 6: points(46) = 6: points(47) = 0

    mSize = 4: nSize = 4

    ' creates a 3Dmesh in model space
    Set meshObj = ThisDrawing.ModelSpace. _
                    Add3DMesh(mSize, nSize, points)

    ' Change the viewing direction of the viewport
    ' to better see the cylinder
    Dim NewDirection(0 To 2) As Double
    NewDirection(0) = -1
    NewDirection(1) = -1
    NewDirection(2) = 1
    ThisDrawing.ActiveViewport.direction = NewDirection
    ThisDrawing.ActiveViewport = ThisDrawing.ActiveViewport

    ZoomAll
End Sub
```

Create Polyface Meshes

A polyface mesh represents the surface of an object defined by faces capable of having numerous vertices. Creating a polyface mesh is similar to creating a rectangular mesh. You create a polyface mesh by creating an instance of a PolyFaceMesh object. The constructor of the PolyFaceMesh object does not accept any parameters. To add a vertex to a polyface mesh, you create a PolyFaceMeshVertex and add it to the PolyFaceMesh object using the AppendVertex method.

As you create the polyface mesh, you can set specific edges to be invisible, assign them to layers, or give them colors. To make an edge invisible, you create an instance of a FaceRecord and set which edges should be invisible and then append the FaceRecord object to the PolyFaceMesh object using the AppendFaceRecord method.

Create a polyface mesh

This example creates a PolyfaceMesh object and changes the viewing direction of the active viewport to display the three-dimensional nature of the mesh.

VB.NET

```
Imports Autodesk.AutoCAD.Runtime
Imports Autodesk.AutoCAD.ApplicationServices
Imports Autodesk.AutoCAD.DatabaseServices
Imports Autodesk.AutoCAD.Geometry

<CommandMethod("CreatePolyfaceMesh")> _
Public Sub CreatePolyfaceMesh()
  '' Get the current document and database, and start a transaction
  Dim acDoc As Document = Application.DocumentManager.MdiActiveDocument
```

```vbnet
    Dim acCurDb As Database = acDoc.Database

    Using acTrans As Transaction = acCurDb.TransactionManager.StartTransaction()
        '' Open the Block table for read
        Dim acBlkTbl As BlockTable
        acBlkTbl = acTrans.GetObject(acCurDb.BlockTableId, _
                                     OpenMode.ForRead)

        '' Open the Block table record Model space for write
        Dim acBlkTblRec As BlockTableRecord
        acBlkTblRec = acTrans.GetObject(acBlkTbl(BlockTableRecord.ModelSpace), _
                                        OpenMode.ForWrite)

        '' Create a polyface mesh
        Dim acPFaceMesh As PolyFaceMesh = New PolyFaceMesh()
        acPFaceMesh.SetDatabaseDefaults()

        '' Add the new object to the block table record and the transaction
        acBlkTblRec.AppendEntity(acPFaceMesh)
        acTrans.AddNewlyCreatedDBObject(acPFaceMesh, True)

        '' Before adding vertexes, the polyline must be in the drawing
        Dim acPts3dPFMesh As Point3dCollection = New Point3dCollection()
        acPts3dPFMesh.Add(New Point3d(4, 7, 0))
        acPts3dPFMesh.Add(New Point3d(5, 7, 0))
        acPts3dPFMesh.Add(New Point3d(6, 7, 0))

        acPts3dPFMesh.Add(New Point3d(4, 6, 0))
        acPts3dPFMesh.Add(New Point3d(5, 6, 0))
        acPts3dPFMesh.Add(New Point3d(6, 6, 1))

        For Each acPt3d As Point3d In acPts3dPFMesh
            Dim acPMeshVer As PolyFaceMeshVertex = New PolyFaceMeshVertex(acPt3d)
            acPFaceMesh.AppendVertex(acPMeshVer)
            acTrans.AddNewlyCreatedDBObject(acPMeshVer, True)
        Next

        Dim acFaceRec1 As FaceRecord = New FaceRecord(1, 2, 5, 4)
        acPFaceMesh.AppendFaceRecord(acFaceRec1)
        acTrans.AddNewlyCreatedDBObject(acFaceRec1, True)

        Dim acFaceRec2 As FaceRecord = New FaceRecord(2, 3, 6, 5)
        acPFaceMesh.AppendFaceRecord(acFaceRec2)
        acTrans.AddNewlyCreatedDBObject(acFaceRec2, True)

        '' Open the active viewport
        Dim acVportTblRec As ViewportTableRecord
        acVportTblRec = acTrans.GetObject(acDoc.Editor.ActiveViewportId, _
                                          OpenMode.ForWrite)

        '' Rotate the view direction of the current viewport
        acVportTblRec.ViewDirection = New Vector3d(-1, -1, 1)
        acDoc.Editor.UpdateTiledViewportsFromDatabase()

        '' Save the new objects to the database
        acTrans.Commit()
    End Using
End Sub
```

C#

```csharp
using Autodesk.AutoCAD.Runtime;
using Autodesk.AutoCAD.ApplicationServices;
using Autodesk.AutoCAD.DatabaseServices;
```

```csharp
using Autodesk.AutoCAD.Geometry;

[CommandMethod("CreatePolyfaceMesh")]
public static void CreatePolyfaceMesh()
{
  // Get the current document and database, and start a transaction
  Document acDoc = Application.DocumentManager.MdiActiveDocument;
  Database acCurDb = acDoc.Database;

  using (Transaction acTrans = acCurDb.TransactionManager.StartTransaction())
  {
      // Open the Block table record for read
      BlockTable acBlkTbl;
      acBlkTbl = acTrans.GetObject(acCurDb.BlockTableId,
                                   OpenMode.ForRead) as BlockTable;

      // Open the Block table record Model space for write
      BlockTableRecord acBlkTblRec;
      acBlkTblRec = acTrans.GetObject(acBlkTbl[BlockTableRecord.ModelSpace],
                                      OpenMode.ForWrite) as BlockTableRecord;

      // Create a polyface mesh
      PolyFaceMesh acPFaceMesh = new PolyFaceMesh();
      acPFaceMesh.SetDatabaseDefaults();

      // Add the new object to the block table record and the transaction
      acBlkTblRec.AppendEntity(acPFaceMesh);
      acTrans.AddNewlyCreatedDBObject(acPFaceMesh, true);

      // Before adding vertexes, the polyline must be in the drawing
      Point3dCollection acPts3dPFMesh = new Point3dCollection();
      acPts3dPFMesh.Add(new Point3d(4, 7, 0));
      acPts3dPFMesh.Add(new Point3d(5, 7, 0));
      acPts3dPFMesh.Add(new Point3d(6, 7, 0));

      acPts3dPFMesh.Add(new Point3d(4, 6, 0));
      acPts3dPFMesh.Add(new Point3d(5, 6, 0));
      acPts3dPFMesh.Add(new Point3d(6, 6, 1));

      foreach (Point3d acPt3d in acPts3dPFMesh)
      {
          PolyFaceMeshVertex acPMeshVer = new PolyFaceMeshVertex(acPt3d);
          acPFaceMesh.AppendVertex(acPMeshVer);
          acTrans.AddNewlyCreatedDBObject(acPMeshVer, true);
      }

      FaceRecord acFaceRec1 = new FaceRecord(1, 2, 5, 4);
      acPFaceMesh.AppendFaceRecord(acFaceRec1);
      acTrans.AddNewlyCreatedDBObject(acFaceRec1, true);

      FaceRecord acFaceRec2 = new FaceRecord(2, 3, 6, 5);
      acPFaceMesh.AppendFaceRecord(acFaceRec2);
      acTrans.AddNewlyCreatedDBObject(acFaceRec2, true);

      // Open the active viewport
      ViewportTableRecord acVportTblRec;
      acVportTblRec = acTrans.GetObject(acDoc.Editor.ActiveViewportId,
                                        OpenMode.ForWrite) as ViewportTableRecord;

      // Rotate the view direction of the current viewport
      acVportTblRec.ViewDirection = new Vector3d(-1, -1, 1);
      acDoc.Editor.UpdateTiledViewportsFromDatabase();

      // Save the new objects to the database
      acTrans.Commit();
```

```
    }
}
```

```
Sub CreatePolyfaceMesh()
    'Define the mesh vertices
    Dim vertex(0 To 17) As Double
    vertex(0) = 4: vertex(1) = 7: vertex(2) = 0
    vertex(3) = 5: vertex(4) = 7: vertex(5) = 0
    vertex(6) = 6: vertex(7) = 7: vertex(8) = 0
    vertex(9) = 4: vertex(10) = 6: vertex(11) = 0
    vertex(12) = 5: vertex(13) = 6: vertex(14) = 0
    vertex(15) = 6: vertex(16) = 6: vertex(17) = 1

    ' Define the face list
    Dim FaceList(0 To 7) As Integer
    FaceList(0) = 1
    FaceList(1) = 2
    FaceList(2) = 5
    FaceList(3) = 4
    FaceList(4) = 2
    FaceList(5) = 3
    FaceList(6) = 6
    FaceList(7) = 5

    ' Create the polyface mesh
    Dim polyfaceMeshObj As AcadPolyfaceMesh
    Set polyfaceMeshObj = ThisDrawing.ModelSpace. _
                         AddPolyfaceMesh(vertex, FaceList)

    ' Change the viewing direction of the viewport to
    ' better see the polyface mesh
    Dim NewDirection(0 To 2) As Double
    NewDirection(0) = -1
    NewDirection(1) = -1
    NewDirection(2) = 1
    ThisDrawing.ActiveViewport.direction = NewDirection
    ThisDrawing.ActiveViewport = ThisDrawing.ActiveViewport

    ZoomAll
End Sub
```

Create Solids

A solid object (Solid3d object) represents the entire volume of an object. Solids are the most informationally complete and least ambiguous of the 3D modeling types. Complex solid shapes are also easier to construct and edit than wireframes and meshes.

You create basic solid shapes, such as a box, sphere, and wedge among others with the member mthods and properties of the Solid3d object. You can also extrude region objects along a path or revolving a 2D object about an axis.

Like meshes, solids are displayed as wireframes until you hide, shade, or render them. Additionally, you can analyze solids for their mass properties (volume, moments of inertia,

center of gravity, and so forth). Use the following MassProperties property you can query the Solid3dMassProperties object associated with the Solid3d object. The Solid3dMassProperties object contains the following properties in which allow you to to analyze the solid: MomentOfInertia, PrincipalAxess, PrincipalMoments, ProductOfInertia, RadiiOfGyration, and Volume.

The display of a solid is affected by the current visual style and 3D modeling related system variables. Some of the system variables that affect the display of a solid are ISOLINES and FACETRES. ISOLINES controls the number of tessellation lines used to visualize curved portions of the wireframe, while FACETRES adjusts the smoothness of shaded and hidden-line objects.

For more information on creating solids, see "Create 3D Objects" in the *AutoCAD User's Guide*.

Create a wedge solid

The following example creates a wedge-shaped solid. The viewing direction of the active viewport is updated to display the three-dimensional nature of the wedge.

VB.NET

```
Imports Autodesk.AutoCAD.Runtime
Imports Autodesk.AutoCAD.ApplicationServices
Imports Autodesk.AutoCAD.DatabaseServices
Imports Autodesk.AutoCAD.Geometry

<CommandMethod("CreateWedge")> _
Public Sub CreateWedge()
  '' Get the current document and database, and start a transaction
  Dim acDoc As Document = Application.DocumentManager.MdiActiveDocument
  Dim acCurDb As Database = acDoc.Database

  Using acTrans As Transaction = acCurDb.TransactionManager.StartTransaction()
      '' Open the Block table for read
      Dim acBlkTbl As BlockTable
      acBlkTbl = acTrans.GetObject(acCurDb.BlockTableId, _
                                   OpenMode.ForRead)

      '' Open the Block table record Model space for write
      Dim acBlkTblRec As BlockTableRecord
      acBlkTblRec = acTrans.GetObject(acBlkTbl(BlockTableRecord.ModelSpace), _
                                      OpenMode.ForWrite)

      '' Create a 3D solid wedge
      Dim acSol3D As Solid3d = New Solid3d()
      acSol3D.SetDatabaseDefaults()
      acSol3D.CreateWedge(10, 15, 20)

      '' Position the center of the 3D solid at (5,5,0)
      acSol3D.TransformBy(Matrix3d.Displacement(New Point3d(5, 5, 0) - _
                                                Point3d.Origin))

      '' Add the new object to the block table record and the transaction
      acBlkTblRec.AppendEntity(acSol3D)
      acTrans.AddNewlyCreatedDBObject(acSol3D, True)

      '' Open the active viewport
      Dim acVportTblRec As ViewportTableRecord
      acVportTblRec = acTrans.GetObject(acDoc.Editor.ActiveViewportId, _
```

```
                                            OpenMode.ForWrite)

        '' Rotate the view direction of the current viewport
        acVportTblRec.ViewDirection = New Vector3d(-1, -1, 1)
        acDoc.Editor.UpdateTiledViewportsFromDatabase()

        '' Save the new objects to the database
        acTrans.Commit()
    End Using
End Sub
```

C#

```csharp
using Autodesk.AutoCAD.Runtime;
using Autodesk.AutoCAD.ApplicationServices;
using Autodesk.AutoCAD.DatabaseServices;
using Autodesk.AutoCAD.Geometry;

[CommandMethod("CreateWedge")]
public static void CreateWedge()
{
    // Get the current document and database, and start a transaction
    Document acDoc = Application.DocumentManager.MdiActiveDocument;
    Database acCurDb = acDoc.Database;

    using (Transaction acTrans = acCurDb.TransactionManager.StartTransaction())
    {
        // Open the Block table record for read
        BlockTable acBlkTbl;
        acBlkTbl = acTrans.GetObject(acCurDb.BlockTableId,
                                     OpenMode.ForRead) as BlockTable;

        // Open the Block table record Model space for write
        BlockTableRecord acBlkTblRec;
        acBlkTblRec = acTrans.GetObject(acBlkTbl[BlockTableRecord.ModelSpace],
                                        OpenMode.ForWrite) as BlockTableRecord;

        // Create a 3D solid wedge
        Solid3d acSol3D = new Solid3d();
        acSol3D.SetDatabaseDefaults();
        acSol3D.CreateWedge(10, 15, 20);

        // Position the center of the 3D solid at (5,5,0)
        acSol3D.TransformBy(Matrix3d.Displacement(new Point3d(5, 5, 0) -
                                                  Point3d.Origin));

        // Add the new object to the block table record and the transaction
        acBlkTblRec.AppendEntity(acSol3D);
        acTrans.AddNewlyCreatedDBObject(acSol3D, true);

        // Open the active viewport
        ViewportTableRecord acVportTblRec;
        acVportTblRec = acTrans.GetObject(acDoc.Editor.ActiveViewportId,
                                          OpenMode.ForWrite) as ViewportTableRecord;

        // Rotate the view direction of the current viewport
        acVportTblRec.ViewDirection = new Vector3d(-1, -1, 1);
        acDoc.Editor.UpdateTiledViewportsFromDatabase();

        // Save the new objects to the database
        acTrans.Commit();
    }
}
```

```
Sub CreateWedge()
    Dim wedgeObj As Acad3DSolid
    Dim center(0 To 2) As Double
    Dim length As Double
    Dim width As Double
    Dim height As Double

    ' Define the wedge
    center(0) = 5#: center(1) = 5#: center(2) = 0
    length = 10#: width = 15#: height = 20#

    ' Create the wedge in model space
    Set wedgeObj = ThisDrawing.ModelSpace. _
                        AddWedge(center, length, width, height)

    ' Change the viewing direction of the viewport
    Dim NewDirection(0 To 2) As Double
    NewDirection(0) = -1
    NewDirection(1) = -1
    NewDirection(2) = 1
    ThisDrawing.ActiveViewport.direction = NewDirection
    ThisDrawing.ActiveViewport = ThisDrawing.ActiveViewport

    ZoomAll
End Sub
```

Edit in 3D

This section describes how to edit 3D objects by, for example, rotating, arraying, and mirroring.

Topics in this section

- [Rotate in 3D](#)
- [Array in 3D](#)
- [Mirror Objects Along a Plane](#)

Rotate in 3D

With the TransformBy method of an object and the Rotation method of a Matrix, you can rotate objects in 2D about a specified point. The direction of rotation for 2D objects is around the Z axis. For 3D objects, the axis of rotation is not limited to the Z axis. When using the Rotation method instead of using the Z axis for the rotation axis, you specify a specific 3D vector.

object to rotate selected   axis of rotation specified   result
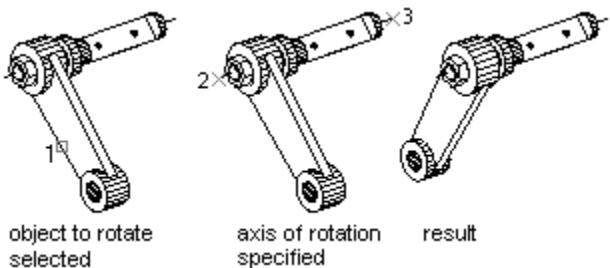
For more information on rotating in 3D, see "Rotate Objects" in the *AutoCAD User's Guide*.

Create a 3D box and rotate it about an axis

This example creates a 3D box. It then defines the axis for rotation and finally rotates the box 30 degrees about the axis.

VB.NET

```
Imports Autodesk.AutoCAD.Runtime
Imports Autodesk.AutoCAD.ApplicationServices
Imports Autodesk.AutoCAD.DatabaseServices
Imports Autodesk.AutoCAD.Geometry

<CommandMethod("Rotate_3DBox")> _
Public Sub Rotate_3DBox()
  '' Get the current document and database, and start a transaction
  Dim acDoc As Document = Application.DocumentManager.MdiActiveDocument
  Dim acCurDb As Database = acDoc.Database

  Using acTrans As Transaction = acCurDb.TransactionManager.StartTransaction()
      '' Open the Block table for read
      Dim acBlkTbl As BlockTable
      acBlkTbl = acTrans.GetObject(acCurDb.BlockTableId, _
                                   OpenMode.ForRead)

      '' Open the Block table record Model space for write
      Dim acBlkTblRec As BlockTableRecord
      acBlkTblRec = acTrans.GetObject(acBlkTbl(BlockTableRecord.ModelSpace), _
                                      OpenMode.ForWrite)

      '' Create a 3D solid box
      Dim acSol3D As Solid3d = New Solid3d()
      acSol3D.SetDatabaseDefaults()
      acSol3D.CreateBox(5, 7, 10)

      '' Position the center of the 3D solid at (5,5,0)
      acSol3D.TransformBy(Matrix3d.Displacement(New Point3d(5, 5, 0) - _
                                                Point3d.Origin))

      Dim curUCSMatrix As Matrix3d = acDoc.Editor.CurrentUserCoordinateSystem
      Dim curUCS As CoordinateSystem3d = curUCSMatrix.CoordinateSystem3d

      '' Rotate the 3D solid 30 degrees around the axis that is
      '' defined by the points (-3,4,0) and (-3,-4,0)
      Dim vRot As Vector3d = New Point3d(-3, 4, 0). _
                                        GetVectorTo(New Point3d(-3, -4, 0))

      acSol3D.TransformBy(Matrix3d.Rotation(0.5236, _
                                            vRot, _
                                            New Point3d(-3, 4, 0)))

      '' Add the new object to the block table record and the transaction
```

```
      acBlkTblRec.AppendEntity(acSol3D)
      acTrans.AddNewlyCreatedDBObject(acSol3D, True)

      '' Save the new objects to the database
      acTrans.Commit()
  End Using
End Sub
```

C#

```csharp
using Autodesk.AutoCAD.Runtime;
using Autodesk.AutoCAD.ApplicationServices;
using Autodesk.AutoCAD.DatabaseServices;
using Autodesk.AutoCAD.Geometry;

[CommandMethod("Rotate_3DBox")]
public static void Rotate_3DBox()
{
  // Get the current document and database, and start a transaction
  Document acDoc = Application.DocumentManager.MdiActiveDocument;
  Database acCurDb = acDoc.Database;

  using (Transaction acTrans = acCurDb.TransactionManager.StartTransaction())
  {
      // Open the Block table for read
      BlockTable acBlkTbl;
      acBlkTbl = acTrans.GetObject(acCurDb.BlockTableId,
                                   OpenMode.ForRead) as BlockTable;

      // Open the Block table record Model space for write
      BlockTableRecord acBlkTblRec;
      acBlkTblRec = acTrans.GetObject(acBlkTbl[BlockTableRecord.ModelSpace],
                                      OpenMode.ForWrite) as BlockTableRecord;

      // Create a 3D solid box
      Solid3d acSol3D = new Solid3d();
      acSol3D.SetDatabaseDefaults();
      acSol3D.CreateBox(5, 7, 10);

      // Position the center of the 3D solid at (5,5,0)
      acSol3D.TransformBy(Matrix3d.Displacement(new Point3d(5, 5, 0) -
                                                Point3d.Origin));

      Matrix3d curUCSMatrix = acDoc.Editor.CurrentUserCoordinateSystem;
      CoordinateSystem3d curUCS = curUCSMatrix.CoordinateSystem3d;

      // Rotate the 3D solid 30 degrees around the axis that is
      // defined by the points (-3,4,0) and (-3,-4,0)
      Vector3d vRot = new Point3d(-3, 4, 0).
                              GetVectorTo(new Point3d(-3, -4, 0));

      acSol3D.TransformBy(Matrix3d.Rotation(0.5236,
                                            vRot,
                                            new Point3d(-3, 4, 0)));

      // Add the new object to the block table record and the transaction
      acBlkTblRec.AppendEntity(acSol3D);
      acTrans.AddNewlyCreatedDBObject(acSol3D, true);

      // Save the new objects to the database
      acTrans.Commit();
  }
}
```

```
Sub Rotate_3DBox()
    Dim boxObj As Acad3DSolid
    Dim length As Double
    Dim width As Double
    Dim height As Double
    Dim center(0 To 2) As Double

    ' Define the box
    center(0) = 5: center(1) = 5: center(2) = 0
    length = 5
    width = 7
    height = 10

    ' Create the box object in model space
    Set boxObj = ThisDrawing.ModelSpace. _
                    AddBox(center, length, width, height)

    ' Define the rotation axis with two points
    Dim rotatePt1(0 To 2) As Double
    Dim rotatePt2(0 To 2) As Double
    Dim rotateAngle As Double
    rotatePt1(0) = -3: rotatePt1(1) = 4: rotatePt1(2) = 0
    rotatePt2(0) = -3: rotatePt2(1) = -4: rotatePt2(2) = 0
    rotateAngle = 30
    rotateAngle = rotateAngle * 3.141592 / 180#

    ' Rotate the box
    boxObj.Rotate3D rotatePt1, rotatePt2, rotateAngle

    ZoomAll
End Sub
```

Array in 3D

With the TransformBy and Clone methods of an object, you can create a 3D rectangular array. In addition to specifying the number of columns (*X* direction) and rows (*Y* direction) like you would for a 2D rectangular array, you also specify the number of levels (*Z* direction).

For more information on using arrays of objects in 3D, see "Create an Array of Objects" in the *AutoCAD User's Guide*.

Create a 3D rectangular array

This example creates a circle and then uses that circle to create a rectangular array of four rows, four columns, and three levels of circles.

VB.NET

```
Imports Autodesk.AutoCAD.Runtime
Imports Autodesk.AutoCAD.ApplicationServices
Imports Autodesk.AutoCAD.DatabaseServices
Imports Autodesk.AutoCAD.Geometry
```

```
Public Shared Function PolarPoints(ByVal pPt As Point2d, _
                                   ByVal dAng As Double, _
                                   ByVal dDist As Double)

    Return New Point2d(pPt.X + dDist * Math.Cos(dAng), _
                       pPt.Y + dDist * Math.Sin(dAng))
End Function

<CommandMethod("CreateRectangular3DArray")> _
Public Sub CreateRectangular3DArray()
  '' Get the current document and database, and start a transaction
  Dim acDoc As Document = Application.DocumentManager.MdiActiveDocument
  Dim acCurDb As Database = acDoc.Database

  Using acTrans As Transaction = acCurDb.TransactionManager.StartTransaction()
      '' Open the Block table for read
      Dim acBlkTbl As BlockTable
      acBlkTbl = acTrans.GetObject(acCurDb.BlockTableId, _
                                   OpenMode.ForRead)

      // Open the Block table record for read
      BlockTable acBlkTbl;
      acBlkTbl = acTrans.GetObject(acCurDb.BlockTableId,
                                   OpenMode.ForRead) as BlockTable;

      // Open the Block table record Model space for write
      BlockTableRecord acBlkTblRec;
      acBlkTblRec = acTrans.GetObject(acBlkTbl[BlockTableRecord.ModelSpace],
                                      OpenMode.ForWrite) as BlockTableRecord;

      '' Create a circle that is at 2,2 with a radius of 0.5
      Dim acCirc As Circle = New Circle()
      acCirc.SetDatabaseDefaults()
      acCirc.Center = New Point3d(2, 2, 0)
      acCirc.Radius = 0.5

      '' Add the new object to the block table record and the transaction
      acBlkTblRec.AppendEntity(acCirc)
      acTrans.AddNewlyCreatedDBObject(acCirc, True)

      '' Create a rectangular array with 4 rows, 4 columns, and 3 levels
      Dim nRows As Integer = 4
      Dim nColumns As Integer = 4
      Dim nLevels As Integer = 3

      '' Set the row, column, and level offsets along with the base array angle
      Dim dRowOffset As Double = 1
      Dim dColumnOffset As Double = 1
      Dim dLevelsOffset As Double = 4
      Dim dArrayAng As Double = 0

      '' Get the angle from X for the current UCS
      Dim curUCSMatrix As Matrix3d = acDoc.Editor.CurrentUserCoordinateSystem
      Dim curUCS As CoordinateSystem3d = curUCSMatrix.CoordinateSystem3d
      Dim acVec2dAng As Vector2d = New Vector2d(curUCS.Xaxis.X, _
                                                curUCS.Xaxis.Y)

      '' If the UCS is rotated, adjust the array angle accordingly
      dArrayAng = dArrayAng + acVec2dAng.Angle

      '' Use the upper-left corner of the objects extents for the array base point
      Dim acExts As Extents3d = acCirc.Bounds.GetValueOrDefault()
      Dim acPt2dArrayBase As Point2d = New Point2d(acExts.MinPoint.X, _
                                                   acExts.MaxPoint.Y)
```

```vbnet
'' Track the objects created for each column
Dim acDBObjCollCols As DBObjectCollection = New DBObjectCollection()
acDBObjCollCols.Add(acCirc)

'' Create the number of objects for the first column
Dim nColumnsCount As Integer = 1
While (nColumns > nColumnsCount)
    Dim acEntClone As Entity = acCirc.Clone()
    acDBObjCollCols.Add(acEntClone)

    '' Caclucate the new point for the copied object (move)
    Dim acPt2dTo As Point2d = PolarPoints(acPt2dArrayBase, _
                                          dArrayAng, _
                                          dColumnOffset * nColumnsCount)

    Dim acVec2d As Vector2d = acPt2dArrayBase.GetVectorTo(acPt2dTo)
    Dim acVec3d As Vector3d = New Vector3d(acVec2d.X, acVec2d.Y, 0)
    acEntClone.TransformBy(Matrix3d.Displacement(acVec3d))

    acBlkTblRec.AppendEntity(acEntClone)
    acTrans.AddNewlyCreatedDBObject(acEntClone, True)

    nColumnsCount = nColumnsCount + 1
End While

'' Set a value in radians for 90 degrees
Dim dAng As Double = 1.5708

'' Track the objects created for each row and column
Dim acDBObjCollLvls As DBObjectCollection = New DBObjectCollection()

For Each acObj As DBObject In acDBObjCollCols
    acDBObjCollLvls.Add(acObj)
Next

'' Create the number of objects for each row
For Each acEnt As Entity In acDBObjCollCols
    Dim nRowsCount As Integer = 1

    While (nRows > nRowsCount)
        Dim acEntClone As Entity = acEnt.Clone()
        acDBObjCollLvls.Add(acEntClone)

        '' Caclucate the new point for the copied object (move)
        Dim acPt2dTo As Point2d = PolarPoints(acPt2dArrayBase, _
                                              dArrayAng + dAng, _
                                              dRowOffset * nRowsCount)

        Dim acVec2d As Vector2d = acPt2dArrayBase.GetVectorTo(acPt2dTo)
        Dim acVec3d As Vector3d = New Vector3d(acVec2d.X, acVec2d.Y, 0)
        acEntClone.TransformBy(Matrix3d.Displacement(acVec3d))

        acBlkTblRec.AppendEntity(acEntClone)
        acTrans.AddNewlyCreatedDBObject(acEntClone, True)

        nRowsCount = nRowsCount + 1
    End While
Next

'' Create the number of levels for a 3D array
For Each acEnt As Entity In acDBObjCollLvls
    Dim nLvlsCount As Integer = 1

    While (nLevels > nLvlsCount)
```

```
            Dim acEntClone As Entity = acEnt.Clone()

            Dim acVec3d As Vector3d = New Vector3d(0, 0, dLevelsOffset *
nLvlsCount)
            acEntClone.TransformBy(Matrix3d.Displacement(acVec3d))

            acBlkTblRec.AppendEntity(acEntClone)
            acTrans.AddNewlyCreatedDBObject(acEntClone, True)

            nLvlsCount = nLvlsCount + 1
        End While
    Next

    '' Save the new objects to the database
    acTrans.Commit()
  End Using
End Sub
```

C#

```csharp
using Autodesk.AutoCAD.Runtime;
using Autodesk.AutoCAD.ApplicationServices;
using Autodesk.AutoCAD.DatabaseServices;
using Autodesk.AutoCAD.Geometry;

static Point2d PolarPoints(Point2d pPt, double dAng, double dDist)
{
    return new Point2d(pPt.X + dDist * Math.Cos(dAng),
 pPt.Y + dDist * Math.Sin(dAng));
}

[CommandMethod("CreateRectangular3DArray")]
public static void CreateRectangular3DArray()
{
  // Get the current document and database, and start a transaction
  Document acDoc = Application.DocumentManager.MdiActiveDocument;
  Database acCurDb = acDoc.Database;

  using (Transaction acTrans = acCurDb.TransactionManager.StartTransaction())
  {
      // Open the Block table record for read
      BlockTable acBlkTbl;
      acBlkTbl = acTrans.GetObject(acCurDb.BlockTableId,
                                   OpenMode.ForRead) as BlockTable;

      // Open the Block table record Model space for write
      BlockTableRecord acBlkTblRec;
      acBlkTblRec = acTrans.GetObject(acBlkTbl[BlockTableRecord.ModelSpace],
                                      OpenMode.ForWrite) as BlockTableRecord;

      // Create a circle that is at 2,2 with a radius of 0.5
      Circle acCirc = new Circle();
      acCirc.SetDatabaseDefaults();
      acCirc.Center = new Point3d(2, 2, 0);
      acCirc.Radius = 0.5;

      // Add the new object to the block table record and the transaction
      acBlkTblRec.AppendEntity(acCirc);
      acTrans.AddNewlyCreatedDBObject(acCirc, true);

      // Create a rectangular array with 4 rows, 4 columns, and 3 levels
      int nRows = 4;
      int nColumns = 4;
      int nLevels = 3;
```

```csharp
            // Set the row, column, and level offsets along with the base array angle
            double dRowOffset = 1;
            double dColumnOffset = 1;
            double dLevelsOffset = 4;
            double dArrayAng = 0;

            // Get the angle from X for the current UCS
            Matrix3d curUCSMatrix = acDoc.Editor.CurrentUserCoordinateSystem;
            CoordinateSystem3d curUCS = curUCSMatrix.CoordinateSystem3d;
            Vector2d acVec2dAng = new Vector2d(curUCS.Xaxis.X,
                                               curUCS.Xaxis.Y);

            // If the UCS is rotated, adjust the array angle accordingly
            dArrayAng = dArrayAng + acVec2dAng.Angle;

            // Use the upper-left corner of the objects extents for the array base point
            Extents3d acExts = acCirc.Bounds.GetValueOrDefault();
            Point2d acPt2dArrayBase = new Point2d(acExts.MinPoint.X,
                                                  acExts.MaxPoint.Y);

            // Track the objects created for each column
            DBObjectCollection acDBObjCollCols = new DBObjectCollection();
            acDBObjCollCols.Add(acCirc);

            // Create the number of objects for the first column
            int nColumnsCount = 1;
            while (nColumns > nColumnsCount)
            {
                Entity acEntClone = acCirc.Clone() as Entity;
                acDBObjCollCols.Add(acEntClone);

                // Caclucate the new point for the copied object (move)
                Point2d acPt2dTo = PolarPoints(acPt2dArrayBase,
                                               dArrayAng,
                                               dColumnOffset * nColumnsCount);

                Vector2d acVec2d = acPt2dArrayBase.GetVectorTo(acPt2dTo);
                Vector3d acVec3d = new Vector3d(acVec2d.X, acVec2d.Y, 0);
                acEntClone.TransformBy(Matrix3d.Displacement(acVec3d));

                acBlkTblRec.AppendEntity(acEntClone);
                acTrans.AddNewlyCreatedDBObject(acEntClone, true);

                nColumnsCount = nColumnsCount + 1;
            }

            // Set a value in radians for 90 degrees
            double dAng = 1.5708;

            // Track the objects created for each row and column
            DBObjectCollection acDBObjCollLvls = new DBObjectCollection();

            foreach (DBObject acObj in acDBObjCollCols)
            {
                acDBObjCollLvls.Add(acObj);
            }

            // Create the number of objects for each row
            foreach (Entity acEnt in acDBObjCollCols)
            {
                int nRowsCount = 1;

                while (nRows > nRowsCount)
                {
```

```csharp
                Entity acEntClone = acEnt.Clone() as Entity;
                acDBObjCollLvls.Add(acEntClone);

                // Caclucate the new point for the copied object (move)
                Point2d acPt2dTo = PolarPoints(acPt2dArrayBase,
                                               dArrayAng + dAng,
                                               dRowOffset * nRowsCount);

                Vector2d acVec2d = acPt2dArrayBase.GetVectorTo(acPt2dTo);
                Vector3d acVec3d = new Vector3d(acVec2d.X, acVec2d.Y, 0);
                acEntClone.TransformBy(Matrix3d.Displacement(acVec3d));

                acBlkTblRec.AppendEntity(acEntClone);
                acTrans.AddNewlyCreatedDBObject(acEntClone, true);

                nRowsCount = nRowsCount + 1;
            }
        }

        // Create the number of levels for a 3D array
        foreach (Entity acEnt in acDBObjCollLvls)
        {
            int nLvlsCount = 1;

            while (nLevels > nLvlsCount)
            {
                Entity acEntClone = acEnt.Clone() as Entity;

                Vector3d acVec3d = new Vector3d(0, 0, dLevelsOffset * nLvlsCount);
                acEntClone.TransformBy(Matrix3d.Displacement(acVec3d));

                acBlkTblRec.AppendEntity(acEntClone);
                acTrans.AddNewlyCreatedDBObject(acEntClone, true);

                nLvlsCount = nLvlsCount + 1;
            }
        }

        // Save the new objects to the database
        acTrans.Commit();
    }
}
```

VBA/ActiveX Code Reference

```vb
Sub CreateRectangular3DArray()
    ' Create the circle
    Dim circleObj As AcadCircle
    Dim center(0 To 2) As Double
    Dim radius As Double
    center(0) = 2: center(1) = 2: center(2) = 0
    radius = 0.5
    Set circleObj = ThisDrawing.ModelSpace. _
                    AddCircle(center, radius)

    ' Define the rectangular array
    Dim numberOfRows As Long
    Dim numberOfColumns As Long
    Dim numberOfLevels As Long
    Dim distanceBwtnRows As Double
    Dim distanceBwtnColumns As Double
    Dim distanceBwtnLevels As Double
    numberOfRows = 4
```

```
    numberOfColumns = 4
    numberOfLevels = 3
    distanceBwtnRows = 1
    distanceBwtnColumns = 1
    distanceBwtnLevels = 4

    ' Create the array of objects
    Dim retObj As Variant
    retObj = circleObj.ArrayRectangular _
                (numberOfRows, numberOfColumns, _
                 numberOfLevels, distanceBwtnRows, _
                 distanceBwtnColumns, distanceBwtnLevels)

    ZoomAll
End Sub
```

Mirror Objects Along a Plane

With the TransformBy method of an object and the Mirroring method of a Matrix, you can mirror objects along a specified mirroring plane specified by three points.

object to mirror   defining a mirror plane   result
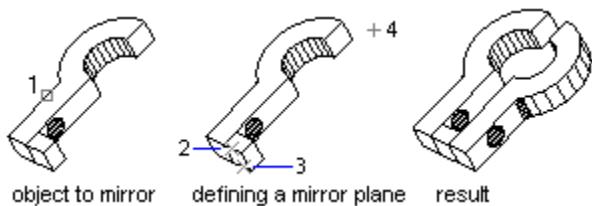
For more information on mirroring objects in 3D, see "Mirror Objects" in the *AutoCAD User's Guide*.

Mirror in 3D

This example creates a box in model space. It then mirrors the box about a plane and colors the mirrored box red.

VB.NET

```
Imports Autodesk.AutoCAD.Runtime
Imports Autodesk.AutoCAD.ApplicationServices
Imports Autodesk.AutoCAD.DatabaseServices
Imports Autodesk.AutoCAD.Geometry

<CommandMethod("MirrorABox3D")> _
Public Sub MirrorABox3D()
  '' Get the current document and database, and start a transaction
  Dim acDoc As Document = Application.DocumentManager.MdiActiveDocument
  Dim acCurDb As Database = acDoc.Database

  Using acTrans As Transaction = acCurDb.TransactionManager.StartTransaction()
      '' Open the Block table for read
      Dim acBlkTbl As BlockTable
      acBlkTbl = acTrans.GetObject(acCurDb.BlockTableId, _
                                OpenMode.ForRead)
```

```
      '' Open the Block table record Model space for write
      Dim acBlkTblRec As BlockTableRecord
      acBlkTblRec = acTrans.GetObject(acBlkTbl(BlockTableRecord.ModelSpace), _
                                       OpenMode.ForWrite)

      '' Create a 3D solid box
      Dim acSol3D As Solid3d = New Solid3d()
      acSol3D.SetDatabaseDefaults()
      acSol3D.CreateBox(5, 7, 10)

      '' Position the center of the 3D solid at (5,5,0)
      acSol3D.TransformBy(Matrix3d.Displacement(New Point3d(5, 5, 0) - _
                                                 Point3d.Origin))

      '' Add the new object to the block table record and the transaction
      acBlkTblRec.AppendEntity(acSol3D)
      acTrans.AddNewlyCreatedDBObject(acSol3D, True)

      '' Create a copy of the original 3D solid and change the color of the copy
      Dim acSol3DCopy As Solid3d = acSol3D.Clone()
      acSol3DCopy.ColorIndex = 1

      '' Define the mirror plane
      Dim acPlane As Plane = New Plane(New Point3d(1.25, 0, 0), _
                                       New Point3d(1.25, 2, 0), _
                                       New Point3d(1.25, 2, 2))

      '' Mirror the 3D solid across the plane
      acSol3DCopy.TransformBy(Matrix3d.Mirroring(acPlane))

      '' Add the new object to the block table record and the transaction
      acBlkTblRec.AppendEntity(acSol3DCopy)
      acTrans.AddNewlyCreatedDBObject(acSol3DCopy, True)

      '' Save the new objects to the database
      acTrans.Commit()
   End Using
End Sub
```

C#

```csharp
using Autodesk.AutoCAD.Runtime;
using Autodesk.AutoCAD.ApplicationServices;
using Autodesk.AutoCAD.DatabaseServices;
using Autodesk.AutoCAD.Geometry;

[CommandMethod("MirrorABox3D")]
public static void MirrorABox3D()
{
  // Get the current document and database, and start a transaction
  Document acDoc = Application.DocumentManager.MdiActiveDocument;
  Database acCurDb = acDoc.Database;

  using (Transaction acTrans = acCurDb.TransactionManager.StartTransaction())
  {
      // Open the Block table record for read
      BlockTable acBlkTbl;
      acBlkTbl = acTrans.GetObject(acCurDb.BlockTableId,
                                   OpenMode.ForRead) as BlockTable;

      // Open the Block table record Model space for write
      BlockTableRecord acBlkTblRec;
      acBlkTblRec = acTrans.GetObject(acBlkTbl[BlockTableRecord.ModelSpace],
                                      OpenMode.ForWrite) as BlockTableRecord;
```

```
      // Create a 3D solid box
      Solid3d acSol3D = new Solid3d();
      acSol3D.SetDatabaseDefaults();
      acSol3D.CreateBox(5, 7, 10);

      // Position the center of the 3D solid at (5,5,0)
      acSol3D.TransformBy(Matrix3d.Displacement(new Point3d(5, 5, 0) -
                                            Point3d.Origin));

      // Add the new object to the block table record and the transaction
      acBlkTblRec.AppendEntity(acSol3D);
      acTrans.AddNewlyCreatedDBObject(acSol3D, true);

      // Create a copy of the original 3D solid and change the color of the copy
      Solid3d acSol3DCopy = acSol3D.Clone() as Solid3d;
      acSol3DCopy.ColorIndex = 1;

      // Define the mirror plane
      Plane acPlane = new Plane(new Point3d(1.25, 0, 0),
                                new Point3d(1.25, 2, 0),
                                new Point3d(1.25, 2, 2));

      // Mirror the 3D solid across the plane
      acSol3DCopy.TransformBy(Matrix3d.Mirroring(acPlane));

      // Add the new object to the block table record and the transaction
      acBlkTblRec.AppendEntity(acSol3DCopy);
      acTrans.AddNewlyCreatedDBObject(acSol3DCopy, true);

      // Save the new objects to the database
      acTrans.Commit();
  }
}
```

⊟ *VBA/ActiveX Code Reference*

```
Sub MirrorABox3D()
    ' Create the box object
    Dim boxObj As Acad3DSolid
    Dim length As Double
    Dim width As Double
    Dim height As Double
    Dim center(0 To 2) As Double
    center(0) = 5#: center(1) = 5#: center(2) = 0
    length = 5#: width = 7: height = 10#

    ' Create the box (3DSolid) object in model space
    Set boxObj = ThisDrawing.ModelSpace. _
                    AddBox(center, length, width, height)

    ' Define the mirroring plane with three points
    Dim mirrorPt1(0 To 2) As Double
    Dim mirrorPt2(0 To 2) As Double
    Dim mirrorPt3(0 To 2) As Double

    mirrorPt1(0) = 1.25: mirrorPt1(1) = 0: mirrorPt1(2) = 0
    mirrorPt2(0) = 1.25: mirrorPt2(1) = 2: mirrorPt2(2) = 0
    mirrorPt3(0) = 1.25: mirrorPt3(1) = 2: mirrorPt3(2) = 2

    ' Mirror the box
    Dim mirrorBoxObj As Acad3DSolid
    Set mirrorBoxObj = boxObj.Mirror3D _
```

```
                                (mirrorPt1, mirrorPt2, mirrorPt3)
        mirrorBoxObj.Color = acRed

        ZoomAll
End Sub
```

Edit 3D Solids

Once you have created a solid, you can create more complex shapes by combining or
subtracting solids. You can join solids, subtract solids from each other, or find the common
volume (overlapping portion) of solids. Use the BooleanOperation method to perform these
combinations. The CheckInterference method allows you to determine if two solids overlap.
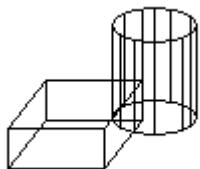
Solids are further modified by obtaining the 2D cross section of a solid or slicing a solid into
two pieces. Use the GetSection method to find cross sections of solids, and the Slice method
for slicing a solid into two pieces.

Find the interference between two solids

This example creates a box and cylinder. It then finds the interference between the two solids
and creates a new solid from that interference. For ease of viewing, the box is colored white,
the cylinder is colored cyan, and the interference solid is colored red.

VB.NET

```
Imports Autodesk.AutoCAD.Runtime
Imports Autodesk.AutoCAD.ApplicationServices
Imports Autodesk.AutoCAD.DatabaseServices
Imports Autodesk.AutoCAD.Geometry

<CommandMethod("FindInterferenceBetweenSolids")> _
Public Sub FindInterferenceBetweenSolids()
  '' Get the current document and database, and start a transaction
  Dim acDoc As Document = Application.DocumentManager.MdiActiveDocument
  Dim acCurDb As Database = acDoc.Database

  Using acTrans As Transaction = acCurDb.TransactionManager.StartTransaction()
      '' Open the Block table for read
      Dim acBlkTbl As BlockTable
      acBlkTbl = acTrans.GetObject(acCurDb.BlockTableId, _
                                   OpenMode.ForRead)

      '' Open the Block table record Model space for write
      Dim acBlkTblRec As BlockTableRecord
      acBlkTblRec = acTrans.GetObject(acBlkTbl(BlockTableRecord.ModelSpace), _
                                      OpenMode.ForWrite)
```

```vb
        '' Create a 3D solid box
        Dim acSol3DBox As Solid3d = New Solid3d()
        acSol3DBox.SetDatabaseDefaults()
        acSol3DBox.CreateBox(5, 7, 10)
        acSol3DBox.ColorIndex = 7

        '' Position the center of the 3D solid at (5,5,0)
        acSol3DBox.TransformBy(Matrix3d.Displacement(New Point3d(5, 5, 0) - _
                                                     Point3d.Origin))

        '' Add the new object to the block table record and the transaction
        acBlkTblRec.AppendEntity(acSol3DBox)
        acTrans.AddNewlyCreatedDBObject(acSol3DBox, True)

        '' Create a 3D solid cylinder
        '' 3D solids are created at (0,0,0) so there is no need to move it
        Dim acSol3DCyl As Solid3d = New Solid3d()
        acSol3DCyl.SetDatabaseDefaults()
        acSol3DCyl.CreateFrustum(20, 5, 5, 5)
        acSol3DCyl.ColorIndex = 4

        '' Add the new object to the block table record and the transaction
        acBlkTblRec.AppendEntity(acSol3DCyl)
        acTrans.AddNewlyCreatedDBObject(acSol3DCyl, True)

        '' Create a 3D solid from the interference of the box and cylinder
        Dim acSol3DCopy As Solid3d = acSol3DCyl.Clone()

        '' Check to see if the 3D solids overlap
        If acSol3DCopy.CheckInterference(acSol3DBox) = True Then
            acSol3DCopy.BooleanOperation(BooleanOperationType.BoolIntersect, _
                                         acSol3DBox.Clone())

            acSol3DCopy.ColorIndex = 1
        End If

        '' Add the new object to the block table record and the transaction
        acBlkTblRec.AppendEntity(acSol3DCopy)
        acTrans.AddNewlyCreatedDBObject(acSol3DCopy, True)

        '' Save the new objects to the database
        acTrans.Commit()
    End Using
End Sub
```

C#

```csharp
using Autodesk.AutoCAD.Runtime;
using Autodesk.AutoCAD.ApplicationServices;
using Autodesk.AutoCAD.DatabaseServices;
using Autodesk.AutoCAD.Geometry;

[CommandMethod("FindInterferenceBetweenSolids")]
public static void FindInterferenceBetweenSolids()
{
    // Get the current document and database, and start a transaction
    Document acDoc = Application.DocumentManager.MdiActiveDocument;
    Database acCurDb = acDoc.Database;

    using (Transaction acTrans = acCurDb.TransactionManager.StartTransaction())
    {
        // Open the Block table record for read
        BlockTable acBlkTbl;
```

```
    acBlkTbl = acTrans.GetObject(acCurDb.BlockTableId,
                                 OpenMode.ForRead) as BlockTable;

    // Open the Block table record Model space for write
    BlockTableRecord acBlkTblRec;
    acBlkTblRec = acTrans.GetObject(acBlkTbl[BlockTableRecord.ModelSpace],
                                    OpenMode.ForWrite) as BlockTableRecord;

    // Create a 3D solid box
    Solid3d acSol3DBox = new Solid3d();
    acSol3DBox.SetDatabaseDefaults();
    acSol3DBox.CreateBox(5, 7, 10);
    acSol3DBox.ColorIndex = 7;

    // Position the center of the 3D solid at (5,5,0)
    acSol3DBox.TransformBy(Matrix3d.Displacement(new Point3d(5, 5, 0) -
                                                 Point3d.Origin));

    // Add the new object to the block table record and the transaction
    acBlkTblRec.AppendEntity(acSol3DBox);
    acTrans.AddNewlyCreatedDBObject(acSol3DBox, true);

    // Create a 3D solid cylinder
    // 3D solids are created at (0,0,0) so there is no need to move it
    Solid3d acSol3DCyl = new Solid3d();
    acSol3DCyl.SetDatabaseDefaults();
    acSol3DCyl.CreateFrustum(20, 5, 5, 5);
    acSol3DCyl.ColorIndex = 4;

    // Add the new object to the block table record and the transaction
    acBlkTblRec.AppendEntity(acSol3DCyl);
    acTrans.AddNewlyCreatedDBObject(acSol3DCyl, true);

    // Create a 3D solid from the interference of the box and cylinder
    Solid3d acSol3DCopy = acSol3DCyl.Clone() as Solid3d;

    // Check to see if the 3D solids overlap
    if (acSol3DCopy.CheckInterference(acSol3DBox) == true)
    {
        acSol3DCopy.BooleanOperation(BooleanOperationType.BoolIntersect,
                                     acSol3DBox.Clone() as Solid3d);

        acSol3DCopy.ColorIndex = 1;
    }

    // Add the new object to the block table record and the transaction
    acBlkTblRec.AppendEntity(acSol3DCopy);
    acTrans.AddNewlyCreatedDBObject(acSol3DCopy, true);

    // Save the new objects to the database
    acTrans.Commit();
  }
}
```

☐ *VBA/ActiveX Code Reference*

```
Sub FindInterferenceBetweenSolids()
    ' Define the box
    Dim boxObj As Acad3DSolid
    Dim length As Double
    Dim width As Double
    Dim height As Double
    Dim center(0 To 2) As Double
```

```
    center(0) = 5: center(1) = 5: center(2) = 0
    length = 5
    width = 7
    height = 10

    ' Create the box object in model space
    ' and color it white
    Set boxObj = ThisDrawing.ModelSpace. _
                    AddBox(center, length, width, height)
    boxObj.Color = acWhite

    ' Define the cylinder
    Dim cylinderObj As Acad3DSolid
    Dim cylinderRadius As Double
    Dim cylinderHeight As Double
    center(0) = 0: center(1) = 0: center(2) = 0
    cylinderRadius = 5
    cylinderHeight = 20

    ' Create the Cylinder and
    ' color it cyan
    Set cylinderObj = ThisDrawing.ModelSpace. _
                        AddCylinder(center, cylinderRadius, cylinderHeight)
    cylinderObj.Color = acCyan

    ' Find the interference between the two solids
    ' and create a new solid from it. Color the
    ' new solid red.
    Dim solidObj As Acad3DSolid
    Set solidObj = boxObj.CheckInterference(cylinderObj, True)
    solidObj.Color = acRed

    ZoomAll
End Sub
```

Slice a solid into two solids

This example creates a box in model space. It then slices the box based on a plane defined by three points. The slice is returned as a 3DSolid.

VB.NET

```
Imports Autodesk.AutoCAD.Runtime
Imports Autodesk.AutoCAD.ApplicationServices
Imports Autodesk.AutoCAD.DatabaseServices
Imports Autodesk.AutoCAD.Geometry

<CommandMethod("SliceABox")> _
Public Sub SliceABox()
  '' Get the current document and database, and start a transaction
  Dim acDoc As Document = Application.DocumentManager.MdiActiveDocument
  Dim acCurDb As Database = acDoc.Database

  Using acTrans As Transaction = acCurDb.TransactionManager.StartTransaction()
      '' Open the Block table for read
      Dim acBlkTbl As BlockTable
      acBlkTbl = acTrans.GetObject(acCurDb.BlockTableId, _
                                   OpenMode.ForRead)

      '' Open the Block table record Model space for write
      Dim acBlkTblRec As BlockTableRecord
      acBlkTblRec = acTrans.GetObject(acBlkTbl(BlockTableRecord.ModelSpace), _
                                      OpenMode.ForWrite)
```

```vb
'' Create a 3D solid box
Dim acSol3D As Solid3d = New Solid3d()
acSol3D.SetDatabaseDefaults()
acSol3D.CreateBox(5, 7, 10)
acSol3D.ColorIndex = 7

'' Position the center of the 3D solid at (5,5,0)
acSol3D.TransformBy(Matrix3d.Displacement(New Point3d(5, 5, 0) - _
                                          Point3d.Origin))

'' Add the new object to the block table record and the transaction
acBlkTblRec.AppendEntity(acSol3D)
acTrans.AddNewlyCreatedDBObject(acSol3D, True)

'' Define the mirror plane
Dim acPlane As Plane = New Plane(New Point3d(1.5, 7.5, 0), _
                       New Point3d(1.5, 7.5, 10), _
                       New Point3d(8.5, 2.5, 10))

Dim acSol3DSlice As Solid3d = acSol3D.Slice(acPlane, True)
acSol3DSlice.ColorIndex = 1

'' Add the new object to the block table record and the transaction
acBlkTblRec.AppendEntity(acSol3DSlice)
acTrans.AddNewlyCreatedDBObject(acSol3DSlice, True)

'' Save the new objects to the database
acTrans.Commit()
  End Using
End Sub
```

C#

```csharp
using Autodesk.AutoCAD.Runtime;
using Autodesk.AutoCAD.ApplicationServices;
using Autodesk.AutoCAD.DatabaseServices;
using Autodesk.AutoCAD.Geometry;

[CommandMethod("SliceABox")]
public static void SliceABox()
{
  // Get the current document and database, and start a transaction
  Document acDoc = Application.DocumentManager.MdiActiveDocument;
  Database acCurDb = acDoc.Database;

  using (Transaction acTrans = acCurDb.TransactionManager.StartTransaction())
  {
      // Open the Block table record for read
      BlockTable acBlkTbl;
      acBlkTbl = acTrans.GetObject(acCurDb.BlockTableId,
                                   OpenMode.ForRead) as BlockTable;

      // Open the Block table record Model space for write
      BlockTableRecord acBlkTblRec;
      acBlkTblRec = acTrans.GetObject(acBlkTbl[BlockTableRecord.ModelSpace],
                                      OpenMode.ForWrite) as BlockTableRecord;

      // Create a 3D solid box
      Solid3d acSol3D = new Solid3d();
      acSol3D.SetDatabaseDefaults();
      acSol3D.CreateBox(5, 7, 10);
      acSol3D.ColorIndex = 7;
```

```
        // Position the center of the 3D solid at (5,5,0)
        acSol3D.TransformBy(Matrix3d.Displacement(new Point3d(5, 5, 0) -
                                                 Point3d.Origin));

        // Add the new object to the block table record and the transaction
        acBlkTblRec.AppendEntity(acSol3D);
        acTrans.AddNewlyCreatedDBObject(acSol3D, true);

        // Define the mirror plane
        Plane acPlane = new Plane(new Point3d(1.5, 7.5, 0),
                                  new Point3d(1.5, 7.5, 10),
                                  new Point3d(8.5, 2.5, 10));

        Solid3d acSol3DSlice = acSol3D.Slice(acPlane, true);
        acSol3DSlice.ColorIndex = 1;

        // Add the new object to the block table record and the transaction
        acBlkTblRec.AppendEntity(acSol3DSlice);
        acTrans.AddNewlyCreatedDBObject(acSol3DSlice, true);

        // Save the new objects to the database
        acTrans.Commit();
    }
}
```

⊟ *VBA/ActiveX Code Reference*

```
Sub SliceABox()
    ' Create the box object
    Dim boxObj As Acad3DSolid
    Dim length As Double
    Dim width As Double
    Dim height As Double
    Dim center(0 To 2) As Double
    center(0) = 5#: center(1) = 5#: center(2) = 0
    length = 5#: width = 7: height = 10#

    ' Create the box (3DSolid) object in model space
    Set boxObj = ThisDrawing.ModelSpace. _
                   AddBox(center, length, width, height)
    boxObj.Color = acWhite

    ' Define the section plane with three points
    Dim slicePt1(0 To 2) As Double
    Dim slicePt2(0 To 2) As Double
    Dim slicePt3(0 To 2) As Double

    slicePt1(0) = 1.5: slicePt1(1) = 7.5: slicePt1(2) = 0
    slicePt2(0) = 1.5: slicePt2(1) = 7.5: slicePt2(2) = 10
    slicePt3(0) = 8.5: slicePt3(1) = 2.5: slicePt3(2) = 10

    ' slice the box and color the new solid red
    Dim sliceObj As Acad3DSolid
    Set sliceObj = boxObj.SliceSolid _
                        (slicePt1, slicePt2, slicePt3, True)
    sliceObj.Color = acRed

    ZoomAll
End Sub
```

8   Define Layouts and Plot

After you have created a drawing with AutoCAD, you usually plot or publish it. A plotted drawing can contain a single view of your drawing or a more complex arrangement of views. You create windows called floating viewports in Paper space in which are used to display various views of the drawing. Depending on your needs, you can plot one or more viewports, or set options that determine what is plotted and how the image fits in the final output format.

Topics in this section

- Model Space and Paper Space
- Layouts
- Viewports
- Plot Your Drawing

Model Space and Paper Space

Model space is the drawing environment in which you create the geometry for your model. Normally, as you begin to draw in Model space, you designate your drawing limits to determine the extents of the drawing environment, and you draw in real world units.

Paper space represents the paper representation of your model as it will be plotted. In Paper space you lay out different views of your drawing, scale views independently from one another, and arrange the different views of your drawing as you want them to be plotted. There can be many different Paper space representations of your drawing.

Layouts

All the geometry of your drawing is contained in layouts. Model space geometry is contained on a single layout named Model. You cannot rename the Model space layout, nor can you create another Model space layout as there can be only one per drawing.

Paper space geometry is also contained on layouts. You can have many different Paper space layouts in your drawing, each representing a different configuration to print. You can change the name of the Paper space layouts.

The "*MODEL_SPACE" BlockTableRecord in the BlockTable contains all the geometry in the Model space layout. Because there can be more than one Paper space layout in a drawing, the "*PAPER_SPACE"BlockTableRecord in the BlockTable points to the last active Paper space layout.

For more information about working with Paper space layouts, see "Create Multiple-View Drawing Layouts (Paper Space)" in the *AutoCAD User's Guide*.

Topics in this section

Layouts and Blocks

The content of any layout is distributed among two different objects: a Layout and BlockTableRecord object. The Layout object contains the plot settings and the visual properties of the layout as it appears in the AutoCAD user interface. The BlockTableRecord object contains the geometry that is displayed on the layout such as annotation, floating viewports, and title blocks. The BlockTableRecord object also includes the Viewport object that controls the display of the drafting aids and layer properties used for the layout.

Each Layout object is associated with one, and only one, BlockTableRecord object. To access the BlockTableRecord object associated with a given layout, use the BlockTableRecordId property. Conversely, each BlockTableRecord object is associated with one, and only one, Layout object. To access the Layout object associated with a given BlockTableRecord, use the LayoutId property for that block. The IsLayout property of a BlockTableRecord can be used to determine if it has an associated Layout object; TRUE is returned if the BlockTableRecord is associated with a Layout object

Plot Settings

A PlotSettings object is similar to a Layout object, as both contain identical plot information and this is because the Layout class is derived from the PlotSettings class. The main difference is that a Layout object has an associated BlockTableRecord object containing the geometry to plot. A PlotSettings object is not associated with a particular BlockTableRecord object. It is simply a named collection of plot settings available for use with any geometry. PlotSettings objects are known as page setups in the AutoCAD user interface and are accessed from the Page Setup Manager.

Layout Settings

Layout settings control the final output of a drawing when plotted or published. These settings affect the paper size, plot scale, plot area, plot origin, and the plot device name. Understanding how to use layout settings ensures that the layout is plotted as expected. Most settings related to outputting a

layout are read-only. Changing the plot settings for a layout requires the use of the PlotSettings and PlotSettingsValidator objects.

Topics in this section

- [Paper Size and Units](#)
- [Plot Origin](#)
- [Plot Area](#)
- [Plot Scale](#)
- [Lineweight Scale](#)
- [Plot Device](#)
- [Query and Set Layout Settings](#)

Paper Size and Units

The choice of paper size depends on the plotter or device configured for output. Each plotter or device has a standard list of available output sizes. The output size for a layout is assigned can be quired with the CanonicalMediaName property.

You can also query the units for a layout using the PlotPaperUnits property. This property returns one of three values defined by the PlotPaperUnit enum: Inches, Millimeters, or Pixels. If your plotter is configured for raster output, the output size is returned in pixels.

Plot Origin

The plot origin is the lower-left corner of the specified plotted area and can be queried with the PlotOrigin property. Typically, the plot origin is set to (0, 0). However, a plot can be centered on the sheet of paper. The PlotCentered property returns if the plot is currently centered; if TRUE the plot is centered. Centering the plot alters the plot origin.

Plot Area

When a layout is plotted, the area in which is plotted is determined by the PlotType property. The value stored in the PlotType property is one of the values defined by the PlotType enum. The PlotType enum defines the following values:

Display

Prints everything that is in the current model space display. This option is unavailable when plotting from a Paper space layout.

Extents

Prints everything that falls within the boundaries of the currently selected space.

Limits

Prints everything that is in the limits of the current space.

View

Prints the view named by the PlotViewName property.

Window

Prints everything in the window specified by the PlotWindowArea property.

Layout

Prints everything that falls within the margins of the specified paper size. This option is not available when printing from Model space.

When you create a new Paper space layout, the default option is Layout.

Plot Scale

Generally, you draw objects at their actual size. When you plot the drawing, you either specify a precise scale or fit the image to the output size. You specify a scale with either a standard or custom plot scale.

A standard scale is used when the UseStandardScale property is set to TRUE. The actual scale at which the plot will be scaled to can be queried with the StdScale property.

A custom scale is used when the UseStandardScale property is set to FALSE. The custom scale at which the plot will be scaled to can be queried with the CustomPrintScale property.

When reviewing an early draft view, a precise scale is not always important. You can set the StdScaleType property to a value of ScaleToFit defined by the StdScaleType enum to plot the layout at the largest possible size that fits the output size.

Lineweight Scale

Lineweights can be scaled proportionately in a layout with the plot scale. Typically, lineweights specify the linewidth of plotted objects and are plotted with the linewidth size regardless of the plot scale. Most often, you use the default plot scale of 1:1 when plotting a layout. However, if you want to plot an E-size layout that is scaled to fit on an A-size sheet of paper, for example, you can specify lineweights to be scaled in proportion to the new plot scale.

The ScaleLineweights property retrurns whether lineweights are scaled or not; a value of TRUE indicates that lineweights are to be scaled when the layout is plotted.

Plot Device

The plot device name is stored in the `PlotConfigurationName` property. The name should match one of the devices on your system, if not the default device will be used.

Query and Set Layout Settings

The following example shows how to query and change the device of the current layout.

VB.NET

```
Imports Autodesk.AutoCAD.Runtime
Imports Autodesk.AutoCAD.ApplicationServices
Imports Autodesk.AutoCAD.DatabaseServices
Imports Autodesk.AutoCAD.PlottingServices

<CommandMethod("ChangePlotSetting")> _
Public Sub ChangePlotSetting()
  '' Get the current document and database, and start a transaction
  Dim acDoc As Document = Application.DocumentManager.MdiActiveDocument
  Dim acCurDb As Database = acDoc.Database

  Using acTrans As Transaction = acCurDb.TransactionManager.StartTransaction()
      '' Reference the Layout Manager
      Dim acLayoutMgr As LayoutManager
      acLayoutMgr = LayoutManager.Current

      '' Get the current layout and output its name in the Command Line window
      Dim acLayout As Layout
      acLayout =
acTrans.GetObject(acLayoutMgr.GetLayoutId(acLayoutMgr.CurrentLayout), _
```

```
                                        OpenMode.ForRead)

        '' Output the name of the current layout and its device
        acDoc.Editor.WriteMessage(vbLf & "Current layout: " & _
                                   acLayout.LayoutName)

        acDoc.Editor.WriteMessage(vbLf & "Current device name: " & _
                                   acLayout.PlotConfigurationName)

        '' Get the PlotInfo from the layout
        Dim acPlInfo As PlotInfo = New PlotInfo()
        acPlInfo.Layout = acLayout.ObjectId

        '' Get a copy of the PlotSettings from the layout
        Dim acPlSet As PlotSettings = New PlotSettings(acLayout.ModelType)
        acPlSet.CopyFrom(acLayout)

        '' Update the PlotConfigurationName property of the PlotSettings object
        Dim acPlSetVdr As PlotSettingsValidator = PlotSettingsValidator.Current
        acPlSetVdr.SetPlotConfigurationName(acPlSet, "DWF6 ePlot.pc3", _
                                             "ANSI_A_(8.50_x_11.00_Inches)")

        '' Update the layout
        acLayout.UpgradeOpen()
        acLayout.CopyFrom(acPlSet)

        '' Output the name of the new device assigned to the layout
        acDoc.Editor.WriteMessage(vbLf & "New device name: " & _
                                   acLayout.PlotConfigurationName)

        '' Save the new objects to the database
        acTrans.Commit()
    End Using
End Sub
```

C#

```
using Autodesk.AutoCAD.Runtime;
using Autodesk.AutoCAD.ApplicationServices;
using Autodesk.AutoCAD.DatabaseServices;
using Autodesk.AutoCAD.PlottingServices;

[CommandMethod("ChangePlotSetting")]
public static void ChangePlotSetting()
{
    // Get the current document and database, and start a transaction
    Document acDoc = Application.DocumentManager.MdiActiveDocument;
    Database acCurDb = acDoc.Database;

    using (Transaction acTrans = acCurDb.TransactionManager.StartTransaction())
    {
        // Reference the Layout Manager
        LayoutManager acLayoutMgr;
        acLayoutMgr = LayoutManager.Current;

        // Get the current layout and output its name in the Command Line window
        Layout acLayout;
        acLayout =
acTrans.GetObject(acLayoutMgr.GetLayoutId(acLayoutMgr.CurrentLayout),
                                  OpenMode.ForRead) as Layout;

        // Output the name of the current layout and its device
        acDoc.Editor.WriteMessage("\nCurrent layout: " +
                                  acLayout.LayoutName);
```

```
        acDoc.Editor.WriteMessage("\nCurrent device name: " +
                                    acLayout.PlotConfigurationName);

        // Get the PlotInfo from the layout
        PlotInfo acPlInfo = new PlotInfo();
        acPlInfo.Layout = acLayout.ObjectId;

        // Get a copy of the PlotSettings from the layout
        PlotSettings acPlSet = new PlotSettings(acLayout.ModelType);
        acPlSet.CopyFrom(acLayout);

        // Update the PlotConfigurationName property of the PlotSettings object
        PlotSettingsValidator acPlSetVdr = PlotSettingsValidator.Current;
        acPlSetVdr.SetPlotConfigurationName(acPlSet, "DWF6 ePlot.pc3",
                                             "ANSI_A_(8.50_x_11.00_Inches)");

        // Update the layout
        acLayout.UpgradeOpen();
        acLayout.CopyFrom(acPlSet);

        // Output the name of the new device assigned to the layout
        acDoc.Editor.WriteMessage("\nNew device name: " +
                                    acLayout.PlotConfigurationName);

        // Save the new objects to the database
        acTrans.Commit();
    }
}
```

⊟ *VBA/ActiveX Code Reference*

```
Public Sub ChangePlotSetting()
    Dim layoutObj As AcadLayout
    Set layoutObj = ThisDrawing.ActiveLayout

    ' Output the name of the current layout and its device
    ThisDrawing.Utility.Prompt vbLf & "Current layout: " & layoutObj.Name
    ThisDrawing.Utility.Prompt vbLf & "Current device name: " & _
                                layoutObj.ConfigName

    ' Change the name of the output device for the current layout
    layoutObj.RefreshPlotDeviceInfo
    layoutObj.ConfigName = "DWF6 ePlot.pc3"
    layoutObj.CanonicalMediaName = "ANSI_A_(8.50_x_11.00_Inches)"

    ' Output the name of the new device assigned to the layout
    ThisDrawing.Utility.Prompt vbLf & "Current device name: " & _
 layoutObj.ConfigName & vbLf
End Sub
```

Viewports

When working in Model space you draw geometry in tile viewports which are represented by ViewportTableRecord objects. You can display one or several different viewports at a time. If several tiled viewports are displayed, editing in one viewport affects all other viewports. However, you can set the magnification, viewpoint, grid, and snap settings individually for each viewport.

In Paper space, you work in floating viewports which are represented by Viewport objects and can contain different views of your model. Floating viewports are treated as objects that you can move, resize, and shape to create a suitable layout. You also can draw objects, such as title blocks or annotations, directly in the Paper space view without affecting the model itself.

tiled viewports   floating viewports
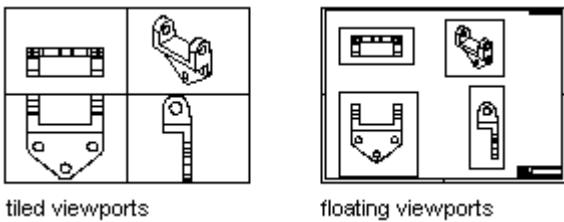
For more information about viewports, see "Display Multiple Views in Model Space" in the *AutoCAD User's Guide*.

Topics in this section

- Floating Viewports
- Create Paper Space Viewports
- Change Viewport Views and Content
- Scale Views Relative to Paper Space
- Scale Pattern Linetypes in Paper Space
- Use Shaded Viewports

Floating Viewports

You cannot edit the model from Paper space. To access the model in a Viewport object, toggle from Paper space to Model space using the SwitchToModelSpace and SwitchToPaperSpace member methods of the Editor object. As a result, you can work with the model while keeping the overall layout visible. In Viewport objects, the editing and view-changing capabilities are almost the same as in ViewportTableRecord objects.

However, you have more control over the individual views. For example, you can freeze or thaw layers in some viewports without affecting others. You can toggle the display of the geometry in a viewport on or off. You can also align views between viewports and scale the views relative to the overall layout.

The following illustration shows how different views of a model can be displayed in Paper space. Each Paper space image represents a Viewport object with a different view. In one view, the layer for dimensions is frozen. Notice that the title block, border, and annotation,

which are drawn in Paper space, do not appear in the Model space view. Also, the layer containing the viewport borders has been frozen.
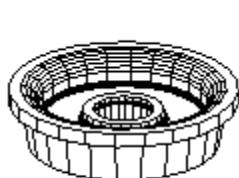
the model
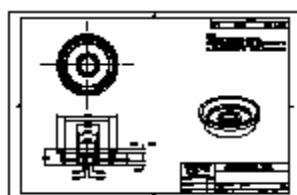
the model displayed
in floating viewports

When you are working in a Viewport object, you can be in either Model or Paper space. You can determine if you are working in Model space by checking the current values of the TILEMODE and CVPORT system variables. The following table breaks down the space and layout you are working in based on the current values of TILEMODE and CVPORT. is 0 and CVPORT is a value other than 2, you are working in Paper space, and if TILEMODE is 0 and CVPORT is 2 then you are working in Model space. If TILEMODE is 1, you are working in Model space on the Model layout.

| | | Current space |
TILEMODE	CVPORT	Status
0	Not equal to 2	Layout other than Model is active and you are working in Paper space.
0	2	Layout other than Model is active and you are working in a floating viewport.
1	Any value	Model layout is active.

NoteBefore switching to Model space on when on a layout, the On property for at least one Viewport object on the layout should be set to TRUE.

When you are in paper space, AutoCAD displays the paper space user coordinate system (UCS) icon in the lower-left corner of the graphics area. The crosshairs indicate that the paper space layout area (not the views in the viewports) can be edited.

To toggle between Model and Paper space

This example shows how to toggle between Model and Paper space.

VB.NET

```
Imports Autodesk.AutoCAD.Runtime
Imports Autodesk.AutoCAD.ApplicationServices
Imports Autodesk.AutoCAD.DatabaseServices

<CommandMethod("ToggleSpace")> _
Public Sub ToggleSpace()
  '' Get the current document
  Dim acDoc As Document = Application.DocumentManager.MdiActiveDocument

  '' Get the current values of CVPORT and TILEMODE
  Dim nCvports As Integer = Application.GetSystemVariable("CVPORT")
  Dim nTilemode As Integer = Application.GetSystemVariable("TILEMODE")

  '' Check to see if the Model layout is active, TILEMODE is 1 when
  '' the Model layout is active
  If nTilemode = 0 Then
      '' Check to see if Model space is active in a viewport,
```

```
        '' CVPORT is 2 if Model space is active
        If nCvports = 2 Then
            acDoc.Editor.SwitchToPaperSpace()
        Else
            acDoc.Editor.SwitchToModelSpace()
        End If
    Else
        '' Switch to the previous Paper space layout
        Application.SetSystemVariable("TILEMODE", 0)
    End If
End Sub
```

C#

```csharp
using Autodesk.AutoCAD.Runtime;
using Autodesk.AutoCAD.ApplicationServices;
using Autodesk.AutoCAD.DatabaseServices;

[CommandMethod("ToggleSpace")]
public static void ToggleSpace()
{
  // Get the current document
  Document acDoc = Application.DocumentManager.MdiActiveDocument;

  // Get the current values of CVPORT and TILEMODE
  object oCvports = Application.GetSystemVariable("CVPORT");
  object oTilemode = Application.GetSystemVariable("TILEMODE");

  // Check to see if the Model layout is active, TILEMODE is 1 when
  // the Model layout is active
  if (System.Convert.ToInt16(oTilemode) == 0)
  {
      // Check to see if Model space is active in a viewport,
      // CVPORT is 2 if Model space is active
      if (System.Convert.ToInt16(oCvports) == 2)
      {
          acDoc.Editor.SwitchToPaperSpace();
      }
      else
      {
          acDoc.Editor.SwitchToModelSpace();
      }
  }
  else
  {
      // Switch to the previous Paper space layout
      Application.SetSystemVariable("TILEMODE", 0);
  }
}
```

⊟ *VBA/ActiveX Code Reference*

```vba
Public Sub ToggleSpace()
    ' Check to see if the Model layout is active
    If ThisDrawing.ActiveLayout.Name <> "Model" Then
        ' Check to see if Model space is active
        If ThisDrawing.MSpace = True Then
            ThisDrawing.MSpace = False
        Else
            ThisDrawing.MSpace = True
        End If
    Else
```

```
        ' Switch to the previous Paper space layout
        ThisDrawing.ActiveSpace = acPaperSpace
    End If
End Sub
```

Create Paper Space Viewports

Paper space viewports are created by creating instances of Viewport objects and adding them to the block reference used by a layout other than Model. The constructor for the Viewport object does not accept any parameters to create a new viewport object. Once an instance of a Viewport object iws created, you can define its placement with the CenterPoint, Width and Height properties.

After creating the Viewport object, you can set properties of the view itself, such as viewing direction (ViewDirection property), lens length for perspective views (LensLength property), and grid display (GridOn property). You can also control properties of the viewport itself, such as layer (Layer property), linetype (Linetype property), and linetype scaling (LinetypeScale property).

Create and enable a floating viewport

This example sets paper space active, creates a floating viewport, defines the view for the viewport, and enables the viewport.

VB.NET

```
Imports System.Runtime.InteropServices

Imports Autodesk.AutoCAD.Runtime
Imports Autodesk.AutoCAD.ApplicationServices
Imports Autodesk.AutoCAD.DatabaseServices
Imports Autodesk.AutoCAD.Geometry

<DllImport("acad.exe", CallingConvention:=CallingConvention.Cdecl, _
 EntryPoint:="?acedSetCurrentVPort@@YA?AW4ErrorStatus@Acad@@PBVAcDbViewport@@@Z")> _
Public Shared Function acedSetCurrentVPort(ByVal AcDbVport As IntPtr) As IntPtr
End Function

<CommandMethod("CreateFloatingViewport")> _
Public Sub CreateFloatingViewport()
  '' Get the current document and database, and start a transaction
  Dim acDoc As Document = Application.DocumentManager.MdiActiveDocument
  Dim acCurDb As Database = acDoc.Database

  Using acTrans As Transaction = acCurDb.TransactionManager.StartTransaction()
      '' Open the Block table for read
      Dim acBlkTbl As BlockTable
      acBlkTbl = acTrans.GetObject(acCurDb.BlockTableId, _
                                   OpenMode.ForRead)

      '' Open the Block table record Paper space for write
      Dim acBlkTblRec As BlockTableRecord
      acBlkTblRec = acTrans.GetObject(acBlkTbl(BlockTableRecord.PaperSpace), _
                                      OpenMode.ForWrite)
```

```
        '' Switch to the previous Paper space layout
        Application.SetSystemVariable("TILEMODE", 0)
        acDoc.Editor.SwitchToPaperSpace()

        '' Create a Viewport
        Dim acVport As Viewport = New Viewport()
        acVport.SetDatabaseDefaults()
        acVport.CenterPoint = New Point3d(3.25, 3, 0)
        acVport.Width = 6
        acVport.Height = 5

        '' Add the new object to the block table record and the transaction
        acBlkTblRec.AppendEntity(acVport)
        acTrans.AddNewlyCreatedDBObject(acVport, True)

        '' Change the view direction
        acVport.ViewDirection = New Vector3d(1, 1, 1)

        '' Enable the viewport
        acVport.On = True

        '' Activate model space in the viewport
        acDoc.Editor.SwitchToModelSpace()

        '' Set the new viewport current via an imported ObjectARX function
        acedSetCurrentVPort(acVport.UnmanagedObject)

        '' Save the new objects to the database
        acTrans.Commit()
    End Using
End Sub
```

C#

```csharp
using System.Runtime.InteropServices;

using Autodesk.AutoCAD.Runtime;
using Autodesk.AutoCAD.ApplicationServices;
using Autodesk.AutoCAD.DatabaseServices;
using Autodesk.AutoCAD.Geometry;

[DllImport("acad.exe", CallingConvention = CallingConvention.Cdecl,
 EntryPoint =
"?acedSetCurrentVPort@@YA?AW4ErrorStatus@Acad@@PBVAcDbViewport@@@Z")]
extern static private int acedSetCurrentVPort(IntPtr AcDbVport);

[CommandMethod("CreateFloatingViewport")]
public static void CreateFloatingViewport()
{
  // Get the current document and database, and start a transaction
  Document acDoc = Application.DocumentManager.MdiActiveDocument;
  Database acCurDb = acDoc.Database;

  using (Transaction acTrans = acCurDb.TransactionManager.StartTransaction())
  {
      // Open the Block table for read
      BlockTable acBlkTbl;
      acBlkTbl = acTrans.GetObject(acCurDb.BlockTableId,
                                  OpenMode.ForRead) as BlockTable;

      // Open the Block table record Paper space for write
      BlockTableRecord acBlkTblRec;
      acBlkTblRec = acTrans.GetObject(acBlkTbl[BlockTableRecord.PaperSpace],
```

```
                                               OpenMode.ForWrite) as BlockTableRecord;

        // Switch to the previous Paper space layout
        Application.SetSystemVariable("TILEMODE", 0);
        acDoc.Editor.SwitchToPaperSpace();

        // Create a Viewport
        Viewport acVport = new Viewport();
        acVport.SetDatabaseDefaults();
        acVport.CenterPoint = new Point3d(3.25, 3, 0);
        acVport.Width = 6;
        acVport.Height = 5;

        // Add the new object to the block table record and the transaction
        acBlkTblRec.AppendEntity(acVport);
        acTrans.AddNewlyCreatedDBObject(acVport, true);

        // Change the view direction
        acVport.ViewDirection = new Vector3d(1, 1, 1);

        // Enable the viewport
        acVport.On = true;

        // Activate model space in the viewport
        acDoc.Editor.SwitchToModelSpace();

        // Set the new viewport current via an imported ObjectARX function
        acedSetCurrentVPort(acVport.UnmanagedObject);

        // Save the new objects to the database
        acTrans.Commit();
    }
}
```

☐ *VBA/ActiveX Code Reference*

```
Sub CreateFloatingViewport()
    ' Set the active space to paper space
    ThisDrawing.ActiveSpace = acPaperSpace

    ' Create the paperspace viewport
    Dim newVport As AcadPViewport
    Dim center(0 To 2) As Double
    center(0) = 3.25
    center(1) = 3
    center(2) = 0
    Set newVport = ThisDrawing.PaperSpace. _
                                AddPViewport(center, 6, 5)

    ' Change the view direction for the viewport
    Dim viewDir(0 To 2) As Double
    viewDir(0) = 1
    viewDir(1) = 1
    viewDir(2) = 1
    newVport.direction = viewDir

    ' Enable the viewport
    newVport.Display True

    ' Switch to model space
    ThisDrawing.MSpace = True

    ' Set newVport current
```

```
        ' (not always necessary but a good idea)
      ThisDrawing.ActivePViewport = newVport
End Sub
```

Note To set or modify aspects of the view (view direction, lens length, and so forth), the
Viewport object's On property must be set to FALSE, and before you can set a viewport
current the On property must be set to TRUE.

Create four floating viewports

This example takes the example from "Create and enable a floating viewport" and continues
it by creating four floating viewports and setting the view of each to top, front, right, and
isometric views, respectively. Each viewport is set to a scale of 1:2. To ensure there is
something to see in these viewports, you may want to create a 3D solid sphere in Model
space before trying this example.

VB.NET

```
Imports System.Runtime.InteropServices

Imports Autodesk.AutoCAD.Runtime
Imports Autodesk.AutoCAD.ApplicationServices
Imports Autodesk.AutoCAD.DatabaseServices
Imports Autodesk.AutoCAD.Geometry

<DllImport("acad.exe", CallingConvention:=CallingConvention.Cdecl, _
 EntryPoint:="?acedSetCurrentVPort@@YA?AW4ErrorStatus@Acad@@PBVAcDbViewport@@@Z")>
_
Public Shared Function acedSetCurrentVPort(ByVal AcDbVport As IntPtr) As IntPtr
End Function

<CommandMethod("FourFloatingViewports")> _
Public Sub FourFloatingViewports()
  '' Get the current document and database, and start a transaction
  Dim acDoc As Document = Application.DocumentManager.MdiActiveDocument
  Dim acCurDb As Database = acDoc.Database

  Using acTrans As Transaction = acCurDb.TransactionManager.StartTransaction()
      '' Open the Block table for read
      Dim acBlkTbl As BlockTable
      acBlkTbl = acTrans.GetObject(acCurDb.BlockTableId, _
                                   OpenMode.ForRead)

      '' Open the Block table record Paper space for write
      Dim acBlkTblRec As BlockTableRecord
      acBlkTblRec = acTrans.GetObject(acBlkTbl(BlockTableRecord.PaperSpace), _
                                      OpenMode.ForWrite)

      '' Switch to the previous Paper space layout
      Application.SetSystemVariable("TILEMODE", 0)
      acDoc.Editor.SwitchToPaperSpace()

      Dim acPt3dCol As Point3dCollection = New Point3dCollection()
      acPt3dCol.Add(New Point3d(2.5, 5.5, 0))
      acPt3dCol.Add(New Point3d(2.5, 2.5, 0))
      acPt3dCol.Add(New Point3d(5.5, 5.5, 0))
      acPt3dCol.Add(New Point3d(5.5, 2.5, 0))

      Dim acVec3dCol As Vector3dCollection = New Vector3dCollection()
      acVec3dCol.Add(New Vector3d(0, 0, 1))
      acVec3dCol.Add(New Vector3d(0, 1, 0))
      acVec3dCol.Add(New Vector3d(1, 0, 0))
```

```vb
        acVec3dCol.Add(New Vector3d(1, 1, 1))

        Dim dWidth As Double = 2.5
        Dim dHeight As Double = 2.5

        Dim acVportLast As Viewport = Nothing
        Dim nCnt As Integer = 0

        For Each acPt3d As Point3d In acPt3dCol
            Dim acVport As Viewport = New Viewport()
            acVport.SetDatabaseDefaults();
            acVport.CenterPoint = acPt3d
            acVport.Width = dWidth
            acVport.Height = dHeight

            '' Add the new object to the block table record and the transaction
            acBlkTblRec.AppendEntity(acVport)
            acTrans.AddNewlyCreatedDBObject(acVport, True)

            '' Change the view direction
            acVport.ViewDirection = acVec3dCol(nCnt)

            '' Enable the viewport
            acVport.On = True

            '' Record the last viewport created
            acVportLast = acVport

            '' Increment the counter by 1
            nCnt = nCnt + 1
        Next

        If acVportLast <> Nothing Then
            '' Activate model space in the viewport
            acDoc.Editor.SwitchToModelSpace()

            '' Set the new viewport current via an imported ObjectARX function
            acedSetCurrentVPort(acVportLast.UnmanagedObject)
        End If

        '' Save the new objects to the database
        acTrans.Commit()
    End Using
End Sub
```

C#

```csharp
using System.Runtime.InteropServices;

using Autodesk.AutoCAD.Runtime;
using Autodesk.AutoCAD.ApplicationServices;
using Autodesk.AutoCAD.DatabaseServices;
using Autodesk.AutoCAD.Geometry;

[DllImport("acad.exe", CallingConvention = CallingConvention.Cdecl,
 EntryPoint =
"?acedSetCurrentVPort@@YA?AW4ErrorStatus@Acad@@PBVAcDbViewport@@@Z")]
extern static private int acedSetCurrentVPort(IntPtr AcDbVport);

[CommandMethod("FourFloatingViewports")]
public static void FourFloatingViewports()
{
  // Get the current document and database, and start a transaction
  Document acDoc = Application.DocumentManager.MdiActiveDocument;
```

```csharp
Database acCurDb = acDoc.Database;

using (Transaction acTrans = acCurDb.TransactionManager.StartTransaction())
{
    // Open the Block table for read
    BlockTable acBlkTbl;
    acBlkTbl = acTrans.GetObject(acCurDb.BlockTableId,
                                 OpenMode.ForRead) as BlockTable;

    // Open the Block table record Paper space for write
    BlockTableRecord acBlkTblRec;
    acBlkTblRec = acTrans.GetObject(acBlkTbl[BlockTableRecord.PaperSpace],
                                    OpenMode.ForWrite) as BlockTableRecord;

    // Switch to the previous Paper space layout
    Application.SetSystemVariable("TILEMODE", 0);
    acDoc.Editor.SwitchToPaperSpace();

    Point3dCollection acPt3dCol = new Point3dCollection();
    acPt3dCol.Add(new Point3d(2.5, 5.5, 0));
    acPt3dCol.Add(new Point3d(2.5, 2.5, 0));
    acPt3dCol.Add(new Point3d(5.5, 5.5, 0));
    acPt3dCol.Add(new Point3d(5.5, 2.5, 0));

    Vector3dCollection acVec3dCol = new Vector3dCollection();
    acVec3dCol.Add(new Vector3d(0, 0, 1));
    acVec3dCol.Add(new Vector3d(0, 1, 0));
    acVec3dCol.Add(new Vector3d(1, 0, 0));
    acVec3dCol.Add(new Vector3d(1, 1, 1));

    double dWidth = 2.5;
    double dHeight = 2.5;

    Viewport acVportLast = null;
    int nCnt = 0;

    foreach (Point3d acPt3d in acPt3dCol)
    {
        Viewport acVport = new Viewport();
        acVport.SetDatabaseDefaults();
        acVport.CenterPoint = acPt3d;
        acVport.Width = dWidth;
        acVport.Height = dHeight;

        // Add the new object to the block table record and the transaction
        acBlkTblRec.AppendEntity(acVport);
        acTrans.AddNewlyCreatedDBObject(acVport, true);

        // Change the view direction
        acVport.ViewDirection = acVec3dCol[nCnt];

        // Enable the viewport
        acVport.On = true;

        // Record the last viewport created
        acVportLast = acVport;

        // Increment the counter by 1
        nCnt = nCnt + 1;
    }

    if (acVportLast != null)
    {
        // Activate model space in the viewport
        acDoc.Editor.SwitchToModelSpace();
```

```
        // Set the new viewport current via an imported ObjectARX function
        acedSetCurrentVPort(acVportLast.UnmanagedObject);
    }

    // Save the new objects to the database
    acTrans.Commit();
  }
}
```

⊟ *VBA/ActiveX Code Reference*

```
Sub FourFloatingViewports()
    Dim topVport, frontVport As AcadPViewport
    Dim rightVport, isoVport As AcadPViewport
    Dim pt(0 To 2) As Double
    Dim viewDir(0 To 2) As Double
    ThisDrawing.ActiveSpace = acPaperSpace
    ThisDrawing.MSpace = True

    ' Take the existing PViewport and make it the topVport
    pt(0) = 2.5: pt(1) = 5.5: pt(2) = 0
    Set topVport = ThisDrawing.ActivePViewport
    ' No need to set Direction for top view
    topVport.center = pt
    topVport.width = 2.5
    topVport.height = 2.5
    topVport.Display True
    topVport.StandardScale = acVp1_2

    ' Create and setup frontVport
    pt(0) = 2.5: pt(1) = 2.5: pt(2) = 0
    Set frontVport = ThisDrawing.PaperSpace. _
                                    AddPViewport(pt, 2.5, 2.5)
    viewDir(0) = 0: viewDir(1) = 1: viewDir(2) = 0
    frontVport.direction = viewDir
    frontVport.Display acOn
    frontVport.StandardScale = acVp1_2

    ' Create and setup rightVport
    pt(0) = 5.5: pt(1) = 5.5: pt(2) = 0
    Set rightVport = ThisDrawing.PaperSpace. _
                                    AddPViewport(pt, 2.5, 2.5)
    viewDir(0) = 1: viewDir(1) = 0: viewDir(2) = 0
    rightVport.direction = viewDir
    rightVport.Display acOn
    rightVport.StandardScale = acVp1_2

    ' Create and set up isoVport
    pt(0) = 5.5: pt(1) = 2.5: pt(2) = 0
    Set isoVport = ThisDrawing.PaperSpace. _
                                    AddPViewport(pt, 2.5, 2.5)
    viewDir(0) = 1: viewDir(1) = 1: viewDir(2) = 1
    isoVport.direction = viewDir
    isoVport.Display acOn
    isoVport.StandardScale = acVp1_2

    ThisDrawing.MSpace = True
    ThisDrawing.ActivePViewport = isoVport

    ' Finish: Perform a regen in all viewports
    ThisDrawing.Regen True
End Sub
```

Change Viewport Views and Content

To change the view within a Viewport object, you manipulate the view of a viewport you use its member properties. The following member properties allow you to adjust the display of the view in a viewport:

- *ViewCenter* - Specifies the view center of the view in the viewport.
- *ViewDirection* - Specifies the vector from the target to the camera of the view in the viewport.
- *ViewHeight* - Specifies the height of the Model Space view within the viewport.
- *ViewTarget* - Specifies the location of the target of the view in the viewport.

Scale Views Relative to Paper Space

Before you plot, you can establish the scale factor for the view in each viewport. Scaling views relative to paper space establishes a consistent scale for each displayed view. For example, the following illustration shows a Paper space view with several viewports—each set to different scales and views.
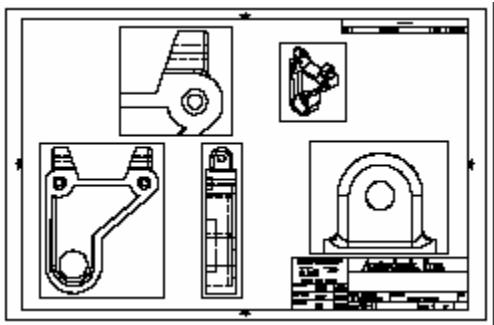
When you work in Paper space, the scale factor represents a ratio between the size of the plotted drawing and the actual size of the model displayed in the viewports. To derive this scale, divide paper space units by Model space units. For a quarter-scale drawing, for example, you specify a scale factor of one Paper space unit to four Model space units (1:4).

The StandardScale and CustomScale properties are used to specify the scale for a viewport. The StandardScale property accepts a value based on the StandardScaleType enum; whereas the CustomScale property accepts a real number which represents the ratio of units between Model and Paper space. For example, the real value equal to 1:4 is 0.25.

The following illustrations show the view of the model at a scale of 1:1, and then the same model view show at scales of 2:1 and 1:2. A scale of 2:1 increases the view of the model to twice the size of the paper space units; while a scale of 1:2 displays the model at half its actual size.
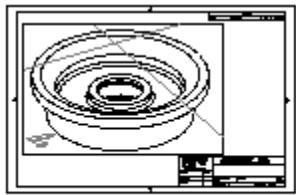
current view
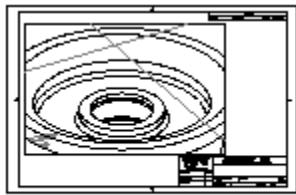
zoom to 2xp
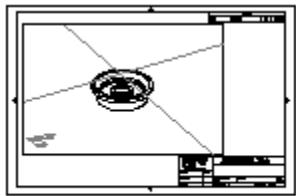
zoom to 0.5xp

Scale Pattern Linetypes in Paper Space

In Paper space, you can scale any type of linetype in two ways. The scale can be based on the drawing units of the space in which the object was created (Model or Paper). The linetype scale also can be a uniform scale based on Paper space units. You can use the PSLTSCALE system variable to maintain the same linetype scaling for objects displayed at different scales in different viewports. It also affects the line display in 3D views.

Tthe following illustration shows the pattern linetype scaled uniformly in Paper space. Notice that the linetype in the two viewports have the same scale, even though each of the viewports have different scales.
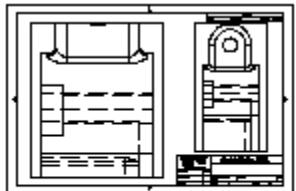
psltscale=1, dashes scaled to paper space
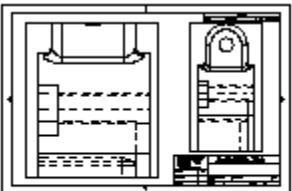
psltscale=0, dashes scaled to space where they were created

Use the `SetSystemVariable` method to set the value of the PSLTSCALE system variable.

Use Shaded Viewports

If your drawing contains 3D faces, meshes, extruded objects, surfaces, or solids, you can plot from Paper space using one of the values defined by the ShadePlotStype enum. Shaded and rendered viewports are previewed, plotted, and plotted to file with full shading and rendering.

The ShadePlot property of the Viewport object is used to set the option for plotting a shaded viewport.

NoteThe hidden lines for the objects in a Viewport object can be enabled with the `HiddenLinesRemoved` property. This property takes a boolean value: TRUE to remove hidden lines or FALSE to draw them.

Plot Your Drawing

You can plot your drawing as it is viewed in Model space, or you can plot one of your prepared Paper space layouts. Plotting from Model space is often preferable when you want to view or verify your drawing prior to creating a Paper space layout. Once your model is ready, you can prepare and plot a Paper space layout.

NoteThe BACKGROUNDPLOT system variable controls if plotting occurs in the background or the foreground. Set BACKGROUNDPLOT to 0 to plot in the foreground.

Plotting involves working with a number of different objects: PlotFactory, PlotEngine, PlotInfo, PlotSettings, PlotSettingsValidator, PlotInfoValidator, and PlotPageInfo. The PlotEngine object is responsible for generating a plot based on the information provided to it by a PlotInfo object.

The PlotEngine object is used to generate the output of a layout. From the PlotEngine object, you can do the following:

- Plot to a file
- Plot to a plotter or printer
- Displays a preview of a layout

Topics in this section

- [Plot from Model Space](#)
- [Plot from Paper Space](#)

Plot from Model Space

Typically, when you plot a large drawing such as a floorplan, you specify a scale to convert the real drawing units into plotted inches or millimeters. However, when you plot from Model space, the defaults that are used if there are no settings specified include plot to system printer, plot the current display, scaled to fit, 0 rotation, and 0,0 offset.

You use a PlotSettings object to modify the plot settings of the layout to plot, and then validate the plot settings with a PlotSettingsValidatorobject before passing the PlotSettings object to a PlotInfo object. Once the Plot Info object is defined, you use a PlotEngine object to output create the plot and pass to it the PlotInfo object that contains the information of the sheet or layout to plot.

Plot the extents of the Model layout

This example establishes several plot settings before generating the final plot. The output generated is a DWF file named Myplot. Before running this example, you might want to add some objects to Model space first.

VB.NET

```
Imports Autodesk.AutoCAD.Runtime
Imports Autodesk.AutoCAD.ApplicationServices
Imports Autodesk.AutoCAD.DatabaseServices
Imports Autodesk.AutoCAD.PlottingServices

<CommandMethod("PlotCurrentLayout")> _
Public Sub PlotCurrentLayout()
  '' Get the current document and database, and start a transaction
  Dim acDoc As Document = Application.DocumentManager.MdiActiveDocument
  Dim acCurDb As Database = acDoc.Database

  Using acTrans As Transaction = acCurDb.TransactionManager.StartTransaction()
      '' Reference the Layout Manager
      Dim acLayoutMgr As LayoutManager
      acLayoutMgr = LayoutManager.Current

      '' Get the current layout and output its name in the Command Line window
      Dim acLayout As Layout
      acLayout =
acTrans.GetObject(acLayoutMgr.GetLayoutId(acLayoutMgr.CurrentLayout), _
                                    OpenMode.ForRead)

      '' Get the PlotInfo from the layout
      Dim acPlInfo As PlotInfo = New PlotInfo()
      acPlInfo.Layout = acLayout.ObjectId

      '' Get a copy of the PlotSettings from the layout
      Dim acPlSet As PlotSettings = New PlotSettings(acLayout.ModelType)
      acPlSet.CopyFrom(acLayout)

      '' Update the PlotSettings object
      Dim acPlSetVdr As PlotSettingsValidator = PlotSettingsValidator.Current

      '' Set the plot type
      acPlSetVdr.SetPlotType(acPlSet, _
                             Autodesk.AutoCAD.DatabaseServices.PlotType.Extents)

      '' Set the plot scale
      acPlSetVdr.SetUseStandardScale(acPlSet, True)
      acPlSetVdr.SetStdScaleType(acPlSet, StdScaleType.ScaleToFit)

      '' Center the plot
      acPlSetVdr.SetPlotCentered(acPlSet, True)

      '' Set the plot device to use
      acPlSetVdr.SetPlotConfigurationName(acPlSet, "DWF6 ePlot.pc3", _
                                          "ANSI_A_(8.50_x_11.00_Inches)")

      '' Set the plot info as an override since it will
      '' not be saved back to the layout
      acPlInfo.OverrideSettings = acPlSet

      '' Validate the plot info
      Dim acPlInfoVdr As PlotInfoValidator = New PlotInfoValidator()
      acPlInfoVdr.MediaMatchingPolicy = MatchingPolicy.MatchEnabled
```

```vb
        acPlInfoVdr.Validate(acPlInfo)

        '' Check to see if a plot is already in progress
        If PlotFactory.ProcessPlotState = Autodesk.AutoCAD.PlottingServices. _
                                    ProcessPlotState.NotPlotting Then
          Using acPlEng As PlotEngine = PlotFactory.CreatePublishEngine()

            '' Track the plot progress with a Progress dialog
            Dim acPlProgDlg As PlotProgressDialog = New
PlotProgressDialog(False, _
                                                    1, _
                                                    True)

            Using (acPlProgDlg)
              '' Define the status messages to display when plotting starts
              acPlProgDlg.PlotMsgString(PlotMessageIndex.DialogTitle) = _
                                              "Plot Progress"
              acPlProgDlg.PlotMsgString(PlotMessageIndex.CancelJobButtonMessag
e) = _
                                              "Cancel Job"

              acPlProgDlg.PlotMsgString(PlotMessageIndex.CancelSheetButtonMessa
ge) = _
                                              "Cancel Sheet"

              acPlProgDlg.PlotMsgString(PlotMessageIndex.SheetSetProgressCaptio
n) = _
                                              "Sheet Set Progress"
              acPlProgDlg.PlotMsgString(PlotMessageIndex.SheetProgressCaption)
= _
                                              "Sheet Progress"

              '' Set the plot progress range
              acPlProgDlg.LowerPlotProgressRange = 0
              acPlProgDlg.UpperPlotProgressRange = 100
              acPlProgDlg.PlotProgressPos = 0

              '' Display the Progress dialog
              acPlProgDlg.OnBeginPlot()
              acPlProgDlg.IsVisible = True

              '' Start to plot the layout
              acPlEng.BeginPlot(acPlProgDlg, Nothing)

              '' Define the plot output
              acPlEng.BeginDocument(acPlInfo, _
                            acDoc.Name, _
                            Nothing, _
                            1, _
                            True, _
                            "c:\myplot")

              '' Display information about the current plot
              acPlProgDlg.PlotMsgString(PlotMessageIndex.Status) = _
                                "Plotting: " & acDoc.Name & _
                                " - " & acLayout.LayoutName

              '' Set the sheet progress range
              acPlProgDlg.OnBeginSheet()
              acPlProgDlg.LowerSheetProgressRange = 0
              acPlProgDlg.UpperSheetProgressRange = 100
              acPlProgDlg.SheetProgressPos = 0

              '' Plot the first sheet/layout
              Dim acPlPageInfo As PlotPageInfo = New PlotPageInfo() _
```

```
                    acPlEng.BeginPage(acPlPageInfo, _
                                      acPlInfo, _
                                      True, _
                                      Nothing)

                acPlEng.BeginGenerateGraphics(Nothing)
                acPlEng.EndGenerateGraphics(Nothing)

                '' Finish plotting the sheet/layout
                acPlEng.EndPage(Nothing)
                acPlProgDlg.SheetProgressPos = 100
                acPlProgDlg.OnEndSheet()

                '' Finish plotting the document
                acPlEng.EndDocument(Nothing)

                '' Finish the plot
                acPlProgDlg.PlotProgressPos = 100
                acPlProgDlg.OnEndPlot()
                acPlEng.EndPlot(Nothing)
            End Using
        End Using
    End If
  End Using
End Sub
```

C#

```csharp
using Autodesk.AutoCAD.Runtime;
using Autodesk.AutoCAD.ApplicationServices;
using Autodesk.AutoCAD.DatabaseServices;
using Autodesk.AutoCAD.PlottingServices;

[CommandMethod("PlotCurrentLayout")]
public static void PlotCurrentLayout()
{
  // Get the current document and database, and start a transaction
  Document acDoc = Application.DocumentManager.MdiActiveDocument;
  Database acCurDb = acDoc.Database;

  using (Transaction acTrans = acCurDb.TransactionManager.StartTransaction())
  {
      // Reference the Layout Manager
      LayoutManager acLayoutMgr;
      acLayoutMgr = LayoutManager.Current;

      // Get the current layout and output its name in the Command Line window
      Layout acLayout;
      acLayout =
acTrans.GetObject(acLayoutMgr.GetLayoutId(acLayoutMgr.CurrentLayout),
                            OpenMode.ForRead) as Layout;

      // Get the PlotInfo from the layout
      PlotInfo acPlInfo = new PlotInfo();
      acPlInfo.Layout = acLayout.ObjectId;

      // Get a copy of the PlotSettings from the layout
      PlotSettings acPlSet = new PlotSettings(acLayout.ModelType);
      acPlSet.CopyFrom(acLayout);

      // Update the PlotSettings object
      PlotSettingsValidator acPlSetVdr = PlotSettingsValidator.Current;

      // Set the plot type
```

```
        acPlSetVdr.SetPlotType(acPlSet,
                          Autodesk.AutoCAD.DatabaseServices.PlotType.Extents);

    // Set the plot scale
    acPlSetVdr.SetUseStandardScale(acPlSet, true);
    acPlSetVdr.SetStdScaleType(acPlSet, StdScaleType.ScaleToFit);

    // Center the plot
    acPlSetVdr.SetPlotCentered(acPlSet, true);

    // Set the plot device to use
    acPlSetVdr.SetPlotConfigurationName(acPlSet, "DWF6 ePlot.pc3",
                                    "ANSI_A_(8.50_x_11.00_Inches)");

    // Set the plot info as an override since it will
    // not be saved back to the layout
    acPlInfo.OverrideSettings = acPlSet;

    // Validate the plot info
    PlotInfoValidator acPlInfoVdr = new PlotInfoValidator();
    acPlInfoVdr.MediaMatchingPolicy = MatchingPolicy.MatchEnabled;
    acPlInfoVdr.Validate(acPlInfo);

    // Check to see if a plot is already in progress
    if (PlotFactory.ProcessPlotState == ProcessPlotState.NotPlotting)
    {
        using (PlotEngine acPlEng = PlotFactory.CreatePublishEngine())
        {
            // Track the plot progress with a Progress dialog
            PlotProgressDialog acPlProgDlg = new PlotProgressDialog(false,
                                                          1,
                                                          true);

            using (acPlProgDlg)
            {
                // Define the status messages to display when plotting starts
                acPlProgDlg.set_PlotMsgString(PlotMessageIndex.DialogTitle,
                                        "Plot Progress");

                acPlProgDlg.set_PlotMsgString(PlotMessageIndex.CancelJobButtonMe
ssage,
                                        "Cancel Job");

                acPlProgDlg.set_PlotMsgString(PlotMessageIndex.CancelSheetButton
Message,
                                        "Cancel Sheet");

                acPlProgDlg.set_PlotMsgString(PlotMessageIndex.SheetSetProgressC
aption,
                                        "Sheet Set Progress");

                acPlProgDlg.set_PlotMsgString(PlotMessageIndex.SheetProgressCapt
ion,
                                        "Sheet Progress");

                // Set the plot progress range
                acPlProgDlg.LowerPlotProgressRange = 0;
                acPlProgDlg.UpperPlotProgressRange = 100;
                acPlProgDlg.PlotProgressPos = 0;

                // Display the Progress dialog
                acPlProgDlg.OnBeginPlot();
                acPlProgDlg.IsVisible = true;

                // Start to plot the layout
```

```
                    acPlEng.BeginPlot(acPlProgDlg, null);

                    // Define the plot output
                    acPlEng.BeginDocument(acPlInfo,
                                          acDoc.Name,
                                          null,
                                          1,
                                          true,
                                          "c:\\myplot");

                    // Display information about the current plot
                    acPlProgDlg.set_PlotMsgString(PlotMessageIndex.Status,
                                                  "Plotting: " + acDoc.Name + " - "
+
                                                  acLayout.LayoutName);

                    // Set the sheet progress range
                    acPlProgDlg.OnBeginSheet();
                    acPlProgDlg.LowerSheetProgressRange = 0;
                    acPlProgDlg.UpperSheetProgressRange = 100;
                    acPlProgDlg.SheetProgressPos = 0;

                    // Plot the first sheet/layout
                    PlotPageInfo acPlPageInfo = new PlotPageInfo();
                    acPlEng.BeginPage(acPlPageInfo,
                                      acPlInfo,
                                      true,
                                      null);

                    acPlEng.BeginGenerateGraphics(null);
                    acPlEng.EndGenerateGraphics(null);

                    // Finish plotting the sheet/layout
                    acPlEng.EndPage(null);
                    acPlProgDlg.SheetProgressPos = 100;
                    acPlProgDlg.OnEndSheet();

                    // Finish plotting the document
                    acPlEng.EndDocument(null);

                    // Finish the plot
                    acPlProgDlg.PlotProgressPos = 100;
                    acPlProgDlg.OnEndPlot();
                    acPlEng.EndPlot(null);
                }
            }
        }
    }
}
```

⊟ *VBA/ActiveX Code Reference*

```
Sub PlotModelSpace()
    ' Verify that the active space is model space
    If ThisDrawing.ActiveSpace = acPaperSpace Then
        ThisDrawing.MSpace = True
        ThisDrawing.ActiveSpace = acModelSpace
    End If

    ' Set the extents and scale of the plot area
    ThisDrawing.ModelSpace.Layout.PlotType = acExtents
    ThisDrawing.ModelSpace.Layout. _
                    StandardScale = acScaleToFit
```

```
    ' Center the plot
    ThisDrawing.ModelSpace.Layout.CenterPlot = True

    ' Set the plot device to use
    ThisDrawing.ModelSpace.Layout. _
                        ConfigName = "DWF6 ePlot.pc3"

    ' Set the plot output size
    ThisDrawing.ModelSpace.Layout. _
                        CanonicalMediaName = "ANSI_A_(8.50_x_11.00_Inches)"

    ' Set the number of copies to one
    ThisDrawing.Plot.NumberOfCopies = 1

    ' Initiate the plot
    ThisDrawing.Plot.PlotToFile "c:\myplot"
End Sub
```

The device name can be specified using the ConfigName property. This device can be overridden in the PlotToDevice method by specifying a PC3 file.

Plot from Paper Space

You can plot a Paper space layout. Plotting a Paper space layout is the same as plotting the Model layout. Before plotting a Paper space layout, be sure that the layout is initialized and the viewports are defined with the settings and desired views of the model.For information on defining viewports on a Paper space layout, see Create Paper Space Viewports. To plot a Paper space layout, refer to Plot from Model Space.

9   Use Events

Events are notifications, or messages, that are sent out by AutoCAD® to inform you about the current state of the session, or alert you that something has happened. For example, when a drawing is saved the BeginSave event is triggered. There are other events triggered when a drawing is closed, a command is started or even when a system variable is changed. Given this information you could write a subroutine, or event handler, that uses these events to track changes to a drawing or the amount of time a user spends working on a particular drawing.

Topics in this section

- [Understand the Events in AutoCAD](#)
- [Guidelines for Event Handlers](#)
- [Register and Unregister Events](#)
- [Handle Application Events](#)
- [Handle Document Events](#)
- [Handle DocumentCollection Events](#)
- [Handle Object Events](#)
- [Register COM Based Events with .NET](#)

Understand the Events in AutoCAD

There are mainy different types of events in AutoCAD. The following is some of the common types of events:

- *Application* - Events respond to the AutoCAD closing, changes to system variables, begin double clicking, and entering and leaving modal states.
- *Database* - Events respond to the saving drawings, addition, deletion, or modification of objects, insertion of block references, attachment and changes to external drawings (xrefs). There are also document level events for system variable changes.
- *Document* - Events respond to the closing drawings, issuing of AutoCAD commands, issuing AutoLISP command or statements, and changes to system variables.
- *DocumentCollection* - Events respond to the creation and destruction of a document, document is actived or deactived, and lock mode changes to a document.
- *Editor* - Events respond to the changes during requests for user input.
- *Graphics* - Events respond to the creation and destruction of views, and view configuration changes.
- *Plotting* - Events respond to plotting a layout.
- *Publishing* - Events respond to publishing a layout.
- *Runtime* - Events respond to loading and unloading modules, and variables changed or are changing.
- *Windows* - Events respond to changes to the status bar, tray items, palettes and InfoCenter.

Subroutines that respond to events are called *event handlers* and are executed automatically each and every time their designated event is triggered. Information contained in the arguments returned by an event, such as a system variable name in the

SystemVariableChanging event, are passed from the event handler to the SystemVariableChangingEventArgs object.

Guidelines for Event Handlers

It is important to remember that events simply provide information on the state or activities taking place within AutoCAD. Although event handlers can be written to respond to those events, AutoCAD might be in the middle of an operation when the event handler is triggered. Event handlers, therefore, have some restrictions on what they can do if they are to provide safe operations in conjunction with AutoCAD and its database.

- Do not rely on the sequence of events.

  When writing event handlers, do not rely on the sequence of events to happen in the exact order you think they occur. For example, if you issue an OPEN command, the events CommandWillStart, DocumentCreateStarted, DocumentCreated, and CommandEnded will all be triggered. However, they may not occur in that exact order each and everytime. The only thing you can rely on is that a most events occur in pairs, a beginning and ending event.

- Do not rely on the sequence of operations.

  If you delete object1 and then object2, do not rely on the fact that you will receive the ObjectErased event for object1 and then for object2. You may receive the ObjectErased event for object2 first.

- Do not attempt any interactive functions from an event handler.

  Attempting to execute interactive functions from within an event handler can cause serious problems, as AutoCAD may still be processing a command at the time the event is triggered. Therefore, you should always avoid requesting for input at the Command prompt, as well as object selection requests, and using the SendStringToExecute method from within event handlers.

- Do not launch a dialog box from within an event handler.

  Dialog boxes are considered interactive functions and can interfere with the current operation of AutoCAD. Message boxes and alert boxes are not considered interactive and can be issued safely; however issuing a message box within some event handlers such as EnterModal, LeaveModal, DocumentActivated, and DocumentToBeDeactivated can result in unexpected sequencing.

- You can write data to any object in the database, but modifying the object that issued the event should be avoided.

  Obviously, any object causing an event to be triggered could still be open and the operation currently in progress. Therefore, avoid modifying an object from an event

handler for the same object. However, you can safely read information from the object triggering an event.

- Do not perform any action from an event handler that might trigger the same event.

  If you perform the same action in an event handler that triggers that same event, you will create an infinite loop. For example, you should never attempt to open an object from within the ObjectOpenedForModify event, or AutoCAD will simply continue to open objects.

- No events are fired while AutoCAD is displaying a modal dialog box.

Register and Unregister Events

Before you can respond to an event, the event must be registered with AutoCAD. You register an event by creating a new event handler of the desired type and then assigning it to the object in which you want to register the event with. Once you are done with an event, it is best to unregister the event to minimize conflicts with other reactors as well as reduce the about of memory and CPU usage that AutoCAD requires to maintain your event handler.

Register an event

You register an event by appending an event handler to an event. The event handler object requires a procedure that you must have defined in your project. Most event handlers require a procedure that accepts two paramters, one of the type Object and another that represents the return arguments of the event. You register an event by using the VB.NET AddHandler statement or the C# += operator.

The following code registers a procedure named appSysVarChanged with an object type of SystemVariableChangedEventHandler to the SystemVariableChanged event. The appSysVarChanged procedure accepts two paramters: Object and SystemVariableChangedEventArgs. The SystemVariableChangedEventArgs object returns the name of the system variabled changed when the event is registered.

VB.NET

```
AddHandler Application.SystemVariableChanged, AddressOf appSysVarChanged
```

C#

```
Application.SystemVariableChanged +=
        new SystemVariableChangedEventHandler(appSysVarChanged)
```

Unregister an event

An event is unregistered by removing an event handler from the event in which it is assigned. You use the same syntax in which was used to register the event handler with an event with the exception you use RemoveHandler or the -= operator.

The following code unregisters a procedure named appSysVarChanged with an object type of SystemVariableChangedEventHandler from the SystemVariableChanged event.

VB.NET

```
RemoveHandler Application.SystemVariableChanged, AddressOf appSysVarChanged
```

C#

```
Application.SystemVariableChanged +=
            new SystemVariableChangedEventHandler(appSysVarChanged)
```

Handle Application Events

Application object events are are used to respond to the application window. Once an Application event is registered, it remains registered until AutoCAD is shutdown or the event is unregistered.

The following events are available for the Application object:

BeginCustomizationMode

> Triggered just before AutoCAD enters customization mode.

BeginDoubleClick

> Triggered when the mouse button is double clicked.

BeginQuit

> Triggered just before an AutoCAD session ends.

DisplayingCustomizeDialog

> Triggered just before the Customize dialog box is displayed.

DisplayingDraftingSettingsDialog

> Triggered just before the Drafting Settings dialog box is displayed.

DisplayingOptionDialog

> Triggered just before the Options dialog box is displayed.

EndCustomizationMode

> Triggered when AutoCAD exits customization mode.

EnterModal

Triggered just before a modal dialog box is displayed.

Idle

Triggered when AutoCAD text.

LeaveModal

Triggered when a modal dialog box is closed.

PreTranslateMessage

Triggered just before a message is translated by AutoCAD.

QuitAborted

Triggered when an attempt to shutdown AutoCAD is aborted.

QuitWillStart

Triggered after the BeginQuit event and before shutdown begins.

SystemVariableChanged

Triggered when an attempt to change a system variable is made.

SystemVariableChanging

Triggered just before an attempt is made at changing a system variable.

Enable an Application object event

This example demonstrates how to register an event handler with the BeginQuit event. Once registered, a message box is displayed before AutoCAD completely shutdown.

VB.NET

```
Imports Autodesk.AutoCAD.Runtime
Imports Autodesk.AutoCAD.ApplicationServices

<CommandMethod("AddAppEvent")> _
Public Sub AddAppEvent()
  AddHandler Application.SystemVariableChanged, AddressOf appSysVarChanged
End Sub

<CommandMethod("RemoveAppEvent")> _
Public Sub RemoveAppEvent()
  RemoveHandler Application.SystemVariableChanged, AddressOf appSysVarChanged
End Sub

Public Sub appSysVarChanged(ByVal senderObj As Object, _
                            ByVal sysVarChEvtArgs As
Autodesk.AutoCAD.ApplicationServices. _
                            SystemVariableChangedEventArgs)

  Dim oVal As Object = Application.GetSystemVariable(sysVarChEvtArgs.Name)
```

```
    '' Display a message box with the system variable name and the new value
    Application.ShowAlertDialog(sysVarChEvtArgs.Name & " was changed." & _
                                vbLf & "New value: " & oVal.ToString())
End Sub
```

C#

```csharp
using Autodesk.AutoCAD.Runtime;
using Autodesk.AutoCAD.ApplicationServices;
[CommandMethod("AddAppEvent")]
public void AddAppEvent()
{
  Application.SystemVariableChanged +=
      new Autodesk.AutoCAD.ApplicationServices.
          SystemVariableChangedEventHandler(appSysVarChanged);
}

[CommandMethod("RemoveAppEvent")]
public void RemoveAppEvent()
{
  Application.SystemVariableChanged -=
      new Autodesk.AutoCAD.ApplicationServices.
          SystemVariableChangedEventHandler(appSysVarChanged);
}

public void appSysVarChanged(object senderObj,
                              Autodesk.AutoCAD.ApplicationServices.
                              SystemVariableChangedEventArgs sysVarChEvtArgs)
{
  object oVal = Application.GetSystemVariable(sysVarChEvtArgs.Name);

  // Display a message box with the system variable name and the new value
  Application.ShowAlertDialog(sysVarChEvtArgs.Name + " was changed." +
                              "\nNew value: " + oVal.ToString());
}
```

⊟ *VBA/ActiveX Code Reference*

```vba
Public WithEvents ACADApp As AcadApplication

Sub Example_AcadApplication_Events()
    ' Intialize the public variable (ACADApp)
    '
    ' Run this procedure first

    Set ACADApp = ThisDrawing.Application
End Sub

Private Sub ACADApp_SysVarChanged(ByVal SysvarName As String, _
                                  ByVal newVal As Variant)
    ' This procedure intercepts an Application SysVarChanged event.

    MsgBox (SysvarName & " was changed." & _
            vbLf & "New value: " & newVal)
End Sub
```

Handle Document Events

Document object events are used to respond to the document window. When a document event is registered, it is only associated with the document object in which it is associated. So if an event needs to be registered with each document, you will want to use the DocumentCreated event of the DocumentCollection object to register events with each new or opened drawing.

The following events are available for Document objects:

BeginDocumentClose

Triggered just after a request is received to close a drawing.

CloseAborted

Triggered when an attempt to close a drawing is aborted.

CloseWillStart

Triggered after the BeginDocumentClose event and before closing the drawing begins.

CommandCancelled

Triggered when a command is cancelled before it completes.

CommandEnded

Triggered immediately after a command completes.

CommandFailed

Triggered when a command fails to complete and is not cancelled.

CommandWillStart

Triggered immediately after a command is issued, but before it completes.

ImpliedSelectionChanged

Triggered when the current pickfirst selection set changes.

LispCancelled

Triggered when the evaluation of a LISP expression is canceled.

LispEnded

Triggered upon completion of evaluating a LISP expression.

LispWillStart

Triggered immediately after AutoCAD receives a request to evaluate a LISP expression.

UnknownCommand

Triggered immediately when an unknown command is entered at the Command prompt.

Enable a Document object event

The following example uses the BeginDocumentClose event to prompt the user if they want to continue closing the current drawing. A message box is displayed with the Yes and No buttons. Clicking No aborts the closing of the drawing by using the Veto method of the arguments that are returned by the event handler.

VB.NET

```
Imports Autodesk.AutoCAD.Runtime
Imports Autodesk.AutoCAD.ApplicationServices

<CommandMethod("AddDocEvent")> _
Public Sub AddDocEvent()
  '' Get the current document
  Dim acDoc As Document = Application.DocumentManager.MdiActiveDocument

  AddHandler acDoc.BeginDocumentClose, AddressOf docBeginDocClose
End Sub

<CommandMethod("RemoveDocEvent")> _
Public Sub RemoveDocEvent()
  '' Get the current document
  Dim acDoc As Document = Application.DocumentManager.MdiActiveDocument

  RemoveHandler acDoc.BeginDocumentClose, AddressOf docBeginDocClose
End Sub

Public Sub docBeginDocClose(ByVal senderObj As Object, _
                            ByVal docBegClsEvtArgs As DocumentBeginCloseEventArgs)

  '' Display a message box prompting to continue closing the document
  If System.Windows.Forms.MessageBox.Show( _
                    "The document is about to be closed." & _
                    vbLf & "Do you want to continue?", _
                    "Close Document", _
                    System.Windows.Forms.MessageBoxButtons.YesNo) = _
                    System.Windows.Forms.DialogResult.No Then
      docBegClsEvtArgs.Veto()
  End If
End If
```

C#

```
using Autodesk.AutoCAD.Runtime;
using Autodesk.AutoCAD.ApplicationServices;

[CommandMethod("AddDocEvent")]
public void AddDocEvent()
{
```

```
  // Get the current document
  Document acDoc = Application.DocumentManager.MdiActiveDocument;

  acDoc.BeginDocumentClose +=
      new DocumentBeginCloseEventHandler(docBeginDocClose);
}

[CommandMethod("RemoveDocEvent")]
public void RemoveDocEvent()
{
  // Get the current document
  Document acDoc = Application.DocumentManager.MdiActiveDocument;

  acDoc.BeginDocumentClose -=
      new DocumentBeginCloseEventHandler(docBeginDocClose);
}

public void docBeginDocClose(object senderObj,
                             DocumentBeginCloseEventArgs docBegClsEvtArgs)
{
  // Display a message box prompting to continue closing the document
  if (System.Windows.Forms.MessageBox.Show(
                      "The document is about to be closed." +
                      "\nDo you want to continue?",
                      "Close Document",
                      System.Windows.Forms.MessageBoxButtons.YesNo) ==
                      System.Windows.Forms.DialogResult.No)
  {
      docBegClsEvtArgs.Veto();
  }
}
```

⊟ *VBA/ActiveX Code Reference*

```
Private Sub AcadDocument_BeginDocClose(Cancel As Boolean)
    ' This procedure intercepts the Document BeginDocClose event.

    If MsgBox("The document is about to be closed." & _
            vbLf & "Do you want to continue?", vbYesNo, _
            "Close Document") = vbNo Then

        ' Veto the document close
        Cancel = True
    End If
End Sub
```

Handle DocumentCollection Events

DocumentCollection object events are used to respond to the open documents in the application. DocumentCollection events, unlike Document object events, remain registered until AutoCAD is shutdown or until they are unregistered.

The following events are available for DocumentCollection objects:

DocumentActivated

Triggered when a document window is activated.

DocumentActivationChanged

Triggered after the active document window is deactivated or destroyed.

DocumentBecameCurrent

Triggered when a document window is set current and is different from the previous active document window.

DocumentCreated

Triggered after a document window is created. Occurs after a new drawing is created or an existing drawing is opened.

DocumentCreateStarted

Triggered just before a document window is created. Occurs before a new drawing is created or an existing drawing is opened.

DocumentCreationCanceled

Triggered when a request to create a new drawing or to open an existing drawing is cancelled.

DocumentDestroyed

Triggered before a document window is destroyed and its associated database object is deleted.

DocumentLockModeChanged

Triggered after the lock mode of a document has changed.

DocumentLockModeChangeVetoed

Triggered after the lock mode change is vetoed.

DocumentLockModeWillChange

Triggered before the lock mode of a document is changed.

DocumentToBeActivated

Triggered when a document is about to be activated.

DocumentToBeDeactivated

Triggered when a document is about to be deactivated.

DocumentToBeDestroyed

Triggered when a document is about to be destroyed.

Enable a DocumentCollection object event

The following example uses the DocumentActivated event to indicate when a drawing window has been activated. A message box with the name of the drawing that is activated is displayed when the event occurs.

VB.NET

```
Imports Autodesk.AutoCAD.Runtime
Imports Autodesk.AutoCAD.ApplicationServices

<CommandMethod("AddDocColEvent")> _
Public Sub AddDocColEvent()
  AddHandler Application.DocumentManager.DocumentActivated, _
      AddressOf docColDocAct
End Sub

<CommandMethod("RemoveDocColEvent")> _
Public Sub RemoveDocColEvent()
  RemoveHandler Application.DocumentManager.DocumentActivated, _
      AddressOf docColDocAct
End Sub

Public Sub docColDocAct(ByVal senderObj As Object, _
                        ByVal docColDocActEvtArgs As DocumentCollectionEventArgs)
  Application.ShowAlertDialog(docColDocActEvtArgs.Document.Name & _
                              " was activated.")
End Sub
```

C#

```
using Autodesk.AutoCAD.Runtime;
using Autodesk.AutoCAD.ApplicationServices;

[CommandMethod("AddDocColEvent")]
public void AddDocColEvent()
{
  Application.DocumentManager.DocumentActivated +=
      new DocumentCollectionEventHandler(docColDocAct);
}

[CommandMethod("RemoveDocColEvent")]
public void RemoveDocColEvent()
{
  Application.DocumentManager.DocumentActivated -=
      new DocumentCollectionEventHandler(docColDocAct);
}

public void docColDocAct(object senderObj,
                         DocumentCollectionEventArgs docColDocActEvtArgs)
{
  Application.ShowAlertDialog(docColDocActEvtArgs.Document.Name +
                              " was activated.");
}
```

```
Private Sub AcadDocument_Activate()
    ' This example intercepts the Document Activate event.

    MsgBox ThisDrawing.Name & " was activated."
End Sub
```

Handle Object Events

Object events are used to respond to the opening, adding, modifying, and erasing of objects from a drawing database. There are two types of object related events: object and database level. Object level events are defined to respond to a specific object in a database, whereas Database level events respond to all objects in a database.

You define an object level event by registering an event handler with the event of a database object. Database level object events are defined by registering an event handler with one of the events of an open Database object.

The following events are available for DBObjects:

Cancelled

> Triggered when the opening of the object is cancelled text.

Copied

> Triggered after the object is cloned.

Erased

> Triggered when the object is flagged to be erased or is unerased.

Goodbye

> Triggered when the object is about to be deleted from memory because its associated database is being destroyed.

Modified

> Triggered when the object is modified.

ModifiedXData

> Triggered when the XData attached to the object is modified.

ModifyUndone

> Triggered when previous changes to the object are being undone.

ObjectClosed

Triggered when the object is closed.

OpenedForModify

Triggered before the object is modified.

Reappended

Triggered when the object is removed from the database after an Undo operation and then re-appended with a Redo operation.

SubObjectModified

Triggered when a subobject of the object is modified.

Unappended

Triggered when the object is removed from the database after an Undo operation.

The following are some of the events used to respond to object changes at the Database level:

ObjectAppended

Triggered when an object is added to a database.

ObjectErased

Triggered when an object is erased or unerased from a database.

ObjectModified

Triggered when an object has been modified.

ObjectOpenedForModify

Triggered before an object is modified.

ObjectReappended

Triggered when an object is removed from a database after an Undo operation and then re-appended with a Redo operation.

ObjectUnappended

Triggered when an object is removed from a database after an Undo operation.

Enable an Object event

This example creates a lightweight polyline with events. The event handler for the polyline then displays the new area whenever the polyline is changed. To trigger the event, simply

change the size of the polyline in AutoCAD. Remember that you must run the
CreatePLineWithEvents subroutine before the event handler is activated.

VB.NET

```vbnet
Imports Autodesk.AutoCAD.Runtime
Imports Autodesk.AutoCAD.ApplicationServices
Imports Autodesk.AutoCAD.DatabaseServices
Imports Autodesk.AutoCAD.Geometry

'' Global variable for polyline object
Dim acPoly As Polyline = Nothing

<CommandMethod("AddPlObjEvent")> _
Public Sub AddPlObjEvent()
  '' Get the current document and database, and start a transaction
  Dim acDoc As Document = Application.DocumentManager.MdiActiveDocument
  Dim acCurDb As Database = acDoc.Database

  Using acTrans As Transaction = acCurDb.TransactionManager.StartTransaction()
      '' Open the Block table record for read
      Dim acBlkTbl As BlockTable
      acBlkTbl = acTrans.GetObject(acCurDb.BlockTableId, _
                                   OpenMode.ForRead)

      '' Open the Block table record Model space for write
      Dim acBlkTblRec As BlockTableRecord
      acBlkTblRec = acTrans.GetObject(acBlkTbl(BlockTableRecord.ModelSpace), _
                                      OpenMode.ForWrite)

      '' Create a closed polyline
      acPoly = New Polyline()
      acPoly.SetDatabaseDefaults()
      acPoly.AddVertexAt(0, New Point2d(1, 1), 0, 0, 0)
      acPoly.AddVertexAt(1, New Point2d(1, 2), 0, 0, 0)
      acPoly.AddVertexAt(2, New Point2d(2, 2), 0, 0, 0)
      acPoly.AddVertexAt(3, New Point2d(3, 3), 0, 0, 0)
      acPoly.AddVertexAt(4, New Point2d(3, 2), 0, 0, 0)
      acPoly.Closed = True

      '' Add the new object to the block table record and the transaction
      acBlkTblRec.AppendEntity(acPoly)
      acTrans.AddNewlyCreatedDBObject(acPoly, True)

      AddHandler acPoly.Modified, AddressOf acPolyMod

      '' Save the new object to the database
      acTrans.Commit()
  End Using
End Sub

<CommandMethod("RemovePlObjEvent")> _
Public Sub RemovePlObjEvent()
  If acPoly <> Nothing Then
      '' Get the current document and database, and start a transaction
      Dim acDoc As Document = Application.DocumentManager.MdiActiveDocument
      Dim acCurDb As Database = acDoc.Database

      Using acTrans As Transaction = acCurDb.TransactionManager.StartTransaction()
          '' Open the polyline for read
          acPoly = acTrans.GetObject(acPoly.ObjectId, _
                                     OpenMode.ForRead)

          If acPoly.IsWriteEnabled = False Then
```

```
            acPoly.UpgradeOpen()
        End If

        RemoveHandler acPoly.Modified, AddressOf acPolyMod
        acPoly = Nothing
    End Using
  End If
End Sub

Public Sub acPolyMod(ByVal senderObj As Object, _
                     ByVal evtArgs As EventArgs)
  Application.ShowAlertDialog("The area of " & _
                             acPoly.ToString() & " is: " & _
                             acPoly.Area)
End Sub
```

C#

```csharp
using Autodesk.AutoCAD.Runtime;
using Autodesk.AutoCAD.ApplicationServices;
using Autodesk.AutoCAD.DatabaseServices;
using Autodesk.AutoCAD.Geometry;

// Global variable for polyline object
Polyline acPoly = null;

[CommandMethod("AddPlObjEvent")]
public void AddPlObjEvent()
{
  // Get the current document and database, and start a transaction
  Document acDoc = Application.DocumentManager.MdiActiveDocument;
  Database acCurDb = acDoc.Database;

  using (Transaction acTrans = acCurDb.TransactionManager.StartTransaction())
  {
      // Open the Block table record for read
      BlockTable acBlkTbl;
      acBlkTbl = acTrans.GetObject(acCurDb.BlockTableId,
                                   OpenMode.ForRead) as BlockTable;

      // Open the Block table record Model space for write
      BlockTableRecord acBlkTblRec;
      acBlkTblRec = acTrans.GetObject(acBlkTbl[BlockTableRecord.ModelSpace],
                                      OpenMode.ForWrite) as BlockTableRecord;

      // Create a closed polyline
      acPoly = new Polyline();
      acPoly.SetDatabaseDefaults();
      acPoly.AddVertexAt(0, new Point2d(1, 1), 0, 0, 0);
      acPoly.AddVertexAt(1, new Point2d(1, 2), 0, 0, 0);
      acPoly.AddVertexAt(2, new Point2d(2, 2), 0, 0, 0);
      acPoly.AddVertexAt(3, new Point2d(3, 3), 0, 0, 0);
      acPoly.AddVertexAt(4, new Point2d(3, 2), 0, 0, 0);
      acPoly.Closed = true;

      // Add the new object to the block table record and the transaction
      acBlkTblRec.AppendEntity(acPoly);
      acTrans.AddNewlyCreatedDBObject(acPoly, true);

      acPoly.Modified += new EventHandler(acPolyMod);

      // Save the new object to the database
      acTrans.Commit();
  }
```

```
}

[CommandMethod("RemovePlObjEvent")]
public void RemovePlObjEvent()
{
  if (acPoly != null)
  {
      // Get the current document and database, and start a transaction
      Document acDoc = Application.DocumentManager.MdiActiveDocument;
      Database acCurDb = acDoc.Database;

      using (Transaction acTrans = acCurDb.TransactionManager.StartTransaction())
      {
          // Open the polyline for read
          acPoly = acTrans.GetObject(acPoly.ObjectId,
                                     OpenMode.ForRead) as Polyline;

          if (acPoly.IsWriteEnabled == false)
          {
              acPoly.UpgradeOpen();
          }

          acPoly.Modified -= new EventHandler(acPolyMod);
          acPoly = null;
      }
  }
}

public void acPolyMod(object senderObj,
                      EventArgs evtArgs)
{
  Application.ShowAlertDialog("The area of " +
                              acPoly.ToString() + " is: " +
                              acPoly.Area);
}
```

⊟ *VBA/ActiveX Code Reference*

```
Public WithEvents PLine As AcadLWPolyline
Sub CreatePLineWithEvents()
    ' This example creates a light weight polyline
    Dim points(0 To 9) As Double
    points(0) = 1: points(1) = 1
    points(2) = 1: points(3) = 2
    points(4) = 2: points(5) = 2
    points(6) = 3: points(7) = 3
    points(8) = 3: points(9) = 2

    Set PLine = ThisDrawing.ModelSpace. _
    AddLightWeightPolyline(points)
    PLine.Closed = True

    ThisDrawing.Application.ZoomAll
End Sub
Private Sub PLine_Modified _
 (ByVal pObject As AutoCAD.IAcadObject)
    ' This event is triggered when the polyline is resized.
    ' If the polyline is deleted the modified event is still
    ' triggered, so we use the error handler to avoid
    ' reading data from a deleted object.

    On Error GoTo ERRORHANDLER
    MsgBox "The area of " & pObject.ObjectName & " is: " _
```

```
        & pObject.Area
    Exit Sub

ERRORHANDLER:
    MsgBox Err.Description
End Sub
```

Register COM Based Events with .NET

The AutoCAD COM Automation library offers some unique events that are not found in the .NET API. Registering events that are in a COM library is different than how you would initialize an event using VB or VBA. You use the VB.NET AddHandler statement or the C# += operator to reigister an event handler with the event. The event handler requires the address of the procedure in which should be called when the event is raised.

Register a COM based event

This example demonstrates how to register the BeginFileDrop event using COM interop. The BeginFileDrop event is associated with the Application object of the AutoCAD COM Automation library. Once the commands are loaded into AutoCAD, enter AddCOMEvent at the Command prompt and then drag and drop a DWG file into the drawing window. A message box will be displayed prompting you to continue. Use the RemoveCOMEvent command to remove the event handler.

VB.NET

```
Imports Autodesk.AutoCAD.Runtime
Imports Autodesk.AutoCAD.ApplicationServices
Imports Autodesk.AutoCAD.DatabaseServices

Imports Autodesk.AutoCAD.Interop
Imports Autodesk.AutoCAD.Interop.Common

'' Global variable for AddCOMEvent and RemoveCOMEvent commands
Dim acAppCom As AcadApplication

<CommandMethod("AddCOMEvent")> _
Public Sub AddCOMEvent()
  '' Set the global variable to hold a reference to the application and
  '' register the BeginFileDrop COM event
  acAppCom = Application.AcadApplication
  AddHandler acAppCom.BeginFileDrop, AddressOf appComBeginFileDrop
End Sub

<CommandMethod("RemoveCOMEvent")> _
Public Sub RemoveCOMEvent()
  '' Unregister the COM event handle
  RemoveHandler acAppCom.BeginFileDrop, AddressOf appComBeginFileDrop
  acAppCom = Nothing
End Sub

Public Sub appComBeginFileDrop(ByVal strFileName As String, _
                               ByRef bCancel As Boolean)
  '' Display a message box prompting to continue inserting the DWG file
  If System.Windows.Forms.MessageBox.Show("AutoCAD is about to load " & _
```

```
                        strFileName & vbLf & _
                        "Do you want to continue loading this file?", _
                        "DWG File Dropped", _
                        System.Windows.Forms.MessageBoxButtons.YesNo) = _
    System.Windows.Forms.DialogResult.No Then
      bCancel = True
  End If
End Sub
```

C#

```
using Autodesk.AutoCAD.Runtime;
using Autodesk.AutoCAD.ApplicationServices;
using Autodesk.AutoCAD.DatabaseServices;

using Autodesk.AutoCAD.Interop;
using Autodesk.AutoCAD.Interop.Common;

// Global variable for AddCOMEvent and RemoveCOMEvent commands
AcadApplication acAppCom;

[CommandMethod("AddCOMEvent")]
public void AddCOMEvent()
{
  // Set the global variable to hold a reference to the application and
  // register the BeginFileDrop COM event
  acAppCom = Application.AcadApplication as AcadApplication;
  acAppCom.BeginFileDrop +=
      new _DAcadApplicationEvents_BeginFileDropEventHandler(appComBeginFileDrop);
}

[CommandMethod("RemoveCOMEvent")]
public void RemoveCOMEvent()
{
  // Unregister the COM event handle
  acAppCom.BeginFileDrop -=
      new _DAcadApplicationEvents_BeginFileDropEventHandler(appComBeginFileDrop);
  acAppCom = null;
}

public void appComBeginFileDrop(string strFileName, ref bool bCancel)
{
  // Display a message box prompting to continue inserting the DWG file
  if (System.Windows.Forms.MessageBox.Show("AutoCAD is about to load " +
strFileName +
                                          "\nDo you want to continue loading this
file?",
                                          "DWG File Dropped",
                                          System.Windows.Forms.MessageBoxButtons.YesN
o) ==
                                          System.Windows.Forms.DialogResult.No)
  {
      bCancel = true;
  }
}
```

⊟ **VBA/ActiveX Code Referemce**

```
Public WithEvents ACADApp As AcadApplication
Sub Example_AcadApplication_Events()
    ' Intialize the public variable (ACADApp)
    ' which will be used to intercept AcadApplication Events
```

```vb
     '
     ' Run this procedure FIRST!
     Set ACADApp = ThisDrawing.Application
End Sub
Private Sub ACADApp_BeginFileDrop _
 (ByVal FileName As String, Cancel As Boolean)
     ' This procedure intercepts an Application BeginFileDrop event.
     '
     ' This event is triggered when a drawing file is dragged
     ' into AutoCAD.
     '
     ' To trigger this example event:
     ' 1) Run the Example_AcadApplication_Events procedure to initialize
     ' the public variable (named ACADApp) linked to this event.
     '
     ' 2) Drag an AutoCAD drawing file into the AutoCAD
     ' application from either the Windows Desktop
     ' or Windows Explorer
     '
     ' Use the "Cancel" variable to stop the loading of the
     ' dragged file, and the "FileName" variable to  notify
     ' the user which file is about to be dragged in.

     If MsgBox("AutoCAD is about to load " & FileName & vbCrLf _
               & "Do you want to continue loading this file?", _
               vbYesNoCancel + vbQuestion) <> vbYes Then
         Cancel = True
     End If
End Sub
```

10   Develop Applications with VB.NET and C#

Many programming tasks involve more than simply working with the AutoCAD .NET API object model. This chapter provides a brief overview of handling errors and distributing your application to others.

Remember, the Microsoft documentation for VB.NET and C# contains more information on these topics.

Topics in this section

- [Handle Errors](#)
- [Distribute Your Application](#)

Handle Errors

Most development environments provide default error handling. For C# and VB.NET, the default reaction to an error is to display an error message and terminate the application. While this behavior is adequate during the development, it is not productive for the user.

Errors raised during execution should be handled and not left to the user to encounter whenever possible. When designing an application, you should catch all possible errors and then determine how to respond to them. In some situations an error can be safely ignored, while in others you will need to handle the error with a specific response in order for the application to continue.

When you catch an error, the default error message is suppressed and the application does not automatically terminate. With the default error message suppressed, you can display a custom error message or have the application handle the error.

In general, error handling is necessary whenever user input is required, during file I/O operations, and when a database or object is being accessed. Even if you are sure a file or object is available, there may be conditions you have not thought of that could cause errors.

NoteMost of the code examples provided in this documentation do not use error handling or the error handling is limited in scope. This keeps the examples simple and to the point. However, as with all programming languages, proper error handling is essential for a robust application.

Topics in this section

- [Define Application Error Types](#)
- [Trap Runtime Errors](#)
- [Respond to User Input Errors](#)

Define Application Error Types

There are three different types of errors you can encounter in your applications: compile-time errors, runtime errors, and logic errors.

- Compile-time errors occur during the construction of your application. These errors consist mostly of syntax mistakes, and variable scope and data type problems. In C# and VB.NET, these types of errors are caught by the development environment. When you enter an incorrect line of code, the line is underlined and the problem with the code line is displayed in a tooltip when the cursor is positioned over the underlined text. Compile-time errors must be corrected before the .NET assembly for the application can be built.
- Runtime errors are a little more difficult to find and correct. They occur during the execution of your code, and often involve information provided by the user or files that are expected to be present. For example, if your application requires the user to enter the name of a drawing and the user enters a name for a drawing that does not exist, a runtime error occurs. To handle runtime errors effectively, you must predict what kinds of problems could happen, trap them, and then write code to handle these situations.
- Logic errors are the most difficult to find and correct. Symptoms of logic errors include situations in which there are no compile-time errors and no runtime errors, but the outcome of your program is still incorrect. This is what programmers refer to as a defect or bug. A defect can be very easy or difficult to track down.

Information on finding and correcting errors can be found in the documentation for your development environment. AutoCAD-specific errors fall into the runtime error category, so these types of errors are covered in this documentation.

Trap Runtime Errors

VB.NET and C# support common and language specific ways to handle errors at runtime. The Try statement is available in both C# and VB.NET to trap errors, while the On Error statement is available in VB.NET and Goto is only in C#.

For more information on the Try, On Error, and Goto statements beyond what is mentioned in this guide, see the documentation that comes with your development environment.

Topics in this section

- Use Try Statements
- Use the Exception Object
- On Error Statements (VB.NET)
- Compare Error Handlers in VBA or VB to .NET

Use Try Statements

In VB.NET and C#, runtime errors can be trapped using a Try statement. This statement literally sets a trap for the system. When an error occurs in a Try statement, the default error handling for the system is bypassed and execution is redirected to its Catch clause.

The Try statement has three forms:

- Try-Catch
- Try-Finally
- Try-Catch-Finally

Try-Catch Statement

The Try-Catch statement is used when you want to respond to an error. This statement traps the error and instead of displaying a default error message and terminating the application, execution is moved to the Catch clause of the Try statement.

The Catch clause can optional contain a single parameter which accepts an Exception object. The Exception object contains information about the error encountered. If the error that is encountered cannot be resolved, you should display a custom error message and exit the application gracefully.

For information on the Exception object, see [Use the Exception Object](#).

Try-Finally Statement

The Try-Finally statement is used when you do not want to provide specific error handling. This statement traps an error, and displays the default error message without terminating the application. When an error is encountered, execution is moved from the Try statement to its Finally clause after Continue is clicked in the default message box. The Try-Finally statement is best used when an application is still being developed and debugged.

Try-Catch-Finally Statement

The Try-Catch-Finally statement is a combination of the Try-Catch and Try-Finally statements. This statement traps the error and instead of displaying a default error message and terminating the application, execution is moved to the Catch clause of the Try statement. After the code is executed in the Catch clause, execution is moved to the Finally clause which gives your application one last chance to either continue execution or to exit gracefully.

Test error handling without and with the Try-Catch-Finally statement

The following examples attempt to open a file named "Drawing123" on the C: drive. If the file is not found, an eFileNotFound error is thrown. The first command does not catch the error thrown by the ReadDwgFile method, so the default message box is displayed when the command is started in AutoCAD. The second command catches the error thrown using the Try-Catch-Finally statement.

VB.NET

```vbnet
Imports Autodesk.AutoCAD.Runtime
Imports Autodesk.AutoCAD.ApplicationServices
Imports Autodesk.AutoCAD.DatabaseServices

<CommandMethod("NoErrorHandler")> _
Public Sub NoErrorHandler()
  '' Create a new database with no document window
  Using acDb As Database = New Database(False, True)
      '' Read the drawing file named "Drawing123.dwg" on the C: drive.
      '' If the "Drawing123.dwg" file does not exist, an eFileNotFound
      '' exception is tossed and the program halts.
      acDb.ReadDwgFile("c:\Drawing123.dwg", _
                       System.IO.FileShare.None, False, "")
  End Using

  '' Message will not be displayed since the exception caused by
  '' ReadDwgFile is not handled.
  Application.ShowAlertDialog("End of command reached")
End Sub

<CommandMethod("ErrorTryCatchFinally")> _
Public Sub ErrorTryCatchFinally()
  '' Create a new database with no document window
  Using acDb As Database = New Database(False, True)
      Try
          '' Read the drawing file named "Drawing123.dwg" on the C: drive.
          '' If the "Drawing123.dwg" file does not exist, an eFileNotFound
          '' exception is tossed and the catch statement handles the error.
          acDb.ReadDwgFile("c:\Drawing123.dwg", _
                           System.IO.FileShare.None, False, "")
      Catch Ex As Autodesk.AutoCAD.Runtime.Exception
          Application.ShowAlertDialog("The following exception was caught:" & _
                                      vbLf & Ex.Message)
      Finally
          '' Message is displayed since the exception caused
          '' by ReadDwgFile is handled.
          Application.ShowAlertDialog("End of command reached")
      End Try
  End Using
End Sub
```

C#

```csharp
using Autodesk.AutoCAD.Runtime;
using Autodesk.AutoCAD.ApplicationServices;
using Autodesk.AutoCAD.DatabaseServices;

[CommandMethod("NoErrorHandler")]
public void NoErrorHandler()
{
  // Create a new database with no document window
  using (Database acDb = new Database(false, true))
  {
      // Read the drawing file named "Drawing123.dwg" on the C: drive.
      // If the "Drawing123.dwg" file does not exist, an eFileNotFound
      // exception is tossed and the program halts.
      acDb.ReadDwgFile("c:\\Drawing123.dwg",
                       System.IO.FileShare.None, false, "");
  }

  // Message will not be displayed since the exception caused by
  // ReadDwgFile is not handled.
```

```
      Application.ShowAlertDialog("End of command reached");
}

[CommandMethod("ErrorTryCatchFinally")]
public void ErrorTryCatchFinally()
{
   // Create a new database with no document window
   using (Database acDb = new Database(false, true))
   {
       try
       {
           // Read the drawing file named "Drawing123.dwg" on the C: drive.
           // If the "Drawing123.dwg" file does not exist, an eFileNotFound
           // exception is tossed and the catch statement handles the error.
           acDb.ReadDwgFile("c:\\Drawing123.dwg",
                            System.IO.FileShare.None, false, "");
       }
       catch (Autodesk.AutoCAD.Runtime.Exception Ex)
       {
           Application.ShowAlertDialog("The following exception was caught:\n" +
                                       Ex.Message);
       }
       finally
       {
           // Message is displayed since the exception caused
           // by ReadDwgFile is handled.
           Application.ShowAlertDialog("End of command reached");
       }
   }
}
```

Use the Exception Object

The Exception object is used to obtain information about the error trapped with the Catch clause of a Try statement. The Exception object used for the AutoCAD .NET API is based on the Exception object of the Microsoft® .NET Framework. To determine the error that is trapped by the Catch clause, you can use the properties of the Exception object. Some of the member properties of the Exception object that you can get information about the error are:

- ErrorStatus - Returns the Autodesk.AutoCAD.Runtime.ErrorStatus enum value assigned to the exception.
- Message - Returns a text message that explains the exception.
- Source - Returns the application or object that caused the exception.
- StackTrace - Returns a string representation of the frames on the call stack when the exception occurred.
- TargetSite - Returns the method that threw the exception.

On Error Statements (VB.NET)

In VB.NET, runtime errors can be trapped with the Try or On Error statement. This statement literally sets a general trap in the application. When an error occurs, this statement automatically detours processing to your specially written error handler. The default error handling for the system is bypassed.

The On Error statement has three forms:

- On Error Resume Next
- On Error GoTo Label
- On Error GoTo 0

NoteBoth Try and On Error statements cannot be used in the same procedure.

On Error Resume Next statement

The On Error Resume Next statement is used when you want to ignore errors. This statement traps the error and instead of displaying an error message and terminating the program, execution is moved to the next line of code and continues processing.

For example, if you wanted to create a procedure to iterate through Model space and change the color of each entity, you know that AutoCAD will throw an error if you try to color an entity on a locked layer. Instead of terminating the program, simply skip the entity on the locked layer and continue processing the remaining entities. The On Error Resume Next statement lets you do just that.

On Error GoTo Label statement

The On Error GoTo Label statement is used when you want to write an explicit error handler. This statement traps the error and instead of displaying an error message and terminating the program, it jumps to a specific location in your code. Your code can then respond to the error in whatever manner is appropriate for your application. For example, you could jump to the beginning of the program using 0 (zero), a line number, or a named label such as ErrNoFileFound. Named labels are defined by using the following syntax:

```
HandlerName:
```

Use the Err object with trapped errors

The Exception object is used with Try statements, while the Err object is used to provide information on the type of error that has been trapped with the On Error statements. This object has several properties: Number, Description, Source, HelpFile, HelpContext, and LastDLLError. The properties of the Err object get filled in with the information for the most current error. The most important properties are the Number and Description properties. The Number property contains the unique error code associated with the error, and the Description property contains the error message that would normally be displayed.

In your error handler you can compare the Number property of the error to an expected value. This will help you determine the nature of the error that was occurred. Once you know what kind of error you are dealing with, you can take the appropriate action.

Compare Error Handlers in VBA or VB to .NET

Error handling in VBA or VB is done using the On Error statements. While the On Error statements can be used with VB.NET without any problems, it is recommended to utilize Try statements instead. Try statements are more flexibility than the On Error Resume Next and On Error GoTo Label statements.

The use of On Error Resume Next and On Error GoTo Label statements can be rewritten using Try-Catch statements. The following shows how an On Error GoTo Label statement can be rewritten using Try-Catch.

On Error - VBA

```
Sub ColorEntities()
    On Error GoTo MyErrorHandler

    Dim entry As Object
    For Each entry In ThisDrawing.ModelSpace
        entry.color = acRed
    Next entry

    ' Important! Exit the subroutine before the error handler
  Exit Sub

MyErrorHandler:
    MsgBox entry.EntityName + " is on a locked layer." + _
                            " The handle is: " + entry.Handle

  Resume Next
End Sub
```

Try-Catch - VB.NET

```
<CommandMethod("ColorEntities")> _
Public Sub ColorEntities()
  '' Get the current document and database, and start a transaction
  Dim acDoc As Document = Application.DocumentManager.MdiActiveDocument
  Dim acCurDb As Database = acDoc.Database

  Using acTrans As Transaction = acCurDb.TransactionManager.StartTransaction()
      '' Open the Block table record for read
      Dim acBlkTbl As BlockTable
      acBlkTbl = acTrans.GetObject(acCurDb.BlockTableId, _
                                OpenMode.ForRead)

      '' Open the Block table record Model space for read
      Dim acBlkTblRec As BlockTableRecord
      acBlkTblRec = acTrans.GetObject(acBlkTbl(BlockTableRecord.ModelSpace), _
                                OpenMode.ForRead)
```

```
        Dim acObjId As ObjectId

        '' Step through each object in Model space
        For Each acObjId In acBlkTblRec
            Try
                Dim acEnt As Entity
                acEnt = acTrans.GetObject(acObjId, _
                                          OpenMode.ForWrite)

                acEnt.ColorIndex = 1
            Catch
                Application.ShowAlertDialog(acObjId.ObjectClass.DxfName & _
                                            " is on a locked layer." & _
                                            " The handle is: " &
acObjId.Handle.ToString())
            End Try
        Next

        acTrans.Commit()
    End Using
End Sub
```

Respond to User Input Errors

The user-input methods provide a certain amount of inherent error trapping in that they require the user to enter a certain type of data. If the user tries to enter some other data, AutoCAD rejects the input and re-prompts the user. Using the PromptXXXOption objects with the appropriate GetXXX or SelectXXX methods provide additional control of the user input but can also introduce additional conditions that must be verified before execution continues. For examples of checking the status of input provided by a user, see Prompt for User Input.

Distribute Your Application

.NET applications can be distributed in two deployable builds: debug and release.

- *Debug build* - Contains debugging related information. .NET assemblies containing debug information are larger in size than a Release build of a .NET assembly.
- *Release build* - Contains no debug related information.

You must choose the type of build to distribute your application in, both build types can be loaded into AutoCAD. Debug builds are usually only used when developing and testing an application, while a Release build is built when you are distributing an application for use on many computers inside or outside of your company.

⊟ *To generate a Release build for a .NET assembly*

Load a .NET assembly

After you have determined the build type of your .NET assembly, you must determine how it will be loaded into AutoCAD. A .NET assembly file can be loaded manually or with demand loading.

- Manually - Use the NETLOAD command at the Command prompt or within an AutoLISP file. For information on loading a .NET assembly with the NETLOAD command, see [Load an Assembly into AutoCAD](#).
- Demand load - Define a key specific to the application you want to load when AutoCAD starts up. The key must be placed under the Application key for the specific release of AutoCAD that you want your application to be loaded in.

  The key for the application can contain the following keys:

  DESCRIPTION

  Description of the .NET assemly and is optional.

  LOADCTRLS

  Controls how and when the .NET assembly is loaded.

    o 1 - Load application upon detection of proxy object
    o 2 - Load the application at startup
    o 4 - Load the application at start of a command
    o 8 - Load the application at the request of a user or another application
    o 16 - Do not load the application
    o 32 - Load the application transparently

  LOADER

  Specifies which .NET assembly file to load.

  MANAGED

  Specifies the file that should be loaded is a .NET assembly or ObjectARX file. Set to 1 for .NET assembly files.

Demand load a .NET application

The following examples create and remove the required keys in the registry to load a .NET assembly file at the startup of AutoCAD. When the RegisterMyApp command is used, the required registry keys are created that will automatically load the application the next time AutoCAD starts. The UnregisterMyApp command removes the demand loading information from the registry so the application is not loaded the next time AutoCAD starts.

VB.NET

```
Imports Microsoft.Win32
Imports System.Reflection

Imports Autodesk.AutoCAD.Runtime
Imports Autodesk.AutoCAD.ApplicationServices

<CommandMethod("RegisterMyApp")> _
```

```vb
Public Sub RegisterMyApp()
  '' Get the AutoCAD Applications key
  Dim sProdKey As String = HostApplicationServices.Current.RegistryProductRootKey
  Dim sAppName As String = "MyApp"

  Dim regAcadProdKey As RegistryKey = Registry.CurrentUser.OpenSubKey(sProdKey)
  Dim regAcadAppKey As RegistryKey = regAcadProdKey.OpenSubKey("Applications",
True)

  '' Check to see if the "MyApp" key exists
  Dim subKeys() As String = regAcadAppKey.GetSubKeyNames()
  For Each sSubKey As String In subKeys
      '' If the application is already registered, exit
      If (sSubKey.Equals(sAppName)) Then
         regAcadAppKey.Close()

         Exit Sub
      End If
  Next

  '' Get the location of this module
  Dim sAssemblyPath As String = Assembly.GetExecutingAssembly().Location

  '' Register the application
  Dim regAppAddInKey As RegistryKey = regAcadAppKey.CreateSubKey(sAppName)
  regAppAddInKey.SetValue("DESCRIPTION", sAppName, RegistryValueKind.String)
  regAppAddInKey.SetValue("LOADCTRLS", 14, RegistryValueKind.DWord)
  regAppAddInKey.SetValue("LOADER", sAssemblyPath, RegistryValueKind.String)
  regAppAddInKey.SetValue("MANAGED", 1, RegistryValueKind.DWord)

  regAcadAppKey.Close()
End Sub

<CommandMethod("UnregisterMyApp")> _
Public Sub UnregisterMyApp()
  '' Get the AutoCAD Applications key
  Dim sProdKey As String = HostApplicationServices.Current.RegistryProductRootKey
  Dim sAppName As String = "MyApp"

  Dim regAcadProdKey As RegistryKey = Registry.CurrentUser.OpenSubKey(sProdKey)
  Dim regAcadAppKey As RegistryKey = regAcadProdKey.OpenSubKey("Applications",
True)

  '' Delete the key for the application
  regAcadAppKey.DeleteSubKeyTree(sAppName)
  regAcadAppKey.Close()
End Sub
```

C#

```csharp
using Microsoft.Win32;
using System.Reflection;

using Autodesk.AutoCAD.Runtime;
using Autodesk.AutoCAD.ApplicationServices;

[CommandMethod("RegisterMyApp")]
public void RegisterMyApp()
{
  // Get the AutoCAD Applications key
  string sProdKey = HostApplicationServices.Current.RegistryProductRootKey;
  string sAppName = "MyApp";

  RegistryKey regAcadProdKey = Registry.CurrentUser.OpenSubKey(sProdKey);
  RegistryKey regAcadAppKey = regAcadProdKey.OpenSubKey("Applications", true);

  // Check to see if the "MyApp" key exists
  string[] subKeys = regAcadAppKey.GetSubKeyNames();
  foreach (string subKey in subKeys)
  {
      // If the application is already registered, exit
      if (subKey.Equals(sAppName))
      {
          regAcadAppKey.Close();
          return;
      }
  }

  // Get the location of this module
  string sAssemblyPath = Assembly.GetExecutingAssembly().Location;

  // Register the application
  RegistryKey regAppAddInKey = regAcadAppKey.CreateSubKey(sAppName);
  regAppAddInKey.SetValue("DESCRIPTION", sAppName, RegistryValueKind.String);
  regAppAddInKey.SetValue("LOADCTRLS", 14, RegistryValueKind.DWord);
  regAppAddInKey.SetValue("LOADER", sAssemblyPath, RegistryValueKind.String);
  regAppAddInKey.SetValue("MANAGED", 1, RegistryValueKind.DWord);

  regAcadAppKey.Close();
}

[CommandMethod("UnregisterMyApp")]
public void UnregisterMyApp()
{
  // Get the AutoCAD Applications key
  string sProdKey = HostApplicationServices.Current.RegistryProductRootKey;
  string sAppName = "MyApp";

  RegistryKey regAcadProdKey = Registry.CurrentUser.OpenSubKey(sProdKey);
  RegistryKey regAcadAppKey = regAcadProdKey.OpenSubKey("Applications", true);

  // Delete the key for the application
  regAcadAppKey.DeleteSubKeyTree(sAppName);
  regAcadAppKey.Close();
}
```

11   VBA/VB to VB.NET and C# Comparison

While most programming languages differ from each other in their syntax and capabilities, there still some fundamental concepts and logic that they share. This appendix serves as a reference to help developers that are familiar with VBA/VB to find the equivalent VB.NET or C# functionality.

Topics in this section

VBA to VB.NET and C# Comparison

The following table compares VBA functions with the similar VB.NET and C# functions and operators. The ActiveX library is indicated by "AutoCAD.Application" and the .NET Managed library equivalents are indicated by "Autodesk.AutoCAD" and the VB.NET or C# equivalents are listed as a function or operator.

Math Functions

ActiveX, VBA, or Visual Basic 6	VB.NET and C# equivalent (same unless noted)
+ (addition operator)	+ (addition operator)
- (subtraction operator)	- (subtraction operator)
* (multiplication operator)	* (multiplication operator)
/ (division operator)	/ (division operator)
^ (exponentiation operator)	^ (exponentiation operator)
Abs function	System.Math.Abs function
Atn function	System.Math.Atan function
Cos function	System.Math.Cos function
Exp function	System.Math.Exp function
Log function	System.Math.Log function
Max function	System.Math.Max function
Min function	System.Math.Min function
	VB.NET
	Mod function
	C#
Mod function	% (operator)
	VB.NET and C#
	System.Math.DivRem function

Sin function	System.Math.Sin function
Sqr function	System.Math.Sqrt function

Conditional and Loop Statements

ActiveX, VBA, or Visual Basic 6	VB.NET and C# equivalent (same unless noted)
	VB.NET
Loop Do Until… statement	Do Until... Loop statement
	C#
	Use do... while statement
	VB.NET
Loop Do While… statement	Do While... Loop statement
	C#
	do... while statement
	VB.NET
For Each...Next statement	For Each...Next statement
	C#
	Foreach and For statements
	VB.NET
If… Then… Else...End If statement	If… Then… Else...End If statement
	C#
	if... else... statement
	VB.NET
Select Case statement	Select Case statement
	C#
	Switch statement
	VB.NET
Wend While… statement	While... Wend statement
	C#
	while... statement

Logic Statements

ActiveX, VBA, or Visual Basic 6	VB.NET and C# equivalent (same unless noted)
	VB.NET
	= (equal to comparison operator)
= (equal to comparison operator)	
	C#
	== (equal to comparison operator)
	VB.NET
	<> (not equal to comparison operator)
<> (not equal to comparison operator)	
	C#
	!= (not equal to comparison operator)
< (less than comparison operator)	< (less than comparison operator)
<= (less than or equal to comparison operator)	<= (less than or equal to comparison operator)
> (greater than comparison operator)	> (greater than comparison operator)
>= (greater than or equal to comparison operator)	>= (greater than or equal to comparison operator)
	VB.NET
	And operator
And function	
	C#
	&& operator
Eqv operator	Not provided, use other bitwise comparison methods instead
Imp operator	Not provided, use = comparison instead
	VB.NET
	object Is *object*
Is operator	
	C#
	object is *object*
	VB.NET
	IsArray function
	or
IsArray function	TypeOf *arrayName* Is Array comparison
	C#
	typeof(*arrayName*) == Array comparison

	VB.NET and C#
	varName.GetType().IsArray
	VB.NET
	IsDBNull function
IsNull function	**C#**
	Use == null comparison
	VB.NET
	Like operator
Like operator	**VB.NET and C#**
	stringVariable.Contains function
	VB.NET
	Not operator
Not operator	**C#**
	!= (not equal to comparison operator)
	VB.NET
	Or function
Or function	**C#**
	|| operator

Data Conversion Functions

ActiveX, VBA, or Visual Basic 6	VB.NET and C# equivalent (same unless noted)
	VB.NET
Asc function	Asc function
	C#
	(int)'*letter*'
AutoCAD.Application.ActiveDocument.Utility.AngleToReal method	Autodesk.AutoCAD.Runtime.Converter. StringToAngle method
AutoCAD.Application.ActiveDocument.Utility.AngleToString method	Autodesk.AutoCAD.Runtime.Converter. AngleToString method
AutoCAD.Application.ActiveDocument.Utility.RealToString method	Autodesk.AutoCAD.Runtime.Converter. DistanceToString function
	VB.NET
CDbl Function	CDbl function
	V**B.NET and C**#
	System.Convert.ToDouble function
	VB.NET
Chr function	Chr function
	VB.NET and C#
	System.Convert.ToChar
	VB.NET
	CInt function
CInt Function	**VB.NET and C#**
	System.Convert.ToInt16, System.Convert.ToInt32, or System.Convert.ToInt64 function
	VB.NET
Fix function	Fix function
	VB.NET and C#
	System.Convert.ToInt16,

	VB.NET and C# equivalent (same unless noted)
	System.Convert.ToInt32, or System.Convert.ToInt64 function
	VB.NET
	Int function
Int function	**VB.NET and C#**
	System.Convert.ToInt16, System.Convert.ToInt32, or System.Convert.ToInt64 function
	VB.NET
	Str function
Str function	**VB.NET and C#**
	System.Convert.ToString function
	VB.NET
	StrConv function
StrConv function	**VB.NET and C#**
	System.Text.Encoding.Convert function

Basic String Manipulation Functions

ActiveX, VBA, or Visual Basic 6	VB.NET and C# equivalent (same unless noted)
	VB.NET
	& or + operator
& operator (concatenate string)	**C#**
	+ operator
	VB.NET
	Len function
Len function	**VB.NET and C#**
	stringVariable.Length property
	VB.NET
	Mid function
Mid function	**VB.NET and C#**
	stringVariable.Substring function

Get Input from the AutoCAD Command Prompt Functions

ActiveX, VBA, or Visual Basic 6	VB.NET and C# equivalent (same unless noted)
AutoCAD.Application.ActiveDocument. Utility.GetAngle method	Autodesk.AutoCAD.ApplicationServices.Application. DocumentManager.MdiActiveDocument.Editor. GetAngle function
AutoCAD.Application.ActiveDocument. Utility.GetCorner method	Autodesk.AutoCAD.ApplicationServices.Application. DocumentManager.MdiActiveDocument.Editor. GetCorner function
AutoCAD.Application.ActiveDocument. Utility.GetDistance method	Autodesk.AutoCAD.ApplicationServices.Application. DocumentManager.MdiActiveDocument.Editor. GetDistance function
AutoCAD.Application.ActiveDocument. Utility.GetEntity method	Autodesk.AutoCAD.ApplicationServices.Application. DocumentManager.MdiActiveDocument.Editor. GetEntity function
AutoCAD.Application.ActiveDocument. Utility.GetInteger method	Autodesk.AutoCAD.ApplicationServices.Application. DocumentManager.MdiActiveDocument.Editor. GetInteger function
AutoCAD.Application.ActiveDocument. Utility.GetKeyword method	Autodesk.AutoCAD.ApplicationServices.Application. DocumentManager.MdiActiveDocument.Editor. GetKeyword function
AutoCAD.Application.ActiveDocument. Utility.GetOrientation method	Autodesk.AutoCAD.ApplicationServices.Application. DocumentManager.MdiActiveDocument.Editor. GetAngle function
AutoCAD.Application.ActiveDocument. Utility.GetPoint method	Autodesk.AutoCAD.ApplicationServices.Application. DocumentManager.MdiActiveDocument.Editor. GetPoint function
AutoCAD.Application.ActiveDocument. Utility.GetReal method	Autodesk.AutoCAD.ApplicationServices.Application. DocumentManager.MdiActiveDocument.Editor. GetDouble function
AutoCAD.Application.ActiveDocument.	Autodesk.AutoCAD.ApplicationServices.Application.

Utility.GetString method	DocumentManager.MdiActiveDocument.Editor.
AutoCAD.Application.ActiveDocument.Autodesk.AutoCAD.EditorInput.	GetString function
Utility.InitializeUserInput	PromptKeywordOptions

Basic AutoCAD Application and Drawing Functions

ActiveX, VBA, or Visual Basic 6	VB.NET and C# equivalent (same unless noted)
AutoCAD.Application.ActiveDocument.	Autodesk.AutoCAD.Geometry.Point2d(*point1*).
Utility.AngleFromXAxis method	GetVectorTo(*point2*).Angle property
	Autodesk.AutoCAD.Runtime.SystemObjects.
AutoCAD.Application.ListARX method	
	DynamicLinker.GetLoadedModules function
	Autodesk.AutoCAD.Runtime.SystemObjects.
AutoCAD.Application.LoadARX method	
	DynamicLinker.LoadModule method
	Autodesk.AutoCAD.Runtime.SystemObjects.
AutoCAD.Application.UnloadARX method	
	DynamicLinker.UnloadModule method
	Autodesk.AutoCAD.ApplicationServices.Application.
	DocumentManager.MdiActiveDocument.
	CloseAndDiscard method
AutoCAD.Application.	or
Documents.Close method	Autodesk.AutoCAD.ApplicationServices.Application.
	DocumentManager.MdiActiveDocument.
	CloseAndSave method
	Autodesk.AutoCAD.ApplicationServices.Application.
AutoCAD.Application.ActiveDocument.	DocumentManager.MdiActiveDocument.
SendCommand method	SendStringToExecute method
	VB.NET
	dictionaryObj = *transactionObj*.
AutoCAD.Application.ActiveDocument.	GetObject(*workingDatabaseObj*.
Dictionaries.Add method	NamedObjectsDictionaryId, *openMode*)
	dictionaryObj.SetAt function
	C#

	dictionaryObj = *transactionObj*.
	GetObject(*workingDatabaseObj*.
	NamedObjectsDictionaryId, *openMode*) as Autodesk.AutoCAD.DatabaseServices.
	DBDictionary;
	dictionaryObj.SetAt function
	VB.NET
	dictionaryObj = *transactionObj*.
	GetObject(*workingDatabaseObj*.
	NamedObjectsDictionaryId, *openMode*)
	dictionaryObj.GetAt function
AutoCAD.Application.ActiveDocument. Dictionaries.Item method	**C#**
	dictionaryObj = *transactionObj*.
	GetObject(*workingDatabaseObj*.
	NamedObjectsDictionaryId, *openMode*) as Autodesk.AutoCAD.DatabaseServices.
	DBDictionary;
	dictionaryObj.GetAt function
	VB.NET
	blockTableObj = *transactionObj*.
	GetObject(*workingDatabaseObj*.BlockTableId, *openMode*)
	blockTableRecordObj = *transactionObj*.
AutoCAD.Application.ActiveDocument. ModelSpace property	GetObject(*blockTableObj*(BlockTableRecord.
	ModelSpace), *openMode*)
	C#
	blockTableObj = *transactionObj*.
	GetObject(*workingDatabaseObj*.BlockTableId, *openMode*) as Autodesk.AutoCAD.

```
DatabaseServices.BlockTable;

blockTableRecordObj = transactionObj.

GetObject(blockTableObj[BlockTableRecord.

ModelSpace], openMode) as

Autodesk.AutoCAD.DatabaseServices.

BlockTableRecord;
```

VB.NET

```
blockTableObj = transactionObj.

GetObject(workingDatabaseObj.BlockTableId, openMode)

blockTableRecordObj = transactionObj.

GetObject(blockTableObj(BlockTableRecord.

ModelSpace), openMode)

dbObj = blockTableRecordObj(index)
```

C#

```
blockTableObj = transactionObj.

GetObject(workingDatabaseObj.BlockTableId, openMode) as Autodesk.AutoCAD.

DatabaseServices.BlockTable;

blockTableRecordObj = transactionObj.

GetObject(blockTableObj[BlockTableRecord.

ModelSpace], openMode) as

Autodesk.AutoCAD.DatabaseServices.

BlockTableRecord;

foreach(objecId in blockTableRecordObj)

{

objObject = transactionObj.GetObject(objecId);

}
```

AutoCAD.Application.ActiveDocument.

ModelSpace.Item method

blockTableObj = *transactionObj*.

GetObject(*workingDatabaseObj*.BlockTableId, *openMode*)

blockTableRecordObj = *transactionObj*.

GetObject(*blockTableObj*(BlockTableRecord.

ModelSpace), *openMode*)

Dim *nCount* As Integer = 0

For Each *objectId* In *blockTableRecordObj*

nCount = *nCount* + 1

Next

C#

AutoCAD.Application.ActiveDocument.

ModelSpace.Count property

blockTableObj = *transactionObj*.

GetObject(*workingDatabaseObj*.BlockTableId,

openMode) as Autodesk.AutoCAD.

DatabaseServices.BlockTable;

blockTableRecordObj = *transactionObj*.

GetObject(*blockTableObj*[BlockTableRecord.

ModelSpace], *openMode*) as Autodesk.

AutoCAD.DatabaseServices.BlockTableRecord;

int cnt = 0;

foreach(*objectId* in *blockTableRecordObj*)

{

cnt = cnt + 1;

}

VB.NET

AutoCAD.Application.ActiveDocument.

ModelSpace.Add<entityname> method

blockTableObj = *transactionObj*.

GetObject(*workingDatabaseObj*.BlockTableId,

openMode)

blockTableRecordObj = transactionObj.

GetObject(*blockTableObj*(BlockTableRecord.

ModelSpace), *openMode*)

blockTableRecordObj.AppendEntity(*someEntity*)

transactionObj.AddNewlyCreatedDBObject(*someE*
True)

C#

blockTableObj = transactionObj.

GetObject(*workingDatabaseObj*.BlockTableId,

openMode) as Autodesk.AutoCAD.

DatabaseServices.BlockTable;

blockTableRecordObj = transactionObj.

GetObject(*blockTableObj*[BlockTableRecord.

ModelSpace], *openMode*) as Autodesk.

AutoCAD.DatabaseServices.BlockTableRecord;

blockTableRecordObj.AppendEntity(*someEntity*);

transactionObj.AddNewlyCreatedDBObject(*someE*
true);

VB.NET

blockTableRecordObj = transactionObj.

GetObject(*workingDatabaseObj*.CurrentSpaceId,
openMode)

AutoCAD.Application.ActiveDocument.

ActiveSpace property

C#

blockTableRecordObj = transactionObj.

GetObject(*workingDatabaseObj*.CurrentSpaceId,

openMode) as Autodesk.AutoCAD.

DatabaseServices.BlockTableRecord;

AutoCAD.Application.ActiveDocument.

VB.NET

PaperSpace property

blockTableObj = transactionObj.

GetObject(*workingDatabaseObj.*BlockTableId, *openMode*)

blockTableRecordObj = transactionObj.

GetObject(*blockTableObj*(BlockTableRecord.

PaperSpace), *openMode*)

C#

blockTableObj = transactionObj.

GetObject(*workingDatabaseObj.*BlockTableId, *openMode*) as Autodesk.AutoCAD.

DatabaseServices.BlockTable;

blockTableRecordObj = transactionObj.

GetObject(*blockTableObj*[BlockTableRecord.

PaperSpace], *openMode*) as

Autodesk.AutoCAD.DatabaseServices.

BlockTableRecord;

VB.NET

layoutObj = transactionObj.

GetObject(*layoutManagerObj.*

GetLayoutId(*layoutManagerObj.*

CurrentLayout), *openMode*)

AutoCAD.Application.ActiveDocument.

blockTableRecordObj = transactionObj.

ActiveLayout property

GetObject(*layoutObj.*BlockTableRecordId, *openMo*

C#

layoutObj = transactionObj.

GetObject(*layoutManagerObj.*

GetLayoutId(*layoutManagerObj.*

	CurrentLayout), *openMode*)
	as Autodesk.AutoCAD.
	DatabaseServices.Layout;
	blockTableRecordObj = transactionObj.
	GetObject(*layoutObj*.BlockTableRecordId,
	openMode) as Autodesk. AutoCAD.
AutoCAD.Application.ActiveDocument.	DatabaseServices.BlockTableRecord; HostApplicationServices.WorkingDatabase.
PurgeAll method	Purge method Autodesk.AutoCAD.ApplicationServices.Application.
AutoCAD.Application.GetVariable method	GetSystemVariable function Autodesk.AutoCAD.ApplicationServices.Application.
AutoCAD.Application.MenuBar property	MenuBar property Autodesk.AutoCAD.ApplicationServices.Application.
AutoCAD.Application.MenuGroup property	MenuGroups property Autodesk.AutoCAD.ApplicationServices.Application.
AutoCAD.Application.ActiveDocument.	DocumentManager.MdiActiveDocument.Editor.
PickfirstSelectionSet property	SelectImplied function
AutoCAD.Application.ActiveDocument.	Not provided, use the Point2d and Point3d classes from th Geometry namespace to calculate a new point
Utility.PolarPoint method AutoCAD.Application.	Autodesk.AutoCAD.ApplicationServices.Application.
Preferences property	Preferences property Autodesk.AutoCAD.ApplicationServices.Application.
AutoCAD.Application.ActiveDocument.	DocumentManager.MdiActiveDocument.Editor.
Utility.Prompt method	WriteMessage method Autodesk.AutoCAD.ApplicationServices.Application.
AutoCAD.Application.Quit method	Quit method
AutoCAD.Application.ActiveDocument.	Not needed/provided
SelectionSets.Add method AutoCAD.Application.ActiveDocument.	Autodesk.AutoCAD.EditorInput.SelectionSet.
SelectionSets.SelectionSet.Item method AutoCAD.Application.ActiveDocument.	*selectionSet*.Item(*object*) method Autodesk.AutoCAD.EditorInput.SelectionSet.
SelectionSets.SelectionSet.Delete method	*selectionSet*.Item(*object*).Delete method

AutoCAD.Application.ActiveDocument.	Autodesk.AutoCAD.ApplicationServices.Application.
SelectionSets.SelectionSet.SelectOnScreen method	DocumentManager.MdiActiveDocument.Editor. GetSelection method
AutoCAD.Application.ActiveDocument.	Autodesk.AutoCAD.EditorInput.SelectionSet.
SelectionSets.SelectionSet.Count property	*selectionSet*.Count property
AutoCAD.Application.ActiveDocument.	Autodesk.AutoCAD.ApplicationServices.Application. DocumentManager.MdiActiveDocument.Editor.
SelectionSets.SelectionSet.SelectAtPoint method	SelectCrossingWindow method
AutoCAD.Application.SetVariable method	Autodesk.AutoCAD.ApplicationServices.Application. SetSystemVariable method
AutoCAD.Application.ActiveDocument.	Not provided, use the Matrix3d class from the Geometry namespace to translate points between different coordinat systems
Utility.TranslateCoordinates method	Autodesk.AutoCAD.ApplicationServices.Application.
AutoCAD.Application.Version property	Version property Autodesk.AutoCAD.ApplicationServices.Application.
	DocumentManager.MdiActiveDocument property
	and
ThisDrawing	HostApplicationServices.WorkingDatabase property

Basic VBA and Visual Basic 6 Functions and Statements

ActiveX, VBA, or Visual Basic 6	VB.NET and C# equivalent (same unless noted)
	VB.NET
	AppActivate Autodesk.AutoCAD.
AppActivate AutoCAD.Application.Caption	ApplicationServices.Application.
function	MainWindow.Text
	VB.NET and C#
	Use the Win32 Calls ShowWindow and SetWindowPos
Dir function	System.IO.Directory.Exists function
	VB.NET
	Error object/method/properties
Error object/method/properties	
	VB.NET and C#
	Try Catch statement with exception handling
	VB.NET
	Function and End Function keywords and use *Return* to return a value
Function and End Function keywords	
	C#
	Define a procedure and use *return* to return a value
	VB.NET
	Input method
Input function	
	VB.NET and C#
	*fileStream.*Read method
	VB.NET
	LBound(*arrayName*) function
LBound(*arrayName*) function	
	VB.NET and C#
	*arrayName.*GetLowerBound function
	VB.NET
Line Input function	LineInput method
	VB.NET and C#

MsgBox function	*fileStream*.Read method MessageBox.Show method **VB.NET**
object(n) syntax	object(n) syntax **C#**
Open function	object[n] syntax System.IO.File.Open function **VB.NET**
ReDim statement	ReDim *arrayName*(*newSize*) **VB.NET and C#**
Set statement	*arrayName*.Resize Not needed/provided **VB.NET**
Shell function	Shell function **VB.NET and C#**
	System.Diagnostics.Process.Start function **VB.NET**
Sub and End Sub keywords	Sub and End Sub keywords **C#**
	Define a procedure **VB.NET**
TypeName function	TypeName function **VB.NET and C#**
	varName.GetType().Name or *varName*.GetType().FullName functions **VB.NET**
UBound(*arrayName*) function	UBound(*arrayName*) function **VB.NET and C#**
	arrayName.GetUpperBound function

Index