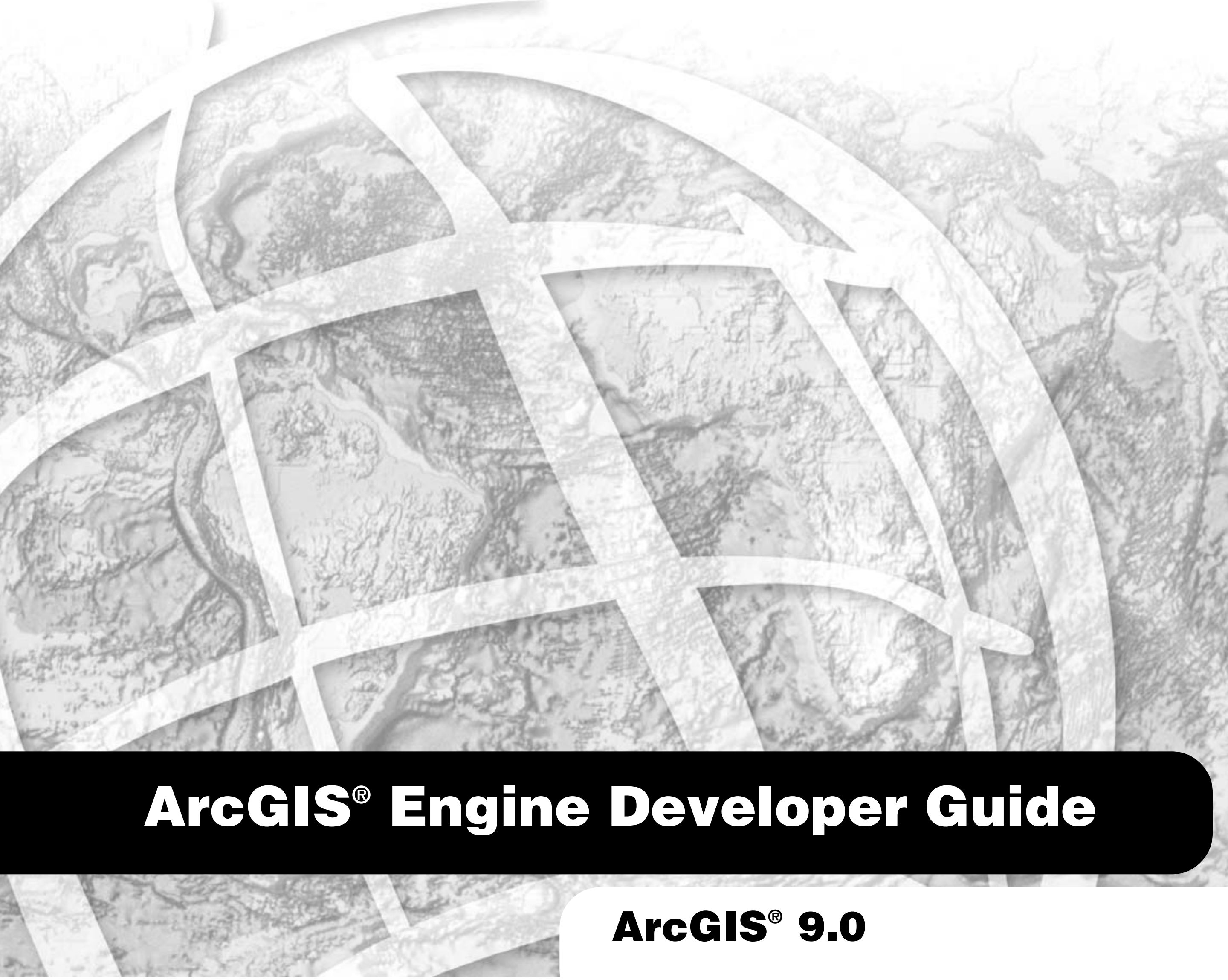


ArcGIS® Engine Developer Guide

ArcGIS® 9.0

PUBLISHED BY
ESRI
380 New York Street
Redlands, California 92373-8100

The information contained in this document is subject to change without notice.

Contributing Writers
Euan Cameron, Chris Davies, Rob Elkins, Kylie Evans, Anne Frankland, Shelly Gill, Sean Jones, Allan Laframboise, Glenn Meister, Dan O'Neill, Rohit Singh, Steve Van Esch, Zhiqian Yu

Contents

1

Introducing ArcGIS Engine

ESRI® ArcGIS® Engine is a platform for building custom stand-alone geographic information systems (GIS) applications which support multiple application programming interfaces (APIs), include advanced GIS functionality, and are built using industry standards.

This chapter will introduce you, the developer, to the ArcGIS Engine developer kit and the ArcGIS Engine Runtime, discussing how to use it and its different components.

Topics covered in this chapter include:

• an overview of ArcGIS 9 • introduction to ArcGIS Engine • ArcGIS Engine users • capabilities of ArcGIS Engine • a description of this book

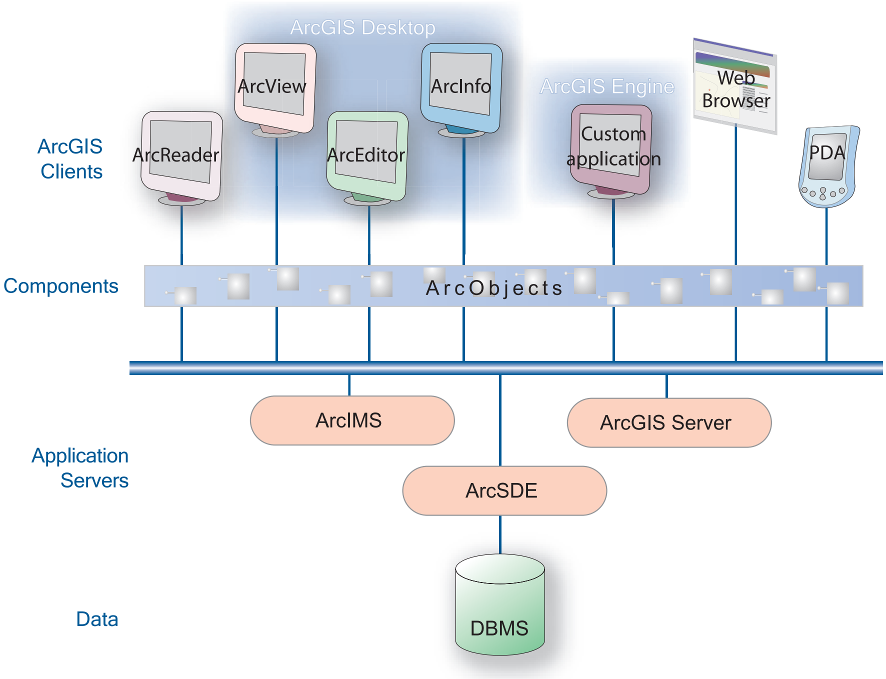
Developers wishing to customize ArcGIS Desktop applications or work with ArcGIS Server should refer to the ArcGIS Desktop Developer Guide *and the* ArcGIS Server Administrator and Developer Guide.

ArcGIS provides a scalable framework for implementing GIS for a single user or for many users on desktops and servers. This book focuses on building and deploying custom applications using ArcGIS Engine. It will be of greatest use to developers who want to embed mapping and GIS functionality in custom applications. It provides an overview of ArcGIS Engine, its components, and the possibilities ArcGIS Engine offers developers who wish to build and deploy custom GIS applications and solutions. In addition, several scenarios are used to illustrate, with code examples, the various types of applications that can be developed with ArcGIS Engine.

AN OVERVIEW OF ArcGIS 9

ArcGIS 9 is an integrated family of GIS software products for building a complete GIS. It is based on a common library of shared GIS software components called ArcObjects. ArcGIS 9 consists of four key parts:

- ArcGIS Desktop—an integrated suite of advanced GIS applications.
- ArcGIS Engine—embeddable GIS component libraries for building custom applications using multiple application programming interfaces.
- ArcGIS Server—a platform for building server-side GIS applications in enterprise and Web computing frameworks. Used for building both Web services and Web applications.
- ArcIMS—GIS Web server to publish maps, data, and metadata through open Internet protocols.



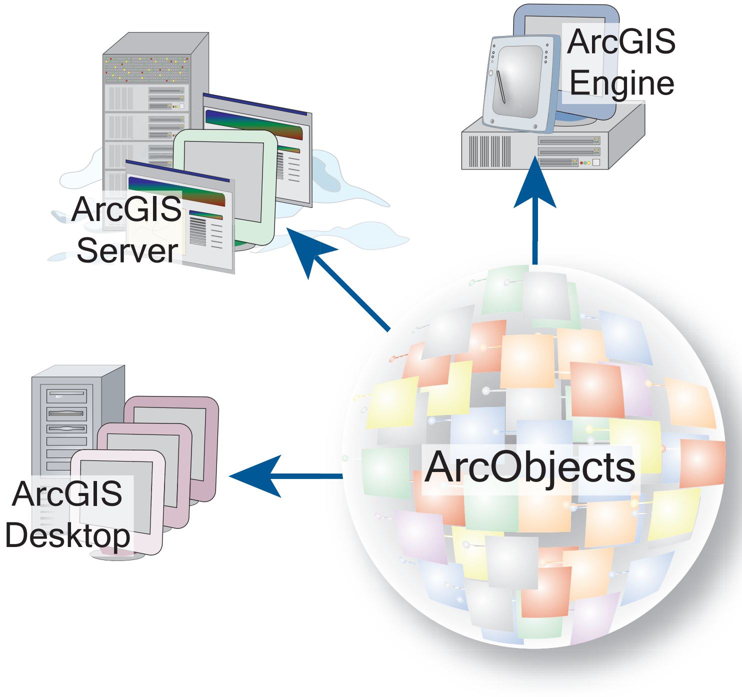
Each of the GIS frameworks also includes the ArcSDE® gateway, an interface for managing geodatabases in numerous relational database management systems (RDBMS).

ArcGIS is a platform for building geographic information systems. ArcGIS 9 extends the system with major new capabilities in the areas of geoprocessing, 3D visualization, and developer tools. Two new products, ArcGIS Engine and ArcGIS Server, are introduced at this release, making ArcGIS a complete system for application and server development.

There is a wide range of possibilities when developing with ArcGIS. Developers can:

- Configure/Customize ArcGIS applications such as ArcMap™ and ArcCatalog™.
- Extend the ArcGIS architecture and data model.
- Embed maps and GIS functionality in other applications with ArcGIS Engine.
- Build and deploy custom desktop applications with ArcGIS Engine.
- Build Web services and applications with ArcGIS Server.

The ArcGIS system is built and extended using software components called ArcObjects™. ArcObjects includes a wide variety of programmable components ranging from fine-grained objects, such as individual geometry objects, to coarse-grained objects, such as a map object that can be used to interact with existing ArcMap documents. These components aggregate comprehensive GIS functionality for developers.

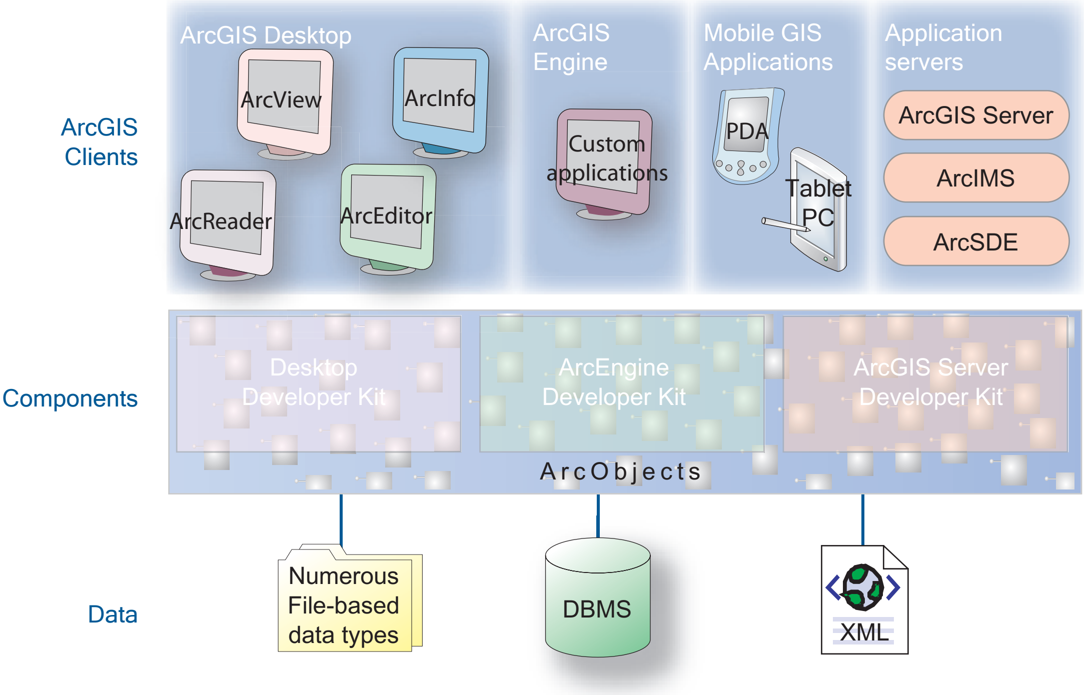


ArcGIS 9 has a common developer experience across all ArcGIS products (Engine, Server, and Desktop). You, as a developer, can work with ArcObjects using standard programming frameworks to extend ArcGIS Desktop, build custom applications with ArcGIS Engine, and to implement enterprise GIS applications using ArcGIS Server.

As noted previously, this book focuses on building and deploying custom applications using ArcGIS Engine. If you wish to customize ArcGIS Desktop applications or work with ArcGIS Server, refer to the *ArcGIS Desktop Developer Guide* and the *ArcGIS Server Administrator and Developer Guide*.

The ArcGIS system is available in a number of programming frameworks including: C++, COM, .NET, and Java™.

Each of the ArcGIS product architectures built with ArcObjects represents alternative application development containers for GIS software developers, including desktops, embeddable engines, and servers.



ArcGIS Desktop includes a series of Windows® desktop application frameworks (for example, applications for map, catalog, toolbox, and globes) with user interface components. ArcGIS Desktop is available at three functional levels (ArcView®, ArcEditor™, and ArcInfo™) and can be customized and extended using the ArcGIS Desktop developer kit.

The software developer kit (SDK) for ArcGIS Desktop is included with ArcView, ArcEditor, and ArcInfo and supports the COM and .NET programming frameworks. Many developers apply the Desktop SDK to add extended functions, new GIS tools, custom user interfaces, and full extensions for improving professional GIS productivity of the ArcGIS Desktop applications.

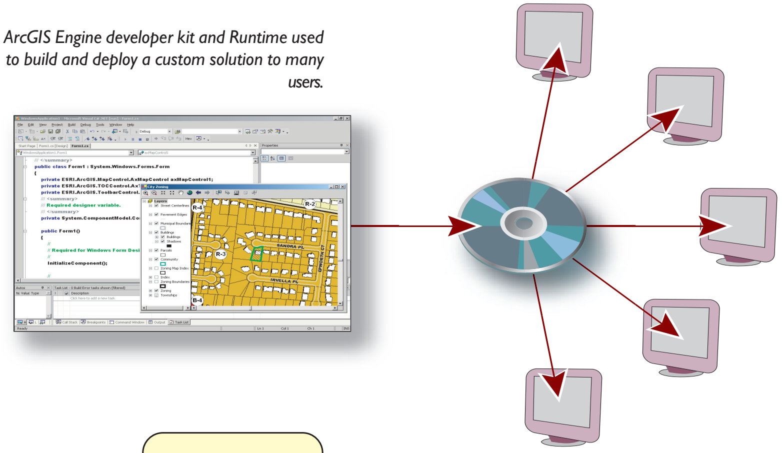
ArcGIS Server defines and implements a set of standard GIS Web services (for example, mapping, data access, data access, geocoding, and so on) as well as supporting enterprise-level application development based on ArcObjects for the server.

The ArcGIS Server developer kit enables developers to build central GIS servers to host GIS functions that are accessed by many users, perform back office processing on large central GIS databases, build and deliver GIS Web applications, and to perform distributed GIS computing.

ArcGIS Engine, the focus of this guide, is a simple, application-neutral programming environment for ArcObjects. Its SDK provides a series of embeddable ArcGIS components that are used outside of the ArcGIS Desktop application framework (for example, mapping objects are managed as a part of ArcGIS Engine, rather than in ArcMap). Using the ArcGIS Engine developer kit, developers can build focused GIS solutions with simple interfaces to access any set of GIS functions or embed GIS logic in existing user applications in order to deploy GIS to broad groups of users.

ArcGIS Engine and its developer kit will be discussed in more detail later in this chapter and throughout this book.

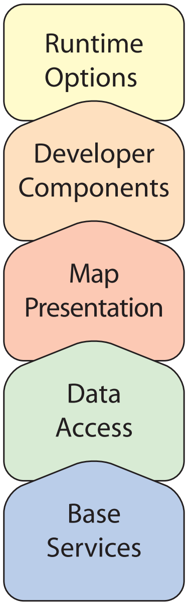
ArcGIS Engine is a complete library of embeddable GIS components for developers to build custom applications. Using ArcGIS Engine, you can embed GIS functions into existing applications, including Microsoft® Office products like Word and Excel, and build focused custom applications that deliver advanced GIS systems to many users. ArcGIS Engine consists of a software development kit and a re-distributable runtime providing the platform for all ArcGIS applications.



ArcGIS Engine developer kit and Runtime used to build and deploy a custom solution to many users.

The five parts of ArcGIS Engine are outlined below:

1. Base Services—The core GIS ArcObjects required for almost any GIS application such as feature geometry and display.
2. Data Access—ArcGIS Engine provides access to a wide variety of raster and vector formats including the power and flexibility of the geodatabase.
3. Map Presentation—ArcObjects for map creation and display with symbology, labeling, and thematic mapping capabilities including custom applications.
4. Developer Components—High-level user interface controls for rapid application development and a comprehensive help system for effective development.
5. Runtime Options—ArcGIS Engine Runtime is deployable with the standard functionality or with additional options for advanced functionality.



Components of ArcGIS Engine

Each of these parts, except runtime options, is made available through the ArcGIS Engine SDK The ArcGIS Engine Runtime and its Options, although integral factors in the development of a custom GIS application, specifically involve application deployment, and so are considered separately.

ARCGIS ENGINE SOFTWARE DEVELOPER KIT

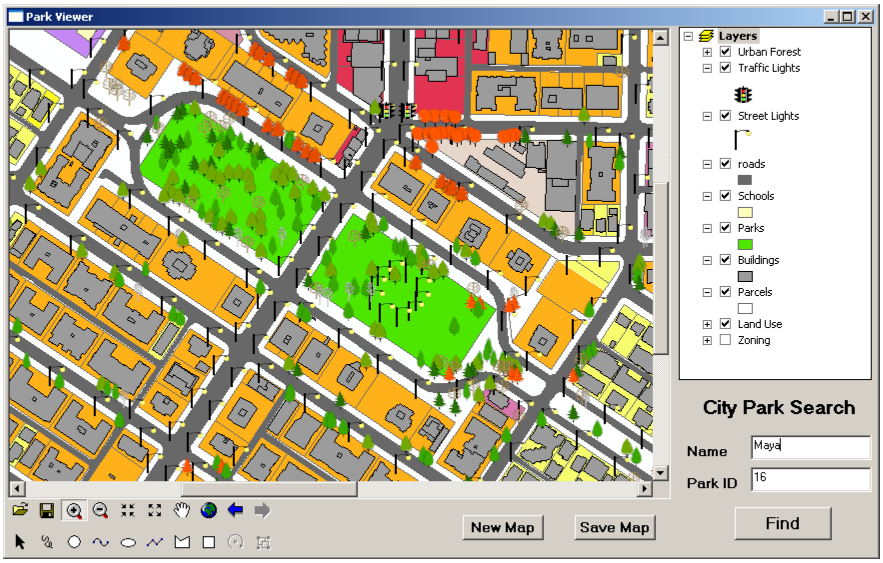
The ArcGIS Engine developer kit is a component-based software development product for building and deploying custom GIS and mapping applications. The ArcGIS Engine developer kit is not an end user product, but rather a toolkit for application developers. It can be used to build basic map viewers or comprehensive and dynamic GIS editing tools. With the ArcGIS Engine developer kit, you, as a developer, have an unprecedented flexibility for creating customized interfaces for maps. You can use one of several APIs to create unique applications or combine ArcGIS Engine components with other software components to realize a synergistic relationship between maps and the information that users manage.

Using ArcGIS Engine, the map itself can be either an incidental element within or the central component of an application. If, for example, the focus of your application is a database with information about businesses, ArcGIS Engine can enable the application to display a form with a map highlighting the business location of interest when your user performs a query on the database.

The ArcGIS Engine developer kit provides access to a large collection of GIS components, or ArcObjects, that fall into the categories discussed earlier—base services, data access, and map presentation. The fourth part of ArcGIS Engine that was discussed, developer components, is also included in the SDK. These are value-added developer controls for creating a high-quality map user interface. The following ArcGIS Controls, or visual components, are provided to assist with application development:

Chapter 3, 'Developing with ArcGIS Controls', discusses each of these visual components in detail.

- MapControl
- PageLayoutControl
- SceneControl
- GlobeControl
- ToolbarControl
- TOCControl
- ReaderControl
- Collection of commands, tools, and menus for use with the ToolbarControl



☑ Urban Forest
☑ Traffic Lights
☑ Street Lights
☑ roads
☑ Schools
☑ Parks
☑ Buildings
☑ Parcels
☑ Land Use
☐ Zoning

City Park Search

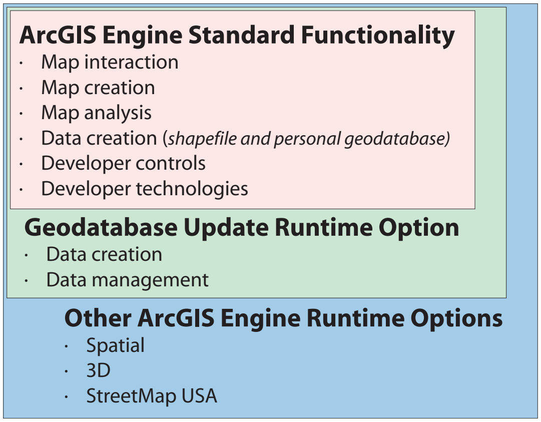
Name: Maya

Park ID: 16

An ArcGIS Engine Controls-based application

ArcGIS Engine Runtime

The final component of ArcGIS Engine is its runtime options. All applications built with the ArcGIS Engine SDK require the ArcGIS Engine Runtime, with appropriate level of license, in order to execute successfully. The ArcGIS Engine Runtime is the platform on which ArcGIS Desktop is built; this allows users of ArcGIS Desktop applications to execute custom Engine-based applications, if permitted by the ArcGIS Engine application developer. There are multiple ArcGIS Engine Runtime Options ranging from standard to enterprise options.



ArcGIS Engine Standard Functionality
- Map interaction
- Map creation
- Map analysis
- Data creation (*shapefile and personal geodatabase)*
- Developer controls
- Developer technologies

Geodatabase Update Runtime Option
- Data creation
- Data management

Other ArcGIS Engine Runtime Options
- Spatial
- 3D
- StreetMap USA

ArcGIS Engine Runtime deployment options

Standard ArcGIS Engine functionality

The standard Engine Runtime provides the core functionality of all ArcGIS applications. This level of ArcGIS Engine Runtime provides the ability to work with several different raster and vector formats. Map presentation and creation along with the ability to explore features by performing a wide range of spatial or attribute searches. This level also makes available basic data creation, editing of shapefiles and simple personal geodatabases, and GIS analysis.

Geodatabase Update Option

The Geodatabase Update Option for the ArcGIS Engine Runtime adds the ability to create and update a multiuser enterprise geodatabase. This includes the ability to work with schemas and versioned geodatabases. The Geodatabase Update Option unlocks the ArcGIS Engine Runtime with the necessary ArcObjects to run custom solutions. These solutions include applications that deal with GIS data automation and compilation and the construction and maintenance of geodatabase features. The Geodatabase Update Option provides the ability to programmatically create geodatabase behaviors such as topologies, subtypes, and geometric networks.

ArcGIS Engine developers with access to an RDBMS via ArcSDE are able to build and deploy multiuser editing applications to endusers that have the ArcGIS Engine Runtime with the Geodatabase Update Option installed and configured.

The availability of the different levels of functionality is controlled by a software authorization file that can be configured by the end user or the developer of the application. For more details on deploying and configuring the ArcGIS Engine Runtime, refer to Chapter 5, 'Licensing and deployment'.

Other ArcGIS Engine options

Three additional runtime options are available for the ArcGIS Engine Runtime:

1. Spatial Option—The ArcGIS Engine Spatial Option provides a powerful set of functions that allow applications to create, query, and analyze cell-based raster data. This type of analysis allows your user to derive information about their data, identify spatial relationships, find suitable locations, and calculate the accumulated cost of traveling from one point to another. Other advanced applications that this runtime option supports include the calculation of slope, aspect, and contours against digital elevation models.
2. 3D Option—The 3D Option for ArcGIS Engine Runtime enables the visualization of data in 3D. This option supplements standard ArcGIS Engine with the components for viewing a surface from multiple viewpoints and determin-

ing what is visible from a chosen location. The SceneControl and GlobeControl provide the interface for viewing multiple layers of 3D and global data for visualizing data, creating surfaces, and analyzing surfaces.

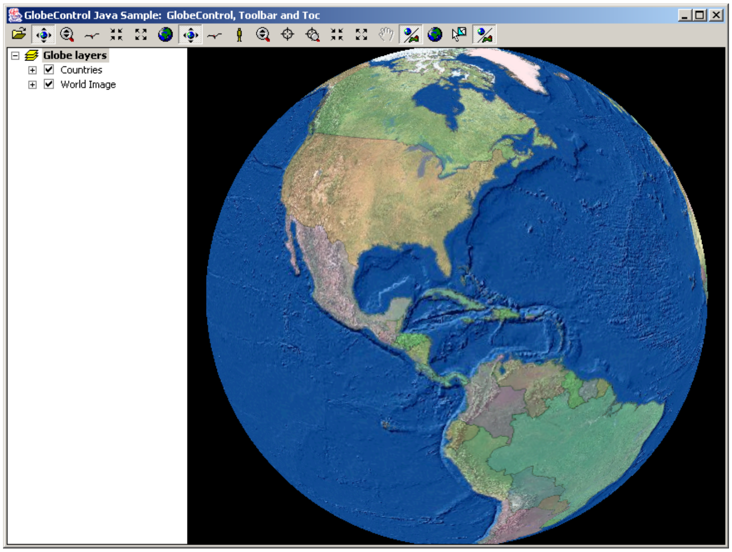
3. StreetMap™ USA Option—The StreetMap Runtime Option provides street-level mapping, address matching, and basic routing for the United States. StreetMap layers automatically manage, label, and draw features such as local landmarks, streets, parks, water bodies, and other features resulting in rich cartographic street network for the US. All data is provided in a compressed format on CD–ROM.

Many users require focused, lightweight access to GIS. They need much less than a complete GIS application such as ArcView, yet require access to sophisticated GIS logic in their applications. In cases where users need focused, customized access to GIS, ArcGIS Engine provides a lower-cost, lightweight option.

STAND-ALONE APPLICATION DEVELOPERS

There are many potential users of GIS-enhanced applications who are not GIS professionals and are just not equipped to take advantage of the comprehensive tools available on the market without a steep learning curve. In order to provide spatial solutions to non-GIS users, developers need the ability to build domain specific, easy-to-use applications that can incorporate the power of a comprehensive GIS system into a user-friendly experience. These applications, if built from scratch, can be an overwhelming development effort and may not be time or cost effective.

You can use the ArcGIS Engine developer kit to successfully build stand-alone applications. There is a wide variety of types of applications that can be built, ranging from graphical user interface (GUI) applications to command line, batch driven applications. GUI applications will make use of the extensive ArcGIS Controls exposed in the developer kit. These controls include everything you need to build a sophisticated front end application. You can leverage your chosen API to integrate the ArcGIS Controls with other third-party components and create a unique user interface for your custom ArcGIS Engine application.



An application built in Java using the GlobeControl

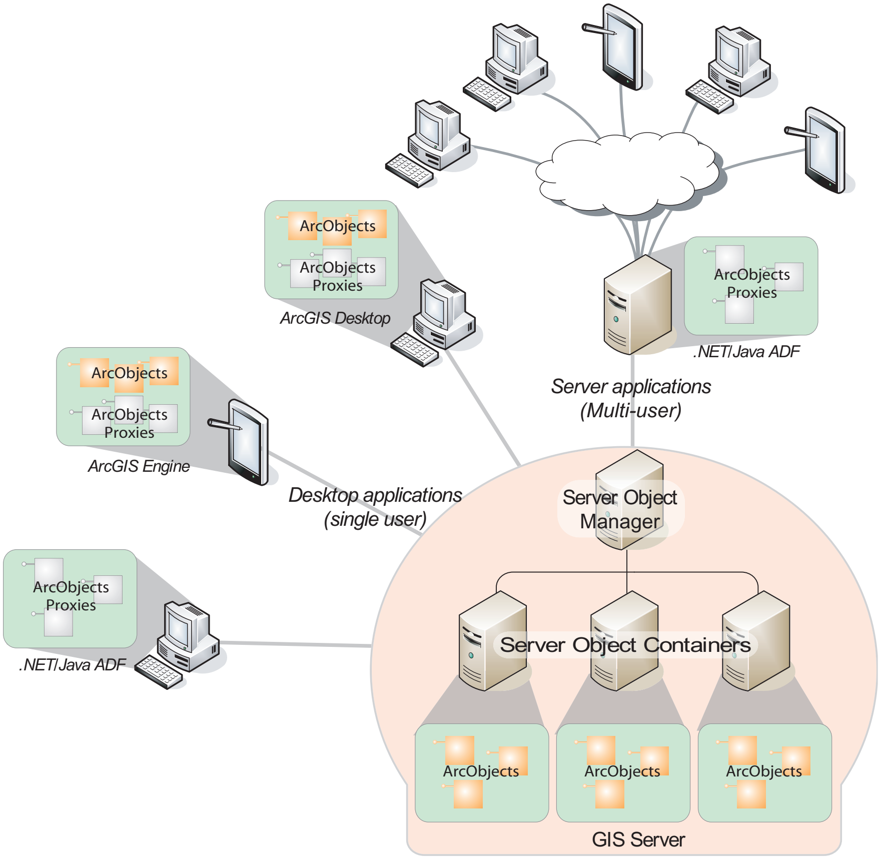
ArcGIS DESKTOP USERS

ArcMap, one of the ArcGIS Desktop applications, is an excellent way to create data and author maps for use in custom applications. The MapControl and PageLayoutControl provided with ArcGIS Engine can work with the map documents created in ArcMap. The SceneControl and GlobeControl can display documents authored in the ArcScene™ and ArcGlobe™ applications. Using the ArcGIS Desktop applications to create and manage maps used in custom applications can save you a lot of development and effort. ArcGIS Desktop also provides tools to build and manage geodatabases, shapefiles, and other forms of spatial data.

The underlying components of ArcGIS Desktop are the same ArcObjects components that make up ArcGIS Engine. This allows every ArcGIS Desktop user the ability to run ArcGIS Engine applications. You can develop and deploy Engine-based applications to ArcGIS Desktop users or extend ArcToolbox™ with a custom toolset built with the ArcGIS Engine developer kit.

ARCGIS SERVER USERS

ArcGIS Server administrators can provide server objects and Web services to ArcGIS Engine applications. This allows the integration of desktop functionality with server functionality.



The capabilities of ArcGIS Engine are extensive. As an ArcGIS Engine developer, you can implement these and many other functions using its developer kit:

The items listed at right, if deployed, are included in the standard ArcGIS Engine Runtime functionality and would not require any of the additional Runtime Options.

- Display a map with multiple map layers such as road, streams, and boundaries.
- Pan and zoom throughout a map.
- Identify features on a map.
- Search for and find features on a map.
- Display labels with text from field values.
- Draw images from aerial photography or satellite imagery.
- Draw graphic features such as points, lines, circles, and polygons.
- Draw descriptive text.
- Select features along lines and inside boxes, areas, polygons, and circles.
- Select features within a specified distance of other features.
- Find and select features with a Structured Query Language (SQL) expression.
- Render features with thematic methods such as value map, class breaks, and dot density.
- Dynamically display real time or time series data.
- Find locations on a map by geocoding addresses or street intersections.
- Transform the coordinate system of your map data.
- Perform geometric operations on shapes to create buffers; calculate differences; and find intersections, union, or inverse intersections of shapes.
- Manipulate the shape or rotation of a map.
- Create and update geographic features and their attributes.

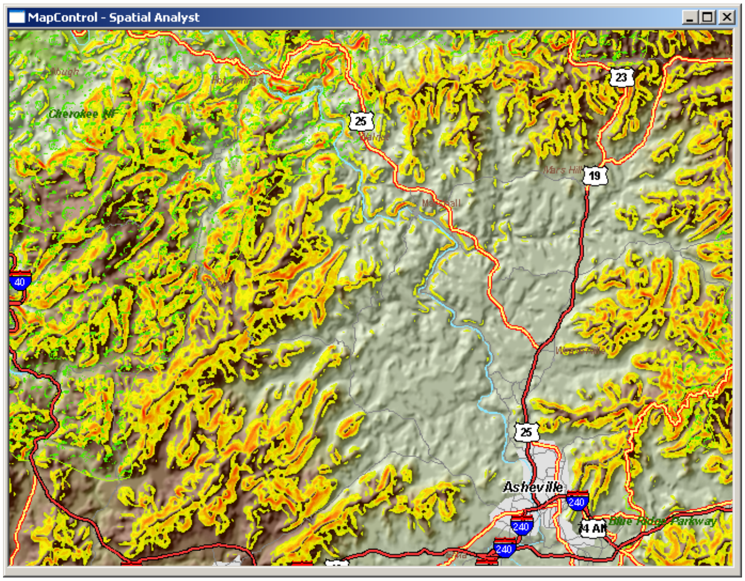
EDITING FEATURES

A software authorization file controls the availability of the various levels of ArcGIS Engine Runtime functionality. For more details on deploying and configuring the ArcGIS Engine Runtime, refer to Chapter 5, 'Licensing and deployment'.

ArcGIS Engine developer kit enables you to build applications that create, modify, and remove vector-shaped features in a geodatabase or shapefile. The standard ArcGIS Engine Runtime is used to run applications that edit shapefiles or the simple features of a personal geodatabase. However, to leverage the full function of the enterprise geodatabase, the Geodatabase Update Option of the ArcGIS Engine Runtime is required.

SPATIAL MODELING AND ANALYSIS

You can extend the capabilities of ArcGIS Engine by adding the Spatial Option to the ArcGIS Engine Runtime. This option provides a broad range of powerful spatial modeling and analysis functions. You can create, query, map, and analyze cell-based raster data; perfom integrated raster/vector analysis; derive new information from existing data; query information across multiple data layers; and fully integrate cell-based raster data with vector data in a custom ArcGIS Engine application.



An application, developed using the MapControl, that utilizes the Spatial Option for the ArcGIS Engine Runtime

For example, you can:

- Convert features (points, lines, or polygons) to rasater
- Create raster buffers based on distance or proximity from freatures or rasters.
- Generate density maps from points features.
- Derive contours, slope, view shed, aspect, and hillshades
- Perform gird classification and display
- Use data from standard formats including TIFF, BIL, IMG, USGS DEM, SDTS, DTED, and many others.

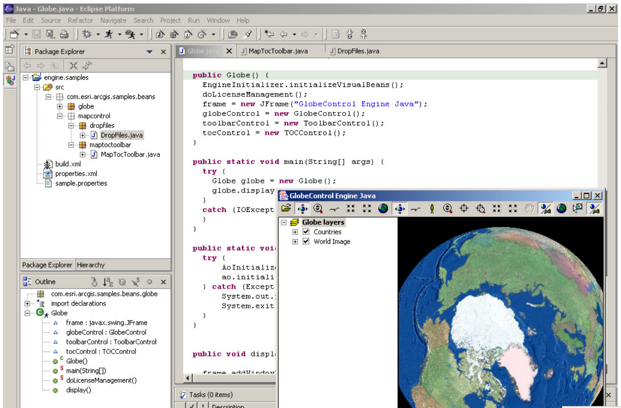
3D visualization and more

The ArcGIS Engine Runtime 3D Option extends the capabilities of ArcGIS Engine even further by enabling you to build applications that effectively visualize and analyze surface and globe data using the Scene and Globe controls. You can create applications that view a surface from multiple viewpoints, query a surface, determine what is visible from a chosen location on a surface, and display a realistic perspective image by draping raster and vector data over a surface.

You can, for example:

- Display Scene and Globe documens
- Perform interactive perspective viewing, including pan and zoom, rotate, tilt, and fly-through simulations, for presentation and analysis.
- Display real-world surface features such as buildings
- Perform viewshed and line-of-sight analysis, spot height interpolation, profiling, and steepest path determination.

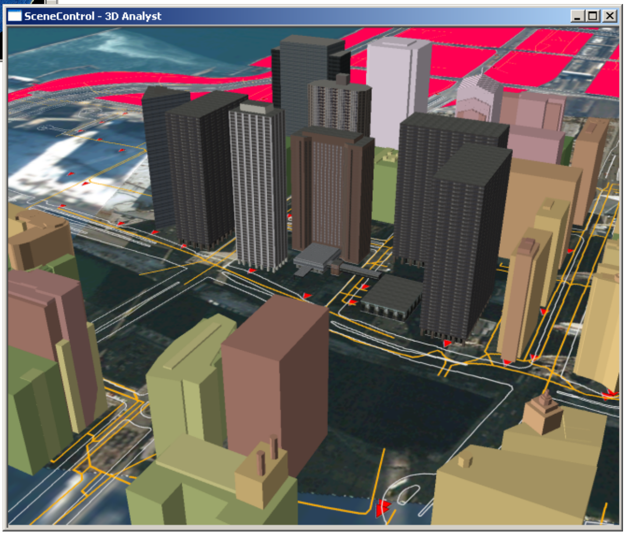
Java code for the inset GlobeControl-based application



```
engine.samples
  src
    com.esri.arcgis.samples.beans
      globe
      mapcontrol
        dropfiles
          DropFiles.java
        maptoctoolbar
          MapTocToolbar.java
  build.xml
  properties.xml
  sample.properties
```

```
com.esri.arcgis.samples.beans.globe
import declarations
Globe
  frame : javax.swing.JFrame
  globeControl : GlobeControl
  toolbarControl : ToolbarControl
  tocControl : TOCControl
  Globe()
  main(String[])
  doLicenseManagement()
  display()
```

```
public Globe() {
  EngineInitializer.initializeVisualBeans();
  doLicenseManagement();
  frame = new JFrame("GlobeControl Engine Java");
  globeControl = new GlobeControl();
  toolbarControl = new ToolbarControl();
  tocControl = new TOCControl();
}

public static void main(String[] args) {
  try {
    Globe globe = new Globe();
    globe.display
  }
  catch (IOExcept
  }
}

public static voi
  try {
     AoInitializ
     ao.initiali
  } catch (Except
     System.out.
     System.exit
  }
}


public void displ

  frame.addWindow
```



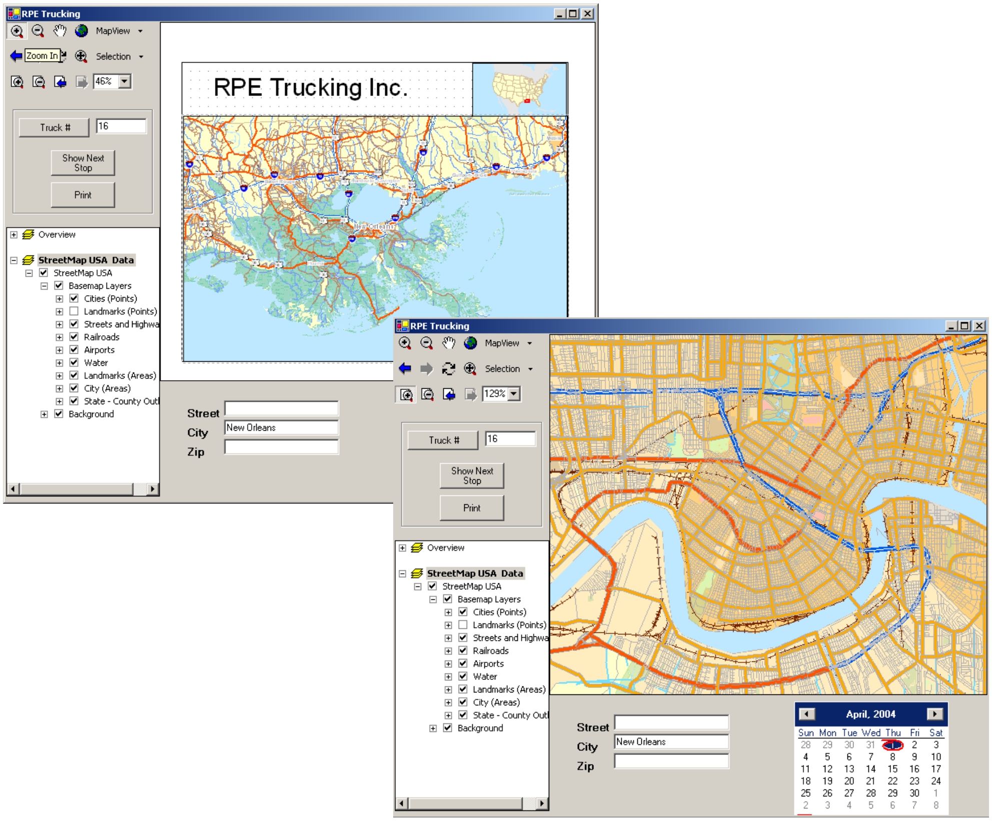
Display of a SceneControl-based application

Street-level data

StreetMap USA provides detailed street data for the entire United States. You can include and use this data in your ArcGIS Engine application through the use of the StreetMap USA Runtime Option. With this option, you can use StreetMap data sources just like any other feature class dataset. The StreetMap USA Option extends the underlying geodatabase object model within ArcObjects so that your applications can seamlessly use StreetMap data sources as geodatabase objects.

This option provides:

- Nationwide address matching
- Street and landmark database
- StreetMap group layer that displays different levels of details at different scales
- Basic street level routing



Zoom: 46%

Truck # 16

Street

City New Orleans

Zip

RPE Trucking

Zoom: 129%

Truck # 16

Street

City New Orleans

Zip

April, 2004

Sun	Mon	Tue	Wed	Thu	Fri	Sat
28	29	30	31	1	2	3
4	5	6	7	8	9	10
11	12	13	14	15	16	17
18	19	20	21	22	23	24
25	26	27	28	29	30	1
2	3	4	5	6	7	8

Once you have the ArcGIS Engine developer kit installed, you are ready to start developing ArcGIS Engine applications. However, good applications require careful planning; working with ArcObjects is no exception. Before beginning your development, feel free to read through and use, as necessary, the dicussions and checklists in this section. They are provided in order to help you formulate your plans and ensure you're getting started on the right foot.

DETERMINE THE TYPE OF APPLICATION

Some examples of ArcGIS Engine applications are provided in Chapter 6, 'Developer scenarios'. Additional samples are included with the ArcGIS Developer Help system.

A wide variety of applications can be developed with ArcGIS Engine. These applications vary from simple consoles that perform operations such as database editing and analyses, to more complex Windows applications that contain controls and visual components for user interaction and geographic data display. In general there are three types of ArcGIS Engine applications:

1. Stand-alone, non-visual applications, such as console and utility applications
2. Stand-alone, visual applications, such as Windows and control-based applications
3. Embedded applications, such as components that are inserted into existing applications

Ultimately, the type of application you develop will depend on the functional requirements of the project at hand.

Checklist:

□ What type of application are you developing? Non-visual, visual, or embedded?

□ Do you plan to migrate the functionality to ArcGIS Desktop or ArcGIS Server products?

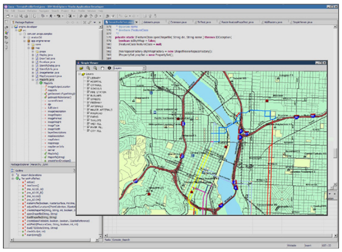
□ What platform do you want to support now and into the future? Windows? UNIX®? Both?

CHOOSE AN API AND DEVELOPMENT ENVIRONMENT

Since ArcGIS Engine provides four developer APIs—COM, .NET, Java, and C++—you can use any development environment that supports these APIs for application development. For non-visual applications, common language choices include C++ and Java. For visual applications, many languages are available with Windows capabilities such as Visual Basic® (VB) 6, C#.NET, Java, and Visual C++® (VC++). Below are some possible APIs, development environments, and languages for ArcGIS Engine.

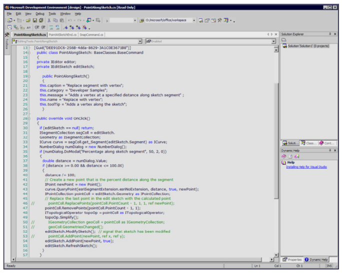
- COM—Visual Studio® 6.0 (VB6, VC++)
- .NET—Visual Studio .NET (VB.Net, C#)
- Java—Eclipse, Websphere Studio, Intelli J, JBuilder™, and so on.
- C++—Visual Studio 6.0, C++ Builder

The environment you choose to develop with will ultimately depend on your programming skills, the functionality you wish to provide end users, and whether



```
 * @param name
 * @return IFeatureClass
 */
private static IFeatureClass openShapefile( String dir, String name ) throws IOException{
    boolean isShpWksp = false;
    IFeatureClass featureClass = null;

    IWorkspaceFactory shpWkspFactory = new ShapefileWorkspaceFactory();
    IPropertySet propSet = new PropertySet();
```

Websphere Studio



Microsoft Development Environment [design] - PointAlongSketch.cs [Read Only]

```
[Guid("DEE91DC6-256B-4dda-8629-3A1C0E3671B8")]
public class PointAlongSketch: BaseClasses.BaseCommand
{
private IEditor editor;
private IEditSketch editSketch;

  public PointAlongSketch()
  {
  this.caption = "Replace segment with vertex";
  this.category = "Developer Samples";
  this.message = "Adds a vertex at a specified distance along sketch segment" ;
  this.name = "Replace with vertex";
  this.toolTip = "Adds a vertex along the sketch";
  }

public override void OnClick()
{
  if (editSketch == null) return;
  ISegmentCollection segColl = editSketch.
  Geometry as ISegmentCollection;
  ICurve curve = segColl.get_Segment(editSketch.Segment) as ICurve;
  NumberDialog numDialog = new NumberDialog();
  if (numDialog.DoModal("Percentage along sketch segment", 50, 2, 0))
  {
    double distance = numDialog.Value;
    if (distance >= 0.00 && distance <= 100.00)
    {
      distance /= 100;
      // Create a new point that is the percent distance along the segment
      IPoint newPoint = new Point();
      curve.QueryPoint(esriSegmentExtension.esriNoExtension, distance, true, newPoint);
      IPointCollection pointColl = editSketch.Geometry as IPointCollection;
      // Replace the last point in the edit sketch with the calculated point
//      pointColl.ReplacePoints(pointColl.PointCount - 1, 1, 1, ref newPoint);
      pointColl.RemovePoints(pointColl.PointCount - 1, 1);
      ITopologicalOperator topoOp = pointColl as ITopologicalOperator;
      topoOp.Simplify();
//      IGeometryCollection geoColl = pointColl as IGeometryCollection;
//      geoColl.GeometriesChanged();
      editSketch.ModifySketch();  // signal that sketch has been modified
//      pointColl.AddPoint(newPoint, ref x, ref y);
      editSketch.AddPoint(newPoint, true);
      editSketch.RefreshSketch();
    }
  }
```

Visual Studio .NET

Throughout most of this book, VB 6 is used as the language to illustrate most coding concepts and is often the easiest language to learn when getting started. See Chapter 4, 'Developer environments', for programming guidelines for VB and some of the other environments supported by the ArcGIS Engine APIs.

or not you are integrating with other existing applications or technologies.

Checklist:

□ What development environment and language are you the most familiar with?

□ Which ArcGIS Engine API do you plan to use?

□ Which development environment and language is best suited for the type of the development you wish to undertake?

DEVELOP YOUR APPLICATION

At this point, assuming that a proper project development plan is in place, you are ready to dive into the ArcGIS Engine SDK and start developing your application. You may want to start by identifying the libraries and objects that will be necessary to provided the functionality for the application. Use the developer help resources to assist you in this process, including the ArcGIS Developer Help system, the Developer Guide series, samples included in the help system, and the ArcGIS Developer Online site.

Each functional group of ArcObjects, or library, used must be referenced in your development environment for your application to compile and run successfully. The various libraries available in ArcGIS Engine are discussed in detail in Chapter 2, 'ArcGIS software architecture'.

ArcGIS Developer Online can be accessed from http://arcgisdeveloperonline.esri.com.

Checklist:

□ Identify the ArcObjects functionality required.

□ What ArcGIS Engine library references will be required?

□ What ArcGIS license will be required to run the application?

□ Are ArcGIS Engine Runtime Options required?

□ How do you plan to deploy the application?

□ Have you implemented the correct license check-out code?

DEPLOY YOUR APPLICATION

Application deployment is an issue that should be considered long before application development begins. ArcGIS Engine applications can be deployed in a number of ways and it is possible to have a number of end-user software and license configurations. Therefore, there are a number of issues that you need to consider.

Checklist:

□ Will they have ArcGIS Desktop installed or the ArcGIS Engine Runtime or both? Which license will your application check-out?

□ What ArcGIS license will your end-users have on their systems? ArcInfo, ArcEditor or ArcView?

Chapter 5, 'Licensing and deployment', discusses the various aspects of this checklist.

□ How should you package and deploy the application?

□ Will I need to provide new versions in the future?

□ How will I distribute the application?

This book, *ArcGIS Engine Developer Guide*, is an introduction for developers who want to build stand-alone GIS applications. This guide will help you, as the developer, become familiar with the ArcGIS Engine object model by introducing all of the ArcGIS Engine developer kit components, discussing relevant aspects of building applications, introducing supported APIs, and providing developer scenarios which produce real-world GIS applications.

To serve the widest base of developers, most of the code samples provided within this book use the COM Visual Basic 6 API. However, the developer scenarios cover the full range of supported APIs and a chapter is devoted to API-specifc usages.

The first two chapters of this book provide an overview of ArcGIS Engine and its capabilities, including architecture and components. The remaining chapters focus on developing applications usages of each particular supported API.

CHAPTER GUIDE

Chapter 1, 'Introducing ArcGIS Engine', gives developers an overview of the ArcGIS Engine product, it's capabilities, and developer resources.

Chapter 2, 'ArcGIS software architecture', describes ArcGIS Engine architecture and how the software components interact inside the system.

'Developing with ArcGIS Controls' is detailed in Chapter 3. It describes each of the controls and provides some considerations for their use in application development.

Chapter 4, 'Developer environments', introduces you to the multiple APIs supported by ArcGIS Engine. This chapter steps through each API from the basics to advanced usage topics.

'Licensing and deployment', issues are addressed in Chapter 5. It details the licensing options, discusses deployment strategies for your application, including initialization and license checking.

Chapter 6, 'Developer scenarios', guides you through the creation and deployment of several types of stand-alone applications utilixing each of the supported APIs

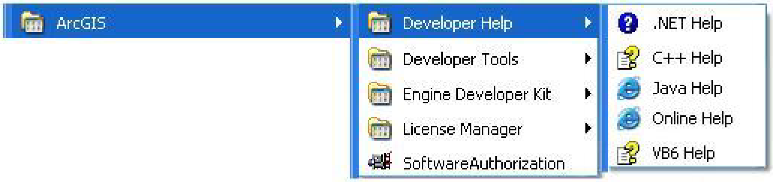
This book also contains a number of appendicies that provide detailed information about the object model diagrams available in the ArcGIS Developer Help system and additional developer resources.

The following topics describe some of the additional resources available to ArcGIS developers. More in-depth coverage on the resources available to developers is covered in Appendix B.

ARCGIS DEVELOPER HELP SYSTEM

The ArcGIS Developer Help system is an essential resource for both the beginning and experienced ArcObjects developers. It contains information on developing with ArcObjects including sample code, technical documents, and object model diagrams. In addition, it also serves as a reference guide containing information on every object within ArcObjects. The help system is available to Visual Basic, .NET, Java, and C++ developers. You can start the ArcGIS Developer Help system through the ArcGIS program group from the Windows Start button.

The VB6 version of ArcGIS Developer Help is installed in a typical installation. Follow the custom installation procedures to access the C++, Java, or .NET versions.

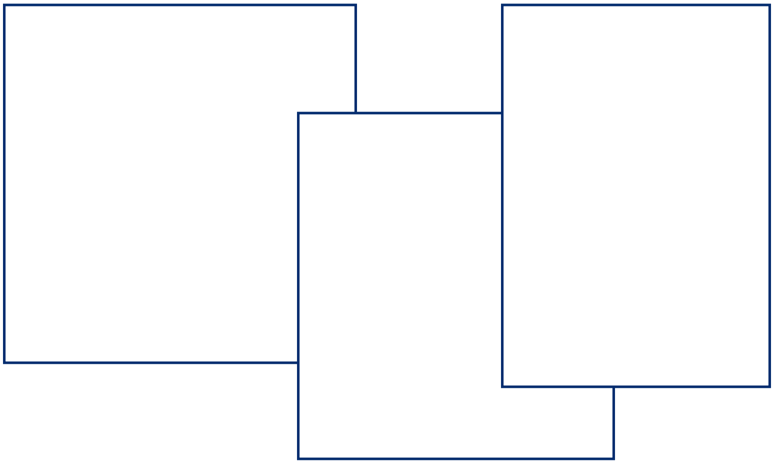


THE ARCGIS DEVELOPER SERIES

This book is one in a series of books for ArcGIS developers.

The *ArcGIS Desktop Developer Guide* is for developers who want to customize or extend one of the ArcGIS Desktop applications, such as ArcMap or ArcCatalog. Developers can use Visual Basic for Applications (VBA) to customize and either Visual Basic, Visual C++, or .NET to extend the applications.

The *ArcGIS Server Administrator and Developer Guide* is for developers who want to use ArcGIS Server to build custom server applications. Server developers can build Web services and Web applications that do simple mapping or include advanced GIS functionality. Several scenarios illustrate with code examples some of the different types of applications that can be developed using one of the multiple ArcGIS Server developer kits. This book also serves as the administration guide to ArcGIS Server.



ARCGIS DEVELOPER ONLINE WEB SITE

ArcGIS Developer Online is a Web-based equivalent of the ArcGIS Developer Help system and is available at the following URL: *http://arcgisdeveloperonline.esri.com.*

The online site has several advantages including being accessible via a Web browser and a connection to the Internet. The site is continually updated, making it the most up-to-date reference for developers.

ESRI SUPPORT CENTER

The ESRI Support Center at *http://support.esri.com* contains software information, technical documents, samples, forums, and a knowledge base for all ArcGIS products.

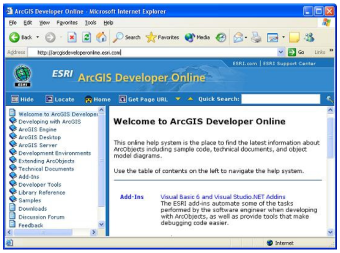
ArcGIS developers can take advantage of the forums, knowledge base, and samples sections to aid in development of their ArcGIS applications.

TRAINING

ESRI offers a number of instructor-led and Web-based training courses for the ArcGIS developer. These courses range from introductory level for VBA to the more advanced courses in component development for ArcGIS Desktop, Engine, and Server.

For more information, visit *http://www.esri.com* and select the Training and Events tab.

The ESRI Virtual Campus can be found directly at *http://campus.esri.com.*


ArcGIS Developer Online - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Search Favorites Media
Address http://arcgisdeveloperonline.esri.com
Go Links
ESRI.com | ESRI Support Center
ESRI ArcGIS Developer Online
Hide Locate Home Get Page URL Quick Search:
Welcome to ArcGIS Developer
Developing with ArcGIS
ArcGIS Engine
ArcGIS Desktop
ArcGIS Server
Development Environments
Extending ArcObjects
Technical Documents
Add-Ins
Developer Tools
Library Reference
Samples
Downloads
Discussion Forum
Feedback
Internet

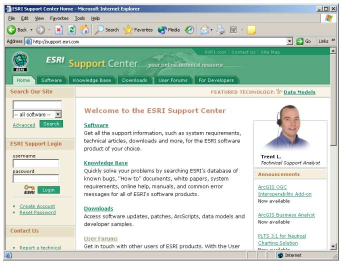
Welcome to ArcGIS Developer Online

This online help system is the place to find the latest information about ArcObjects including sample code, technical documents, and object model diagrams.

Use the table of contents on the left to navigate the help system.

Add-Ins — Visual Basic 6 and Visual Studio.NET Addins
The ESRI add-ins automate some of the tasks performed by the software engineer when developing with ArcObjects, as well as provide tools that make debugging code easier.

ArcGIS Developer Online at http://arcgisdeveloperonline.esri.com


ESRI Support Center Home - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Search Favorites Media
Address http://support.esri.com
Go Links
ESRI.com | Contact Us | Site Map
ESRI Support Center your online technical resource
Home Software Knowledge Base Downloads User Forums For Developers
Search Our Site
-- all software --
Advanced Search
ESRI Support Login
username
password
Login
Create Account
Reset Password
Contact Us
Report a technical
FEATURED TECHNOLOGY: Data Models
Trent L.
Technical Support Analyst
Announcements
ArcGIS OGC Interoperability Add-on
Now available
ArcGIS Business Analyst
Now available
PLTS 3.1 for Nautical Charting Solution
Internet

Welcome to the ESRI Support Center

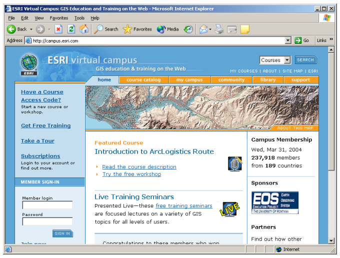
Software
Get all the support information, such as system requirements, technical articles, downloads and more, for the ESRI software product of your choice.

Knowledge Base
Quickly solve your problems by searching ESRI's database of known bugs, "How to" documents, white papers, system requirements, online help, manuals, and common error messages for all of ESRI's software products.

Downloads
Access software updates, patches, ArcScripts, data models and developer samples.

User Forums
Get in touch with other users of ESRI products. With the User

The ESRI Support Center at http://support.esri.com


ESRI Virtual Campus: GIS Education and Training on the Web - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Search Favorites Media
Address http://campus.esri.com
Go Links
ESRI virtual campus
GIS education & training on the Web
Courses SEARCH
MY COURSES | ABOUT | SITE MAP | ESRI
home course catalog my campus community library support
ABOUT THIS MAP
Have a Course Access Code?
Get Free Training
Take a Tour
Subscriptions
MEMBER SIGN-IN
Member login
Password
SIGN IN
Campus Membership
Sponsors
EOS Earth Observing System Education Project The University of Montana
Partners
LIVE
Internet

Start a new course or workshop.

Login to your account or find out more.

Featured Course
Introduction to ArcLogistics Route
- Read the course description
- Try the free workshop

Live Training Seminars
Presented Live—these free training seminars are focused lectures on a variety of GIS topics for all levels of users.

Congratulations to these members who won

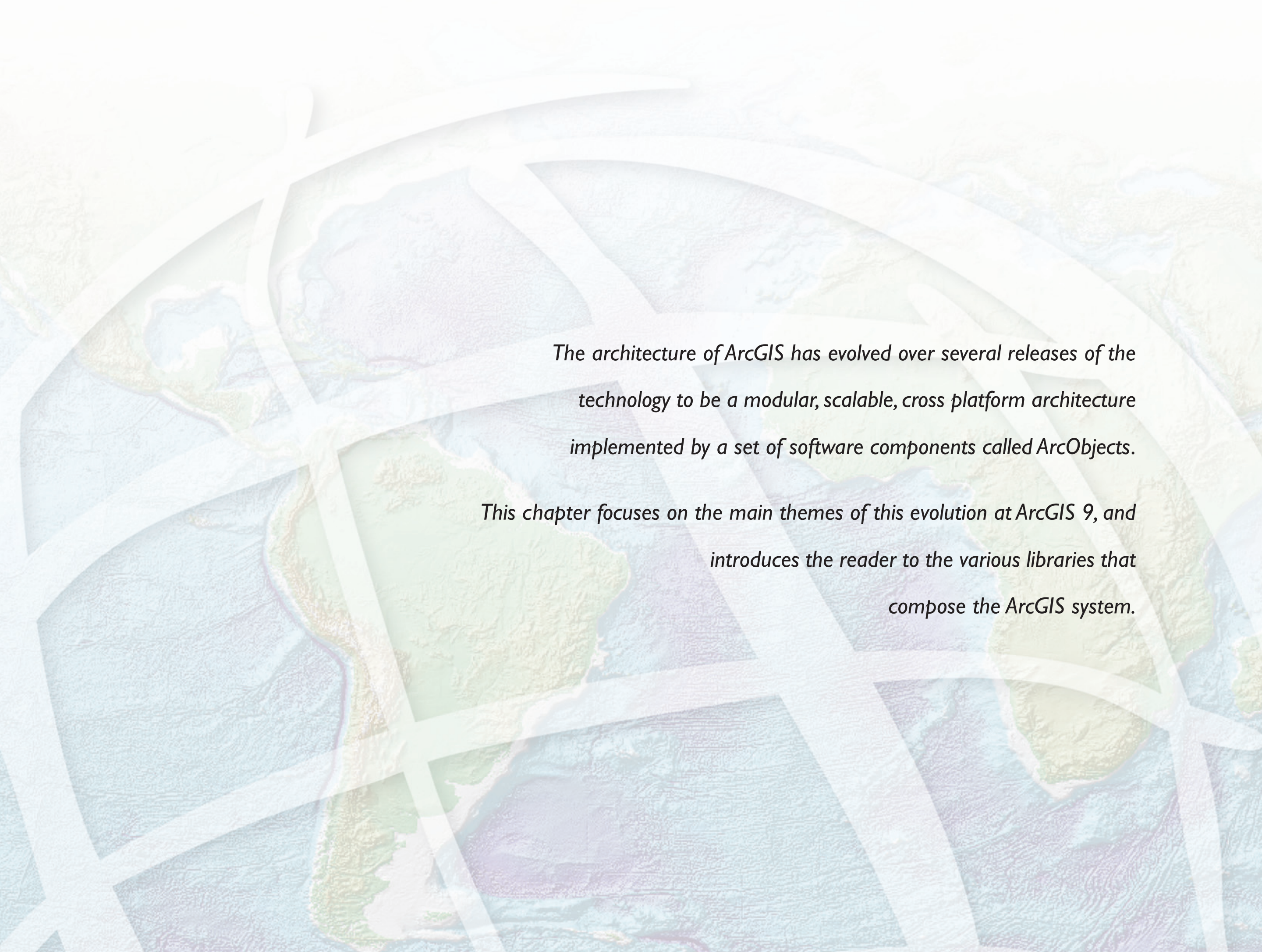
Wed, Mar 31, 2004
237,918 members
from 189 countries

Find out how other

The ESRI Virtual Campus at http://campus.esri.com

2

ArcGIS software architecture



The architecture of ArcGIS has evolved over several releases of the technology to be a modular, scalable, cross platform architecture implemented by a set of software components called ArcObjects.

This chapter focuses on the main themes of this evolution at ArcGIS 9, and introduces the reader to the various libraries that compose the ArcGIS system.

The ArcGIS Architecture has evolved over several releases of the technology to be a modular, scalable, cross platform architecture implemented by a set of software components called ArcObjects.

This section focuses on the main themes of this evolution at ArcGIS 9 and introduces the reader to the libraries that compose the ArcGIS system.

The ArcGIS software architecture supports a number of products, each with its unique set of requirements. The components that make up ArcGIS, ArcObjects, are designed and built to support this. This chapter introduces you to ArcObjects.

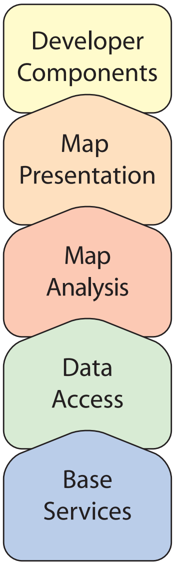
ArcObjects is a set of platform-independent software components, written in C++, that provide services to support GIS applications on the desktop, in the form of thick and thin clients, and on the server.

For a detailed explanation of COM see the COM section of Chapter 4, 'Developer environments'.

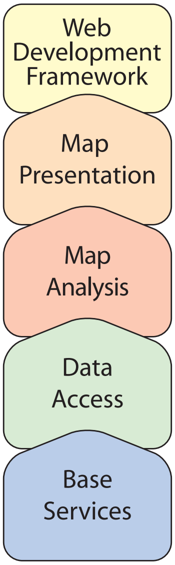
As stated, the language chosen to develop ArcObjects was C++; in addition to this language, ArcObjects makes use of the Microsoft Component Object Model (COM). COM is often thought of as simply specifying how objects are implemented and built in memory and how these objects communicate with one another. While this is true, COM also provides a solid infrastructure at the operating system level to support any system built using COM. On Microsoft Windows operating systems, the COM infrastructure is built directly into the operating system. For operating systems other than Microsoft Windows, this infrastructure must be provided for the ArcObjects system to function.

Not all ArcObjects components are created equal. The requirements of a particular object, in addition to its basic functionality, vary depending on the final end use of the object. This end use broadly falls into one of the three ArcGIS product families:

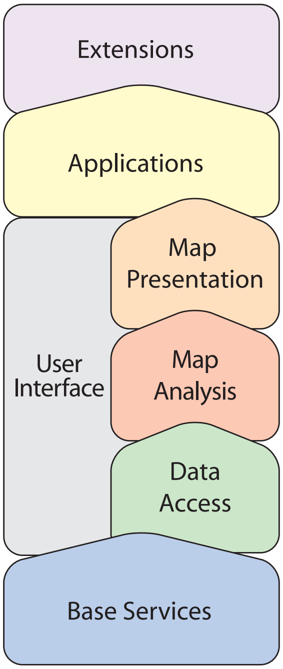
- ArcGIS Engine—Use of the object is within a custom application. Objects within the Engine must support a variety of uses; simple map dialog boxes, multithreaded servers, and complex Windows desktop applications are all possible uses of Engine objects. The dependencies of the objects within the Engine must be well understood. The impact of adding dependencies external to ArcObjects must be carefully reviewed, since new dependencies may introduce undesirable complexity to the installation of the application built on the Engine.
- ArcGIS Server—The object is used within the server framework, where clients of the object are most often remote. The remoteness of the client can vary from local, possibly on the same machine or network, to distant, where clients can be on the Internet. Objects running within the server must be scalable and thread safe to allow execution in a multithreaded environment.
- ArcGIS Desktop—Use of the object is within one of the ArcGIS Desktop applications. ArcGIS Desktop applications have a rich user experience, with applications containing many dialog boxes and property pages that allow end users to work effectively with the functionality of the object. Objects that contain properties that are to be modified by users of these applications should have property pages created for these properties. Not all objects require property pages.



ArcGIS Engine



ArcGIS Server



ArcGIS Desktop

Many of the ArcObjects that make up ArcGIS are used within all three of the ArcGIS products. The product diagrams on these pages show that the objects within the broad categories, Base Services, Data Access, Map Analysis, and Map Presentation, are contained in all three products. These four categories contain the majority of the GIS functionality exposed to developers and users in ArcGIS.

This commonality of function between all the products is important for developers to understand, since it means that when working in a particular category, much of the development effort can be transferred between the ArcGIS products with little change to the software. After all, this is exactly how the ArcGIS architecture is developed. Code reuse is a major benefit of building a modular architecture, but code reuse does not simply come from creating components in a modular fashion.

The ArcGIS architecture provides rich functionality to the developer, but it is not a closed system. The ArcGIS architecture is extendable by developers external to ESRI. Developers have been extending the architecture for a number of years, and the ArcGIS 9 architecture is no different; it, too, can be extended. However, ArcGIS 9 introduces many new possibilities for the use of objects created by ESRI and you. To realize these possibilities, components must meet additional requirements to ensure that they will operate successfully within this new and significantly enhanced ArcGIS system. Externally, some of the changes from ArcGIS 8 to ArcGIS 9 appear superficial, an example being the breakup of the type libraries into smaller libraries. That, along with the fact that the objects with their methods and properties, that were present at 8.3 are still available at 9.0, masks the fact that internally ArcObjects has undergone some significant work.

The main focus of the changes made to the ArcGIS architecture at 9.0 revolves around four key concepts:

- Modularity—A modular system where the dependencies between components are well-defined in a flexible system.
- Scalability—ArcObjects must perform well in all intended operating environments, from single user desktop applications to multiuser/multithreaded server applications.
- Multiple Platform Support—ArcObjects for the Engine and Server should be capable of running on multiple computing platforms.
- Compatibility—ArcObjects 9.0 should remain equivalent, both functionally and programmatically, to ArcObjects 8.3.

MODULARITY

The esriCore object library, shipped as part of ArcGIS 8.3, effectively packaged all of ArcObjects into one large block of GIS functionality; there was no distinction between components. The ArcObjects components were divided into smaller groups of components, these groups being packaged in DLLs. The one large library, while simplifying the task of development for external developers, prevented the software from being modular. Adding the type information to all the DLLs, while possible, would have greatly increased the burden on external developers and hence was not an option. In addition the DLL structure did not always reflect the best modular breakup of software components based on functionality and dependency.

ESRI has developed a modular architecture for ArcGIS 9 by a process of analyzing features and functions and matching that with end user requirements and deployment options based on the three ArcGIS product families. Developers who have extended the ArcGIS 8 architecture with custom components are encouraged to go through the same process to restructure their source code into similar modular structures.

There is always a trade-off in performance and manageability when considering architecture modularity. For each criteria, thought is given to the end use and the modularity required to support that. As an example, the system could be divided up into many small DLLs with only a few objects in each. While this provides a flexible system for deployment options, at minimum memory requirements it would affect performance due to the large number of DLLs being loaded and unloaded. Conversely, one large DLL containing all objects is not a suitable solution either. Knowing the requirements of the components allows them to be effectively packaged into DLLs.

An obvious functionality split to make is user interface and nonuser interface code. UI libraries tend to be included only with the ArcGIS Desktop products.

The ArcGIS 9 architecture is divided into a number of libraries. It is possible for a library to have any number of DLLs and EXEs within it. The requirements that components must meet to be within a library are well-defined. For instance, a library such as esriGeometry (from the Base Services set of modules) has the requirement of being thread safe, scalable, with no user interface components, and deployable on a number of computing platforms. These requirements are different from libraries such as esriArcMap (from the Applications category) which does have user interface components and is a Windows-only library.

All the components in the library will share the same set of requirements placed on the library. It is not possible to subdivide a library into smaller pieces for distribution. The library defines the namespace for all components within it and is seen in a form suitable for your chosen API.

- Type Library—COM
- .NET Interop Assembly—.NET
- Java Package—Java
- Header File—C++

SCALABILITY

The ArcObjects components within ArcGIS Engine and ArcGIS Server must be scalable. Engine objects are scalable because they can be used in many different types of applications; some require scalability while others do not. Server objects are required to be scalable to ensure that the server can handle many users connecting to it, and as the configuration of the server grows so does the performance of the ArcObjects components running on the server.

The scalability of a system is achieved using a number of variables involving the hardware and software of the system. In this regard, ArcObjects supports scalability with the effective use of memory within the objects and the ability to execute the objects within multithreaded processes.

For this discussion, thread safety refers to concurrent object access from multiple threads.

There are two considerations when multithreaded applications are discussed: thread safety and scalability. It is important for all objects to be thread safe, but simply having thread-safe objects does not automatically mean that creating multithreaded applications is straightforward or indeed that the resulting application will provide vastly improved performance.

The ArcObjects components contained in the Base Services, Data Access, Map Analysis, and Map Presentation categories are all thread safe. This means that application developers can use them in multithreaded applications; however programmers must still write multithreaded code in such a way as to avoid application failures due to deadlock situations and so forth.

In addition to the ArcObjects components being thread safe for ArcGIS 9, the apartment threading model used by ArcObjects was analyzed to ensure that ArcObjects could be run efficiently in a multithreaded process. A model referred to as "Threads in Isolation" was used to ensure that the ArcObjects architecture is used efficiently.

The classic singleton per process model means that all threads of an application will still access the main thread hosting the singleton objects. This effectively reduces the application to a single threaded application.

This model works by reducing cross-thread communication to an absolute minimum or better still, removing it entirely. For this to work, the singleton objects at ArcGIS 9 were changed to be singletons per thread and not singletons per process. The resource overhead of hosting multiple singletons in a process was outweighed by the performance gain of stopping cross-thread communication where the singleton object is created in one thread (normally the Main STA) and the accessing object is in another thread.

ArcGIS is an extensible system and for the Threads in Isolation model to work, all singleton objects must adhere to this rule. If you are creating singleton objects as part of your development, you must ensure that these objects adhere to the rule.

MULTIPLE PLATFORM SUPPORT

As stated earlier, ArcObjects components are C++ objects, meaning that any computing platform with a C++ compiler can potentially be a platform for ArcObjects. In addition to the C++ compiler, the platform must also support some basic services required by ArcObjects.

Microsoft Windows is a little endian platform, while Sun Solaris is a big endian platform.

While many of the platform differences do not affect the way in which ArcObjects components are developed, there are areas where differences do affect the way code is developed. The byte order of different computing architectures varies between little endian and big endian. This is most readily seen when objects read and write data to disk. Data written using one computing platform will not be compatible if read using another platform, unless some decoding is performed. All of the ArcGIS Engine and ArcGIS Server objects support this multiple platform persistence model. ArcObjects components always persist themselves using the little endian model; when the objects read persisted data, it is converted to the appropriate native byte order. In addition to the byte order differences, there are other areas of functionality that differ between platforms; the directory structure, for example, uses different separators for Windows and UNIX—'\' and '/', respectively. Another example is the platform specific areas of functionality such as OleDB.

COMPATIBILITY

While the aim of ArcGIS releases is to limit the change in the APIs, developers should still test their software thoroughly with later releases.

Maintaining compatibility of the ArcGIS system between releases is important to ensure that external developers are not burdened with changing their code to work with the latest release of the technology. Maintaining compatibility at the object level was a primary goal of the ArcGIS 9 development effort. While this object-level compatibility has been maintained, there are some changes between the ArcGIS 8 and ArcGIS 9 architectures that will affect developers, mainly related to the compilation of the software.

While the changes required to software created for use with ArcGIS 8 to work with ArcGIS 9 are minimal, it is important to understand that, to realize any existing investment in the ArcObjects architecture at ArcGIS 9, you must review your developments with respect to the ArcGIS Engine, ArcGIS Server, and ArcGIS Desktop.

ESRI understands the importance of a unified software architecture and has made numerous changes for ArcGIS 9 in order that the investment in ArcObjects can be realized on multiple products. If you have been involved in creating extensions to the ArcGIS architecture for ArcGIS 8, you should think how the new ArcGIS 9 architecture affects the way your components are implemented.

The functionality of ArcObjects can be accessed using four application programming interfaces (APIs). The choice of which API to use is not a simple one and will depend on a number of factors including; the ArcGIS product that you are developing with, the end user functionality that you are developing and your development experience with particular languages. The supported 4 APIs supported by ArcGIS Engine are:

It is important not to confuse the Visual C++ support available through the COM API, and the native C++ API.

- COM—Any COM compliant language (Visual Basic, Visual C++, Delphi, etc) can be used with this API.
- .NET—Visual Basic .NET and C# are supported by this API.
- Java—Sun™ Java2 Platform Standard Edition
- C++—Microsoft VC++ 6.0, Microsoft VC++ .NET 2003, Sun Solaris Forte 6 Update 2, Linux GCC 3.2

When working with ArcObjects developers can consume functionality exposed by the ArcObjects or extend the functionality of ArcObjects with their own components. When referring to these APIs there are differences with respect to consuming and extending the ArcObjects architecture.

CONSUMING API

Since ArcObjects are developed in C++, there are some case where data types compatible with C++ have been used for performance reasons. These performance considerations mostly affect the internals of ArcObjects, hence using one of the generic interfaces should not adversely affect performance or your ArcObjects developments.

All 4 APIs support consuming the functionality of the ArcObjects; however, not all Interfaces implemented by ArcObjects are supported on all platforms. In some cases interfaces make use of data types that are not compatible with an API. In situations like this an alternative implementation of the interface is provided for developers to use. The naming convention of a 'GEN' postfix on the interface name is used to signify this kind of interface; IFoo would have an IFooGEN interface. This alternative interface is usable by all APIs; however if the non generic interface is supported by the API it is possible to continue to use the API specific interface.

EXTENDING API

Extending ArcObjects is all about creating your own objects and adding them to the ArcObjects architecture. ArcObjects is written to be extensible in almost all areas. Support for extending the Architecture varies between the APIs, and in some cases varies between languages of an API.

The COM API provides the most possibilities of extending the system. The limitation within this API is with the Visual Basic language. Visual basic does not support the implementation of interfaces that have one or more of the following characteristics:

- The interface inherits from an interface other than IUnknown or IDispatch. ICurve which inherits from IGeometry cannot be implemented in VB for this reason.
- Method names on an interface starting with an underscore ("_"). You will not find functions beginning with "_" in ArcObjects.
- A parameter of a method uses a data type not supported by Visual Basic. IActiveView cannot be implemented in Visual Basic for this reason.

In addition to the limitations on the interfaces supported by VB, the binary re-use technique of COM Aggregation is not supported by VB. This means that certain parts of the Architecture cannot be extended; Custom Features is one such example. In reality the above limitations of Visual Basic have little effect on the vast majority of developers, since the percentage of ArcObjects affected is small, and for this small percentage it is unlikely that developers will have a need to extend the architecture. Other COM languages such as Visual C++ do not have any of these limitations.

The .NET API supports extending ArcObjects fully, with the one exception being interfaces that make use of non OLE Automation compliant data types (see the table below for a complete list of all OLE Automation compliant datatypes).

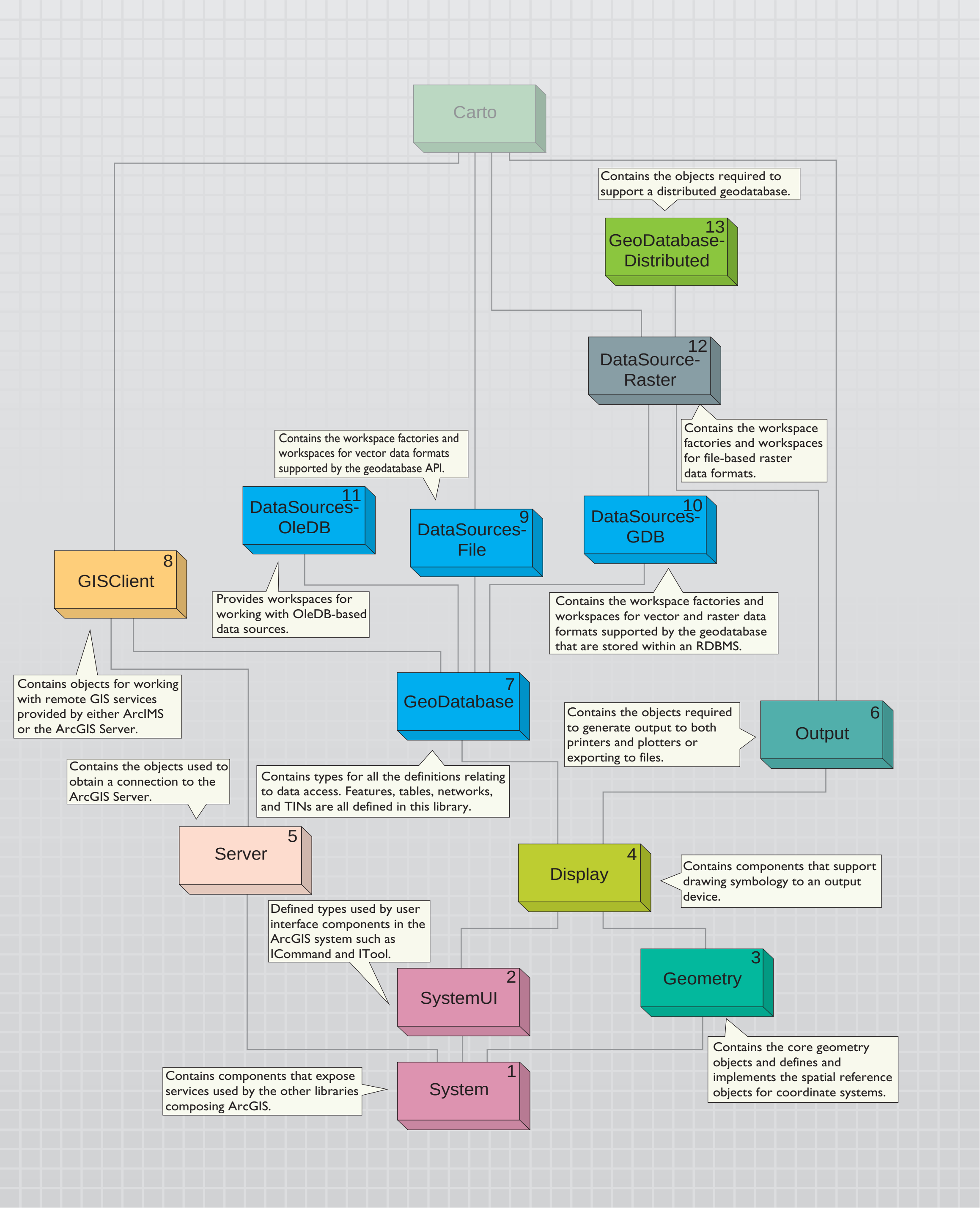
The majority of differences between the APIs support for ArcObjects revolves around data types.All APIs fully support the automation compliant data types show on the right. Differences occur with data types that are not ole automation compliant.

Type	Description
Boolean	Data item that can have the value True or False.
unsigned char	8-bit unsigned data item.
double	64-bit IEEE floating-point number.
float	32-bit IEEE floating-point number.
int	Signed integer, whose size is system dependent.
long	32-bit signed integer.
short	16-bit signed integer.
BSTR	Length-prefixed string.
CURRENCY	8-byte, fixed-point number.
DATE	64-bit, floating-point fractional number of days since Dec 30, 1899.
SCODE	For 16-bit systems - Built-in error that corresponds to VT_ERROR.
Typedef enum myenum	Signed integer, whose size is system dependent.
Interface IDispatch *	Pointer to the IDispatch interface.
Interface IUnknown *	Pointer to an interface that does not derive from IDispatch.
dispinterface Typename *	Pointer to an interface derived from IDispatch.
Coclass Typename *	Pointer to a coclass name (VT_UNKNOWN).
[oleautomation] interface Typename *	Pointer to an interface that derives from IDispatch.
SAFEARRAY (TypeName)	TypeName is any of the above types. Array of these types.
TypeName*	TypeName is any of the above types. Pointer to a type.
Decimal	96-bit unsigned binary integer scaled by a variable power of 10. A decimal data type that provides a size and scale for a number (as in coordinates).

OLE Automation Data types

The Java and C++ APIs have similar limited support for extending ArcObjects. Developers of these APIs are restricted to only being able to create custom commands and tools. These commands and tools can then be used with the ToolbarControl. This may appear to be a severe limitation, but despite this restriction these APIs still have much to attract the developer. The ToolbarControl along with the other ArcGIS Controls offers a rich development environment to work with. The ArcGIS Desktop applications are rich professional GIS applications with a lot of functionality, but if viewed simply the applications can be broken down into a series of Toolbars, along with a TOC and map viewing area. The desktop applications are all extended by adding new commands and tools. In a similar way developers can build applications with rich functionality using any of the 4 ArcGIS Engine APIs.

The COM and .NET APIs are only supported on the Microsoft Windows platform, while the Java and C++ APIs are supported on all the platforms supported by ArcGIS Engine.



Contains the objects required to support a distributed geodatabase.

Contains the workspace factories and workspaces for file-based raster data formats.

Contains the workspace factories and workspaces for vector data formats supported by the geodatabase API.

Provides workspaces for working with OleDB-based data sources.

Contains the workspace factories and workspaces for vector and raster data formats supported by the geodatabase that are stored within an RDBMS.

Contains objects for working with remote GIS services provided by either ArcIMS or the ArcGIS Server.

Contains the objects required to generate output to both printers and plotters or exporting to files.

Contains the objects used to obtain a connection to the ArcGIS Server.

Contains types for all the definitions relating to data access. Features, tables, networks, and TINs are all defined in this library.

Contains components that support drawing symbology to an output device.

Defined types used by user interface components in the ArcGIS system such as ICommand and ITool.

Contains the core geometry objects and defines and implements the spatial reference objects for coordinate systems.

Contains components that expose services used by the other libraries composing ArcGIS.

For a comprehensive discussion on each library, refer to the library overview topics, a part of the library reference section of the ArcGIS Developer Help system.

The libraries contained within the ArcGIS Engine are summarized below. The diagrams that accompany this section indicate the library architecture of the ArcGIS Engine. Understanding the library structure, their dependencies and basic functionality will help you as a developer navigate through the components of ArcGIS Engine.

The libraries are discussed in dependency order. The diagrams show this with sequential numbers in the upper right corner of the library block. For example, System, as the library at the base of the ArcGIS architecture, is numbered one while GeoDatabase, numbered seven, depends on the six libraries that proceed it in the diagram—System, SystemUI, Geometry, Display, Server, and Output.

System

The System library is the lowest level library in the ArcGIS architecture. The library contains components that expose services used by the other libraries composing ArcGIS. There are a number of interfaces defined within System that can be implemented by the developer. The AoInitializer object is defined in System, all developers must use this object to initialize and un-initialize the ArcGIS Engine in applications that make use of Engine functionality. The developer does not extend this library but can extend the ArcGIS system by implementing interfaces contained within this library.

SystemUI

Knowing the library dependency order is important since it affects the way in which developers interact with the libraries as they develop software.As an example C++ developers must include the type libraries in the library dependency order to ensure correct compilation. Understanding the dependencies also helps when deploying your developments.

The SystemUI library contains the interface definitions for user interface components that can be extended within the ArcGIS Engine. These include the *ICommand*, *ITool,* and *IToolControl* interfaces. The developer uses these interfaces to extend the UI components that the ArcGIS Engine's developer components use. The objects contained within this library are utility objects available to the developer to simplify some user interface developments. The developer does not extend this library but can extend the ArcGIS system by implementing interfaces contained within this library.

Geometry

The Geometry library handles the geometry, or shape, of features stored in feature classes or other graphical elements. The fundamental geometry objects that most users will interact with are *Point*, *MultiPoint*, *Polyline* and *Polygon*. Besides those top-level entities are geometries that serve as building blocks for polylines and polygons. Those are the primitives that compose the geometries. They are *Segment*, *Pat*h, and *Ring*. Polylines and polygons are composed of a sequence of connected segments that form a *Path*. A segment consists of two distinguished points, the start and the end point, and an element type that defines the curve from start to end. The kinds of segments are *CircularArc*, *Line*, *EllipticArc,* and *BezierCurve*. All geometry objects can have Z, M, and IDs associated with their vertices. The fundamental geometry objects all support geometric operations such as *Buffer*, *Clip*, and so on. The geometry primitives are not meant to be extended by developers.

Entities within a GIS refer to real-world features; the location of these real-world features is defined by a geometry along with a spatial reference. Spatial reference objects, for both projected and geographic coordinate systems are included in the Geometry library. Developers can extend the spatial reference system by adding new spatial references and projections between spatial references.

Display

The Display library contains objects used for the display of GIS data. In addition to the main display objects responsible for the actual output of the image, the library contains objects that represent symbols and colors used to control the properties of entities drawn on the display. The library also contains objects that provide the user with visual feedback when interacting with the display. Developers most often interact with the Display through a view similar to the ones provided by the *Map* or *PageLayout* objects. All parts of the library can be extended; commonly extended are symbols, colors, and display feedbacks.

Server

The Server library contains objects that allow you to connect and work with ArcGIS Servers. Developers gain access to an ArcGIS Server using the *GISServerConnection* object. The *GISServerConnection* object gives access to the *ServerObjectManager*. Using this object, a developer works with *ServerContext* objects to manipulate ArcObjects running on the server. The Server library is not extended by developers. Developers can also use the GISClient library when interacting with the ArcGIS Server.

Output

The Output library is used to create graphical output to devices, such as printers and plotters, and hardcopy formats such as enhanced metafiles and raster image formats (JPG, BMP, etcetera). The developer uses the objects in the library with other parts of the ArcGIS system to create graphical output. Usually these would be objects in the Display and Carto libraries. Developers can extend the Output library for custom devices and export formats.

GeoDatabase

The GeoDatabase library provides the programming API for the geodatabase. The geodatabase is a repository of geographic data built on standard industry relational and object relational database technology. The objects within the library provide a unified programming model for all supported data sources within ArcGIS. The GeoDatabase library defines many of the interfaces that are implemented by data source providers higher in the architecture. The geodatabase can be extended by developers to support specialized types of data objects (Features, Classes, etcetera); in addition, it can also have custom vector data sources added using the *PlugInDataSource* objects. The native data types supported by the geodatabase cannot be extended.

GISClient

The GISClient library allows developers to consume Web services; these Web services can be provided by ArcIMS and ArcGIS Server. The library includes objects for connecting to GIS servers to make use of Web services. There is support for ArcIMS Image and Feature Services. The library provides a common programming model for working with ArcGIS Server objects in a stateless manner either directly or through a Web service catalog. The ArcObjects components running on the ArcGIS Server are not accessible through the GISClient interface. To gain direct access to ArcObjects running on the server, you should use functionality in the Server library.

DataSourcesFile

The DataSourcesFile library contains the implementation of the GeoDatabase API for file-based data sources. These file based data sources include shapefile, coverage, TIN, CAD, SDC, StreetMap, and VPF. The DataSourcesFile library is not extended by developers.

DataSourcesGDB

The DataSourcesGDB library contains the implementation of the GeoDatabase API for the database data sources. These data sources include Microsoft Access and relational database management systems supported by ArcSDE—IBM® DB2®, Informix®, Microsoft SQL Server™, and Oracle®. The DataSourcesGDB library is not extended by developers.

DataSourcesOleDB

The DataSourcesOleDB library contains the implementation of the GeoDatabase API for the Microsoft OLE DB data sources. This library is only available on the Microsoft Windows operating system. These data sources include any OLE DB supported data provider and text file workspaces. The DataSourcesOleDB library is not extended by developers.

DataSourcesRaster

The DataSourcesRaster library contains the implementation of the GeoDatabase API for the raster data sources. These data sources include relational database management systems supported by ArcSDE—IBM DB2, Informix, Microsoft SQL Server, and Oracle—along with supported RDO raster file formats. Developers do not extend this library when support for new raster formats is required, rather they extend RDO. The DataSourcesRaster library is not extended by developers.

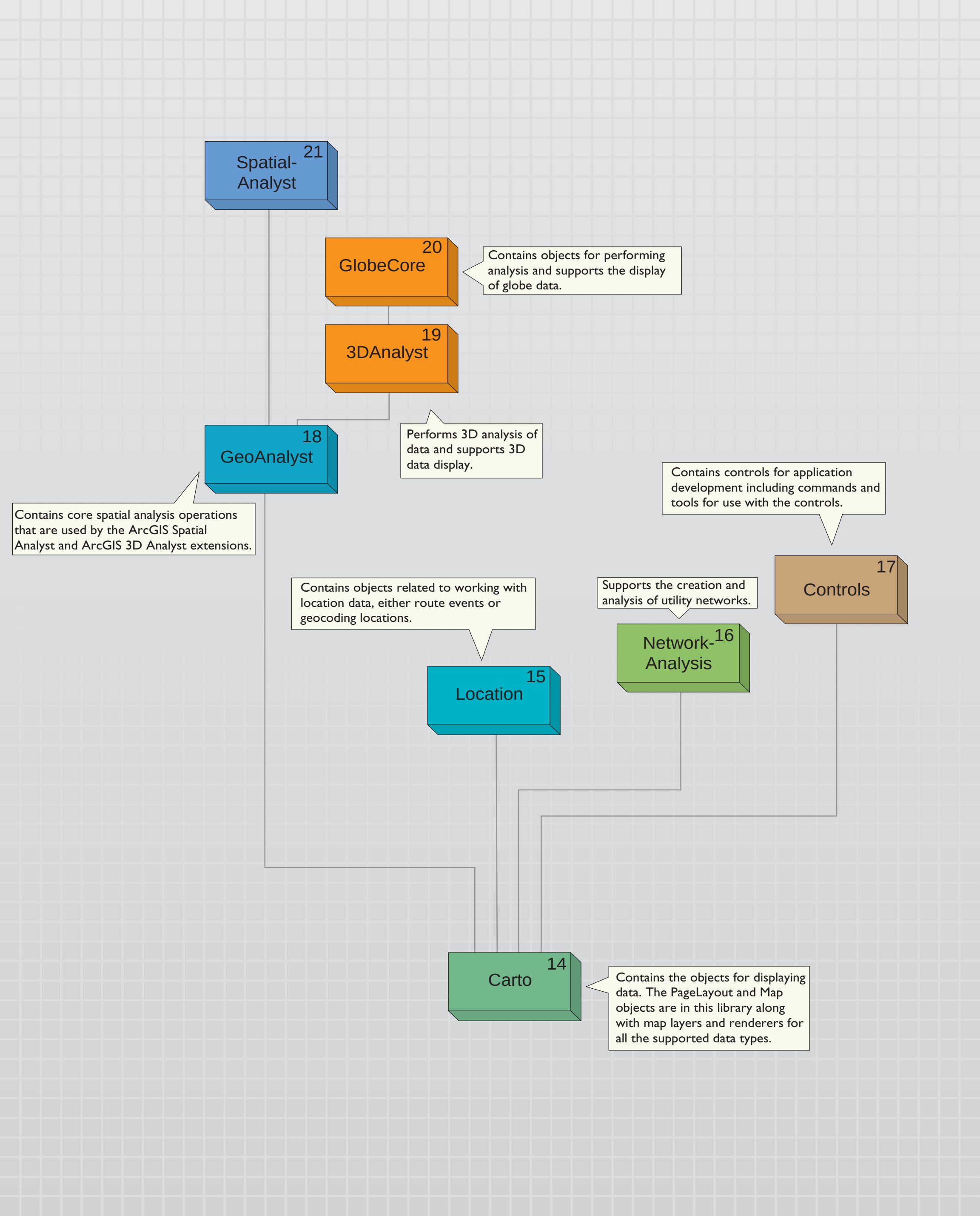
Raster Data Objects (RDO) is a COM API that provides display and analysis support for file based raster data.

GeoDatabaseDistributed

The GeoDatabaseDistributed library supports distributed access to an enterprise geodatabase by providing tools for importing data into and exporting data out of a geodatabase. The GeoDatabaseDistributed library is not extended by developers.

Carto

The Carto library supports the creation and display of maps; these maps can consist of data in one map or a page with many maps and associated marginalia. The *PageLayout* object is a container for hosting one or more maps and their associated marginalia: North arrows, legends, scale bars, etcetera. The *Map* object is a container of layers. The *Map* object has properties that operate on all layers



Contains objects for performing analysis and supports the display of globe data.

Performs 3D analysis of data and supports 3D data display.

Contains core spatial analysis operations that are used by the ArcGIS Spatial Analyst and ArcGIS 3D Analyst extensions.

Contains controls for application development including commands and tools for use with the controls.

Supports the creation and analysis of utility networks.

Contains objects related to working with location data, either route events or geocoding locations.

Contains the objects for displaying data. The PageLayout and Map objects are in this library along with map layers and renderers for all the supported data types.

within the map—spatial reference, map scale, etcetera—along with methods that manipulate the map's layers. There are many different types of layers that can be added to a map. Different data sources often have an associated layer responsible for displaying the data on the map, vector features are handled by the *FeatureLayer* object, raster data by the *RasterLayer*, TIN data by the *TinLayer*, and so on. Layers can, if required, handle all the drawing operations for their associated data, but it is more common for layers to have an associated *Renderer* object. The properties of the *Renderer* object control how the data is displayed in the map. Renderers commonly use symbols from the Display library for the actual drawing, the renderer simply matches a particular symbol with the properties of the entity that is to be drawn. A *Map* object, along with a *PageLayout* object, can contain elements. An element has geometry to define its location on the map or page, along with behavior that controls the display of the element. There are elements for basic shapes, text labels, complex marginalia, and so on. The Carto library also contains support for map annotation and dynamic labeling.

The ArcGIS Server uses the MapServer object for its MapService.

Although developers can directly make use of the *Map* or *PageLayout* objects in their applications, it more common for developers to use a higher level object such as the *MapControl*, *PageLayoutControl,* or an ArcGIS application. These higher level objects simplify some tasks, although they always provide access to the lower level *Map* and *PageLayout* objects, allowing the developer fine control of the objects.

The *Map* and *PageLayout* objects are not the only objects in Carto that expose the behavior of map and page drawing. The *MxdServer* and *MapServer* objects both support the rendering of maps and pages, but instead of rendering to a window, these objects render directly to a file.

Using the *MapDocument* object developers can persist the state of the map and page layout within a map document (.mxd), which can be used in ArcMap or one of the ArcGIS controls.

The Carto library is commonly extended in a number of areas. Custom renderers, layers, etcetera, are common. A custom layer is often the easiest method of adding custom data support to a mapping application.

Location

The Location library contains objects that support geocoding and working with route events. The geocoding functionality can be accessed through fine-grained objects for full control, or the *GeocodeServer* objects offers a simplified API. Developers can create their own geocoding objects. The linear referencing functionality provides objects for adding events to linear features and rendering these events using a variety of drawing options. The developer can extend the linear reference functionality.

NetworkAnalysis

The NetworkAnalysis library provides objects for populating a geodatabase with network data and objects to analyze the network when it is loaded in the geodatabase. Developers can extend this library to support custom network tracing. The library is meant to work with utility networks: gas lines, electricity supply lines, etcetera.

Controls

The contents of the Map and PageLayout controls can be specified programmatically, or they can load Map Documents

The ReaderControl only supports Published Map Files (PMF)

ArcGIS Engine comes with over 150 commands.

The Controls library is used by developers to build or extend applications with ArcGIS functionality. The ArcGIS Controls simplify the development process by encapsulating ArcObjects and providing a coarser grained API. Although the controls encapsulate the fine grained ArcObjects they do not restrict access to them. The *MapControl* and *PageLayoutControl* encapsulate the Carto library's *Map* and *PageLayout* objects respectively. The *ReaderControl* encapsulates both the *Map* and *PageLayout* objects, and provides a simplified API when working with the control. If the map publisher has granted permission the developer can access the internal objects, in a similar way to the *Map* and *PageLayout* controls. The library also contains the *TOCControl* that implements a table of contents and a *ToolbarControl* for hosting commands and tools that work with a suitable control.

Developers extend the Controls library by creating their own commands and tools for use with the controls. To support this the library has the *HookHelper* object. This object makes it straightforward to create a command that works with any of the controls, in addition to ArcGIS applications such as ArcMap.

GeoAnalyst

The GeoAnalyst library contains objects that support core spatial analysis functions. These functions are used within both the SpatialAnalyst and 3DAnalyst libraries. Developers can extend the library by creating a new type of raster operation. A license for either the ArcGIS Spatial Analyst or 3D Analyst™ extension or the ArcGIS Engine Runtime Spatial or 3D option is required to make use of the objects in this library.

3DAnalyst

The 3DAnalyst library contains objects for working with 3D scenes in a similar way that the Carto library contains objects for working with 2D maps. The *Scene* object is one of the main objects of the library since it is the container for data similar to the *Map* object. The *Camera* and *Target* objects specify how the scene is viewed regarding the positioning of the features relative to the observer. A scene consists of one or more layers; these layers specify the data in the scene and how the data is drawn.

It is not common for developers to extend this library. A license for either the ArcGIS 3D Analyst extension or the ArcGIS Engine Runtime 3D option is required to work with objects in this library.

GlobeCore

The GlobeCore library contains objects for working with globe data in a similar way that the Carto library contains objects for working with 2D maps. The *Globe* object is one of the main objects of the library since it is the container for data similar to the *Map* object. The *GlobeCamera* object specifies how the globe is viewed regarding the positioning of the globe relative to the observer. The globe can have one or more layers; these layers specify the data on the globe and how the data is drawn.

The GlobeCore library has a developer control along with a set of commands and tools to use with this control. This control can be used in conjunction with the objects in the Controls library.

It is not common for developers to extend this library. A license for either the ArcGIS 3D Analyst extension or the ArcGIS Engine Runtime 3D option is required to work with objects in this library.

SpatialAnalyst

The SpatialAnalyst library contains objects for performing spatial analysis on raster and vector data. Developers most commonly consume the objects within this library and do not extend it. A license for either the ArcGIS Spatial Analyst extension or the ArcGIS Engine Runtime Spatial option is required to work with objects in this library.

3

Developing with ArcGIS Controls

ArcGIS Engine provides a number of high-level developer controls that enable you to build or extend applications with ArcGIS functionality and create a high-quality map-based user interface. These include the MapControl, PageLayoutControl, ReaderControl, TOCControl, and ToolbarControl. The GlobeControl and SceneControl are also available but applications using these controls must be be authorized with the ArcGIS Engine 3D option.

This chapter includes:

• an overview of each control • a discussion of themes and concepts common to each of the ArcGIS Controls • considerations for building applications with or without the ToolbarControl.

The ArcGIS Controls are high-level developer components that firstly enable developers to build and extend applications with ArcGIS functionality and secondly provide a graphical user interface (GUI).

Each of the following ArcGIS Controls is available as an ActiveX control, .Net Windows control, and Visual JavaBean:

- MapControl—similar to the 'data'
- PageLayoutControl
- ToolbarControl
- TOCControl (Table of Contents Control)
- SceneControl
- GlobeControl
- ReaderControl

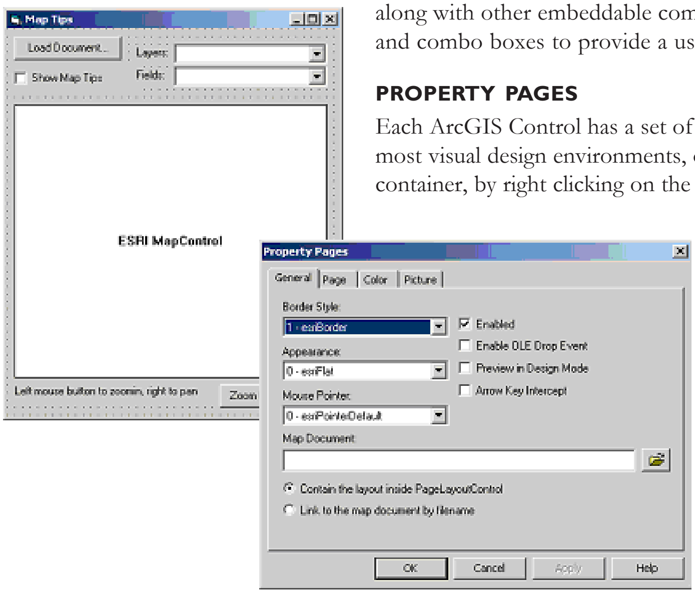
Deployment of applications built with either the GlobeControl or SceneControl requires the ArcGIS Engine 3D Option.

The ArcGIS Controls can be used to build applications in two ways: Firstly, the ArcGIS Control(s) can be embedded into an existing application to add additional mapping capability, or secondly, the ArcGIS Control(s) can be used to create a new stand-alone application. In either case, an individual ArcGIS Control can be embedded into an application or the TOCControl and ToolbarControl can be used in conjunction with another ArcGIS Control to provide part of the applications framework.

There are some common themes and concepts, applicable to all of the ArcGIS Controls, that should be understood in order to effectively build applications using the ArcGIS Controls.

EMBEDDABLE COMPONENTS

Each ArcGIS Control is an embeddable component that can be dropped within a container form or dialog provided by a visual design environment. Once within a container the ArcGIS Control can be resized and repositioned along with other embeddable components such as command buttons and combo boxes to provide a user interface in the application.



Border Style: 1 - esriBorder

☑ Enabled

☐ Enable OLE Drop Event

☐ Preview in Design Mode

☐ Arrow Key Intercept

Appearance: 0 - esriFlat

Mouse Pointer: 0 - esriPointerDefault

Map Document:

● Contain the layout inside PageLayoutControl

○ Link to the map document by filename

All properties accessible via the property pages can be set in code by the developer.

PROPERTY PAGES

Each ArcGIS Control has a set of property pages that are accessible in most visual design environments, once the control is embedded within a container, by right clicking on the control and choosing 'Properties' from the context menu. These property pages provide shortcuts to a selection of a control's properties and methods, and allow a developer to build an application with little or no code.

ARCOBJECTS

Each ArcGIS Control simplifies the development process by encapsulating coarse grained ArcObjects, while still providing access to finer grained ArcObjects. For example, the PageLayoutControl encapsulates the *PageLayout* object. The *PageLayout* contains at least one *MapFrame* element containing a *Map* and the *Map* can contain multiple raster, feature, or custom *Layer* objects. Each ArcGIS Control provides shortcuts to frequently used properties and methods on the ArcObjects they encapsulate. For example, the MapControl has a *SpatialReference* property that is a shortcut to the *SpatialReference* property of the *Map* object. Each ArcGIS Control also has some helper methods that perform common tasks. For example, the MapControl has an *AddShapeFile* method. The ArcGIS Controls are typically a starting point for developing applications not only because they provide user interface, but by providing a direct route into the object model.

EVENTS

Each ArcGIS Control fires events in response to keyboard and mouse interactions by the end user. Other events fire in response to actions occurring within the controls. For example, when a map document is loaded into the MapControl the *OnMapReplaced* event is fired, or when an object is dragged over the MapControl via drag and drop the *OnOleDrop* event is fired.

BUDDY CONTROLS

For more details on how the controls work together, see the TOCControl and ToolbarControl sections below.

The ToolbarControl and TOCControl each work in conjunction with one other 'buddy control'. Typically the 'buddy control' is a MapControl, PageLayoutControl, ReaderControl, SceneControl, or GlobeControl. The 'buddy control' can be set at design time though the control property pages (in development environments that support property page capability) or programmatically using the SetBuddyControl.

CONTROL COMMANDS

The ArcGIS Engine provides a set of commands, tools, and menus that work with the ArcGIS Controls. For example, there is a suite of map navigation, feature selection, and graphic element commands that work with the MapControl and PageLayoutControl. Likewise there is a suite of commands for the SceneControl, GlobeControl, and ReaderControl. For applications using an individual control, these commands can work directly with the control by programmatically creating a new instance of the command and passing the control to the command's *OnCreate* event. For applications using the ToolbarControl in conjunction with a 'buddy control', these commands can be added to the ToolbarControl either through the property pages at design time, or programmatically, or at run-time by the end user if the ToolbarControl is in customize mode.

Developers can also extend the suite of commands provided by the ArcGIS Engine by creating their own custom commands, tools, and menus to work with the ArcGIS Controls. The *HookHelper*, *GlobeHookHelper*, and *SceneHookHelper* objects can be used to simplify this development. Refer to the 'Building Applications' scenario in Chapter 6, 'Developer Scenarios' to see how to build a custom command using the *HookHelper* object.

MAP AUTHORING

The ArcGIS Desktop applications can be used to pre-author documents that can be loaded into the ArcGIS Controls, to quickly produce high quality mapping. For example, ArcMap can be used to author map documents that can be loaded into the MapControl and PageLayoutControl. Pre-authoring documents can save a substantial amount of time as it saves having to programmatically build up maps and symbology from scratch. Once a document is loaded into an ArcGIS Control any layers, elements, and symbols can be accessed programmatically through the object model if their appearance subsequently needs changing.

The table below summarizes the types of document that can be loaded into each ArcGIS Control.

					Published Map Files (*.pmf)		
	Map Document (*.mxd, *.mxt)	Layer Files (*.lyr)	Scene Document (*.sxd, *.sxt)	Globe Document (*.3dd, *.sdt)	no permission to load in a customized application (ArcReader application only	permission to load in a customized application	permission to load a customized application and unrestricted access to its contents
MapControl	Yes	Yes	No	No	No	No	Yes
PageLayoutControl	Yes	**Yes	No	No	No	No	Yes
SceneControl	No	**Yes	Yes	No	No	No	No
GlobeControl	No	**Yes	No	Yes	No	No	No
ReaderControl	No	No	No	No	No	Yes	Yes
*ArcReaderControl	No	No	No	No	No	Yes	Yes

* *The ArcReaderControl is only available with the ArcGIS Publisher extension. However, it is listed here due to its similarity with the ReaderControl.*

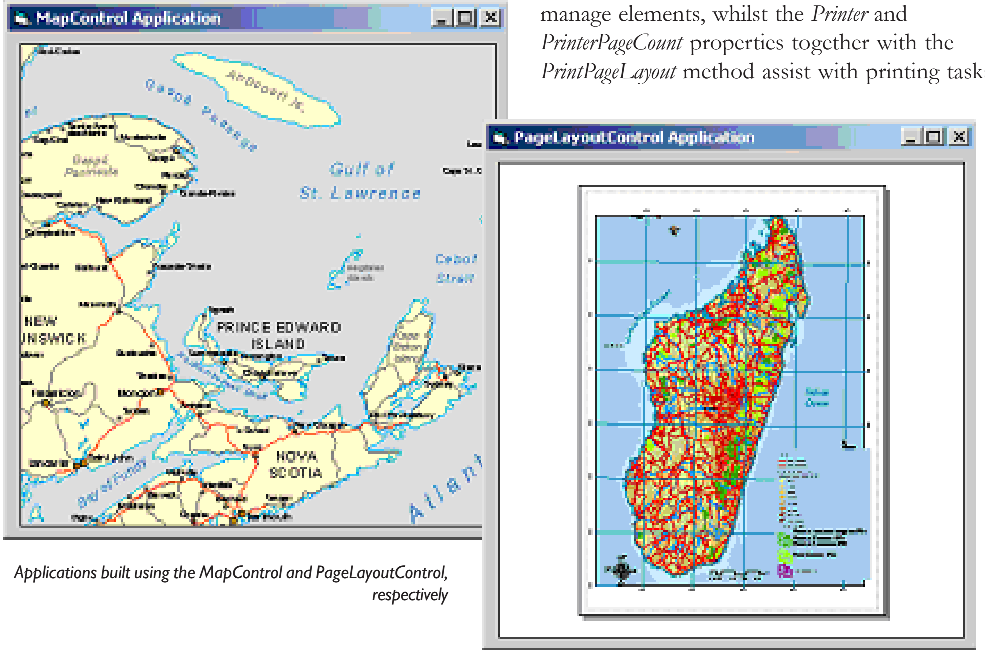
** *There are no properties available on the ArcGIS Controls to directly load Layer files (*.lyr). However, they can be loaded indirectly via the* MapDocument *object.*

The MapControl and PageLayoutControl correspond to the 'data' and 'layout' views of the ArcMap desktop application. The MapControl encapsulates the *Map* object and the PageLayoutControl encapsulates the *PageLayout* object. Map documents authored with the ArcMap application can be loaded into the MapControl and PageLayoutControl, to save the developer programmatically composing the cartography.

The map document can be set at design time though the MapControl and PageLayoutControl property pages (in development environments that support property page capability) and the control can be set to 'link' or 'contain' the map document. When 'linking' the control will read the map document whenever the control is created on the container and will display the most recent updates to the map document. When 'containing' the control will copy the contents of the map document into the control and will not display any further updates made to the map document from that point onwards. Alternatively, a map document can be loaded into the control programmatically using the *LoadMxFile* method.

Not only can the MapControl and PageLayoutControl read map documents, they can also write map documents (*.mxd). Both controls implement the *IMxdContents* interface that enables the *MapDocument* object to write the contents of the MapControl and PageLayoutControl to a new map document.

Helper methods such as *TrackRectangle*, *TrackPolygon*, *TrackLine*, and *TrackCircle* exist on the MapControl for tracking or 'rubber banding' shapes on the display. The *VisibleRegion* property can be used to change the shape of the MapControl's display area. Helper methods such as *FindElementByName* and *LocateFrontElement* exist on the PageLayoutControl to help the developer manage elements, whilst the *Printer* and *PrinterPageCount* properties together with the *PrintPageLayout* method assist with printing tasks.

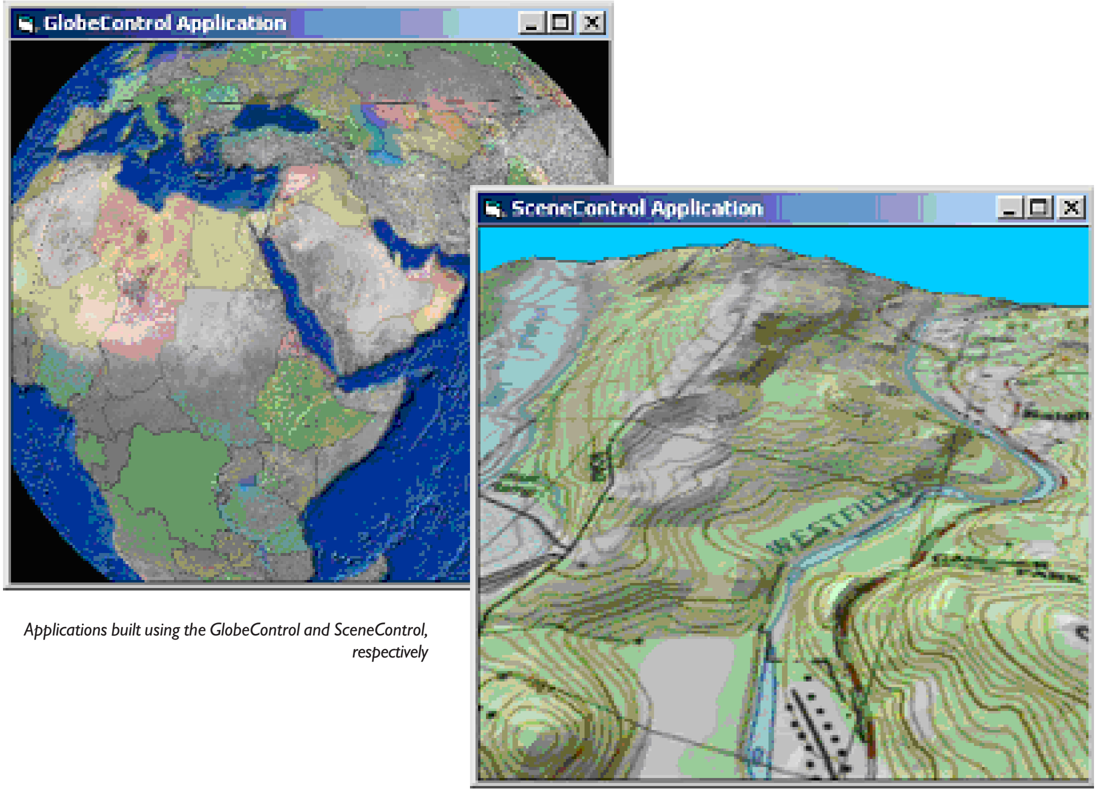


Applications built using the MapControl and PageLayoutControl, respectively

Deployment of applications built with either the GlobeControl or SceneControl requires the ArcGIS Engine 3D Option.

The GlobeControl and SceneControl correspond to the 3D views of the ArcGlobe and ArcScene desktop applications. The GlobeControl encapsulates the *GlobeViewer* object and the SceneControl encapsulates the *SceneViewer* object. Globe and Scene documents authored with the ArcGlobe and ArcScene applications can be loaded into the GlobeControl and SceneControl respectively, to save the developer programmatically composing the cartography.

Both the GlobeControl and SceneControl have built in navigation capability that allows the end user to move around the 3D view and visualize the 3D data, without having to use the available control commands or a custom command. To use the built in navigation, the *Navigate* property must be set either through the property pages or programmatically. The end user can use the left mouse button navigate backwards and forwards and to the left and right of the display, and the right mouse button to zoom in and out on the display.



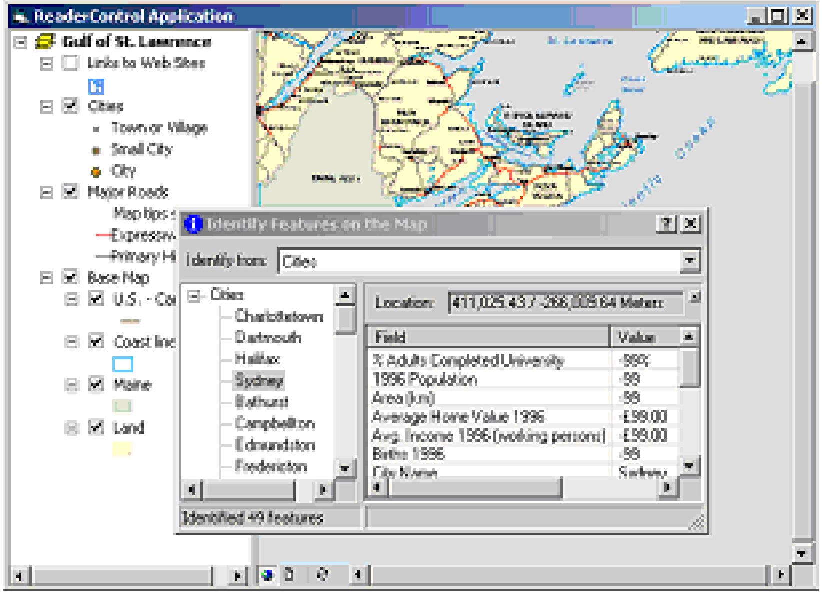
Applications built using the GlobeControl and SceneControl, respectively

ArcReader is available for download. To get your free copy, go to http://www.esri.com/software/arcgis/arcreader.

The ReaderControl corresponds to the 'data' and 'layout' views of the ArcReader™ desktop application, together with its Table of Contents. The ReaderControl also contains the internal windows and tools used by the ArcReader desktop application, such as the Find window and the Identify tool. Published Map Files (PMF) authored with the ArcMap desktop application and published with the ArcGIS Publisher extension can be loaded into the ReaderControl, if published with permission to load into a customized ArcReader application.

The ReaderControl has a simple self-contained object model that exposes all the functionality of the ArcReader desktop application and does not require access to ArcObjects. As such, developing applications with the ReaderControl does not require previous experience with ArcObjects. However, if a Published Map File was published with unrestricted access to its contents, developers can access the underlying ArcObjects and develop with the ReaderControl in a similar way to the MapControl and PageLayoutControl.

While the ArcReaderControl is not available with ArcGIS Engine, it is mentioned here due to its similarity with the ReaderControl; the ArcReaderControl has the same simple self-contained object model as the ReaderControl. However, the ArcReaderControl cannot be used as a 'buddy control' to work in conjunction with the TOCControl or ToolbarControl, nor can developers access any underlying ArcObjects. Developing with the ArcReaderControl requires the ArcGIS Publisher extension and applications built with the ArcReaderControl can be deployed on to any machine that has the free ArcReader desktop application.



Identify from: Cities

Location: 411,025.43 / -268,009.64 Meters

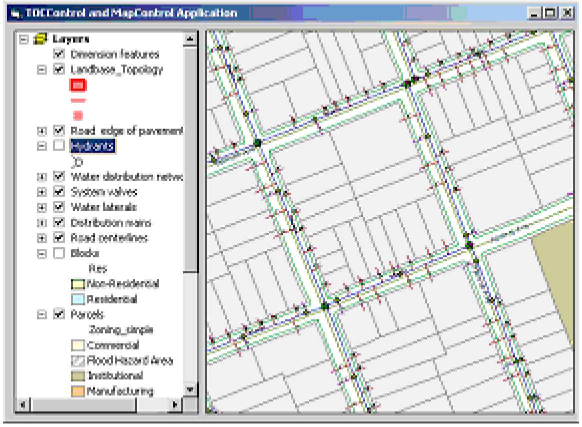
Field	Value
% Adults Completed University	-99%
1996 Population	-99
Area (km)	-99
Average Home Value 1996	-£99.00
Avg. Income 1996 (working persons)	-£99.00
Births 1996	-99
City Name	Sydney

Identified 49 features

A ReaderControl application

TOCCONTROL

The TOCControl works in conjunction with a 'buddy control'. The 'buddy control' can be a MapControl, PageLayoutControl, ReaderControl, SceneControl, or GlobeControl. The 'buddy control' can be set at design time though the TOCControl property pages (in development environments that support property page capability) or programmatically using the *SetBuddyControl* method when the container hosting the TOCControl is displayed. Each TOCControl 'buddy control' implements the *ITOCBuddy* interface. The TOCControl uses the 'buddy control' to display an interactive tree view of its map, layer and symbology contents, and to keep its contents synchronized with the 'buddy control'. For example, if the TOCControl has a MapControl as its 'buddy', and a map layer is removed from the MapContol, the map layer will also be removed from the TOCControl. Likewise, if the end user interacts with the TOCControl to uncheck a map layers visibility, the layer will no longer be visible within the MapControl.

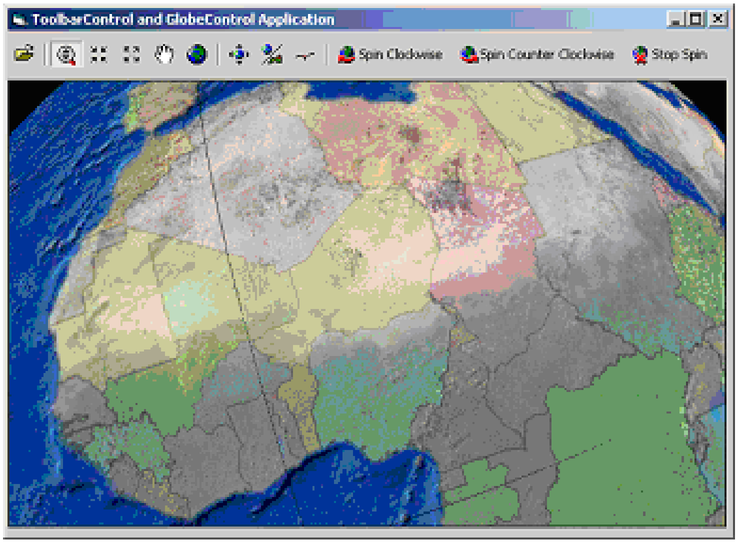


A TOCControl application

TOOLBARCONTROL

The ToolbarControl works in conjunction with a 'buddy control'. The 'buddy control' can be a MapControl, PageLayoutControl, ReaderControl, SceneControl, or GlobeControl. The 'buddy control' can be set at design time though the ToolbarControl property pages (in development environments that support property page capability) or programmatically using the *SetBuddyControl* method when the container hosting the ToolbarControl is displayed. The ToolbarControl hosts a panel of commands, tools, tool controls, and menus that work with the display of the 'buddy control'.

An application that uses the ToolbarControl and the GlobeControl



Each ToolbarControl 'buddy control' implements the *IToolbarBuddy* interface. This interface is used to set the *CurrentTool* property of the 'buddy control'. For example, imagine a ToolbarControl that is hosting a 'Page Zoom In' tool and has a PageLayoutControl as its 'buddy'. When the end user clicks on the 'Page Zoom In' tool on the ToolbarControl it will become the *CurrentTool* of the PageLayoutControl. The implementation of the 'Page Zoom In' tool will query the ToolbarControl to access its 'buddy control' (the PageLayoutControl) and retrieve the PageLayout. It will then provide the implementation for displaying the rectangle dragged by the end user and changing the extent of the PageLayout.

The ToolbarControl is typically used in conjunction with a 'buddy control' and a selection of the control commands to quickly provide a functional GIS application. The ToolbarControl is not only providing a part of the user interface, it is also providing a part of the applications framework. ArcGIS Desktop applications like ArcMap, ArcGlobe, and ArcScene have a powerful and flexible framework that include user interface components such as toolbars, commands, menus, dockable windows, and status bars. This framework enables the end user to customize the application by allowing them to reposition, add, and remove most of these user interface components.

Many development environments provide some pieces of a framework in the form of simple dialogs, forms, and multiple docking interface (MDI) applications. They also provide generic user interface components like buttons, status bars, and list boxes. However, a substantial amount of coding can still be required to provide toolbars and menus that host commands, especially if they need to be customized by the end user.

The ToolbarControl and the objects within its library can supply pieces of a framework similar to the ArcGIS Desktop application framework. The developer can use some or all of these framework pieces when building an application with the ToolbarControl.

COMMANDS

ArcGIS Engine provides several suites of control commands that work with the ArcGIS Controls to perform some specific action. Developers can extend this suite of control commands by creating their own customized commands that perform some specific piece of work. All of these command objects implement the *ICommand* interface that is used by the ToolbarControl to call methods and access properties at appropriate times.

The *ICommand::OnCreate* method is called shortly after the *Command* object is hosted on the ToolbarControl. The method is passed a handle or 'hook' to the application that the command will work with. The implementation of a command normally tests to see if the 'hook' object is supported (that is, the command tests to see that the 'hook' is an object that the command can work with). If the 'hook' is not supported the command disables itself. If the 'hook' is supported the command stores the 'hook' for later use. For example, if an 'Open Map Document' command is to work with the MapControl or PageLayoutControl and they are passed to the *OnCreate* method as the 'hook', the command will store the 'hook' for later use. If the ToolbarControl is passed to the *OnCreate* event as the 'hook', the command would normally check the type of 'buddy control' being used in conjunction with the ToolbarControl using the Buddy property. For example, if a command hosted on the ToolbarControl only works with the ReaderControl and the ToolbarControl 'buddy' is a MapControl, the command should disable itself.

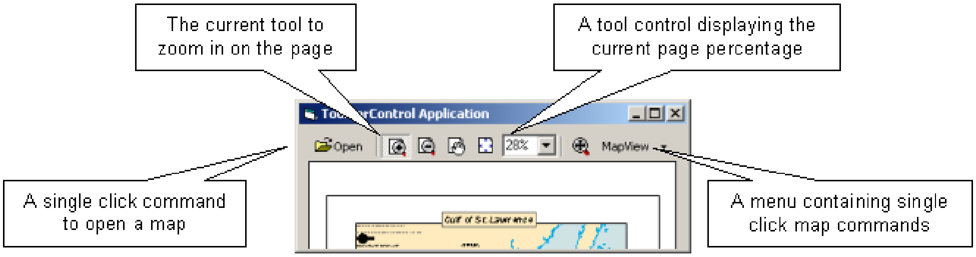
To help developers create custom commands to work with the ArcGIS Controls and the ArcGIS desktop applications, *HookHelper*, *GlobeHookHelper*, and *SceneHookHelper* objects exist.

- The *HookHelper* is used for custom commands that work with the MapControl, PageLayoutControl, ToolbarControl, and the ArcMap desktop application.
- The *SceneHookHelper* is used for custom commands that work with the SceneControl, ToolbarControl, and the ArcScene desktop application.
- The *GlobeHookHelper* is used for custom commands that work with the GlobeControl, ToolbarControl, and the ArcGlobe desktop application.

Rather than the developer adding code into a commands *OnCreate* method to determine the type of 'hook' passed to the command, the helper object handles this. The helper objects are used to hold into the 'hook' and return *ActiveView*, *PageLayout*, *Map*, *Globe*, and *Scene* objects (depending on the type of helper object) regardless of the type of 'hook' that is passed. Refer to the 'Building Applications' scenario in Chapter 6, 'Developer Scenarios' to see how to build a custom command using the *HookHelper* object that works with a MapControl, PageLayoutControl, and ToolbarControl.

The *ICommand::OnClick* method is called when the end user clicks on a command item hosted on the ToolbarControl. Depending on the type of command it will typically do some work using the 'hook' to access the required objects from the 'buddy control'. There are three types of command:

- A single click command implementing the *ICommand* interface that responds to a single click. A click results in a call to the *ICommand::OnClick* method, and some action is performed. By changing the *ICommand::Checked* value simple command items can behave like a toggle. Single click commands are the only types of command that can be hosted on a menu.
- A command item or tool implementing both the *ICommand* and *ITool* interfaces, that requires end user interaction with the display of the 'buddy control'. The ToolbarControl maintains one *CurrentTool*. When the end user clicks the tool on the ToolbarControl it becomes the *CurrentTool*, and the previous tool is deactivated. The ToolbarControl will set the *CurrentTool* of the 'buddy control'. While the tool is the *CurrentTool* it will receive mouse and key events from the 'buddy control'.
- A command item or tool control implementing both the *ICommand* and *IToolControl* interfaces. This is typically a user interface component like a Listbox or ComboBox hosted on the ToolbarControl. The ToolbarControl hosts a small window supplied by a window handle from the *IToolControl::hWnd* property. Only a single instance of a particular tool control can be added to the ToolbarControl.



The current tool to zoom in on the page

A tool control displaying the current page percentage

A single click command to open a map

A menu containing single click map commands

Commands can be added to ToolbarControl in two ways, firstly, by specifying a *UID* object that uniquely identifies a command (using a GUID), or secondly by supplying an instance of an existing *Command* object to the *AddItem* method. Where possible commands should be added to the ToolbarControl by specifying a UID. If a UID is supplied, the ToolbarControl can identify whether this command has previously been added, and if so can reuse the previous instance of the command. When an existing instance of a *Command* object is added to the ToolbarControl there is no unique identifier for the command, and multiple instances of the same command can exist on the ToolbarControl.

TOOLBARITEM

A ToolbarItem is a single command or menu hosted on a ToolbarControl or ToolbarMenu. The *IToolbarItem* interface has properties to determine the appearance of the item to the end user. For example, whether the item has a vertical line to its left signifying that it begins a *Group* and whether the *Style* of the item displays with a bitmap, a caption, or both. The *Command* and *Menu* properties return the actual command or menu that the ToolbarItem represents.

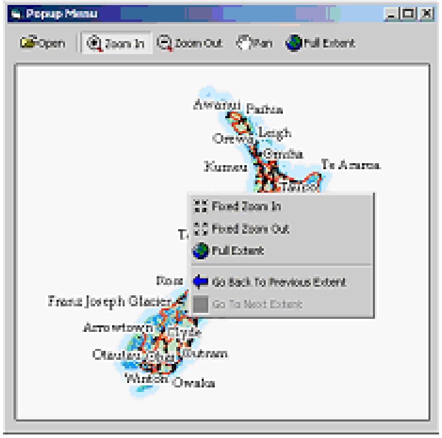
UPDATING COMMANDS

By default the ToolbarControl updates itself automatically every half a second, to ensure that the appearance of each ToolbarItem hosted on the ToolbarControl is synchronised with the *Enabled*, *Bitmap*, and *Caption* properties of its underlying command. Changing the *UpdateInterval* property can alter the frequency of the update. An *UpdateInterval* of 0 will stop any updates from happening automatically, and the developer must call the *Update* method programmatically to refresh the state of each ToolbarItem.

The first time the *Update* method is called in an application, the ToolbarControl will check whether the *ICommand::OnCreate* method of each ToolbarItem's underlying command has been called. If the method has not been called the ToolbarControl is automatically passed as the 'hook' to the *ICommand::OnCreate* method.

TOOLBARMENU

The ToolbarControl can host an item that is a drop down menu. A ToolbarMenu item presents a vertical list of single click command items. The user must select one of the command items on the ToolbarMenu, or click outside of the ToolbarMenu to make it disappear. A ToolbarMenu can only host command items (no tools or tool controls are permitted). The ToolbarMenu itself can be either hosted on the ToolbarControl, hosted on another ToolbarMenu as a 'sub menu', or it can appear as a 'popup menu' and used for a right click context menu. Refer to the 'Building Applications' scenario in Chapter 6, 'Developer Scenarios' to see how to build a 'popup menu' hosting some control commands that work with the PageLayoutControl.



Hosting the ToolbarMenu as a popoup

COMMAND POOL

Each ToolbarControl and ToolbarMenu has a *CommandPool* that is used to manage the collection of *Command* objects that it is using. Normally, a developer will not interact with the *CommandPool*. When a command is added to the ToolbarControl either through the property pages of the ToolbarControl or programmatically the command is automatically added to the *CommandPool*. *Command* objects are added to the *CommandPool* either as a *UID* object that uniquely identifies the command (using a GUID) or as an existing instance of a *Command* object.

If an existing instance of a *Command* object is added there is no unique identifier for the command and multiple instances of the same command can exist in the *CommandPool*. If a *UID* object is supplied, the *CommandPool* can identify whether the command already exists in the *CommandPool*, and if so can reuse the previous instance of the command. The *CommandPool* manages this by tracking whether the *OnCreate* method of a command has been called. If the *OnCreate* method has been called it will reuse the command and increment its UsageCount.

For example, if a 'Zoom In' tool is added to a ToolbarControl twice, with the UID supplied, when one of the 'Zoom In' items on the ToolbarControl is selected and appears 'pressed' the other 'Zoom In' item will also appear 'pressed' because they are both using the same *Command* object. When an application contains multiple ToolbarControls or ToolbarMenus the developer should ensure each ToolbarControl and ToolbarMenu uses the same *CommandPool*, to ensure that only one instance of a command is created in the application.

CUSTOMIZATION

The ToolbarControl has a *Customize* property that can be set to put the ToolbarControl into customize mode. This changes the behavior of the ToolbarControl and allows the end user to rearrange, remove, and add items as well as change their appearance.

- Use the left mouse button to select an item on the ToolbarControl, either drag the selected item to a new position or drag and drop the item off the ToolbarControl to remove it.

- Use the right mouse button to select an item and display a customize menu. The customize menu can be used to remove the item or change the Style (bitmap, caption or both) and Grouping of the ToolbarItem.

While the ToolbarControl is in Customize mode the developer can programmatically launch the modeless CustomizeDialog. The CustomizeDialog lists all of the control commands, together with any custom commands, toolsets, and menus. It does this by reading entries from the 'ESRI Controls Commands', 'ESRI Controls Toolbars,' and 'ESRI Controls Menus' component categories. If required the developer can change the CustomizeDialog to use alternative component categories. The end user can add these commands, toolsets and menus to the ToolbarControl either by dragging and dropping them onto the ToolbarControl or double clicking on them.

The CustomizeDialog is modeless to allow the user to interact with the ToolbarControl. When the CustomizeDialog is launched with the *StartDialog* method, the method call returns immediately while the CustomizeDialog remains open on the screen. In order to keep a reference to the CustomizeDialog while it is open, it is sensible practice to store a class level variable to the CustomizeDialog and to listen to its *ICustomizeDialogEvents*. Refer to the 'Building Applications' scenario in Chapter 6, 'Developer Scenarios' to see how to display the CustomizeDialog when the ToolbarControl is in Customize mode.

OPERATION STACK

The ToolbarControl has an OperationStack that is used to manage undo and redo functionality. Operations are added to the operation stack by each ToolbarItem's underlying command, so that the operation can be rolled forward and then rolled back as desired. For example, when a graphic element is moved, the operation can be undone by moving the graphic back to its original location. Whether or not a command makes use of an OperationStack depends upon its implementation. Typically a developer creates a single ControlsOperationStack for an application (by default the OperationStack property is Nothing) and sets it into each ToolbarControl. Undo and Redo commands can be added to the ToolbarControl that step through the OperationStack.

While building applications with the ToolbarControl can quickly provide pieces of a framework similar to the ArcGIS Desktop application framework, there are times when the ToolbarControl is not required for an application:

- the visual appearance of the ToolbarControl may not match that of the application
- the overhead of implementing *Command* objects for the ToolbarControl is not required
- there is an existing application framework present in the application
- the ToolbarControl and the commands it hosts do not easily work across multiple 'buddy controls'

In such circumstances, the developer must work directly with the MapControl, PageLayoutControl, SceneControl, GlobeControl, or ReaderControl. Any additional user interface components needed by the application such as command buttons, status bars, and list boxes may be supplied by development environment.

For example, building map navigation functionality into a MapControl application can be achieved by:

- setting the resulting *Envelope* of the *IMapControl2::TrackRectangle* method into the *IMapControl2::Extent* property within the MapControl's *OnMouseDown* event to create 'Zoom In' functionality.
- setting the *Envelope* of the *IMapControl2::FullExtent* property into the *IMapControl2::Extent* property to create 'Full Extent' functionality. This code could be placed within the *Click* event of command button supplied by the development environment.

Alternatively, the controls commands that are provided with ArcGIS Engine, or any custom commands that make use of either the *HookHelper*, *SceneHookHelper*, or *GlobeHookHelper* objects will work directly with an individual ArcGIS Control. However, the developer becomes responsible for calling *ICommand::OnCreate* and *ICommand::OnClick* methods at the appropriate times and reading properties on the *ICommand* interface to build up the user interface as follows:

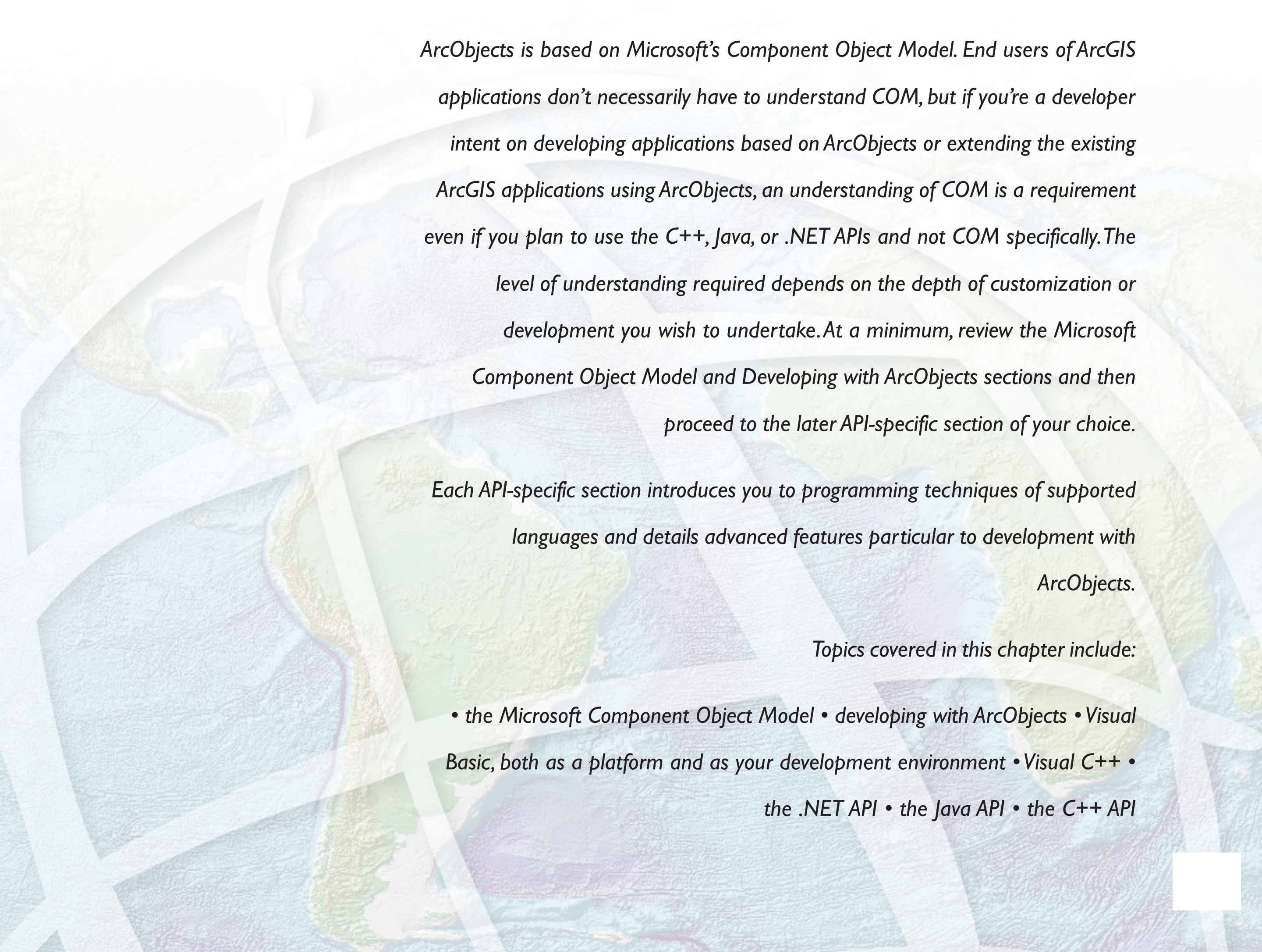
- A new instance of a command is created programmatically and the individual ArcGIS Control is passed to the *OnCreate* event. For example, if the 3D 'Zoom FullExtent' command is to work with the GlobeControl, the GlobeControl must be passed as the 'hook' to the *OnCreate* method.
- A developer can use the *CommandPool* object without the ToolbarControl in order to manage the commands used by an application. The *CommandPool* will provide support for calling the *OnCreate* method of each command based on its *Hook* property.
- If the command only implements the *ICommand* interface, the developer can call the *OnClick* method at the appropriate time to perform the specific action. If the command is a tool that implements both the *ICommand* and *ITool* interfaces the developer must set the tool to be the *CurrentTool* in the ArcGIS

Control. The ArcGIS Control will send any keyboard and mouse events to the tool.

- A Command's *Enabled*, *Caption*, and *Bitmap* properties can be read and set into the properties of a command button supplied by the development environment to build up the user interface of the application.

While this approach to building applications requires more programming on the behalf of the developer, building from scratch does allow more flexibility.

4 Developer environments



ArcObjects is based on Microsoft's Component Object Model. End users of ArcGIS applications don't necessarily have to understand COM, but if you're a developer intent on developing applications based on ArcObjects or extending the existing ArcGIS applications using ArcObjects, an understanding of COM is a requirement even if you plan to use the C++, Java, or .NET APIs and not COM specifically. The level of understanding required depends on the depth of customization or development you wish to undertake. At a minimum, review the Microsoft Component Object Model and Developing with ArcObjects sections and then proceed to the later API-specific section of your choice.

Each API-specific section introduces you to programming techniques of supported languages and details advanced features particular to development with ArcObjects.

Topics covered in this chapter include:

• the Microsoft Component Object Model • developing with ArcObjects • Visual Basic, both as a platform and as your development environment • Visual C++ • the .NET API • the Java API • the C++ API

Before discussing COM specifically, it is worth considering the wider use of software components in general. There are a number of factors driving the motivation behind software components, but the principal one is the fact that software development is a costly and time-consuming venture.

In an ideal world, it should be possible to write a piece of code once and then reuse it again and again using a variety of development tools, even in circumstances that the original developer did not foresee. Ideally, changes to the code's functionality made by the original developer could be deployed without requiring existing users to change or recompile their code.

Early attempts at producing reusable chunks of code revolved around the creation of class libraries, usually developed in C++. These early attempts suffered from several limitations, notably difficulty of sharing parts of the system (it is very difficult to share binary C++ components—most attempts have only shared source code), problems of persistence and updating C++ components without recompiling, lack of good modeling languages and tools, and proprietary interfaces and customization tools.

To counteract these and other problems, many software engineers have adopted component-based approaches to system development. A software component is a binary unit of reusable code.

Several different but overlapping standards have emerged for developing and sharing components. For building interactive desktop applications, Microsoft's COM is the de facto standard. On the Internet, JavaBeans is viable technology. At a coarser grain appropriate for application-level interoperability, the Object Management Group (OMG) has specified the common object request broker architecture (CORBA).

ESRI chose COM as the component technology for ArcGIS because it is a mature technology that offers good performance, many of today's development tools support it, and there are a multitude of third-party components that can be used to extend the functionality of ArcObjects.

To understand COM (and therefore all COM-based technologies), it's important to realize that it isn't an object-oriented language but a protocol or standard. COM is more than just a technology; it is a methodology of software development. COM defines a protocol that connects one software component, or module, with another. By making use of this protocol, it's possible to build reusable software components that can be dynamically interchanged in a distributed system.

COM also defines a programming model, known as interface-based programming. Objects encapsulate the manipulation methods and the data that characterize each instantiated object behind a well-defined interface. This promotes structured and safe system development since the client of an object is protected from knowing any of the details of how a particular method is implemented. COM doesn't specify how an application should be structured. As an application programmer working with COM, language, structure, and implementation details are left up to you.

The key to the success of components is that they implement, in a very practical way, many of the object-oriented principles now commonly accepted in software engineering. Components facilitate software reuse because they are self-contained building blocks that can easily be assembled into larger systems.

COM does specify an object model and programming requirements that enable COM objects to interact with other COM objects. These objects can be within a single process, in other processes, or even on remote machines. They can be written in other languages and may have been developed in very different ways. That is why COM is referred to as a binary specification or standard—it is a standard that applies after a program has been translated to binary machine code.

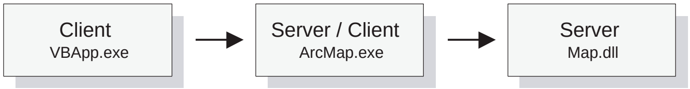
COM allows these objects to be reused at a binary level, meaning that third party developers do not require access to source code, header files, or object libraries in order to extend the system even at the lowest level.

COMPONENTS, OBJECTS, CLIENTS, AND SERVERS

Different texts use the terms components, objects, clients, and servers to mean different things. (To add to the confusion, various texts refer to the same thing using all of these terms.) Therefore, it is worthwhile to define the terminology that this book will use.

COM is a client/server architecture. The server (or object) provides some functionality, and the client uses that functionality. COM facilitates the communication between the client and the object. An object can at the same time be a server to a client and be a client of some other object's services.

Objects are instances of COM classes that make services available for use by a client. Hence it is normal to talk of clients and objects instead of clients and servers. These objects are often referred to as COM objects and component objects. This book will refer to them simply as objects.



The client and its servers can exist in the same process or in a different process space. In-process servers are packaged in Dynamic Link Library (DLL) form, and these DLLs are loaded into the client's address space when the client first accesses the server. Out-of-process servers are packaged in executables (EXE) and run in their own address space. COM makes the differences transparent to the client.

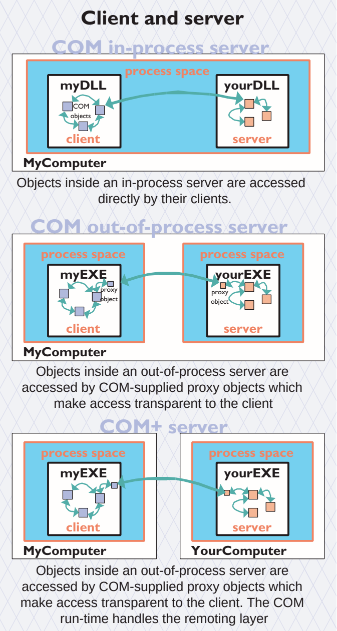
When creating COM objects, the developer must be aware of the type of server that the objects will reside in, but if the creator of the object has implemented them correctly, the packaging does not affect the use of the objects by the client.

There are pros and cons to each method of packaging that are symmetrically opposite. DLLs are faster to load into memory, and calling a DLL function is faster. EXEs, on the other hand, provide a more robust solution (if the server fails, the client will not crash), and security is better handled since the server has its own security context.

In a distributed system, EXEs are more flexible, and it does not matter if the server has a different byte ordering from the client. The majority of ArcObjects servers are packaged as in-process servers (DLLs). Later, you will see the performance benefits associated with in-process servers.

In a COM system, the client, or user of functionality, is completely isolated from the provider of that functionality, the object. All the client needs to know is that the functionality is available; with this knowledge, the client can make method calls to the object and expect the object to honor them. In this way, COM is said to act as a contract between client and object. If the object breaks that contract, the behavior of the system will be unspecified. In this way, COM development is based on trust between the implementer and the user of functionality.

In the ArcGIS applications there are many objects that provide, via their interfaces, thousands of properties and methods. When you use the ESRI object libraries you can assume that all these properties and interfaces have been fully implemented, and if they are present on the object diagrams, they are there to use.



Client and server

Objects inside an in-process server are accessed directly by their clients.

Objects inside an out-of-process server are accessed by COM-supplied proxy objects which make access transparent to the client

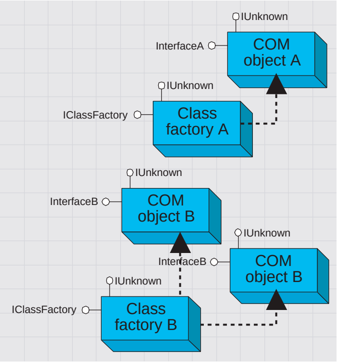
Objects inside an out-of-process server are accessed by COM-supplied proxy objects which make access transparent to the client. The COM run-time handles the remoting layer

CLASS FACTORY

Within each server there is an object called a class factory that the COM runtime interacts with in order to instantiate objects of a particular class. For every corresponding COM class there is a class factory. Normally, when a client requests an object from a server, the appropriate class factory creates a new object and passes out that object to the client.

SINGLETON OBJECTS

While this is the normal implementation, it is not the only implementation possible. The class factory can also create an instance of the object the first time and, with subsequent calls, pass out the same object to clients. This type of implementation creates what is known as a singleton object since there is only one instance of the object per process.



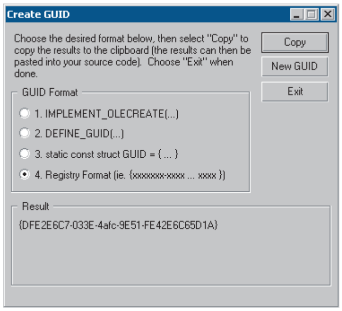
A server *is a binary file that contains all the code required by one or more COM classes. This includes both the code that works with COM to instantiate objects into memory and the code to perform the methods supported by the objects contained within the server.*

GLOBALLY UNIQUE IDENTIFIERS

A distributed system potentially has many thousands of interfaces, classes, and servers, all of which must be referenced when locating and binding clients and objects together at runtime. Clearly, using human-readable names would lead to the potential for clashes, hence COM uses Globally Unique Identifiers (GUIDs), 128 bit numbers that are virtually guaranteed to be unique in the world. It is possible to generate 10 million GUIDs per second until the year 5770 A.D., and each one would be unique.

The COM API defines a function that can be used to generate GUIDs; in addition, all COM-compliant development tools automatically assign GUIDs when appropriate. GUIDs are the same as Universally Unique Identifiers (UUIDs), defined by the Open Group's Distributed Computing Environment (DCE) specification. Below is a sample GUID in registry format.

```
{E6BDAA76-4D35-11D0-98BE-00805F7CED21}
```



Choose the desired format below, then select "Copy" to copy the results to the clipboard (the results can then be pasted into your source code). Choose "Exit" when done.

GUID Format

○ 1. IMPLEMENT_OLECREATE(...)
○ 2. DEFINE_GUID(...)
○ 3. static const struct GUID = { ... }
● 4. Registry Format (ie. {xxxxxxx-xxxx ... xxxx })

Result

{DFE2E6C7-033E-4afc-9E51-FE42E6C65D1A}

GUIDGEN.EXE is a utility that ships with Microsoft's Visual Studio and provides an easy-to-use user interface for generating GUIDs. It can be found in the directory <VS Install Dir>\Common\Tools.

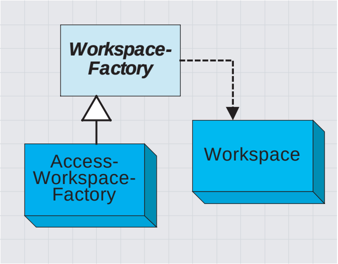
The acronym GUID is commonly pronounced "gwid".

COM CLASSES AND INTERFACES

Developing with COM means developing using interfaces, the so-called interface-based programming model. All communication between objects is made via their interfaces. COM interfaces are abstract, meaning there is no implementation associated with an interface; the code associated with an interface comes from a class implementation. The interface sets out what requests can be made of an object that chooses to implement the interface.

How an interface is implemented differs between objects. Thus the objects inherit the type of interface, not its implementation, which is called type inheritance. Functionality is modeled abstractly with the interfaces and implemented within a class implementation. Classes and interfaces are often referred to as the "what" and "how" of COM. The interface defines what an object can do, and the class defines how it is done.

COM classes provide the code associated with one or more interfaces, thus encapsulating the functionality entirely within the class. Two classes can both have the same interface, but they may implement them quite differently. By implementing these interfaces in this way, COM displays classic object-oriented polymorphic behavior. COM does not support the concept of multiple inheritance; however, this is not a shortcoming since individual classes can implement multiple inter-



This is a simplified portion of the geodatabase object model showing type inheritance among abstract classes and coclasses and instantiation of classes.

faces. See the diagram to the left on polymorphic behavior.

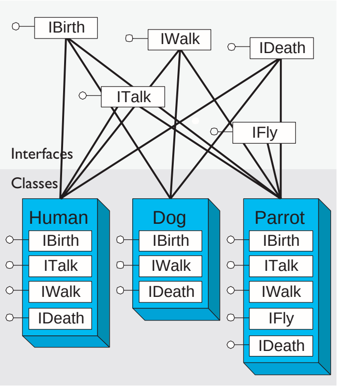
Within ArcObjects are three types of classes that the developer must be aware of: abstract classes, coclasses, and classes. An abstract class cannot be created; it is solely a specification for instances of subclasses (through type inheritance). ArcObjects Dataset or Geometry classes are examples of abstract classes. An object of type Geometry cannot be created, but an object of type Polyline can. This Polyline object in turn implements the interfaces defined within the Geometry base class, hence any interfaces defined within object-based classes are accessible from the coclass.

A coclass is a publicly creatable class. In other words, it is possible for COM to create an instance of that class and give the resultant object to the client in order for the client to use the services defined by the interfaces of that class. A class cannot be publicly created, but objects of this class can be created by other objects within ArcObjects and given to clients to use.

To the left is a diagram that illustrates the polymorphic behavior exhibited in COM classes when implementing interfaces. Notice that both the *Human* and *Parrot* classes implement the *ITalk* interface. The *ITalk* interface defines the methods and properties, such as *StartTalking*, *StopTalking*, or *Language*, but clearly the two classes implement these differently.

INSIDE INTERFACES

COM interfaces are how COM objects communicate with each other. When working with COM objects, the developer never works with the COM object directly but gains access to the object via one of its interfaces. COM interfaces are designed to be a grouping of logically related functions. The virtual functions are called by the client and implemented by the server; in this way an object's interfaces are the contract between the client and object. The client of an object is holding an interface pointer onto that object. This interface pointer is referred to as an opaque pointer since the client cannot gain any knowledge of the implementation details within an object or direct access to an object's state data. The client must communicate through the member functions of the interface. This allows COM to provide a binary standard through which all objects can effectively communicate.



This diagram shows how common behavior, expressed as interfaces, can be shared among multiple objects, animals in this example, to support polymorphism.

Interfaces allow developers to model functionality abstractly. Visual C++ developers see interfaces as a collection of pure virtual functions, while Visual Basic developers see an interface as a collection of properties, functions, and subroutines.

The concept of the interface is fundamental in COM. The COM Specification (Microsoft, 1995) emphasizes these four points when discussing COM interfaces:

1. An interface is not a class. An interface cannot be instantiated by itself since it carries no implementation.
2. An interface is not an object. An interface is a related group of functions and is the binary standard through which clients and objects communicate.

3. Interfaces are strongly typed. Every interface has its own interface identifier, thereby eliminating the possibility of a collision between interfaces of the same human-readable name.
4. Interfaces are immutable. Interfaces are never versioned. Once defined and published, an interface cannot be changed.

Once an interface has been published, it is not possible to change the external signature of that interface. It is possible at any time to change the implementation details of an object that exposes an interface. This change may be a minor bug fix or a complete reworking of the underlying algorithm; the clients of the interface do not care since the interface appears the same to them. This means that when upgrades to the servers are deployed in the form of new DLLs and EXEs, existing clients need not be recompiled to make use of the new functionality. If the external signature of the interface is no longer sufficient, a new interface is created to expose the new functions. Old or deprecated interfaces are not removed from a class to ensure all existing client applications can continue to communicate with the newly upgraded server. Newer clients will have the choice of using the old or new interfaces.

An interface's permanence is not restricted to simply its method signatures, but it extends to its semantic behavior as well. For example, an interface defines two methods, A and B, with no restrictions placed on their use. It breaks the COM contract if at a subsequent release Method A requires that Method B be executed first. A change like this would force possible recompilations of clients.

THE IUNKNOWN INTERFACE

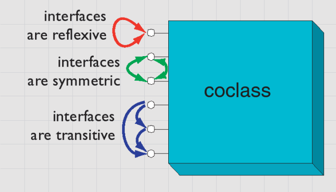
All COM interfaces derive from the *IUnknown* interface, and all COM objects must implement this interface. The *IUnknown* interface performs two tasks: it controls object lifetime and provides runtime type support. It is through the *IUnknown* interface that clients maintain a reference on an object while it is in use—leaving the actual lifetime management to the object itself.

Object lifetime is controlled with two methods, *AddRef* and *Release*, and an internal reference counter. Every object must have an implementation of *IUnknown* in order to control its own lifetime. Anytime an interface pointer is created or duplicated, the *AddRef* method is called, and when the client no longer requires this pointer, the corresponding *Release* method is called. When the reference count reaches zero, the object destroys itself.

The name IUnknown *came from a 1988 internal Microsoft paper called* Object Architecture: Dealing with the Unknown *– or –* Type Safety in a Dynamically Extensible Class Library.

Clients also use *IUnknown* to acquire other interfaces on an object. *QueryInterface* is the method that a client calls when another interface on the object is required. When a client calls *QueryInterface*, the object provides an interface and calls *AddRef*. In fact, it is the responsibility of any COM method that returns an interface to increment the reference count for the object on behalf of the caller. The client must call the *Release* method when the interface is no longer needed. The client calls *AddRef* explicitly only when an interface is duplicated.

When developing a COM object, the developer must obey the rules of *QueryInterface*. These rules dictate that interfaces for an object are symmetric, transitive, and reflexive and are always available for the lifetime of an object. For the client this means that, given a valid interface to an object, it is always valid to ask the object, via a call to *QueryInterface*, for any other interface on that object including itself. It is not possible to support an interface and later deny access to that interface, perhaps because of time or security constraints. Other mechanisms



The rules of QueryInterface *dictate that interfaces of an object are reflexive, symmetric, and transitive. It is always possible, holding a valid interface pointer on an object, to get any other interface on that object.*

must be used to provide this level of functionality. Some classes support the concept of optional interfaces. Depending on the coclass, they may optionally implement an interface; this does not break this rule since the interface is either always available or always not available on the class.

When requested for a particular interface, the *QueryInterface* method can return an already assigned piece of memory for that requested interface, or it can allocate a new piece of memory and return that. The only case when the same piece of memory must be returned is when the *IUnknown* interface is requested. When comparing two interface pointers to see if they point to the same object, it is important that a simple comparison not be performed. To correctly compare two interface pointers to see if they are for the same object, they both must be queried for their *IUnknown*, and the comparison must be performed on the *IUnknown* pointers. In this way, the *IUnknown* interface is said to define a COM object's identity.

The method QueryInterface *is often referred to by the abbreviation* QI.

It's good practice in Visual Basic to call *Release* explicitly by assigning an interface equal to *Nothing* to release any resources it's holding. Even if you don't call *Release*, Visual Basic will automatically call it when you no longer need the object—that is, when it goes out of scope. With global variables, you must explicitly call *Release*. In Visual Basic, the system performs all these reference-counting operations for you, making the use of COM objects relatively straightforward.

Since IUnknown *is fundamental to all COM objects, in general there are no references to* IUnknown *in any of the ArcObjects documentation and class diagrams.*

In C++, however, you must increment and decrement the reference count to allow an object to correctly control its own lifetime. Likewise, the *QueryInterface* method must be called when asking for another interface. In C++ the use of smart pointers simplifies much of this. These smart pointers are class based and hence have appropriate constructors, destructors, and overloaded operators to automate much of the reference counting and query interface operations.

Smart pointers are a class-based smart type and are covered in detail later in this chapter.

INTERFACE DEFINITION LANGUAGE

Microsoft Interface Definition Language (MIDL) is used to describe COM objects including their interfaces. This MIDL is an extension of the Interface Definition Language (IDL) defined by the Distributed Computing Environment (DCE), where it used to define remote procedure calls between clients and servers. The MIDL extensions include most of the Object Definition Language (ODL) statements and attributes. ODL was used in the early days of OLE Automation for the creation of type libraries.

MIDL is commonly referred to simply as IDL.

The IDL defines the public interface that developers use when working with ArcObjects. When compiled, the IDL creates a type library.

TYPE LIBRARY

A type library is best thought of as a binary version of an IDL file. It contains a binary description of all coclasses, interfaces, methods, and types contained within a server or servers.

There are several COM interfaces provided by Microsoft that work with type libraries. Two of these interfaces are *ITypeInfo* and *ITypeLib*. By utilizing these standard COM interfaces, various development tools and compilers can gain information about the coclasses and interfaces supported by a particular library.

In order to support the concept of a language-independent development set of components, all relevant data concerning the ArcObjects libraries is shipped inside type libraries. There are no header files, source files, or object files supplied or needed by external developers.

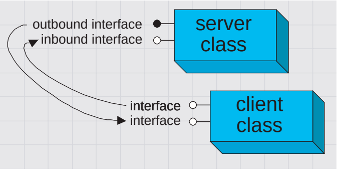
INBOUND AND OUTBOUND INTERFACES

Interfaces can be either inbound or outbound. An inbound interface is the most common kind—the client makes calls to functions within the interface contained on an object. An outbound interface is one where the object makes calls to the client—a technique analogous to the traditional callback mechanism.

There are differences in the ways these interfaces are implemented. The implementer of an inbound interface must implement all functions of the interface; failure to do so breaks the contract of COM. This is also true for outbound interfaces. If you use Visual Basic, you don't have to implement all functions present on the interface since it provides stub methods for the methods you don't implement. On the other hand, if you use C++ you must implement all the pure virtual functions to compile the class.

Connection points is a specific methodology for working with outbound COM interfaces. The connection point architecture defines how the communication between objects is set up and taken down. Connection points are not the most efficient way of initializing bidirectional object communication, but they are in common use because many development tools and environments support them.

Dispatch event interfaces



In the diagrams in this book and the ArcObjects object model diagrams, outbound interfaces are depicted with a solid circle on the interface jack.

There are some objects with ArcObjects that support two outbound event interfaces that look similar to the methods they support. An example of two such interfaces are the *IDocumentEvents* and the *IDocumentEventsDisp*. The "Disp" suffix denotes a pure Dispatch interface. These dispatch interfaces are used by VBA when dealing with certain application events, such as loading documents. A VBA programmer works with the dispatch interfaces, while a developer using another development language uses the nonpure dispatch interface. Since these dispatch event interfaces are application specific, the details are discussed in the application chapters of the book, not the framework chapter.

Default interfaces

Every COM object has a default interface that is returned when the object is created if no other interface is specified. All the objects within the ESRI object libraries have *IUnknown* as their default interface, with a few exceptions.

The reason for making IUnknown *the default interface is because the VB object browser hides information for the default interface. The fact that it hides* IUnknown *is not important for VB developers.*

The default interface of the *Application* object for both ArcCatalog and ArcMap is the *IApplication* interface. These uses of non*IUnknown* default interfaces are a requirement of Visual Basic for Applications and are found on the ArcMap and ArcCatalog application-level objects.

This means that variables that hold interface pointers must be declared in a certain way. For more details, see the coding sections later in this chapter. When COM objects are created, any of the supported interfaces can be requested at creation time.

IDispatch interface

COM supports three types of binding:

Binding is the term given to the process of matching the location of a function given a pointer to an object.

1. Late. This is where type discovery is left until runtime. Method calls made by the client but not implemented by the object will fail at execution time.
2. ID. Method IDs are stored at compile time, but execution of the method is still performed through a higher-level function.
3. Custom vTable (early). Binding is performed at compile time. The client can then make method calls directly into the object.

Binding type	In process DLL	Out of process DLL
Late binding	22,250	5,000
Custom vTable binding	825,000	20,000

This table shows the number of function calls that can be made per second on a typical Pentium® III machine.

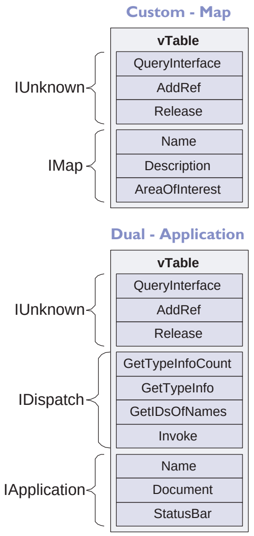
The *IDispatch* interface supports late- and ID-binding languages. The *IDispatch* interface has methods that allow clients to ask the object what methods it supports.

Assuming the required method is supported, the client executes the method by calling the *IDispatch::Invoke* method. This method, in turn, calls the required method and returns the status and any parameters back to the client on completion of the method call.

Clearly, this is not the most efficient way to make calls on a COM object. Late binding requires a call to the object to retrieve the list of method IDs; the client must then construct the call to the *Invoke* method and call it. The *Invoke* method must then unpack the method parameters and call the function.

All these steps add significant overhead to the time it takes to execute a method. In addition, every object must have an implementation for *IDispatch*, which makes all objects larger and adds to their development time.

ID binding offers a slight improvement over late binding in that the method IDs are cached at compile time, which means the initial call to retrieve the IDs is not required. However, there is still significant call overhead because the *IDispatch::Invoke* method is still called in order to execute the required method on the object.



These diagrams summarize the custom and IDispatch *interfaces for two classes in ArcObjects.The layout of the vTable displays the differences. It also illustrates the importance of implementing all methods—if one method is missing, the vTable will have the wrong layout, and hence, the wrong function pointer would be returned to the client, resulting in a system crash.*

Early binding, often referred to as custom vTable binding, does not use the *IDispatch* interface. Instead, a type library provides the required information at compile time to allow the client to know the layout of the server object. At runtime, the client makes method calls directly into the object. This is the fastest method of calling object methods and also has the benefit of compile-time type checking.

Objects that support both *IDispatch* and custom vTable are referred to as dual interface objects. The object classes within the ESRI object libraries do not implement the *IDispatch* interface; this means that these object libraries cannot be used with late-binding scripting languages such as JavaScript or VBScript since these languages require that all COM servers accessed support the *IDispatch* interface.

Careful examination of the ArcGIS class diagrams indicates that the *Application* objects support *IDispatch* because there is a requirement in VBA for the *IDispatch* interface.

All ActiveX controls support *IDispatch*. This means it is possible to use the various ActiveX controls shipped with ArcObjects to access functionality from within scripting environments.

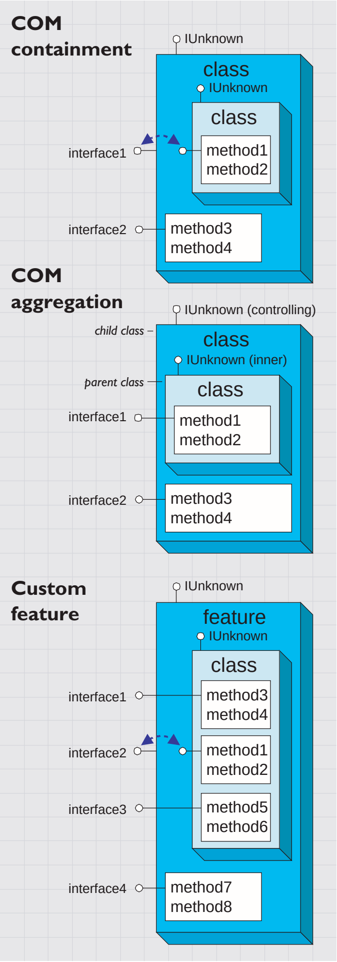
Interfaces that directly inherit from an interface other than IUnknown cannot be implemented in VB.

INTERFACE INHERITANCE

An interface consists of a group of methods and properties. If one interface inherits from another, then all of the methods and properties in the parent are directly available in the inheriting object.

The underlying principle here is interface inheritance, rather than the implementation inheritance you may have seen in languages such as SmallTalk and C++. In implementation inheritance, an object inherits actual code from its parent; in interface inheritance, it's the definitions of the methods of the object that are passed on. The coclass that implements the interfaces must provide the implementation for all inherited interfaces.

Implementation inheritance is not supported in a heterogeneous development environment because of the need to access source and header files. For reuse of code, COM uses the principles of aggregation and containment. Both of these are binary-reuse techniques.



AGGREGATION AND CONTAINMENT

For a third-party developer to make use of existing objects, using either containment or aggregation, the only requirement is that the server housing the contained or aggregated object is installed on both the developer and target release machines. Not all development languages support aggregation.

The simplest form of binary reuse is containment. Containment allows modification of the original object's method behavior but not the method's signature. With containment, the contained object (inner) has no knowledge that it is contained within another object (outer). The outer object must implement all the interfaces supported by the inner. When requests are made on these interfaces, the outer object simply delegates them to the inner. To support new functionality, the outer object can either implement one of the interfaces without passing the calls on or implement an entirely new interface in addition to those interfaces from the inner object.

COM aggregation involves an outer object that controls which interfaces it chooses to expose from an inner object. Aggregation does not allow modification of the original object's method behavior. The inner object is aware that it is being aggregated into another object and forwards any *QueryInterface* calls to the outer (controlling) object so that the object as a whole obeys the laws of COM.

To the clients of an object using aggregation, there is no way to distinguish which interfaces the outer object implements and which interfaces the inner object implements.

Custom features make use of both containment and aggregation. The developer aggregates the interfaces where no customizations are required and contains those that are to be customized. The individual methods on the contained interfaces can then either be implemented in the customized class, thus providing custom functionality, or the method call can be passed to the appropriate method on the contained interface.

Aggregation is important in this case since there are some hidden interfaces defined on a feature that cannot be contained. For more information on custom features, see Volume 2, Chapter 8, 'Accessing the geodatabase'.

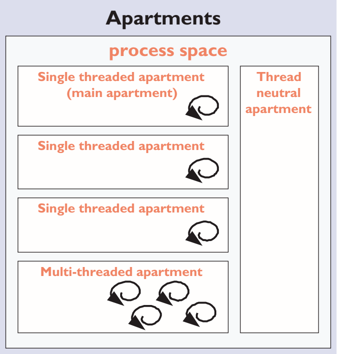
Visual Basic 6 does not support aggregation, so it can't be used to create custom features.

THREADS, APARTMENTS, AND MARSHALING

Although an understanding of apartments and threading is not essential in the use of ArcObjects, basic knowledge will help you understand some of the implications with certain development environments highlighted later in this chapter.

A thread is a process flow through an application. There are potentially many threads within Windows applications. An apartment is a group of threads that work with contexts within a process. With COM+, a context belongs to one apartment. There are potentially many types of contexts; security is an example of a type of context. Before successfully communicating with each other, objects must have compatible contexts.

COM supports two types of apartments: single-threaded apartment and multithreaded apartment (MTA). COM+ supports the additional thread-neutral apartment (TNA). A process can have any number of STAs; each process creates one STA called the main apartment. Threads that are created as apartment threaded are placed in an STA. All user-interface code is placed in an STA to prevent deadlock situations. A process can only have one MTA. A thread that is started as multithreaded is placed in the MTA. The TNA has no threads permanently associated with it; rather, threads enter and leave the apartment when appropriate.


Apartments
process space
Single threaded apartment (main apartment)
Single threaded apartment
Single threaded apartment
Multi-threaded apartment
Thread neutral apartment

Think of the SCM (pronounced scum) as the COM runtime environment. The SCM interacts with objects, servers, and the operating system and provides the transparency between clients and the objects that they work with.

In-process objects have an entry in the registry, the ThreadingModel, that informs the COM service control manager (SCM) into which apartment to place the object. If the object's requested apartment is compatible with the creator's apartment, the object is placed in that apartment; otherwise, the SCM will find or create the appropriate apartment. If no threading model is defined, the object will be placed in the main apartment of the process. The ThreadingModel registry entry can have the following values:

1. *Apartment*. Object must be executed within the STA. Normally used by UI objects.
2. *Free*. Object must be executed within the MTA. Objects creating threads are normally placed in the MTA.
3. *Both*. Object is compatible with all apartment types. The object will be created in the same apartment as the creator.
4. *Neutral*. Objects must execute in the TNA. Used by objects to ensure there is no thread switch when called from other apartments. This is only available under COM+.

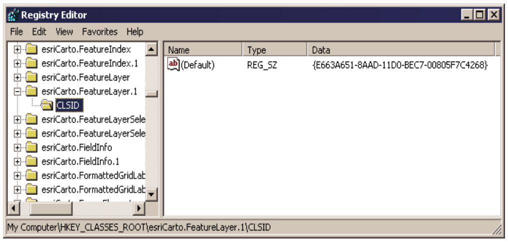
Marshaling enables a client to make interface function calls to objects in other apartments transparently. Marshaling can occur between COM apartments on different machines, between COM apartments in different process spaces, and between COM apartments in the same process space (STA to MTA, for example). COM provides a standard marshaler that handles function calls that use automation-compliant data types (see table below). Nonautomation data types can be handled by the standard marshaler as long as proxy stub code is generated; otherwise, custom-marshaling code is required.

Type	Description
Boolean	Data item that can have the value True or False
unsigned char	8-bit unsigned data item
double	64-bit IEEE floating-point number
float	32-bit IEEE floating-point number
int	Signed integer, whose size is system dependent
long	32-bit signed integer
short	16-bit signed integer
BSTR	Length-prefixed string
CURRENCY	8-byte, fixed-point number
DATE	64-bit, floating-point fractional number of days since Dec 30, 1899
SCODE	For 16-bit systems - Built-in error that corresponds to VT_ERROR
Typedef enum myenum	Signed integer, whose size is system dependent
Interface IDispatch *	Pointer to the IDispatch interface
Interface IUnknown *	Pointer to an interface that does not derive from IDispatch
dispinterface Typename *	Pointer to an interface derived from IDispatch
Coclass Typename *	Pointer to a coclass name (VT_UNKNOWN)
[oleautomation] interface Typename *	Pointer to an interface that derives from IDispatch
SAFEARRAY(TypeName)	TypeName is any of the above types. Array of these types
TypeName*	TypeName is any of the above types. Pointer to a type
Decimal	96-bit unsigned binary integer scaled by a variable power of 10. A decimal data type that provides a size and scale for a number (as in coordinates)

COMPONENT CATEGORY

Component categories are used by client applications to find all COM classes of a particular type that are installed on the system efficiently. For example, a client application may support a data export function in which you can specify the output format—a component category could be used to find all the data export classes for the various formats. If component categories are not used, the application has to instantiate each object and interrogate it to see if it supports the required functionality, which is not a practical approach. Component categories support the extensibility of COM by allowing the developer of the client application to create and work with classes that belong to a particular category. If at a later date a new class is added to the category, the client application need not be changed to take advantage of the new class; it will automatically pick up the new class the next time the category is read.

COM AND THE REGISTRY



Name	Type	Data
(Default)	REG_SZ	{E663A651-8AAD-11D0-BEC7-00805F7C4268}

My Computer\HKEY_CLASSES_ROOT\esriCarto.FeatureLayer.1\CLSID

ESRI keys in the Windows system registry

COM makes use of the Windows system registry to store information about the various parts that compose a COM system. The classes, interfaces, DLLs, EXEs, type libraries, and so forth, are all given unique identifiers (GUIDs) that the SCM uses when referencing these components. To see an example of this, run regedit, then open HKEY_CLASSES_ROOT. This opens a list of all the classes registered on the system.

COM makes use of the registry for a number of housekeeping tasks, but the most important and most easily understood is the use of the registry when instantiating COM objects into memory. In the simplest case, that of an in-process server, the steps are as follows:

1. Client requests the services of a COM object.
2. SCM looks for the requested objects registry entry by searching on the class ID (a GUID).
3. DLL is located and loaded into memory. The SCM calls a function within the DLL called *DllGetClassObject*, passing the desired class as the first argument.
4. The class object normally implements the interface *IClassFactory*. The SCM calls the method *CreateInstance* on this interface to instantiate the appropriate object into memory.
5. Finally, the SCM asks the newly created object for the interface that the client requested and passes that interface back to the client. At this stage, the SCM drops out of the equation, and the client and object communicate directly.

The function DllGetClassObject *is the function that makes a DLL a COM DLL. Other functions, such as* DllRegisterServer *and* DllUnregisterServer*, are nice to have but not essential for a DLL to function as a COM DLL.*

From the above sequence of steps, it is easy to imagine how changes in the object's packaging (DLL versus EXE) make little difference to the client of the object. COM handles these differences.

AUTOMATION

Automation is the technology used by individual objects or entire applications to provide access to their encapsulated functionality via a late-bound language. Commonly, automation is thought of as writing macros, where these macros can access many applications in order for a task to be done. ArcObjects, as already stated, does not support the IDispatch interface; hence, it cannot be used alone by an automation controller.

It is possible to instantiate an instance of ArcMap by cocreating the document object and making calls into ArcMap via the document object or one of its connected objects. There are, however, problems with this approach since the automation controller instance and the ArcMap instance are running in separate processes. Many of the objects contained within ArcObjects are process dependent, and therefore simple Automation will not work.

ArcGIS applications are built using ArcObjects and can be developed via several APIs. These include COM (VB, VC++, Delphi™, MainWin), .NET (VB.NET and C#), Java, and C++. Some APIs are more suitable than others for developing certain applications. This is briefly discussed later, but you should also read the appropriate developer guide for the product you are working with for more information and recommendations on which API to use.

The subsequent sections of this chapter cover some general guidelines and considerations when developing with ArcObjects regardless of the API. Some of the more common API languages each have a section describing the development environment, programming techniques, resources, and other issues you must consider when developing with ArcObjects.

CODING STANDARDS

Each of the language-specific sections begins with a section on coding standards for that language. These standards are used internally at ESRI and are followed by the samples that ship with the software.

For simplicity, some samples will not follow the coding standards. As an example, it is recommended that when coding in Visual Basic, all types defined within an ESRI object library are prefixed with the library name, for example, esriGeometry.IPolyline. *This is only done in samples where a name clash will occur. Omitting this text makes the code easier to understand for developers new to ArcObjects.*

To understand why standards and guidelines are important, consider that in any large software development project, there are many backgrounds represented by the team members. Each programmer has personal opinions concerning how code should look and be built. If each programmer engineers code differently, it becomes increasingly difficult to share work and ideas. On a successful team, the developers adapt their coding styles to the tone set by the group. Often, this means adapting one's code to match the style of existing code in the system.

Initially, this may seem burdensome, but adopting a uniform programming style and set of techniques invariably increases software quality. When all the code in a project conforms to a standard set of styles and conventions, less time is wasted learning the particular syntactic quirks of individual programmers, and more time can be spent reviewing, debugging, and extending the code. Even at a social level, uniform style encourages team-oriented, rather than individualist, outlooks—leading to greater team unity, productivity, and ultimately, better software.

GENERAL CODING TIPS AND RESOURCES

This section on general coding tips will benefit all developers working with ArcObjects no matter what language they are using. Code examples are shown in VBA, however.

Class diagrams

Getting help with the object model is fundamental to successfully working with ArcObjects. The appendix, 'Reading the object model diagrams', provides a detailed introduction of the class diagrams and shows many of the common routes through the objects. The class diagrams are most useful if viewed in the early learning process in printed form. This allows developers to appreciate the overall structure of the object model implemented by ArcObjects. When you are comfortable with the overall structure, the PDF files included with the software distribution can be more effective to work with. The PDF files are searchable; you can use the Search dialog box in Acrobat Reader to find classes and interfaces quickly.

Object browsers

In addition to the class diagram PDF files, the type library information can be viewed using a number of object browsers depending on your development platform.

Visual Basic and .NET have built-in object browsers; OLEView (a free utility from Microsoft) also displays type library information. The best object viewer to use in this environment is the ESRI object viewer. This object viewer can be used to view type information for any type library that you reference within it. Information on the classes and interfaces can be displayed in Visual Basic, Visual C++, or object diagram format. The object browsers can view coclasses and classes but cannot be used to view abstract classes. Abstract classes are only viewable on the object diagrams, where their use is solely to simplify the models.

Java and C++ developers should refer to the ArcObjects JavaDoc or ArcGIS Developer Help.

Component help

All interfaces and coclasses are documented in the component help file. Ultimately, this will be the help most commonly accessed when you get to know the object models better.

For Visual Basic and .NET developers this is a compiled HTML file that can be viewed by itself or when using an IDE. If the cursor is over an ESRI type when the F1 key is pressed, the appropriate page in the ArcObjects Class Help in the ArcGIS Developer Help system is displayed in the compiled HTML viewer.

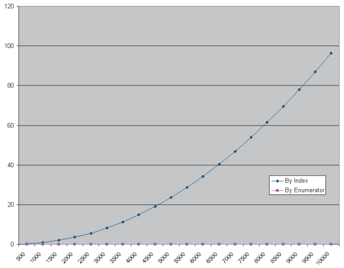
For Java and C++ developers, refer to ArcObjects JavaDoc or ArcGIS Developer Help system

Code wizards

There are a number of code generation wizards available to help with the creation of boiler plate code in Visual Basic, Visual C++, and .NET. While these wizards are useful in removing the tediousness in common tasks, they do not excuse you as the developer from understanding the underlying principles of the generated code. The main objective should be to read the accompanying documentation and understand the limitations of these tools.

Indexing of collections

All collection-like objects in ArcObjects are zero-based for their indexing. This is not the case with all development environments; Visual Basic has both zero- and one-based collections. As a general rule, if the collection base is not known, assume that the collection base is zero. This ensures that a runtime error will be raised when the collection is first accessed (assuming the access of the collection does not start at zero). Assuming a base of one means the first element of a zero-based collection would be missed and an error would only be raised if the end of the collection were reached when the code is executed.



This graph shows the performance benefits of accessing a collection using an enumerator opposed to the elements index. As expected, the graph shows a classic power trend line ($y=cx^b$).

Accessing collection elements

When accessing elements of a collection sequentially, it is best to use an enumerator interface. This provides the fastest method of walking through the collection. The reason for this is that each time an element is requested by index, internally an enumerator is used to locate the element. Hence, if the collection is looped over getting each element in turn, the time taken increases by power ($y=cx^b$).

Enumerator use

When requesting an enumerator interface from an object, the client has no idea how the object has implemented this interface. The object may create a new enumerator, or it may decide for efficiency to return a previously created enumerator. If a previous enumerator is passed to the client, the position of the element pointer will be at the last accessed element. To ensure that the enumerator is at the start of the collection, the client should reset the enumerator before use.

Error handling

Exception handling is language specific, and since COM is language neutral, exceptions are not supported.

All methods of interfaces, in other words, methods callable from other objects, should handle internal errors and signify success or failure via an appropriate *HRESULT*. COM does not support passing exceptions out of interface method calls. COM supports the notion of a COM exception. A COM exception utilizes the COM error object by populating it with relevant information and returning an appropriate *HRESULT* to signify failure. Clients, on receiving the *HRESULT*, can then interrogate the COM *Error* object for contextual information about the error. Languages, such as Visual Basic, implement their own form of exception handling. For more information, see the appropriate API language that you are developing with.

Notification interfaces

There are a number of interfaces in ArcObjects that have no methods. These are known as notification interfaces. Their purpose is to inform the application framework that the class that implements them supports a particular set of functionality. For instance, the application framework uses these interfaces to determine if a menu object is a root-level menu (*IRootLevelMenu*) or a context menu (*IShortcutMenu*).

Client-side storage

Some ArcObjects methods expect interface pointers to point to valid objects prior to making the method call. This is known as client storage since the client allocates the memory needed for the object before the method call. Say you have a polygon, and you want to get its bounding box. To do this, use the *QueryEnvelope* method on *IPolygon*. If you write the following code:

```
Dim pEnv As IEnvelope
pPolygon.QueryEnvelope pEnv
```

you'll get an error because the *QueryEnvelope* method expects you (the client) to create the *Envelope*. The method will modify the envelope you pass in and return the changed one back to you. The correct code is shown below.

```
Dim pEnv As IEnvelope
Set pEnv = New Envelope
pPolygon.QueryEnvelope pEnv
```

How do you know when to create and when not to create? In general, all methods that begin with "Query", such as *QueryEnvelope*, expect you to create the object. If the method name is *GetEnvelope*, then an object will be created for you. The reason for this client-side storage is performance. Where it is anticipated that the method on an object will be called in a tight loop, the parameters need only be created once and simply populated. This is faster than creating new objects inside the method each time.

Property by value and by reference

Occasionally, you will see a property that can be set by value or by reference, meaning that it has both a *put_XXX* and a *putref_XXX* method. On first appearance this may seem odd—why does a property need to support both? A Visual C++ developer sees this as simply giving the client the opportunity to pass ownership of a resource over to the server (using the *putref_XXX* method). A Visual Basic developer will see this as quite different; indeed, it is likely because of the Visual Basic developer that both *By Reference* and *By Value* are supported on the property.

To illustrate this, assume there are two text boxes on a form, Text1 and Text2. With a *propput*, it is possible to do the following in Visual Basic:

```
Text1.text = Text2.text
```

It is also possible to write this:

```
Text1.text = Text2
```

or this:

```
Text1 = Text2
```

DISPIDs *are unique IDs given to properties and methods in order for the* IDispatch *interface to efficiently call the appropriate method using the* Invoke *method.*

All these cases make use of the *propput* method to assign the text string of text box Text2 to the text string of text box Text1. The second and third cases work because, since no specific property is stated, Visual Basic looks for the property with a *DISPID* of 0.

This all makes sense assuming that it is the text string property of the text box that is manipulated. What happens if the actual object referenced by the variable Text2 is to be assigned to the variable Text1? If there was only a *propput* method it would not be possible, hence the need for a *propputref* method. With the *propputref* method, the following code will achieve the setting of the object reference.

Notice the use of the "Set".

```
Set Text1 = Text2
```

Initializing Outbound interfaces

When initializing an Outbound interface, it is important to only initialize the variable if the variable does not already listen to events from the server object. Failure to follow this rule will result in an infinite loop.

As an example, assume there is a variable *ViewEvents* that has been dimensioned as:

```
Private WithEvents ViewEvents As Map
```

To correctly sink this event handler, you can write code within the *OnClick* event of a UI button control, like this:

```
Private Sub UIButtonControl1_Click()
  Dim pMxDoc As IMxDocument
  Set pMxDoc = ThisDocument

  ' Check to see that the map is different than what is currently connected
  If (Not ViewEvents Is pMxDoc.FocusMap) Then
    ' Sink the event since listener has not been initialized with this map
    Set ViewEvents = pMxDoc.FocusMap
  End If
End Sub
```

Notice in the above code the use of the *Is* keyword to check for object identity.

DATABASE CONSIDERATIONS

When programming against the database, there are a number of rules that must be followed to ensure that the code will be optimal. These rules are detailed below.

If you are going to edit data programmatically, that is, not use the editing tools in ArcMap, you need to follow these rules in order to ensure that custom object behavior (such as network topology maintenance or triggering of custom feature-defined methods) is correctly invoked in response to the changes your application makes to the database. You must also follow these rules in order to ensure that your changes are made within the multiuser editing (long transaction) framework.

Edit sessions

Make all changes to the geodatabase within an edit session, which is bracketed between *StartEditing* and *StopEditing* method calls on the *IWorkspaceEdit* interface found on the*Workspace* object.

This behavior is required for any multiuser update of the database. Starting an edit session gives the application a state of the database that is guaranteed not to change, except for changes made by the editing application.

In addition, starting an edit session turns on behavior in the geodatabase such that a query against the database is guaranteed to return a reference to an existing object in memory if the object was previously retrieved and is still in use.

This behavior is required for correct application behavior when navigating between a cluster of related objects while making modifications to objects. In other words, when you are not within an edit session, the database can create a new instance of a COM object each time the application requests a particular object from the database.

Edit operations

Group your changes into edit operations, which are bracketed between the *StartEditOperation* and *StopEditOperation* method calls on the *IWorkspaceEdit* interface.

You may make all your changes within a single edit operation if so required. Edit operations can be undone and redone. If you are working with data stored in ArcSDE®, creating at least one edit operation is a requirement. There is no additional overhead to creating an edit operation.

Recycling and nonrecycling cursors

Use nonrecycling search cursors to select or fetch objects that are to be updated. Recycling cursors should only be used for read-only operations, such as drawing and querying features.

Nonrecycling cursors within an edit session create new objects only if the object to be returned does not already exist in memory.

Fetching properties using query filters

Always fetch all properties of the object; query filters should always use "*". For efficient database access, the number of properties of an object retrieved from the database can be specified. As an example, drawing a feature requires only the *OID* and the *Shape* of the feature, hence the simpler renderers only retrieve these two columns from the database. This optimization speeds up drawing but is not suitable when editing features.

If all properties are not fetched, then object-specific code that is triggered may not find the properties that the method requires. For example, a custom feature developer might write code to update attributes A and B whenever the geometry of a feature changes. If only the geometry was retrieved, then attributes A and B would be found to be missing within the *OnChanged* method. This would cause the *OnChanged* method to return an error, which would cause the *Store* to return an error and the edit operation to fail.

Marking changed objects

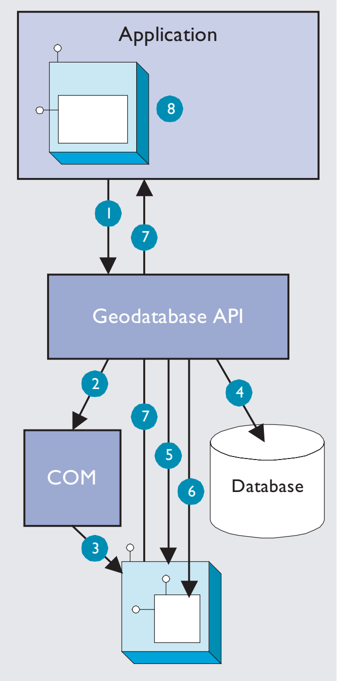
After changing an object, mark the object as changed (and ensure that it is updated in the database) by calling *Store* on the object. Delete an object by calling the *Delete* method on the object. Set versions of these calls also exist and should be used if the operation is being performed on a set of objects to ensure optimal performance.

Calling these methods guarantees that all necessary polymorphic object behavior built into the geodatabase is executed (for example, updating of network topology or updating of specific columns in response to changes in other columns in ESRI-supplied objects). It also guarantees that developer-supplied behavior is correctly triggered.

Update and insert cursors

Never use update cursors or insert cursors to update or insert objects into object and feature classes in an already loaded geodatabase that has active behavior.

Update and insert cursors are bulk cursor APIs for use during initial database loading. If used on an object or feature class with active behavior, they will bypass all object-specific behavior associated with object creation (such as topology creation) and with attribute or geometry updating (such as automatic recalculation of other dependent columns).



The diagram above clearly shows that the Feature*, which is a COM object, has another COM object for its geometry. The* Shape *property of the feature simply passes the* IGeometry *interface pointer to this geometry object out to the caller that requested the shape. This means that if more than one client requested the shape, all clients point to the same geometry object. Hence, this geometry object must be treated as read-only. No changes should be performed on the geometry returned from this property, even if the changes are temporary. Anytime a change is to be made to a feature's shape, the change must be made on the geometry returned by the* ShapeCopy *property, and the updated geometry should subsequently be assigned to the* Shape *property.*

Shape and ShapeCopy geometry property

Make use of a *Feature* object's *Shape* and *ShapeCopy* properties to optimally retrieve the geometry of a feature. To better understand how these properties relate to a feature's geometry, refer to the diagram to the left to see how features coming from a data source are instantiated into memory for use within an application.

Features are instantiated from the data source using the following sequence:

1. The application requests a *Feature* object from a data source by calling the appropriate geodatabase API method calls.
2. The geodatabase makes a request to COM to create a vanilla COM object of the desired COM class (normally this class is *esriGeoDatabase.Feature*).
3. COM creates the *Feature* COM object.
4. The geodatabase gets attribute and geometry data from a data source.
5. The vanilla *Feature* object is populated with appropriate attributes.
6. The *Geometry* COM object is created and a reference is set in the *Feature* object.
7. The *Feature* object is passed to the application.
8. The *Feature* object exists in the application until it is no longer required.

USING A TYPE LIBRARY

Since objects from ArcObjects do not implement *IDispatch*, it is essential to make use of a type library in order for the compiler to early-bind to the correct data types. This applies to all development environments, although for Visual Basic, Visual C++, and .NET, there are wizards that help you set this reference.

The type libraries required by ArcObjects are located within the ArcGIS install folder. For example, the COM type libraries can be found in the COM folder while the .NET Interop assemblies are within the DotNet folder. Many different files can contain type library information, including EXEs, DLLs, OLE custom controls (OCXs), and object libraries (OLBs).

COM DATA TYPES

COM objects talk via their interfaces, and hence, all data types used must be supported by IDL. IDL supports a large number of data types; however, not all languages that support COM support these data types. Because of this, ArcObjects does not make use of all the data types available in IDL but limits the majority of interfaces to the data type supported by Visual Basic. The table below shows the data types supported by IDL and their corresponding types in a variety of languages.

Language	IDL	Microsoft C++	Visual Basic	Java
Base types	boolean	unsigned char	unsupported	char
	byte	unsigned char	unsupported	char
	small	char	unsupported	char
	short	short	Integer	short
	long	long	Long	int
	hyper	__int64	unsupported	long
	float	float	Single	float
	double	double	Double	double
	char	unsigned char	unsupported	char
	wchar_t	wchar_t	Integer	short
	enum	enum	Enum	int
	Interface Pointer	Interface Pointer	Interface Ref.	Interface Ref.
Extended types	VARIANT	VARIANT	Variant	ms.com.Variant
	BSTR	BSTR	String	java.lang.String
	VARIANT_BOOL	short (-1/0)	Boolean	[true/false]

Note the extended data types at the bottom of the table: *VARIANT*, *BSTR*, and *VARIANT_BOOL*. While it is possible to pass strings using data types like *char* and *wchar_t*, these are not supported in languages such as Visual Basic. Visual Basic uses *BSTRs* as its text data type. A *BSTR* is a length-prefixed wide character array, where the pointer to the array points to the text contained within it and not the length prefix. Visual C++ maps *VARIANT_BOOL* values onto 0 and –1 for the *False* and *True* values, respectively. This is different from the normal mapping of 0 and 1. Hence, when writing C++ code, be sure to use the correct macros—*VARIANT_FALSE* and *VARIANT_TRUE*—not *False* and *True*.

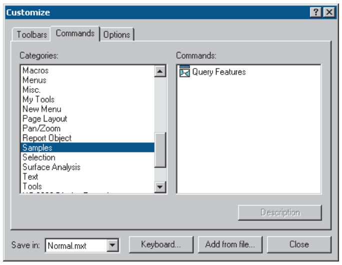
USING COMPONENT CATEGORIES

Component categories are used extensively in ArcObjects so that developers can extend the system without requiring any changes to the ArcObjects code that will work with the new functionality.

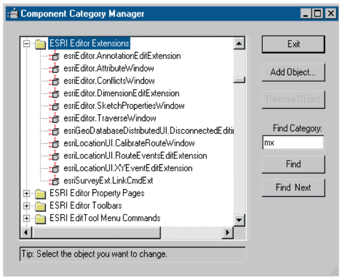
ArcObjects uses component categories in two ways. The first requires classes to be registered in the respective component category at all times, for example, ESRI Mx Extensions. Classes, if present in that component category, have an object that implements *IExtension* interface and are instantiated when the ArcMap application is started. If the class is removed from the component category, the extension will not load, even if the map document (MXD file) is referencing that extension.

The second use is when the application framework uses the component category to locate classes and display them to a user to allow some user customization to occur. Unlike the first method, the application remembers (inside its map document) the objects being used and will subsequently load them from the map document. An example of this is the commands used within ArcMap. ArcMap reads the ESRI Mx Commands category when the Customization dialog box is displayed to the user. This is the only time the category is read. Once the user selects a command and adds it to a toolbar, the map document is used to determine what commands should be instantiated. Later, when this chapter covers debugging Visual Basic code, you'll see the importance of this.

Now that you've seen two uses of component categories, you will see how to get your classes registered into the correct component category. Development environments have various levels of support for component categories; ESRI pro-



The Customize dialog box in ArcMap and ArcCatalog


Component Category Manager
ESRI Editor Extensions
esriEditor.AnnotationEditExtension
esriEditor.AttributeWindow
esriEditor.ConflictsWindow
esriEditor.DimensionEditExtension
esriEditor.SketchPropertiesWindow
esriEditor.TraverseWindow
esriGeoDatabaseDistributedUI.DisconnectedEditi
esriLocationUI.CalibrateRouteWindow
esriLocationUI.RouteEventsEditExtension
esriLocationUI.XYEventEditExtension
esriSurveyExt.LinkCmdExt
ESRI Editor Property Pages
ESRI Editor Toolbars
ESRI EditTool Menu Commands
Exit
Add Object...
Remove Object
Find Category:
mx
Find
Find Next
Tip: Select the object you want to change.

The Component Category Manager

vides two ways of adding classes to a component category. The first can only be used for Commands and command bars that are added to either ArcMap or ArcCatalog. Using the Add From File button on the Customize dialog box (shown to the left), it is possible to select a server. All classes in that server are then added to either the ESRI Gx Commands or the ESRI Mx Commands, depending on the application being customized. While this utility is useful, it is limited since it adds all the classes found in the server. It is not possible to remove classes, and it only supports two of the many component categories implemented within ArcObjects.

Distributed with ArcGIS applications is a utility application called the Component Category Manager, shown to the left. This small application allows you to add and remove classes from any of the component categories on your system, not just ArcObjects ones. Expanding a category displays a list of classes in the category. You can then use the Add Object button to display a checklist of all the classes found in the server. You check the required classes, and these checked classes are then added to the category.

Using these ESRI tools is not the only method to interact with component categories. During the installation of the server on the target user's machine, it is possible to add the relevant information to the Registry using a registry script. Below is one such script. The first line tells Windows for which version of regedit this script is intended. The last line, starting with "[HKEY_LOCAL_", executes the registry command; all the other lines are comments in the file.

```
REGEDIT4
; This Registry Script enters coclasses into their appropriate Component
Category
; Use this script during installation of the components

; Coclass: Exporter.ExportingExtension
; CLSID: {E233797D-020B-4AD4-935C-F659EB237065}
; Component Category: ESRI Mx Extensions
[HKEY_LOCAL_MACHINE\SOFTWARE\Classes\CLSID\{E233797D-020B-4AD4-935C-
F659EB237065}\Implemented Categories\{B56A7C45-83D4-11D2-A2E9-080009B6F22B}]
```

The last line in the code above is one continuous line in the script.

The last method is for the self-registration code of the server to add the relevant classes within the server to the appropriate categories. Not all development environments allow this to be set up. Visual Basic has no support for component categories, although there is an add-in that adds this functionality. See the sections on Visual Basic Developer Add-ins and Active Template Library (ATL) later in this chapter.

The tables below summarize suggested naming standards for the various elements of your Visual Basic projects.

Module Type	Prefix
Form	frm
Class	cls
Standard	bas
Project	prj

Name your modules according to the overall function they provide; do not leave any with default names (such as "Form1", "Class1", or "Module1"). Additionally, prefix the names of forms, classes, and standard modules with three letters that denote the type of module, as shown in the table above.

Control Type	Prefix
Check box	chk
Combo box	cbo
Command button	cmd
Common dialog	cdl
Form	frm
Frame	fra
Graph	gph
Grid	grd
Image	img
Image list	iml
Label	lbl
List box	lst
List view	lvw
Map control	map
Masked edit	msk
Menu	mnu
OLE client	ole
Option button	opt
Picture box	pic
Progress bar	pbr
Rich text box	rtf
Scroll bar	srl
Slider	sld
Status bar	sbr
Tab strip	tab
Text box	txt
Timer	tmr
Tool bar	tbr
Tree view	tvw

As with modules, name your controls according to the function they provide; do not leave them with default names since this leads to decreased maintainability. Use the three-letter prefixes above to identify the type of the control.

This section is intended for both VB6 and VBA developers. Differences in the development environments are clearly marked throughout the text.

USER INTERFACE STANDARDS

Consider preloading forms to increase the responsiveness of your application. Be careful not to preload too many (preloading three or four forms is fine).

Use resource files (.res) instead of external files when working with bitmap files, icons, and related files.

Make use of constructors and destructors to set variable references that are only set when the class is loaded. These are the VB functions: *Class_Initialize()* and *Class_Terminate()* or *Form_Load()* and *Form_Unload()*. Set all variables to *Nothing* when the object is destroyed.

Make sure the tab order is set correctly for the form. Do not add scroll bars to the tabbing sequence; it is too confusing.

Add access keys to those labels that identify controls of special importance on the form (use the *TabIndex* property).

Use system colors where possible instead of hard-coded colors.

Variable declaration

- Always use *Option Explicit* (or turn on Require Variable Declaration in the VB Options dialog box). This forces all variables to be declared before use and thereby prevents careless mistakes.
- Use *Public* and *Private* to declare variables at module scope and *Dim* in local scope. (*Dim* and *Private* mean the same at *Module* scope; however, using *Private* is more informative.) Do not use *Global* anymore; it is available only for backward compatibility with VB 3.0 and earlier.
- Always provide an explicit type for variables, arguments, and functions. Otherwise, they default to *Variant*, which is less efficient.
- Only declare one variable per line unless the type is specified for each variable.

This line causes *count* to be declared as a *Variant*, which is likely to be unintended.

```
Dim count, max As Long
```

This line declares both *count* and *max* as *Long*, the intended type.

```
Dim count As Long, max As Long
```

These lines also declare *count* and *max* as *Long* and are more readable.

```
Dim count As Long
Dim max As Long
```

Parentheses

Use parentheses to make operator precedence and logic comparison statements easier to read.

```
Result = ((x * 24) / (y / 12)) + 42
If ((Not pFoo Is Nothing) And (Counter > 200)) Then
```

Use the following notation for naming variables and constants:

[<libraryName.>][<scope_>]<type><name>

<name> describes how the variable is used or what it contains.The <scope> and <type> portions should always be lowercase, and the <name> should use mixed case.

Library Name	Library
esriGeometry	ESRI Object Library
stdole	Standard OLE COM Library
<empty>	Simple variable data type

<libraryName>

Prefix	Variable scope
c	constant within a form or class
g	public variable defined in a class form or standard module
m	private variable defined in a class or form
<empty>	local variable

<scope>

Prefix	Data Type
b	Boolean
by	byte or unsigned char
d	double
fn	function
h	handle
i	int (integer)
l	long
p	a pointer
s	string

<type>

Order of conditional determination

Visual Basic, unlike languages such as C and C++, performs conditional tests on all parts of the condition, even if the first part of the condition is *False*. This means you must not perform conditional tests on objects and interfaces that had their validity tested in an earlier part of the conditional statement.

```
' The following line will raise a runtime error if pFoo is NULL
If ((Not pFoo Is Nothing) And (TypeOf pFoo.Thing Is IBar)) then
End If

' The correct way to test this code is
If (Not pFoo Is Nothing) Then
  If (TypeOf pFoo.Thing Is IBar) Then
    ' Perform action on IBar thing of Foo
  End If
End If
```

Indentation

Use two spaces for indentation or a tab width of two. Since there is only ever one editor for VB code, formatting is not as critical an issue as it is for C++ code.

Default properties

Avoid using default properties except for the most common cases. They lead to decreased legibility.

Intermodule referencing

When accessing intermodule data or functions, always qualify the reference with the module name. This makes the code more readable and results in more efficient runtime binding.

Multiple property operations

When performing multiple operations against different properties of the same object, use a *With … End With* statement. It is more efficient than specifying the object each time.

```
With frmHello
  .Caption = "Hello world"
  .Font = "Playbill"
  .Left = (Screen.Width - .Width) / 2
  .Top  = (Screen.Height - .Height) / 2
End With
```

Arrays

For arrays, never change *Option Base* to anything other than zero (which is the default). Use *LBound* and *UBound* to iterate over all items in an array.

```
myArray = GetSomeArray
For i = LBound(myArray) To UBound(myArray)
  MsgBox cstr(myArray(i))
Next I
```

Bitwise operators

Since *And*, *Or*, and *Not* are bitwise operators, ensure that all conditions using them test only for Boolean values (unless, of course, bitwise semantics are what is intended).

```
If (Not pFoo Is Nothing) Then
  ' Valid Foo do something with it
End If
```

Type suffixes

Refrain from using type suffixes on variables or function names (such as *myString$* or *Right$(myString)*), unless they are needed to distinguish 16-bit from 32-bit numbers.

Ambiguous type matching

For ambiguous type matching, use explicit conversion operators (such as *CSng*, *CDbl*, and *CStr*), instead of relying on VB to pick which one will be used.

Simple image display

Use an *ImageControl* rather than a *PictureBox* for simple image display. It is much more efficient.

Error handling

Always use *On Error* to ensure fault-tolerant code. For each function that does error checking, use *On Error* to jump to a single error handler for the routine that deals with all exceptional conditions that are likely to be encountered. After the error handler processes the error—usually by displaying a message—it should proceed by issuing one of the recovery statements shown on the table to the left.

Error handling in Visual Basic is not the same as general error handling in COM (see the section 'Working with HRESULTs').

Recovery Statement	Frequency	Meaning
Exit Sub	usually	Function failed, pass control back to caller
Raise	often	Raise a new error code in the caller's scope
Resume	rarely	Error condition removed, reattempt offending statement
Resume Next	very rarely	Ignore error and continue with next statement

Event functions

Refrain from placing more than a few lines of code in event functions to prevent highly fractured and unorganized code. Event functions should simply dispatch to reusable functions elsewhere.

Memory management

To ensure efficient use of memory resources, the following points should be considered:

- Unload forms regularly. Do not keep many forms loaded but invisible since this consumes system resources.
- Be aware that referencing a form-scoped variable causes the form to be loaded.
- Set unused objects to *Nothing* to free up their memory.
- Make use of *Class_Initialize()* and *Class_Terminate()* to allocate and destroy resources.

While Wend constructs

Avoid *While … Wend* constructs. Use the *Do While … Loop* or *Do Until ... Loop* instead because you can conditionally branch out of this construct.

```
pFoos.Reset
Set pFoo = pFoos.Next
Do While (Not pFoo Is Nothing)
  If (pFoo.Answer = "Done") Then Exit Loop
  Set pFoo = pFoos.Next
Loop
```

The Visual Basic Virtual Machine

The VBVM was called the VB Runtime in earlier versions of the software.

The Visual Basic Virtual Machine (VBVM) contains the intrinsic Visual Basic controls and services, such as starting and ending a Visual Basic application, required to successfully execute all Visual Basic developed code.

The VBVM is packaged as a DLL that must be installed on any machine wanting to execute code written with Visual Basic, even if the code has been compiled to native code. If the dependencies of any Visual Basic compiled file are viewed, the file msvbvm60.dll is listed; this is the DLL housing the Virtual Machine.

For more information on the services provided by the VBVM, see the sections 'Interacting with the *IUnknown* interface' and 'Working with HRESULTs' in this chapter.

Interacting with the IUnknown interface

The section on COM contains a lengthy section on the *IUnknown* interface and how it forms the basis on which all of COM is built. Visual Basic hides this interface from developers and performs the required interactions (*QueryInterface*, *AddRef*, and *Release* function calls) on the developer's behalf. It achieves this because of functionality contained within the VBVM. This simplifies development with COM for many developers, but to work successfully with ArcObjects, you must understand what the VBVM is doing.

Visual Basic developers are used to dimensioning variables as follows:

```
Dim pColn as New Collection  'Create a new collection object
PColn.Add "Foo", "Bar"       'Add element to collection
```

It is worth considering what is happening at this point. From a quick inspection of the code it looks like the first line creates a collection object and gives the developer a handle on that object in the form of *pColn*. The developer then calls a method on the object *Add*. Earlier in the chapter you learned that objects talk via their interfaces, never through a direct handle on the object itself. Remember, objects expose their services via their interfaces. If this is true, something isn't adding up.

What is actually happening is some "VB magic" performed by the VBVM and some trickery by the Visual Basic Editor in the way that it presents objects and interfaces. The first line of code instantiates an instance of the collection class, then assigns the default interface for that object, *_Collection*, to the variable *pColn*. It is this interface, *_Collection*, that has the methods defined on it. Visual Basic has hidden the interface-based programming to simplify the developer experience.

This is not an issue if all the functionality implemented by the object can be accessed via one interface, but it is an issue when there are multiple interfaces on an object that provides services.

The Visual Basic Editor backs this up by hiding default interfaces from the IntelliSense completion list and the object browser. By default, any interfaces that begin with an underscore, "_", are not displayed in the object browser (to display these interfaces, turn Show Hidden Member on, although this will still not display default interfaces).

You have already learned that the majority of ArcObjects have *IUnknown* as their default interface and that Visual Basic does not expose any of *IUnknown*'s methods, namely, *QueryInterface*, *AddRef*, and *Release*. Assume you have a class *Foo* that supports three interfaces, *IUnknown* (the default interface), *IFoo*, and *IBar*. This means that if you were to dimension the variable *pFoo* as below, the variable *pFoo* would point to the *IUnknown* interfaces.

```
Dim pFoo As New Foo    ' Create a new Foo object
pFoo.??????
```

Since Visual Basic does not allow direct access to the methods of *IUnknown*, you would immediately have to *QI* for an interface with methods on it that you can call. Because of this, the correct way to dimension a variable that will hold pointers to interfaces is as follows:

```
Dim pFoo As IFoo   ' Variable will hold pointer to IFoo interface
Set pFoo = New Foo ' Create Instance of Foo object and QI for IFoo
```

Now that you have a pointer to one of the object's interfaces, it is an easy matter to request from the object any of its other interfaces.

```
Dim pBar as IBar  'Dim variable to hold pointer to interface
Set pBar = pFoo   'QI for IBar interface
```

By convention, most classes have an interface with the same name as the class with an "I" prefix; this tends to be the interface most commonly used when working with the object. You are not restricted to which interface you request when instantiating an object; any supported interface can be requested, hence the code below is valid.

```
Dim pBar as IBar
Set pBar = New Foo  'CoCreate Object
Set pFoo = pBar     'QI for interface
```

Objects control their own lifetime, which requires clients to call *AddRef* anytime an interface pointer is duplicated by assigning it to another variable and to call *Release* anytime the interface pointer is no longer required. Ensuring that there are a matching number of *AddRefs* and *Releases* is important, and fortunately, Visual Basic performs these calls automatically. This ensures that objects do not "leak". Even when interface pointers are reused, Visual Basic will correctly call release on the old interface before assigning the new interface to the variable. The code below illustrates these concepts; note the reference count on the object at the various stages of code execution.

```
Private Sub VBMagic()
  ' Dim a variable to the IUnknown interface on the simple object
  Dim pUnk As IUnknown

  ' Co Create simpleobject asking for the IUnknown interface
  Set pUnk = New SimpleObject 'refCount = 1

  ' We need access to methods lets QI for a useful interface
  ' Define the interface we are to request
  Dim pMagic As ISimpleObject

  ' Perform the QI operation
  Set pMagic = punk 'refCount = 2

  ' Dim another variable to hold another interface on the object
  Dim pMagic2 As IAnotherInterface

  ' QI for that interface
  Set pMagic2 = pMagic 'refCount = 3

  ' Release the interface pointer
  Set pMagic2 = Nothing 'refCount = 2

  ' Release the interface
  Set pMagic = Nothing 'refCount = 1

  ' Now reuse the pUnk variable - what will VB do for this?
  Set pUnk = New SimpleObject 'refCount = 1, then 0, then 1

  ' Let the interface variable go out of scope and VB to tidy up
End Sub 'refCount = 0
```

See Visual Basic Magic sample on the disk for this code. You are encouraged to run the sample and step though the code. This object also uses an ATL C++ project to define the SimpleObject *and its interfaces; you are encouraged to look at this code to learn a simple implementation of a C++ ATL object.*

Often interfaces have properties that are actually pointers to other interfaces. Visual Basic allows you to access these properties in a shorthand fashion by chaining interfaces together. For instance, assume that you have a pointer to the *IFoo* interface, and that interface has a property called *Gak* that is an *IGak* interface with the method *DoSomething()*. You have a choice on how to access the *DoSomething* method. The first method is the long-handed way.

```
  Dim pGak as IGak
  Set pGak = pFoo     'Assign IGak interface to local variable
  pGak.DoSomething     'Call method on IGak interface
```

Alternatively, you can chain the interfaces and accomplish the same thing on one line of code.

```
  pFoo.Gak.DoSomething  'Call method on IGak interface
```

When looking at the sample code, you will see both methods. Normally the former method is used on the simpler samples, as it explicitly tells you what interfaces are being worked with. More complex samples use the shorthand method.

This technique of chaining interfaces together can always be used to get the value of a property, but it cannot always be used to set the value of a property. Interface chaining can only be used to set a property if all the interfaces in the chain are set by reference. For instance, the code below would execute successfully.

```
Dim pMxDoc As ImxDocument
Set pMxDoc = ThisDocument
pMxDoc.FocusMap.Layers(0).Name = "Foo"
```

The above example works because both the *Layer* of the *Map* and the *Map* of the document are returned by reference. The lines of code below would not work since the *Extent* envelope is set by value on the active view.

```
pMxDoc.ActiveView.Extent.Width = 32
```

The reason that this does not work is that the VBVM expands the interface chain in order to get the end property. Because an interface in the chain is dealt with by value, the VBVM has its own copy of the variable, not the one chained. To set the *Width* property of the extent envelope in the above example, the VBVM must write code similar to this:

```
Dim pActiveView as IActiveView
Set pActiveView = pMxDoc.ActiveView

Dim pEnv as IEnvelope
Set pEnv = pActiveView.Extent  ' This is a get by value,

PEnv.Width = 32   ' The VBVM has set its copy of the Extent and not
                 ' the copy inside the ActiveView
```

For this to work the VBVM requires the extra line below.

```
pActiveView.Extent = pEnv  ' This is a set by value,
```

Accessing ArcObjects

To find out what library an ArcObject is in, review the object model diagrams or the developer help or use the LibraryLocator tool in your developer kit tools directory.

You will now see some specific uses of the create instance and query interface operations that involve ArcObjects. To use an ArcGIS object in Visual Basic or VBA, you must first reference the ESRI library that contains that object. If you are using VBA inside of ArcMap or ArcCatalog, most of the common ESRI object libraries are already referenced for you. In standalone Visual Basic applications or components you will have to manually reference the required libraries.

You will start by identifying a simple object and an interface that it supports. In this case, you will use a *Point* object and the *IPoint* interface. One way to set the coordinates of the point is to invoke the *PutCoords* method on the *IPoint* interface and pass in the coordinate values.

```
Dim pPt As IPoint
Set pPt = New Point
pPt.PutCoords 100, 100
```

IID is short for Interface Identifier, a GUID.

The first line of this simple code fragment illustrates the use of a variable to hold a reference to the interface that the object supports. The line reads the *IID* for the *IPoint* interface from the ESRI object library. You may find it less ambiguous (as per the coding guidelines), particularly if you reference other object libraries in the same project to precede the interface name with the library name, for example:

```
Dim pPt As esriGeometry.IPoint
```

That way, if there happens to be another *IPoint* referenced in your project, there won't be any ambiguity as to which one you are referring to.

Coclass is an abbreviation of component object class.

The second line of the fragment creates an instance of the object or coclass, then performs a *QI* operation for the *IPoint* interface that it assigns to *pPt*.

A QI is required since the default interface of the object is IUnknown. *Since the pPt variable was declared as type* IPoint, *the default* IUnknown *interface was QI'd for the* IPoint *interface.*

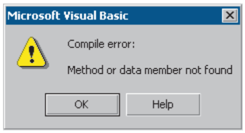
With a name for the coclass as common as *Point*, you may want to precede the coclass name with the library name, for example:

```
Set pPt = New esriGeometry.Point
```

The last line of the code fragment invokes the *PutCoords* method. If a method can't be located on the interface, an error will be shown at compile time.

Working with HRESULTs

So far you have seen that all COM methods signify success or failure via an *HRESULT* that is returned from the method; no exceptions are raised outside of the interface. You have also learned that Visual Basic raises exceptions when errors are encountered. In Visual Basic, *HRESULTs* are never returned from method calls, and to confuse you further when errors do occur, Visual Basic throws an exception. How can this be? The answer lies with the Visual Basic Virtual Machine. It is the VBVM that receives the *HRESULT*; if this is anything other than *S_OK*, the VBVM throws the exception. If it was able to retrieve any worthwhile error information from the COM error object, it populates the Visual Basic *Err* object with that information. In this way, the VBVM handles all *HRESULTs* returned from the client.


Microsoft Visual Basic

Compile error:

Method or data member not found

This is the compilation error message shown when a method or property is not found on an interface.

When implementing interfaces in Visual Basic, it is good coding practice to raise an *HRESULT* error to inform the caller that an error has occurred. Normally, this is done when a method has not been implemented.

```
' Defined in Module
Const E_NOTIMPL = &H80004001 'Constant that represents HRESULT
'Added to any method not implemented
On Error GoTo 0
Err.Raise E_NOTIMPL
```

You must also write code to handle the possibility that an *HRESULT* other than *S_OK* is returned. When this happens, an error handler should be called and the error dealt with. This may mean simply telling the user, or perhaps it may mean automatically dealing with the error and continuing with the function. The choice depends on the circumstances. Below is a very simple error handler that will catch any error that occurs within the function and report it to the user. Note the use of the *Err* object to provide the user with some description of the error.

```
Private Sub Test()
  On Error GoTo ErrorHandler
  ' Do something here
  Exit Sub    ' Must exit sub here before error handler
ErrorHandler:
  Msgbox "Error In Application - Description " & Err.Description
End Sub
```

Working with properties

Some properties refer to specific interfaces in the ESRI object library, and other properties have values that are standard data types, such as strings, numeric expressions, Boolean values, and so forth. For interface references, declare an interface variable and use the *Set* statement to assign the interface reference to the property. For other values, declare a variable with an explicit data type or use Visual Basic's *Variant* data type. Then, use a simple assignment statement to assign the value to the variable.

Properties that are interfaces can either be set by reference or set by value. Properties that are set by value do not require the *Set* statement.

```
Dim pEnv As IEnvelope
Set pEnv = pActiveView.Extent  'Get extent property of view
pEnv.Expand 0.5, 0.5, True     'Shrink envelope
pActiveView.Extent = pEnv      'Set By Value extent back on IActiveView

Dim pFeatureLayer as IfeatureLayer
Set pFeatureLayer = New FeatureLayer    'Create New Layer
Set pFeatureLayer.FeatureClass = pClass 'Set ByRef a class into layer
```

As you might expect, some properties are read-only, others are write-only, and still others are read–write. All the object browsers and the ArcObjects Class Help (found in the ArcGIS Developer Help system) provide this information. If you attempt to use a property and either forget or misuse the *Set* keyword, Visual Basic will fail the compilation of the source code with a method or "data member not found error message". This error may seem strange since it may be given for trying to assign a value to a read-only property. The reason for the message is that Visual Basic is attempting to find a method in the type library that maps to the property name. In the above examples, the underlying method calls in the type library are *put_Extent* and *putref_FeatureClass*.

Working with methods

Methods perform some action and may or may not return a value. In some instances, a method returns a value that's an interface; for example, in the code fragment below, *EditSelection* returns an enumerated feature interface:

```
Dim pApp As IApplication
Dim pEditor As IEditor
Dim pEnumFeat As IEnumFeature 'Holds the selection
Dim pID As New UID
'Get a handle to the Editor extension
pID = "esriEditor.Editor"
Set pApp = Application
Set pEditor = pApp.FindExtensionByCLSID(pID)
'Get the selection
Set pEnumFeat = pEditor.EditSelection
```

In other instances, a method returns a Boolean value that reflects the success of an operation or writes data to a parameter; for example, the *DoModalOpen* method of *GxDialog* returns a value of *True* if a selection occurs and writes the selection to an *IEnumGxObject* parameter.

Be careful not to confuse the idea of a Visual Basic return value from a method call with the idea that all COM methods must return an *HRESULT*. The VBVM is able to read type library information and set up the return value of the VB method call to be the appropriate parameter of the COM method.

Working with events

Events let you know when something has occurred. You can add code to respond to an event. For example, a command button has a *Click* event. You add code to perform some action when the user clicks the control. You can also add events that certain objects generate. VBA and Visual Basic let you declare a variable with the keyword *WithEvents*. *WithEvents* tells the development environment that the object variable will be used to respond to the object's events. This is sometimes referred to as an "event sink". The declaration must be made in a class module or a form. Here's how you declare a variable and expose the events of an object in the *Declarations* section:

```
Private WithEvents m_pViewEvents as Map
```

Visual Basic only supports one outbound interface (marked as the default outbound interface in the IDL) per coclass. To get around this limitation, the coclasses that implement more than one outbound interface have an associated dummy coclass that allows access to the secondary outbound interface. These coclasses have the same name as the outbound interface they contain, minus the I.

```
Private WithEvents m_pMapEvents as MapEvents
```

Once you've declared the variable, search for its name in the Object combo box at the top left of the Code window. Then, inspect the list of events you can attach code to in the Procedure/Events combo box at the top right of the Code window.

Not all procedures of the outbound event interface need to be stubbed out, as Visual Basic will stub out any unimplemented methods. This is different from inbound interfaces, where all methods must be stubbed out for compilation to occur.

Before the methods are called, the hookup between the event source and sink must be made. This is done by setting the variable that represents the sink to the event source.

```
Set m_pMapEvents = pMxDoc.FocusMap
```

Pointers to valid objects as parameters

Some ArcGIS methods expect interfaces for some of their parameters. The interface pointers passed can point to an instanced object before the method call or after the method call is completed.

For example, if you have a polygon (*pPolygon*) whose center point you want to find, you can write code like this:

```
Dim pArea As IArea
Dim pPt As IPoint
Set pArea = pPolygon       ' QI for IArea on pPolygon
Set pPt = pArea.Center
```

You don't need to create *pPt* because the *Center* method creates a *Point* object for you and passes back a reference to the object via its *IPoint* interface. Only methods that use client-side storage require you to create the object prior to the method call.

Passing data between modules

When passing data between modules it is best to use accessor and mutator functions that manipulate some private member variable. This provides data encapsulation, which is a fundamental technique in object-oriented programming. Public variables should never be used.

For instance, you might have decided that a variable has a valid range of 1–100. If you were to allow other developers direct access to that variable, they could set the value to an illegal value. The only way of coping with these illegal values is to check them before they get used. This is both error prone and tiresome to program. The technique of declaring all variables private member variables of the class and providing accessor and mutator functions for manipulating these variables will solve this problem.

In the example below, these properties are added to the default interface of the class. Notice the technique used to raise an error to the client.

```
Private m_lPercentage As Long

Public Property Get Percentage() As Long
  Percentage = m_lPercentage
End Property

Public Property Let Percentage(ByVal lNewValue As Long)
  If (lNewValue >= 0) And (lNewValue <= 100) Then
    m_lPercentage = lNewValue
  Else
    Err.Raise vbObjectError + 29566, "MyProj.MyObject", _
    "Invalid Percentage Value. Valid values (0 -> 100)"
  End If
End Property
```

When you write code to pass an object reference from one form, class, or module to another, for example:

```
  Private Property Set PointCoord(ByRef pPt As IPoint)
    Set m_pPoint = pPt
  End Property
```

your code passes a pointer to an instance of the *IPoint* interface. This means that you are only passing the reference to the interface, not the interface itself; if you add the *ByVal* keyword (as follows), the interface is passed by value.

```
  Private Property Let PointCoord(ByVal pPt As IPoint)
    Set m_pPoint = pPt
  End Property
```

In both of these cases the object pointed to by the interfaces is always passed by reference. In order to pass the object by value, a clone of the object must be made, and that is passed.

Using the *TypeOf* keyword

To check whether an object supports an interface, you can use Visual Basic's *TypeOf* keyword. For example, given an item selected in the ArcMap table of contents, you can test whether it is a *FeatureLayer* using the following code:

```
Dim pDoc As IMxDocument
Dim pUnk As IUnknown
Dim pFeatLyr As IGeoFeatureLayer
Set pDoc = ThisDocument
Set pUnk = pDoc.SelectedItem
If TypeOf pUnk Is IGeoFeatureLayer Then  ' can we QI for IGeoFeatureLayer?
  Set pFeatLyr = pUnk                     ' actually QI happens here
  ' Do something with pFeatLyr
End If
```

Using the *Is* operator

If your code requires you to compare two interface reference variables, you can use the *Is* operator. Typically, you can use the *Is* operator in the following circumstances:

- To check if you have a valid interface.

```
Dim pPt As IPoint
Set pPt = New Point
If (Not pPt Is Nothing) Then 'a valid pointer?
  ...' do something with pPt
End If
```

- To check if two interface variables refer to the same actual object. Say you've got two interface variables of type *IPoint*, *pPt1*, and *pPt2*. Are they pointing to the same object? If they are, then *pPt1* Is *pPt2*.

The *Is* keyword works with the COM identity of an object. Below is an example that illustrates the use of the *Is* keyword when finding out if a certain method on an interface returns a copy of or a reference to the same real object.

In the following example, the *Extent* property on a map (*IMap*) returns a copy, while the *ActiveView* property on a document (*IMxDocument*) always returns a reference to the real object.

```
Dim pDoc As IMxDocument
Dim pEnv1 As IEnvelope, pEnv2 as IEnvelope
Dim pActView1 As IActiveView
Dim pActView2 as IActiveView
Set pDoc = ThisDocument
Set pEnv1 = pDoc.ActiveView.Extent
Set pEnv2 = pDoc.ActiveView.Extent
Set pActView1 = pDoc.ActiveView
Set pActView2 = pDoc.ActiveView
' Extent returns a copy,
' so pEnv1 Is pEnv2 returns False
Debug.Print pEnv1 Is pEnv2
' ActiveView returns a reference,
' so pActView1 Is pActView2
Debug.Print pActView1 Is pActView2
```

Iterating through a collection

Enumerators can support other methods, but these two methods are common among all enumerators.

In your work with ArcMap and ArcCatalog, you'll discover that in many cases you'll be working with collections. You can iterate through these collections with an enumerator. An enumerator is an interface that provides methods for traversing a list of elements. Enumerator interfaces typically begin with *IEnum* and have two methods: *Next* and *Reset*. *Next* returns the next element in the set and advances the internal pointer, and *Reset* resets the internal pointer to the beginning.

Here is some VBA code that loops through the selected features (*IEnumFeature*) in a map. To try the code, add the States sample layer to the map and use the Select tool to select multiple features (drag a rectangle to do this). Add the code to a VBA macro, then execute the macro. The name of each selected state will be printed in the debug window.

```
Dim pDoc As IMxDocument
Dim pEnumFeat As IEnumFeature
Dim pFeat As IFeature
Set pDoc = ThisDocument
Set pEnumFeat = pDoc.FocusMap.FeatureSelection
Set pFeat = pEnumFeat.Next
Do While (Not pFeat Is Nothing)
  Debug.Print pFeat.Value(pFeat.Fields.FindField("state_name"))
  Set pFeat = pEnumFeat.Next
Loop
```

Some collection objects, the Visual Basic Collection being one, implement a special interface called *_NewEnum*. This interface, because of the _ prefix, is hidden, but Visual Basic developers can still use it to simplify iterating through a collection. The Visual Basic *For Each* construct works with this interface to perform the *Reset* and *Next* steps through a collection.

```
Dim pColn as Collection
Set pColn = GetCollection()' Collection returned from some function

Dim thing as Variant     ' VB uses methods on _NewEnum to step through
For Each thing in pColn  ' an enumerator.
  MsgBox Cstr(thing)
Next
```

In the previous section of this chapter, we focused primarily on how to write code in the VBA development environment embedded within the ArcGIS Desktop applications. This section focuses on particular issues related to creating ActiveX DLLs that can be added to the applications and writing external standalone applications using the Visual Basic development environment.

Creating COM components

Most developers use Visual Basic to create a COM component that works with ArcMap or ArcCatalog. Earlier in this chapter you learned that since the ESRI applications are COM clients—their architecture supports the use of software components that adhere to the COM specification—you can build components with different languages including Visual Basic. These components can then be added to the applications easily. For information about packaging and deploying COM components that you've built with Visual Basic, see the last section of this chapter.

This section is not intended as a Visual Basic tutorial; rather, it highlights aspects of Visual Basic that you should know in order to be effective when working with ArcObjects.

In Visual Basic you can build a COM component that will work with ArcMap or ArcCatalog by creating an ActiveX DLL. This section will review the rudimentary steps involved. Note that these steps are not all-inclusive. Your project may involve other requirements.

1. Start Visual Basic. In the New Project dialog box, create an ActiveX DLL Project.
2. In the Properties window, make sure that the Instancing property for the initial class module and any other class modules you add to the Project is set to 5—MultiUse.
3. Reference the ESRI Object Libraries that you will require.
4. Implement the required interfaces. When you implement an interface in a class module, the class provides its own versions of all the public procedures specified in the type library of the interface. In addition to providing a mapping between the interface prototypes and your procedures, the *Implements* statement causes the class to accept COM *QueryInterface* calls for the specified interface ID. You must include all the public procedures involved. A missing member in an implementation of an interface or class causes an error. If you don't put code in one of the procedures in a class you are implementing, you can raise the appropriate error (*Const E_NOTIMPL* = &H80004001). That way, if someone else uses the class, they'll understand that a member is not implemented.
5. Add any additional code that's needed.
6. Establish the Project Name and other properties to identify the component. In the Project Properties dialog box, the Project Name you specify will be used as the name of the component's type library. It can be combined with the name of each class the component provides to produce unique class names (these names are also called ProgIDs). These names appear in the Component Category Manager. Save the project.

The ESRI VB Add-In interface implementer can be used to automate steps 3 and 4.

Visual Basic automatically generates the necessary GUIDs for the classes, interfaces, and libraries. Setting binary compatibility forces VB to reuse the GUIDs from a previous compilation of the DLL. This is essential since ArcMap stores the GUIDs of commands in the document for subsequent loading.

7. Compile the DLL.
8. Set the component's Version Compatibility to binary. As your code evolves, it's good practice to set the components to Binary Compatibility so, if you make changes to a component, you'll be warned that you're breaking compatibility. For additional information, see the 'Binary compatibility mode' help topic in the Visual Basic online help.
9. Save the project.
10. Make the component available to the application. You can add a component to a document or template by clicking the Add from file button in the Customize dialog box's Commands tab. In addition, you can register a component in the Component Category Manager.

Implementing interfaces

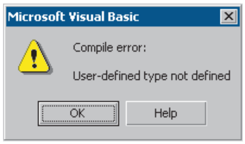
You implement interfaces differently in Visual Basic depending on whether they are inbound or outbound interfaces. An outbound interface is seen by Visual Basic as an event source and is supported through the *WithEvents* keyword. To handle the outbound interface, *IActiveViewEvents*, in Visual Basic (the default outbound interface of the *Map* class), use the *WithEvents* keyword and provide appropriate functions to handle the events.

```
Private WithEvents ViewEvents As Map

Private Sub ViewEvents_SelectionChanged()
  ' User changed feature selection update my feature list form
  UpdateMyFeatureForm
End Sub
```

Inbound interfaces are supported with the *Implements* keyword. However, unlike the outbound interface, all the methods defined on the interface must be stubbed out. This ensures that the vTable is correctly formed when the object is instantiated. Not all of the methods have to be fully coded, but the stub functions must be there. If the implementation is blank, an appropriate return code should be given to any client to inform them that the method is not implemented (see the section 'Working with HRESULTs'). To implement the *IExtension* interface, code similar to below is required. Note that all the methods are implemented.

```
Private m_pApp As IApplication
Implements IExtension
 Private Property Get IExtension_Name() As String
  IExtension_Name = "Sample Extension"
End Property

Private Sub IExtension_Startup(ByRef initializationData As Variant)
  Set m_pApp = initializationData
End Sub

Private Sub IExtension_Shutdown()
  Set m_pApp = Nothing
End Sub
```

Setting references to the ESRI object libraries



Microsoft Visual Basic

Compile error:

User-defined type not defined

The principal difference between working with the VBA development environment embedded in the applications and working with Visual Basic is that the latter environment requires that you load the appropriate object libraries so that any object variables that you declare can be found. If you don't add the reference, you'll get the error message to the left. In addition, the global variables *ThisDocument* and *Application* are not available to you.

Adding a reference to an object library

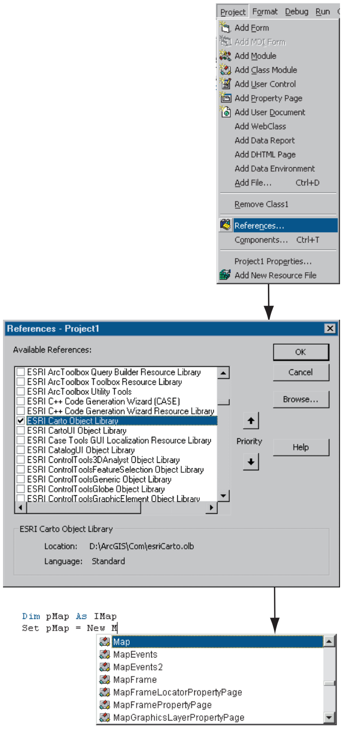
Depending on what you want your code to do, you may need to add several ESRI object and extension libraries. You can determine what library an object belongs to by reviewing the object model diagrams, searching developer help, or using the LibraryLocator tool located in the tools directory of your developer kit.

To display the References dialog box in which you can set the references you need, select References in the Visual Basic Project menu.

After you set a reference to an object library by selecting the check box next to its name, you can find a specific object and its methods and properties in the object browser.

If you are not using any objects in a referenced library, you should clear the check box for that reference to minimize the number of object references Visual Basic must resolve, thus reducing the time it takes your project to compile. You should not remove a reference for an item that is used in your project.

You can't remove the "Visual Basic for Applications" and "Visual Basic objects and procedures" references because they are necessary for running Visual Basic.



References - Project1

Available References:

☐ ESRI ArcToolbox Query Builder Resource Library
☐ ESRI ArcToolbox Toolbox Resource Library
☐ ESRI ArcToolbox Utility Tools
☐ ESRI C++ Code Generation Wizard (CASE)
☐ ESRI C++ Code Generation Wizard Resource Library
☑ ESRI Carto Object Library
☐ ESRI CartoUI Object Library
☐ ESRI Case Tools GUI Localization Resource Library
☐ ESRI CatalogUI Object Library
☐ ESRI ControlTools3DAnalyst Object Library
☐ ESRI ControlToolsFeatureSelection Object Library
☐ ESRI ControlToolsGeneric Object Library
☐ ESRI ControlToolsGlobe Object Library
☐ ESRI ControlToolsGraphicElement Object Library

ESRI Carto Object Library

Location: D:\ArcGIS\Com\esriCarto.olb

Language: Standard

```
Dim pMap As IMap
Set pMap = New M
```

After the applicable ESRI object libraries are referenced, all the types contained within them are available to Visual Basic. IntelliSense will also work with the contents of the object libraries.

Referring to a document

Each VBA project (Normal, Project, TemplateProject) has a class called *ThisDocument*, which represents the document object. Anywhere you write code in VBA you can reference the document as *ThisDocument*. Further, if you are writing your code in the *ThisDocument* Code window, you have direct access to all the methods and properties on *IDocument*. This is not available in Visual Basic. You must first get a reference to the *Application* and then the document. When adding both extensions and commands to ArcGIS applications, a pointer to the *IApplication* interface is provided.

```
Implements IExtension
Private m_pApp As IApplication

Private Sub IExtension_Startup(ByRef initializationData As Variant)
  Set m_pApp = initializationData   ' Assign IApplication
End Sub

Implements ICommand
Private m_pApp As IApplication

Private Sub ICommand_OnCreate(ByVal hook As Object)
  Set m_pApp = hook                 ' QI for IApplication
End Sub
```

Now that a reference to the application is in an *IApplication* pointer member variable, the document, and hence all other objects, can be accessed from any method within the class.

```
Dim pDoc as IDocument
Set pDoc = m_pApp.Document
MsgBox pDoc.Name
```

Getting to an object

In the previous example, navigating around the objects within ArcMap was a straightforward process since a pointer to the *Application* object, the root object of most of the ArcGIS application's objects, was passed to the object via one of its interfaces. This, however, is not the case with all interfaces that are implemented within the ArcObjects application framework. There are cases when you may implement an object that exists within the framework and there is no possibility to traverse the object hierarchy from that object. This is because very few objects support a reference to their parent object (the *IDocument* interface has a property named *Parent* that references the *IApplication* interface). In order to give developers access to the application object, there is a singleton object that provides a pointer to the running application object. The code below illustrates its use.

Singletons are objects that only support one instance of the object. These objects have a class factory that ensures that anytime an object is requested, a pointer to an already existing object is returned.

```
Dim pAppRef As New AppRef
Dim pApp as IApplication
Set pApp = pAppRef
```

You must be careful to ensure that this object is only used where the implementation will only ever run within ArcMap and ArcCatalog. For instance, it would not be a good idea to make use of this function from within a custom feature since that would restrict what applications could be used to view the feature class.

Running ArcMap with a command line argument

You can start ArcMap from the command line and pass it an argument that is either the pathname of a document (.mxd) or the pathname of a template (.mxt). In the former case, ArcMap will open the document; in the latter case, ArcMap will create a new document based on the template specified.

You can also pass an argument and create an instance of ArcMap by supplying arguments to the Win32 API's *ShellExecute* function or Visual Basic's *Shell* function as follows:

In Visual Basic, it is not possible to determine the command line used to start the application. There is a sample on disk that provides this functionality. It can be found at <ArcGIS Developer Kit install>\samples\COM Techniques\Command Line.

```
Dim ret As Variant
ret = Shell("d:\arcgis\bin\arcmap.exe _
  d:\arcgis\bin\templates\LetterPortrait.mxt", vbNormalFocus)
```

By default, *Shell* runs other programs asynchronously. This means that ArcMap might not finish executing before the statements following the *Shell* function are executed.

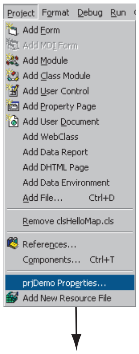
To execute a program and wait until it is terminated, you must call three Win32 API functions. First, call the *CreateProcessA* function to load and execute ArcMap. Next, call the *WaitForSingleObject* function, which forces the operating system to wait until ArcMap has been terminated. Finally, when the user has terminated the application, call the *CloseHandle* function to release the application's 32-bit identifier to the system pool.

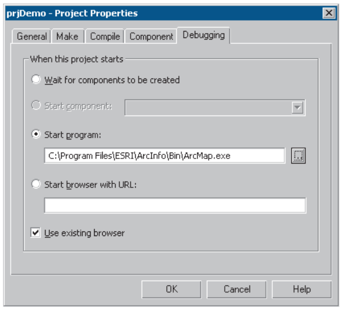
DEBUGGING VISUAL BASIC CODE

Visual Basic has a debugger integrated into its development environment. This is in many cases a valuable tool when debugging Visual Basic code; however, in some cases it is not possible to use the VB debugger. The use of the debugger and these special cases are discussed below.

Running the code within an application

It is possible to use the Visual Basic debugger to debug your ArcObjects-based source code even when ActiveX DLLs are the target server. The application that will host your DLL must be set as the Debug application. To do this, select the appropriate application, ArcMap.exe, for instance, and set it as the Start Program in the Debugging Options of the Project Properties.




prjDemo - Project Properties

When this project starts

○ Wait for components to be created

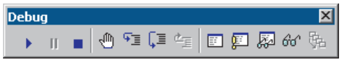
○ Start component:

● Start program:

C:\Program Files\ESRI\ArcInfo\Bin\ArcMap.exe

○ Start browser with URL:

☑ Use existing browser



Using commands on the Debug toolbar, ArcMap can be started and the DLL loaded and debugged. Break points can be set, lines stepped over, functions stepped into, and variables checked. Moving the line pointer in the left-hand margin can also set the current execution line.

Visual Basic debugger issues

In many cases, the Visual Basic debugger will work without any problems; however, there are two problems when using the debugger that is supplied with Visual Basic 6. Both of these problems exist because of the way that Visual Basic implements its debugger.

Normally when running a tool within ArcMap, the DLL is loaded into ArcMap address space, and calls are made directly into the DLL. When debugging, this is not the case. Visual Basic makes changes to the registry so that the class identifier (CLSID) for your DLL does not point to your DLL but, instead, points to the Visual Basic Debug DLL (VB6debug.dll). The Debug DLL must then support all the interfaces implemented by your class on the fly. With the VB Debug DLL loaded into ArcMap, any method calls that come into the DLL are forwarded on to Visual Basic, where the code to be debugged is executed. The two problems with this are caused by the changes made to the Registry and the cross-process space method calling. When these restrictions are first encountered, it can be confusing since the object works outside the debugger or at least until it hits the area of problem code.

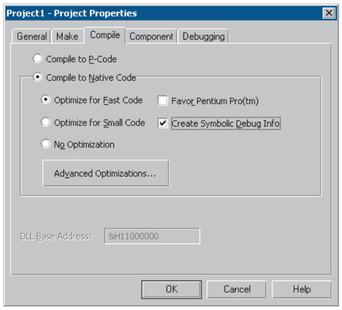
Since the method calls made from ArcMap to the custom tool are across apartments, there is a requirement for the interfaces to be marshaled. This marshaling causes problems in certain circumstances. Most data types can be automatically marshaled by the system, but there are a few that require custom code because the standard marshaler does not support the data types. If one of these data types is used by an interface within the custom tool and there is no custom marshaling code, the debugger will fail with an "Interface not supported" error.

The registry manipulation also breaks the support for component categories. Any time there is a request on a component category, the category manager within COM will be unable to find your component because, rather than asking whether your DLL belongs to the component category, COM is asking whether the VB debugger DLL belongs to the component category, which it obviously doesn't. What this means is that anytime a component category is used to automate the loading of a DLL, the DLL cannot be debugged using the Visual Basic debugger.

This obviously causes problems for many of the ways to extend the framework. The most common way to extend the framework is to add a command or tool. How component categories were used in this instance was discussed previously. Remember the component category was only used to build the list of commands in the dialog box. This means that if the command to be debugged is already present on a toolbar, the Visual Basic debugger can be used. Hence, the procedure for debugging Visual Basic objects that implement the *ICommand* interface is to ensure that the command is added to a toolbar when ArcMap is executed standalone and, after saving the document, loading ArcMap through the debugger.

In some cases, such as extensions and property pages, it is not possible to use the Visual Basic debugger. If you have access to the Visual C++ debugger, you can use one of the options outlined below. Fortunately, there are a number of ESRI Visual Basic Add-ins that make it possible to track down the problem quickly and effectively. The add-ins described below, in the section 'Visual Basic Developer Add-ins', provide error log information including line and module details. A sample output from an error log is given below; note the call stack information along with line numbers.

```
Error Log saved on : 8/28/2000 - 10:39:04 AM
Record Call Stack Sequence - Bottom line is error line.

   chkVisible_MouseUp C:\Source\MapControl\Commands\frmLayer.frm Line : 196
   RefreshMap C:\Source\MapControl\Commands\frmLayer.frm Line : 20

Description
   Object variable or With block variable not set
```


Project1 - Project Properties

○ Compile to P-Code
● Compile to Native Code
● Optimize for Fast Code
○ Optimize for Small Code
○ No Optimization
☐ Favor Pentium Pro(tm)
☑ Create Symbolic Debug Info

DLL Base Address: &H11000000

Create debug symbol information using the Create Symbolic Debug info option on the Compile tab of the Project Properties dialog box.

Alternatives to the Visual Basic debugger

If the Visual Basic debugger and add-ins do not provide enough information, the Visual C++ debugger can be used, either on its own or with C++ ATL wrapper classes. The Visual C++ debugger does not run the object to be debugged out of process from ArcMap, which means that none of the above issues apply. Common debug commands are given in the section 'Debugging tips in Visual Studio'. Both of the techniques below require the Visual Basic project to be compiled with debug symbol information.

The Visual C++ Debugger can work with this symbolic debug information and the source files.

Visual C++ debugger

It is possible to use the Visual C++ debugger directly by attaching to a running process that has the Visual Basic object to be debugged loaded and setting a break point in the Visual Basic file. When the line of code is reached, the debugger will halt execution and step you into the source file at the correct line. The required steps are shown below.

1. Start an appropriate application, such as ArcMap.exe.
2. Start Microsoft Visual C++.
3. Attach to the ArcMap process using Menu option Build -> Start Debug -> Attach to process.
4. Load the appropriate Visual Basic Source file into the Visual C++ debugger and set the break point.
5. Call the method within ArcMap.

No changes can be made to the source code within the debugger, and variables cannot be inspected, but code execution can be viewed and altered. This is often sufficient to determine what is wrong, especially with logic-related problems.

ATL rrapper classes

Using the ATL, you can create a class that implements the same interfaces as the Visual Basic class. When you create the ATL object, you create the Visual Basic object. All method calls are then passed to the Visual Basic Object for execution. You debug the contained object by setting a break point in the appropriate C++ wrapper method, and when the code reaches the break point, the debugger is stepped into the Visual Basic code. For more information on this technique, look at the ATL debugger sample in the Developer Samples of the ArcGIS Developer Help system.

Developing in Visual C++ is a large and complex subject, as it provides a much lower level of interaction with the underlying Windows APIs and COM APIs when compared to other development environments.

While this can be a hindrance for rapid application development, it is the most flexible approach. A number of design patterns like COM aggregation and singletons are possible in Visual C++ that are not possible in Visual Basic 6. By using standard class libraries like Active Template Library (ATL), the complex COM plumbing code can be hidden. However it is still important to have a thorough understanding of the underlying ATL COM implementation.

The documentation in this section is based on Microsoft Visual C++ version 6, and provides some guidance for ArcGIS development in this environment. With the release of Visual Studio C++.Net, (also referred to as VC7), many new enhancements are available to the C++ developer. While VC7 can work with the managed .Net environment, and it is possible to work with the ArcGIS .Net API, this will only add an additional overhead to access the underlying ArcGIS COM objects. So for the purposes of ArcGIS development in VC7 it is recommended to work the "traditional" way, that is directly with the ArcGIS COM interfaces and objects.

There are many enhancements to ATL in VC7. Some of the relevant changes are covered in the topic, 'ATL and Visual C++.NET', later in this section.

With the addition of Visual C#.Net language, is worth considering porting VC++ code to this environment and using the ArcGIS .Net API. The syntax of C# is not unlike C++, but the resulting code is generally simpler and more consistent.

This section is intended to serve two main purposes:

1. To familiarize you with general Visual C++ coding style and debugging, beginning with a discussion on ATL.
2. To detail specific usage requirements and recommendations for working with the ArcObjects programming platform in Visual C++.

WORKING WITH ATL

This section cannot hope to cover all the topics that a developer working with ATL should know in order to become an effective ATL C++ developer, but it will serve as an introduction to getting started with ATL. ATL helps you implement COM objects and it saves typing, but it does not excuse you from knowing C++ and how to develop COM objects.

ATL is the recommended framework for implementing COM objects. The ATL code can be combined with MFC (Microsoft Foundation Class Library) code which provides more support for writing applications. An alternative to MFC is to use the Windows Template Library (WTL). This is based on the ATL template methodology and provides many wrappers for window classes and other application support for ATL. WTL is available for download from Microsoft, at the time of writing version 7.1 is the latest and can be used with Visual C++ versions 6 and Visual C++.Net (VC7).

ATL in brief

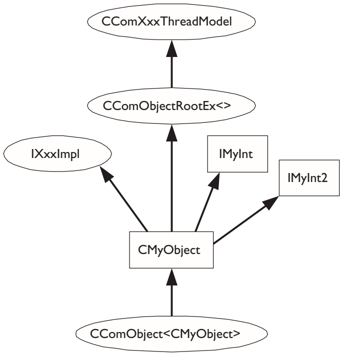
ATL is a set of C++ template classes designed to be small, fast, and extensible, based loosely on the Standard Template Library (STL). STL provides generic template classes for C++ objects like vectors, stacks and queues. ATL also provides a set of wizards that extend the Visual Studio development environment. These wizards automate some of the tedious plumbing code that all ATL projects must have. The wizards include, but are not limited, to the following:

- Application—used to initialize an ATL C++ project.
- Object—used to create COM objects. Both C++ and IDL code is generated, along with the appropriate code to support the creation of the objects at runtime.
- Property—used to add properties to interfaces.
- Method—used to add methods to interfaces; both the Property and Method Wizards require you to know some IDL syntax.
- Interface Implementation—used to implement stub functions for existing interfaces.
- Connection Point Implement — used to implement outbound events interfaces.

Typically these are accessed by a right-click on a project, class or interface in Visual Studio Workspace/Class View.

ATL provides base classes for implementing COM objects, as well as implementations for some of the common COM interfaces, including *IUnknown*, *IDispatch*, and *IClassFactory*. There are also classes that provide support for ActiveX controls and their containers.

ATL provides the required services for exposing ATL-based COM objects—these being registration, server lifetime and class objects.



The hierarchical layers of ATL

These template classes build a hierarchy that sandwiches your class. These inheritances are shown below. The *CComxxxThreadModel* supports thread-safe access to global, instance, and static data. The *CComObjectRootEx* provides the behavior for the *IUnknown* methods. The interfaces at the second level represent the interfaces that the class will implement; these come in two varieties. The *IxxxImpl* are ATL-supplied interfaces that also include an implementation; the other interfaces have pure virtual functions that must be fully implemented within your class. The *CComObject* class inherits your class; this class provides the implementation of the *IUnknown* methods along with the object instantiation and lifetime control.

ATL and DTC

A more detailed discussion on Direct To COM (DTC), follows in the later section, 'Direct-To-COM Smart Types'.

Along with smart types, covered later in this chapter, Direct-To-COM (DTC) provides some useful compiler extensions you can use when creating ATL-based objects. The functions *__declspec* and *__uuidof* are two such functions, but the most useful is the *#import* command.

COM interfaces are defined in IDL, then compiled by the Microsoft IDL compiler (MIDL.exe). This results in the creation of a type library and header files. The project uses these files automatically when compiling software that references these interfaces. This approach is limited in that when working with interfaces you must have access to the IDL files. As a developer of ArcGIS, you only have access to the ArcGIS type library information contained in ".olb" and ".ocx" files. While it is possible to engineer a header file from a type library, it is a tedious process. The *#import* command automates the creation of the necessary files required by the compiler. Since the command was developed to support DTC when using it to import ArcGIS type libraries there are a number of parameters that must be passed so that the correct import takes place. For further information on this process, see the later section 'Importing ArcGIS type libraries'.

Handling errors in ATL

It is possible to just return an E_FAIL *HRESULT* code to indicate the failure within a method, however this does not give any indication to the caller of the nature of the failure. There are a number of windows standard *HRESULT* available, for example E_INVALIDARG (One or more arguments are invalid), E_POINTER (Invalid Pointer). These error codes are listed in the window header file *winerror.h*. Not all development environments have comprehensive support for *HRESULT,* Visual Basic clients often see error results as "Automation Error – Unspecified Error". ATL provides a simple mechanism for working with the COM error information object that can provide an error string description, as well as an error code.

When creating an ATL object, the Object Wizard has an option to support *ISupportErrorInfo*. If you toggle the option on, when the wizard completes your object will implement the interface *ISupportErrorInfo* and a method will be added that looks something like this:

```
STDMETHODIMP MyClass::InterfaceSupportsErrorInfo(REFIID riid)
{
  static const IID* arr[] =
  {
    &IID_IMyClass,
  };

  for (int i = 0; i < sizeof(arr) / sizeof(arr[0]); i++)
  {
    if (InlineIsEqualGUID(*arr[i], riid))
      return S_OK;
  }

  return S_FALSE;
}
```

It is now possible to return rich error messages by calling one of the ATL error functions. These functions even work with resource files to ensure easy internationalization of the message strings.

Although Visual C++ does support an exception mechanism (try ... catch), it is not recommended to mix this with COM code. If an exception unwinds out of a COM interface there is no guarantee the client will be able to catch this and the most likely result is a crash.

```
  // Return a simple string
  AtlReportError(CLSID_MyClass, _T("No connection to Database."),
IID_IMyClass, E_FAIL);
  // Get the Error Text from a resource string
  AtlReportError(CLSID_MyClass, IDS_DBERROR, IID_IMyClass, E_FAIL,
_Module.m_hInstResource);
```

To extract an error string from a failed method use the windows function *GetErrorInfo*. This is used to retrieve the last IErrorInfo object on the current thread and clears the current error state.

Linking ATL code

One of the primary purposes of ATL is to support the creation of small fast objects. To support this, ATL gives the developer a number of choices when compiling and linking the source code. Choices must be made about how to link or dynamically access the C runtime (CRT) libraries, the registration code, and the various ATL utility functions. If no CRT calls are made in the code, this can be removed from the link. If CRT calls are made and the linker switch *_ATL_MIN_CRT* is not removed from the link line the error shown below will generate during the link stage of the build. When compiling a debug build, there will probably not be a problem, however, depending on the code written there may be problems when compiling a release build. If you receive this error either remove the CRT calls or change the linker switches.

```
LIBCMT.lib(crt0.obj) : error LNK2001: unresolved external symbol _main
ReleaseMinSize/History.dll : fatal error LNK1120: 1 unresolved externals
Error executing link.exe.
```

If the utilities code is dynamically loaded at runtime, you must ensure that the appropriate DLL (ATL.DLL) is installed and registered on the user's system. The ArcGIS runtime installation at 9.0 will install ATL.dll. The table below shows the various choices and the related linker switches.

	Symbols	CRT	Utilities	Registrar
Debug		yes	static	dynamic
RelMinDepend	_ATL_MIN_CRT _ATL_STATIC_REGISTRY	no	static	static
RelMinSize	_ATL_MIN_CRT _ATL_DLL	no	dynamic	dynamic

By default there are build configurations for ANSI and Unicode builds. A component that is built with ANSI compilation will run on Windows9x, however considering that ArcGIS is only supported on unicode operating systems (Windows NT®, Windows 2000 and Windows XP), these configurations are redundant. To delete a configuration in Visual Studio, select "Build / Configurations ...". Then delete *Win32 Debug*, *Win32 Release MinSize*, *Win32 Release MinDependency*.

Registration of a COM component

The ATL project wizard generates the windows standard entry points for registration. This code will register the .dll's type library and execute a registry script file (.rgs) script for each COM object within the dll. Additional C++ code to perform other registration tasks can be inserted into these functions.

```
STDAPI DllRegisterServer(void)
{
  // registers object in .rgs, typelib and all interfaces in typelib
  // TRUE instructs the type library to be registered
  return _Module.RegisterServer(TRUE);
}

STDAPI DllUnregisterServer(void)
{
  return _Module.UnregisterServer(TRUE);
}
```

ATL provides a text file format ".rgs" that is parsed by ATLs registrar component when a .dll is registered and unregistered. The .rgs file is built into a .dll as a custom resource. The file can be edited to add additional registry entries and contains, ProgID, ClassID and component category entries to place in the registry. The syntax describes keys, values and names and sub keys to be added or removed from the registry. The format can be summarized as follows:

```
[NoRemove | ForceRemove | val] Name | [ = s 'Value' | d ' Value' | b 'Value ]
{
  .. optional subkeys for the registry
}
```

NoRemove signifies that the registry key should not be removed on un-registration. *ForceRemove* will ensure the key and sub-keys are removed before registering the new keys. The *s*, *d* and *b* values indicate string (enclosed with apostrophes), double word (32bit integer value) and binary registry values. A typical registration script is shown below.

```
HKCR
{
  SimpleObject.SimpleCOMObject.1 = s 'SimpleCOMObject Class'
  {
    CLSID = s '{2AFFC10E-ECFB-4697-8B3D-0405650B7CFB}'
  }
  SimpleObject.SimpleCOMObject = s 'SimpleCOMObject Class'
  {
    CLSID = s '{2AFFC10E-ECFB-4697-8B3D-0405650B7CFB}'
    CurVer = s 'SimpleObject.SimpleCOMObject.1'
  }
  NoRemove CLSID
  {
    ForceRemove {2AFFC10E-ECFB-4697-8B3D-0405650B7CFB} = s 'SimpleCOMObject
Class'
    {
      ProgID = s 'SimpleObject.SimpleCOMObject.1'
      VersionIndependentProgID = s 'SimpleObject.SimpleCOMObject'
      InprocServer32 = s '%MODULE%'
      {
        val ThreadingModel = s 'Apartment'
      }
      'TypeLib' = s '{855DD226-5938-489D-986E-149600FEDD63}'
      'Implemented Categories'
      {
        {7DD95801-9882-11CF-9FA9-00AA006C42C4}
      }
    }
  }
}
```

NoRemove CLSID ensures the registry key "CLSID" is never removed. This is the subkey below which all COM objects register their ProgID's and GUIDS, so its removal would result in a serious corruption of the registry. *InprocServer32* is the standard COM mechanism that relates a component GUID to a .dll file, ATL will insert the correct module name using the %MODULE% variable. Other entries under the GUID specify the ProgID, threading model and type-library to use with this component.

To register a COM CoClass into a component category, there are two approaches, the recommended approach is illustrated above, place GUIDs for component categories beneath an 'Implemented Categories' key, which in turn is under the GUID of the CoClass. The second approach is to use ATL macros in an objects header file; BEGIN_CATEGORY_MAP, IMPLEMENTED_CATEGORY and END_CATEGORY_MAP. However these macros do not correctly remove registry entries as explained in MSDN article *Q279459 BUG: Component Category Registry Entries Not Removed in ATL Component*. A header file is supplied with the GUIDs of all the component categories used by ArcGIS, this is available in \Program Files\ArcGIS\include\CatIDs\ArcCATIDs.h.

If the GUID of a component is changed during development, or the type library name is changed, then it is important to keep the .rgs content consistent with these changes, otherwise the registry will be incorrect and object creation can fail.

Debugging ATL code

In addition to the standard Visual Studio facilities, ATL provides a number of debugging options that provide specific support for debugging COM objects. The output of these debugging options is displayed in the Visual C++ Output window. The *QueryInterface* call can be debugged by setting the symbol *_ATL_DEBUG_QI*, *AddRef* and *Release* calls with the symbol *_ATL_DEBUG_INTERFACES*, and leaked objects can be traced by monitoring the list of leaked interfaces at termination time when the *_ATL_DEBUG_INTERFACES* symbol is defined. The leaked interfaces list has entries like the following:

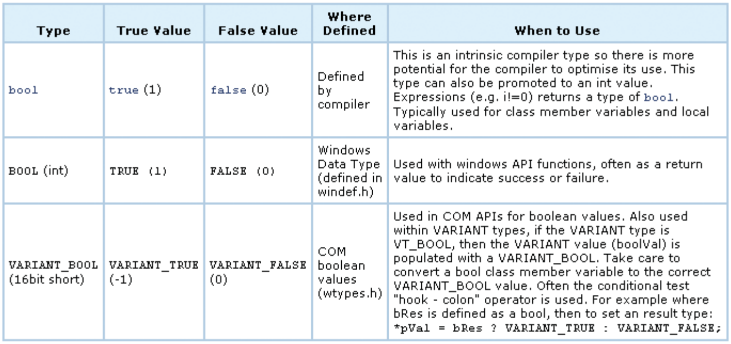
```
INTERFACE LEAK: RefCount = 1, MaxRefCount = 3, {Allocation = 10}
```

On its own, this does not tell you much apart from the fact that one of your objects is leaking because an interface pointer has not been released. However, the *Allocation* number allows you to automatically break when that interface is obtained by setting the *m_nIndexBreakAt* member of the *CComModule* at server start-up time. This in turn calls the function *DebugBreak()* to force the execution of the code to stop at the relevant place in the debugger. For this to work the program flow must be the same, but it can be very useful.

```
extern "C"
BOOL WINAPI DllMain(HINSTANCE hInstance, DWORD dwReason, LPVOID /
*lpReserved*/)
{
  if (dwReason == DLL_PROCESS_ATTACH)
  {
    _Module.Init(ObjectMap, hInstance, &LIBID_HISTORYLib);
    DisableThreadLibraryCalls(hInstance);
    _Module.m_nIndexBreakAt = 10;
  }
  else if (dwReason == DLL_PROCESS_DETACH)
  {
    _Module.Term();
  }
  return TRUE;
}
```

Boolean types

Historically ANSI C did not have a boolean data type and used int value instead, where 0 represents false and non zero represents true. However the bool datatype has now become part of ANSI C++. COM API's are language independent and define a different boolean type VARIANT_BOOL. Additionally win32 API uses a different BOOL type. It is important to use the correct type at the appropriate time. The following table summarizes their usage:



Type	True Value	False Value	Where Defined	When to Use
bool	true (1)	false (0)	Defined by compiler	This is an intrinsic compiler type so there is more potential for the compiler to optimise its use. This type can also be promoted to an int value. Expressions (e.g. i!=0) returns a type of bool. Typically used for class member variables and local variables.
BOOL (int)	TRUE (1)	FALSE (0)	Windows Data Type (defined in windef.h)	Used with windows API functions, often as a return value to indicate success or failure.
VARIANT_BOOL (16bit short)	VARIANT_TRUE (-1)	VARIANT_FALSE (0)	COM boolean values (wtypes.h)	Used in COM APIs for boolean values. Also used within VARIANT types, if the VARIANT type is VT_BOOL, then the VARIANT value (boolVal) is populated with a VARIANT_BOOL. Take care to convert a bool class member variable to the correct VARIANT_BOOL value. Often the conditional test "hook - colon" operator is used. For example where bRes is defined as a bool, then to set an result type: *pVal = bRes ? VARIANT_TRUE : VARIANT_FALSE;

String types

Considering that strings (sequences of text characters) are a simple concept, they have unfortunately become a complex and confusing topic in C++. The two main reasons for this confusion are the lack of C++ support for variable length strings combined with the requirement to support ANSI and Unicode character sets within the same code. However as ArcGIS is only available on Unicode platforms, it may simplify development to remove the ANSI requirements.

The C++ convention for strings is an array of characters terminated with a 0. This is not always good for performance when calculating lengths of large strings. To support variable length strings, the character arrays can be dynamically allocated and released on the heap, typically using *malloc* and *free* or *new* and *delete*. Consequently a number of wrapper classes provide this support; CString defined in MFC and WTL, is the most widely used. Additionally for COM usage the BSTR type is defined and the ATL wrapper class CComBSTR is available.

To allow for international character sets, Microsoft Windows migrated from a 8bit ANSI characters strings (8bit char) representation (found on win95, win98 and winME platforms) to a 16bit Unicode characters (16bit unsigned short). Unicode is synonymous with wide characters (wchar_t). In COM APIs OLECHAR is the type used and is defined to be wchar_t on Windows. Windows operating systems like winNT, win2000 and winXP natively support Unicode characters. To allow the same C++ code to be compiled for ANSI and UNICODE platforms, compiler switches are used to change windows API functions (e.g. SetWindowText) to resolve to an ANSI version (SetWindowTextA) or a Unicode version (SetWindowTextW). Additionally character independent types (TCHAR defined in tchar.h) were introduced to represent a character, on an ANSI build this is defined to be a *char* and on a Unicode build this is a *wchar_t* (typedef defined as unsigned short). To perform standard C string manipulation there are typically three different definitions of the same function, for example for a case insensitive comparison *strcmp* provides the ANSI version, *wcscmp* provides the Unicode version and *_tcscmp* provides the TCHAR version. There is also a fourth version *_mbscmp* which is a variation of the 8bit ANSI version that will interpret multi-byte character sequences (MBCS) within the 8 bit string.

```
// Initialize some fixed length strings
char*    pNameANSI = "Bill";     // 5 bytes (4 characters plus a terminator)
wchar_t* pNameUNICODE = L"Bill";  // 10 bytes (4 16bit characters plus a
                                        16bit terminator)
TCHAR*   pNameTCHAR = _T("Bill"); // either 5 or 10 depending on compiler
                                        settings
```

COM APIs represent variable length strings with a BSTR type; this is a pointer to a sequence of OLECHAR characters, which is defined as Unicode characters and is the same as a wchar_t. A BSTR must be allocated and released with the SysAllocString and SysFreeString windows functions and unlike C strings they can contain embedded zero characters (although this is unusual). The BSTR also has a count value, which is stored 4 bytes before the BSTR pointer address. The CComBSTR wrappers are often used to manage the life time of a string.

Do not pass a pointer to a C style array of Unicode characters (OLECHAR or wchar_t) to a function expecting a BSTR. The compiler will not raise an error as the types are identical. However the function receiving the BSTR can behave incorrectly or crash when accessing the string length (which will be random memory values).

```
ipFoo->put_WindowTitle(L"Hello");            // This is bad!
ipFoo->put_WindowTitle(CComBSTR(L"Hello")); // This correctly initializes
                                                and passes a BSTR
```

ATL provides conversion macros to switch strings between ANSI (A), TCHAR (T), Unicode (W) and OLECHAR (OLE). Additionally the types can have a const modifier (C). These macros use the abbreviations shown in brackets with a "2" between them. For example to convert between OLECHAR (e.g. a input BSTR) to const TCHAR (for use in a windows function) use the OLE2CT conversion macro. To convert ANSI to Unicode use A2W. These macros require the USES_CONVERSION macro to be placed at the top of a method, this will create some local variables that are used by the conversion macros. When the source and destination character sets are different and the destination type in not a BSTR, the macro allocates the destination string on the call stack (using the _alloca runtime function). It's important to realize this especially when using these macros within a loop; the stack may grow very large and run out of stack space.

To check if two CComBSTR strings are different, do not use the not equal ("!=") operator. The "==" operator performs a case sensitive comparison of the string contents, however "!=" will compare pointer values and not the string contents, typically returning false.

```
STDMETHODIMP CFoo::put_WindowTitle(BSTR bstrTitle)
{
  USES_CONVERSION;
  if (::SysStringLen(bstrTitle) == 0)
    return E_INVALIDARG;

  ::SetWindowText(m_hWnd, OLE2CT(bstrTitle));

  return S_OK;
}
```

Implementing non-creatable classes

These Non-Creatable Classes are COM objects that cannot be created by *CoCreateInstance*. Instead the object is created within a method call of a different object and an interface pointer to the non-creatable class is returned. This type of object is found in abundance in the Geodatabase model. For example *FeatureClass* is non-creatable and can only be obtained by calling one of a number of methods; one example is the *IFeatureWorkspace::OpenFeatureClass* method.

One advantage of a non-creatable class is that it can be initialized with private data using method calls that are not exposed in a COM API. Below is a simplified example of returning a non-creatable object:

```
// Foo is a CoCreateable object
IFooPtr ipFoo;
HRESULT hr = ipFoo.CreateInstance(CLSID_Foo);

// Bar is a Non-Creatable object,cannot use ipBar.CreateInstance(CLSID_Bar)
IBarPtr ipBar;
// Use a method on Foo to create a new Bar object
hr = ipFoo->CreateBar(&ipBar);
ipBar->DoSomething();
```

The steps required to change a CoCreatable ATL class into a non-creatable class are shown below:

1. Add “noncreatable” to the idl coclass attributes.

```
[
  uuid(DCB87952-0716-4873-852B-F56AE8F9BC42),
  noncreatable
]
coclass Bar
{
  [default] interface IUnknown;
  interface IBar;
};
```

2. Change the class factory implementation to fail any CoCreate of the non-creatable class. This happens via ATLs object map in the main dll module.

```
BEGIN_OBJECT_MAP(ObjectMap)
  OBJECT_ENTRY(CLSID_Foo, CFoo)                // Creatable object
  OBJECT_ENTRY_NON_CREATEABLE(CLSID_Bar, CBar)  // Non Creatable object
END_OBJECT_MAP()
```

3. Optionally, the registry entries can be removed. First remove the registry script for the object from the resources (Bar.rgs in this example). Then change the class definition DECLARE_REGISTRY_RESOURCEID(IDR_BAR) to DECLARE_NO_REGISTRY().

4. To create the non-creatable object inside a method use the CComObject template to supply the implementation of CreateInstance.

```
// Get NonCreatable object Bar (implementing IBar) from COM object Foo
STDMETHODIMP CFoo::CreateBar(IBar **pVal)
{
  if (pVal==0) return E_POINTER;
```

```
    // Smart pointer to non-creatable object Bar
    IBarPtr ipBar = 0;

    // C++ Pointer to Bar, with ATL template to supply CreateInstance imple-
mentation
    CComObject<CBar>* pBar = 0;

    HRESULT hr = CComObject<CBar>::CreateInstance(&pBar);
    if (SUCCEEDED(hr))
    {
      // Increment the ref count from 0 to 1 to protect the object
      // from being released in any initialization code.
      pBar->AddRef();

      // Call C++ methods (not exposed to COM) to initialize the Bar object
      pBar->InitialiseBar(10);

      // QI to IBar and hold a smart pointer reference to the object Bar
      hr = pBar->QueryInterface(IID_IBar, (void**)&ipBar);

      pBar->Release();
    }

    // return IBar pointer to the caller
    *pVal = ipBar.Detach();

    return S_OK;
  }
```

ATL in Visual C++.NET

Visual C++ version 6 is used for the majority of this help. However with the release of Visual C++.Net (also known as VC7), there are enhancements and changes that are relevent to the ArcGIS ATL developer, some of these are summerized below:

Attribute-based programming—This is a major change introduced in VC7. Attributes are inserted in the source code enclosed in square brackets, for example [coclass]. Attributes are designed to simplify COM programming and .NET Framework common language runtime development. When you include attributes in your source files, the compiler works with provider dynamic-link libraries (DLL) to insert code or modify the code in the generated object files. There are attributes that aid in the creation of .idl files, interfaces, type libraries, and other COM elements. In the integrated development environment (IDE), attributes are supported by the wizards and by the Properties window. The ATL wizards make extensive use of attributes to inject the ATL boiler plate code into the class. Consequently typical COM CoClass header files in VC7 contain much less ATL code than at VC6. As IDL is generated from attributes, there is typically no .idl file present in COM projects as before and the idl file is generated at compile time.

Build configurations—There are only two default build configurations in VC7, these are ANSI Debug and Release based builds. As ArcGIS is only available on

Unicode platforms it is recommended to change these by modifying the project properties. The general project properties page has an option for "Character Set", simply change this from "Use Multi-Byte Character Set" to "Use Unicode Character Set".

Character conversion macros—The character conversion macros (USES_CONVERSION, W2A, W2CT etc) have improved alternative versions. These no longer allocate space on the stack, so they can be used in loops without running out of stack space. The USES_CONVERSION macro is also no longer required. They are now implemented as classes and begin with a "C", for example CW2A, CW2CT.

Safe array support—This is available with CComSafeArray and CComSafeArrayBound classes.

Module level global—The module level global CComModule _module has been split into a number of related classes for example CAtlComModule and CAtlWinModule. To retrieve the resource module instance use the following code: _AtlBaseModule.GetResourceInstance();

String support—General variable length string support is now available through CString in ATL. This is defined in the header file atlstr.h and cstringt.h. If ATL is combined with MFC this defaults to MFC's CString implementation.

File path handling—A collection of related functions for processing the components of file paths are available through the CPath class defined in atlpath.h.

ATLServer—This is a new selection of ATL classes designed for writing web apps, XML web services and other Server applications.

#import issues—When using #import, a few modifications are required compared to VC6. For example the #import of esriSystem requires an exclude or rename of "GetObject" and the #import of esriGeometry requires an exclude or rename of "ISegment".

ATL REFERENCES

Microsoft MSDN provides a wealth of documentation, articles and samples, that are installed with Visual Studio products. ATL reference documentation for Visual Studio version 6 is under:

MSDN Library - October 2001 / Visual Tools and Languages / Visual Studio 6.0 Documentation / Visual C++ Documentation / Reference / Active Template Library

Additional documentation is also available on the MSDN Web site at http://www.msdn.microsoft.com

The following books are also useful:

Grimes, Richard. *ATL COM Programmer's Reference.* Chicago: Wrox Press Inc., 1988

Grimes, Richard. *Professional ATL COM Programming*. Chicago: Wrox Press Inc., 1988

Grimes, Richard, and Reilly Stockton, and Alex Stockton, and Julian Templeman. *Beginning ATL 3 COM Programming*. Chicago: Wrox Press Inc. 1999.

King, Brad and George Shepherd. *Inside ATL*. Redmond, WA: Microsoft Press, 1999.

Rector, Brent, and Chris Sells, and Jim Springfield. *ATL Internals*. Reading, MA: Addison-Wesley, 1999.

SMART TYPES

Smart types are objects that behave like types. They are C++ class implementations that encapsulate a data type, wrapping it with operators and functions that make working with the underlying type easier and less error prone. When these smart types encapsulate an interface pointer, they are referred to as *smart pointers*. Smart pointers work with the *IUnknown* interface to ensure that resource allocation and deallocation is correctly managed. They accomplish this by various functions, construct and destruct methods, and overloaded operators. There are numerous smart types available to the C++ programmer. The two main types of smart types covered here are defined by Direct-To-COM and the Active Template Library.

Smart types can make the task of working with COM interfaces and data types easier, since many of the API calls are moved into a class implementation; however they must be used with caution, and never without a clear understanding of how they are interacting with the encapsulated data type.

Direct-To-COM smart types

The smart type classes supplied with DTC are known as the Compiler COM Support Classes and consist of:

- *_com_error*—this class represents an exception condition in one of the COM support classes. This object encapsulates the *HRESULT* and the *IErrorInfo* COM exception object.
- *_com_ptr_t*—this class encapsulates a COM interface pointer. See below for common uses.
- *_bstr_t*—this class encapsulates the *BSTR* data type. The functions and operators on this class are not as rich as the ATL *CComBSTR* smart type, hence this is not normally used.
- *_variant_t*—this class encapsulates the *VARIANT* data type. The functions and operators on this class are not as rich as the ATL *CComVariant* smart type, hence this is not normally used.

To define a smart pointer for an interface you can use the macro *_COM_SMARTPTR_TYPEDEF* like this:

```
  _COM_SMARTPTR_TYPEDEF(IFoo, __uuidof(IFoo));
```

The compiler expands this as follows:

```
  typedef _com_ptr_t< _com_IIID<IFoo, __uuidof(IFoo)> > IFooPtr;
```

Once declared, it is simply a matter of declaring a variable as the type of the interface and appending *Ptr* to the end of the interface. Below are some common uses of this smart pointer that you will see in the numerous C++ samples.

```
// Get a CLSID GUID constant
extern "C" const GUID __declspec(selectany) CLSID_Foo = \
   {0x2f3b470c,0xb01f,0x11d3,{0x83,0x8e,0x00,0x00,0x00,0x00,0x00,0x00}};
```

```
// Declare Smart Pointers for IFoo, IBar and IGak interfaces
_COM_SMARTPTR_TYPEDEF(IFoo, __uuidof(IFoo));
_COM_SMARTPTR_TYPEDEF(IBar, __uuidof(IBar));
_COM_SMARTPTR_TYPEDEF(IGak, __uuidof(IGak));

STDMETHODIMP SomeClass::Do()
{
  // Create Instance of Foo class and QueryInterface (QI) for IFoo interface
  IFooPtr ipFoo;
  HRESULT hr = ipFoo.CreateInstance(CLSID_Foo);
  if (FAILED(hr)) return hr;

  // Call method on IFoo to get IBar
  IBarPtr ipBar;
  hr = ipFoo->get_Bar(&ipBar);
  if (FAILED(hr)) return hr;

  // QI IBar interface for IGak interface
  IGakPtr ipGak(ipBar);

  // Call method on IGak
  hr = ipGak->DoSomething();
  if (FAILED(hr)) return hr;

  // Explicitly call Release()
  ipGak = 0;
  ipBar = 0;

  // Let destructor call IFoo's Release
  return S_OK;
}
```

One of the main advantages of using the DTC smart pointers is that they are automatically generated from the "#import" compiler statement for all interface and CoClass definitions in a type library. For more details on this functionality, see the later section 'Importing ArcGIS Type Libraries'.

Although is is possible to create an object implicitly in a DTC smart pointers constructor, for example:

```
IFooPtr ipFoo(CLSID_Foo)
```

This will raise a C++ exception if there is an error during object creation (e.g. if the ".dll" file containing the object implementation was accidentally deleted). This exception will typically be unhandled and cause a crash. A more robust approach is to avoid exceptions in COM and call CreateInstance explicitly and handle the failure code, for example:

```
IFooPtr ipFoo;
HRESULT hr = ipFoo.CreateInstance(CLSID_Foo);
if (FAILED(hr))
  return hr; // return object creation failure code to caller
```

Active Template Library smart types

ATL defines various smart types, as seen in the list below. You are free to combine both the ATL and DTC smart types in your code. However it is typical to use the DTC for smart pointers as they are easily generated by importing type libraries. For BSTR and VARIANT types the ATL versions for CComBSTR, CComVariant are typically used..

ATL smart types:

- *CComPtr*—class encapsulates a COM interface pointer by wrapping the *AddRef* and *Release* methods of the *IUnknown* interface.
- *CComQIPtr*—class encapsulates a COM interface and supports all three methods of the *IUnknown* interface: *QueryInterface*, *AddRef*, and *Release*.
- *CComBSTR*—class encapsulates the *BSTR* data type.
- *CComVariant*—class encapsulates the *VARIANT* data type
- *CRegKey*—class provides methods for manipulating Windows registry entries.
- *CComDispatchDriver*—class provides methods for getting and setting properties, and calling methods through an object's *IDispatch* interface.
- *CSecurityDescriptor*—Class provides methods for setting up and working with the Discretionary Access Control List (DACL).

This section examines the first four smart types and their uses. The example code below written with ATL smart pointers, looks like the following:

```
// Get a CLSID GUID constant
extern "C" const GUID __declspec(selectany) CLSID_Foo = \
   {0x2f3b470c,0xb01f,0x11d3,{0x83,0x8e,0x00,0x00,0x00,0x00,0x00,0x00}};

STDMETHODIMP SomeClass::Do ()
{
  // Create Instance of Foo class and QI for IFoo interface
  CComPtr<IFoo> ipFoo;
  HRESULT hr = CoCreateInstance(CLSID_Foo, NULL, CLSCTX_INPROC_SERVER,
IID_IFoo, (void **)&ipFoo);
   if (FAILED(hr)) return hr;

  // Call method on IFoo to get IBar
  CComPtr<IBar> ipBar;
  HRESULT hr = ipFoo->get_Bar(&ipBar);
  if (FAILED(hr)) return hr;

  // IBar interface for IGak interface
  CComQIPtr<IGak> ipGak(ipBar);

  // Call method on IGak
  hr  = ipGak->DoSomething();
  if (FAILED(hr)) return hr;

  // Explicitly call Release()
  ipGak = 0;
```

Differences in implementation of equality ("== ") operator for smart pointer comparisons. The COM specification states object indentity is performed by comparing the pointer values of IUnknown. The DTC smart pointers will perform necessary QI and comparison when using the "==" operator. However the ATL smart pointers will not do this and you must use the ATL IsEqualObject() method.*

```
  ipBar = 0;

  // Let destructor call Foo's Release
  return S_OK;
}
```

The most common smart pointer seen in the Visual C++ samples is the *DTC* type. In the examples below, which illustrate the *BSTR* and *VARIANT* data types, the *DTC* pointers are used. When working with *CComBSTR*, use the text mapping L"" to declare constant *OLECHAR* strings. For more information on string data types see the information on 'ATL String Types' later in this Visual ++ section. *CComVariant* derives directly from the VARIANT data type, meaning that there is no overloading with its implementation, which in turn simplifies it use. It has a rich set of constructors and functions that make working with *VARIANT*s straightforward; there are even methods for reading and writing from streams. Be sure to call the *Clear* method before reusing the variable.

```
  ipFoo->put_Name(CComBSTR(L"NewName"));
  if FAILED(hr)) return hr;

  // Create a VT_I4 variant (signed long)
  CComVariant vValue(12);

  // Change its data type to a string
  hr = vValue.ChangeType(VT_BSTR);
  if (FAILED(hr)) return hr;
```

Some method calls in idl are marked as being optional and take a variant parameter. However in VC++ these parameters still have to be supplied. To signify that a parameter value is not supplied a variant is passed specifying an error code or type DISP_E_PARAMNOTFOUND:

```
  CComBSTR documentFilename(L"World.mxd");

  CComVariant noPassword;
  noPassword.vt = VT_ERROR;
  noPassword.scode = DISP_E_PARAMNOTFOUND;
  HRESULT hr = ipMapControl->LoadMxFile(documentFilename, noPassword);
```

When working with *CComBSTR* and *CComVariant*, the *Detach()* function releases the underlying data type from the smart type and can be used when passing a result as an [out] parameter of a method. The use of the Detach method with *CComBSTR* is shown below:

```
STDMETHODIMP CFoo::get_Name(BSTR* name)
{
  if (name==0) return E_POINTER;
  CComBSTR bsName(L"FooBar");
  *name = bsName.Detach();
}
```

CComVariant(VARIANT_TRUE) will create a short integer variant (type VT_I2) and not a boolean variant (type VT_BOOL) as expected. You can use CComVariant(true) to create a boolean variant.

CComVariant myVar(ipSmartPointer) will result in a variant type of boolean (VT_BOOL) and not a variant with an object reference (VT_UNKNOWN) as expected. It is better to pass unambiguous types to constructors, i.e. types which are not themselves smart types with overloaded cast operators.

```
// Perform QI it IUnknown
IUnknownPtr ipUnk = ipSmartPointer;
// Ensure we use IUnknown* constructor of CComVariant
CComVariant myVar2(ipUnk.GetInterfacePtr());
```

A common practice with smart pointers is to use *Detach()* to return an object from a method call. When returning an interface pointer the COM standard is to increment reference count of the [out] parameter inside the method implementation. It is the callers responsibility to call Release when the pointer is no longer required. Consequently care must be taken to avoid calling *Detach()* directly on a member variable, a typical pattern is show below:

```
STDMETHODIMP CFoo::get_Bar(IBar **pVal)
{
  if (pVal==0) return E_POINTER;

  // Constructing a local smart pointer using another smart pointer
  // results in an AddRef (if pointer is not 0).
  IBarPtr ipBar(m_ipBar);

  // Detach will clear the local smart pointer and the
  // interface is written into the output parameter.
  *pVal = ipBar.Detach();

  // This can be combined into one line
  // *pVal = IBarPtr(m_ipBar).Detach();

  return S_OK;
}
```

The above pattern has the same result as the following code, note that a conditional test for a 0 pointer is required before AddRef can be called, calling AddRef (or any method) on a 0 pointer will result in an access violation exception and typically crash the application:

```
STDMETHODIMP CFoo::get_Bar(IBar **pVal)
{
  if (pVal==0) return E_POINTER;

  // copy the interface pointer (no AddRef) into the output parameter
  *pVal = m_ipBar;

  // Make sure interface pointer is non 0 before calling AddRef
  if (*pVal)
    *pVal->AddRef();

  return S_OK;
}
```

When using a smart pointer to receive an object from from an [out] parameter on a method, use the smart pointer "&" de-reference operator. This will cause the previous interface pointer in the smart pointer to be released. The smart pointer is then populated with the new [out] value. The implementation of the method will have already incremented the object reference count. This will be released when the smart pointer goes out of scope:

```
{
 IFooPtr ipFoo1, ipFoo2;
 ipFoo1.CreateInstance(CLSID_Foo);
 ipFoo2.CreateInstance(CLSID_Foo);

 // Initalise ipBar Smart pointer from Foo1
 IBarPtr ipBar;
 ipFoo1->get_Bar(&ipBar);

 // The "&" de-reference will call Release on ipBar
 // ipBar is then repopulate with a new instance of IBar
 ipFoo2->get_Bar(&ipBar);
}
// ipBar goes out of scope and the smart pointer destructor calls Release
```

Naming conventions

Type names

All type names (*class, struct, enum,* and *typedef*) begin with an uppercase letter and use mixed case for the rest of the name:

```
class Foo : public CObject { . . .};
struct Bar { . . .};
enum ShapeType { . . . };
typedef int* FooInt;
```

Typedefs for function pointers (callbacks) append Proc to the end of their names.

```
typedef void (*FooProgressProc)(int step);
```

Enumeration values all begin with a lowercase string that identifies the project; in the case of ArcObjects this is esri, and each string occurs on separate lines:

```
typedef enum esriQuuxness
 {
  esriQLow,
  esriQMedium,
  esriQHigh
} esriQuuxness;
```

Function names

Name functions using the following conventions:

For simple accessor and mutator functions, use Get<Property> and Set<Property>:

```
int GetSize();
void SetSize(int size);
```

If the client is providing storage for the result, use Query<Property>:

```
void QuerySize(int& size);
```

Here are some suggestions for a naming convention.These help identify the variables usage and type and so reduce coding errors.This is an abridged Hungarian notation:

[<scope>_]<type><name>

Prefix	Variable scope
m	Instance class members
c	Static class member (including constants)
g	Globally static variable
<empty>	local variable or struct or public class member

<type>

Prefix	Data Type
b	Boolean
by	byte or unsigned char
cx / cy	short used as size
d	double
dw	DWORD, double word or unsigned long
f	float
fn	function
h	handle
i	int (integer)
ip	smart pointer
l	long
p	a pointer
s	string
sz	ASCIIZ null-terminated string
w	WORD unsigned int
x, y	short used as coordinates

<name> describes how the variable is used or what it contains.The <scope> and <type> portions should always be lowercase, and the <name> should use mixed case:

Variable Name	Description
m_hWnd	a handle to a HWND
ipEnvelope	a smart pointer to a COM interface
m_pUnkOuter	a pointer to an object
c_isLoaded	a static class member
g_pWindowList	a global pointer to an object

For state functions, use Set<State and Is<State> or Can<State>:

```
bool IsFileDirty();
void SetFileDirty(bool dirty);
bool CanConnect();
```

Where the semantics of an operation are obvious from the types of arguments, leave type names out of the function names.

Instead of:

```
AddDatabase(Database& db);
```

consider using:

```
Add(Database& db);
```

Instead of:

```
ConvertFoo2Bar(Foo* foo, Bar* bar);
```

consider using:

```
Convert(Foo* foo, Bar* bar)
```

If a client relinquishes ownership of some data to an object, use Give<Property>. If an object relinquishes ownership of some data to a client, use Take<Property>:

```
void GiveGraphic(Graphic* graphic);
Graphic* TakeGraphic(int itemNum);
```

Use function overloading when a particular operation works with different argument types:

```
void Append(const CString& text);
void Append(int number);
```

Argument names

Use descriptive argument names in function declarations. The argument name should clearly indicate what purpose the argument serves:

```
bool Send(int messageID, const char* address, const char* message);
```

DEBUGGING TIPS IN DEVELOPER STUDIO

Visual C++ comes with a feature-rich debugger. These tips will help you get the most from your debugging session.

Backing up after failure

When a function call has failed and you'd like to know why (by stepping into it), you don't have to restart the application. Use the Set Next Statement command to reposition the program cursor back to the statement that failed (right-click on the statement to bring up the debugging context menu). Then, just step into the function.

Edit and Continue

Visual Studio 6 allows changes to source code to be made during a debugging session. The changes can be recompiled and incorporated into the executing code without stopping the debugger. There are some limitations to the type of changes that can be made, in this case the debug session must be restarted. This feature is enabled by default, the settings are available in "Settings" of the project menu then Select “C/C++ tab. Select “General” from the “Category” group box. In the Debug info group box, select “Program Database for Edit and Continue”.

Unicode string display

Set your debugger options to display Unicode strings (click the Tools menu, click Options, click Debug, then check the Display Unicode Strings check box).

Variable value display

Pause the cursor over a variable name in the source code to see its current value. If it is a structure, click it and bring up the QuickWatch dialog box (the Eyeglasses icon or Shift+F9) or drag and drop it into the Watch window.

Undocking windows

If the Output window (or any docked window, for that matter) seems too small to you, try undocking it to make it a real window. Just right-click it and toggle the Docking View item.

Conditional break points

Use conditional break points when you need to stop at a break point only once some condition is reached (a for-loop reaching a particular counter value). To do so, set the break point normally, then bring up the Breakpoints window (Ctrl+B or Alt+F9). Select the specific break point you just set and then click the Condition button to display a dialog in which you specify the break point condition.

Preloading DLLs

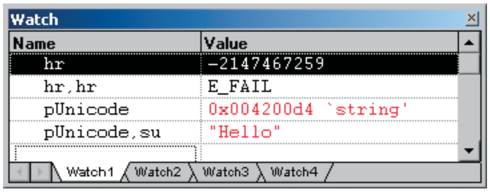
You can preload DLLs that you wish to debug before executing the program. This allows you to set break points up front rather than wait until the DLL has been loaded during program execution. (Click Project, click Settings, click Debug, click Category, then click Additional DLLs.) Then, click in the list area below to add any DLLs you wish to have preloaded.

Changing display formats

You can change the display format of variables in the QuickWatch dialog box or in the Watch window using the formatting symbols in the following table.

Symbol	Format	Value	Displays
d, i	signed decimal integer	0xF000F065	-268373915
u	unsigned decimal integer	0x0065	101
o	unsigned octal integer	0xF065	0170145
x, X	hexadecimal integer	61541	0x0000F065
l, h	long or short prefix for d, I, u, o, x, X	00406042, hx	0x0C22
f	signed floating-point	3./2.	1.500000
e	signed scientific notation	3./2.	1.500000e+00
g	e or f, whichever is shorter	3./2.	1.5
c	single character	0x0065	'e'
s	string	0x0012FDE8	"Hello"
su	Unicode string		"Hello"
hr	string	0	S_OK

To use a formatting symbol, type the variable name followed by a comma and the appropriate symbol. For example, if var has a value of 0x0065, and you want to see the value in character form, type var,c in the Name column on the tab of the Watch window. When you press ENTER, the character-format value appears: var,c = 'e'. Likewise, assuming that *hr* is a variable holding *HRESULTS*, view a human-readable form of the *HRESULT* by typing "hr,hr" in the Name column.



Watch

Name	Value
hr	-2147467259
hr,hr	E_FAIL
pUnicode	0x004200d4 `string'
pUnicode,su	"Hello"

You can use the formatting symbols shown in the following table to format the contents of memory locations.

Symbol	Format	Value
ma	64 ASCII characters	0x0012ffac .4...0...".0W&.. .....1W&.0.:W..1 ....."..1.JO&.1.2 .."..1...0y....1
m	16 bytes in hex, followed by 16 ASCII characters	0x0012ffac B3 34 CB 00 84 30 94 80 FF 22 8A 30 57 26 00 00 .4...0....".0W&..
mb	16 bytes in hex, followed by 16 ASCII characters	0x0012ffac B3 34 CB 00 84 30 94 80 FF 22 8A 30 57 26 00 00 .4...0...".0W&..
mw	8 words	0x0012ffac 34B3 00CB 3084 8094 22FF 308A 2657 0000
md	4 double-words	0x0012ffac 00CB34B3 80943084 308A22FF 00002657
mu	2-byte characters (Unicode)	0x0012fc60 8478 77f4 ffff ffff 0000 0000 0000 0000

With the memory location formatting symbols, you can type any value or expression that evaluates to a location. To display the value of a character array as a string, precede the array name with an ampersand, *&yourname*. A formatting character can also follow an expression:

- *rep+1,x*
- *alps[0],mb*
- *xloc,g*
- *count,d*

To watch the value at an address or the value pointed to by a register, use the *BY, WO,* or *DW* operator:

- *BY* returns the contents of the byte pointed at.
- *WO* returns the contents of the word pointed at.
- *DW* returns the contents of the doubleword pointed at.

Follow the operator with a variable, register, or constant. If the *BY, WO,* or *DW* operator is followed by a variable, then the environment watches the byte, word, or doubleword at the address contained in the variable.

You can also use the context operator { } to display the contents of any location.

To display a Unicode string in the Watch window or the QuickWatch dialog box, use the su format specifier. To display data bytes with Unicode characters in the Watch window or the QuickWatch dialog box, use the mu format specifier.

Keyboard shortcuts

There are numerous keyboard shortcuts that make working with the Visual Studio editor faster. Some of the more useful keyboard shortcuts follow.

The text editor uses many of the standard shortcut keys used by Windows applications, such as Word. Some specific source code editing shortcuts are listed below.

Shortcut	Action
Alt+F8	Correctly indent selected code based on surrounding lines.
Ctrl+]	Find the matching brace.
Ctrl+J	Display list of members.
Ctrl+Spacebar	Complete the word, once the number of letters entered allows the editor to recognize it. Use full when completing function and variable names.
Tab	Indents selection one tab stop to the right.
Shift+Tab	Indents selection one tab to the left.

Below is a table of common keyboard shortcuts used in the debugger.

Shortcut	Action
F9	Add or remove breakpoint from current line.
Ctrl+Shift+F9	Remove all breakpoints.
Ctrl+F9	Disable breakpoints.
Ctrl+Alt+A	Display auto window and move cursor into it.
Ctrl+Alt+C	Display call stack window and move cursor into it.
Ctrl+Alt+L	Display locals window and move cursor into it.
Ctrl+Alt+A	Display auto window and move cursor into it.
Shift+F5	End debugging session.
F11	Execute code one statement at a time, stepping into functions.
F10	Execute code one statement at a time, stepping over functions.
Ctrl+Shift+F5	Restart a debugging session.
Ctrl+F10	Resume execution from current statement to selected statement.
F5	Run the application.
Ctrl+F5	Run the application without the debugger.
Ctrl+Shift+F10	Set the next statement.
Ctrl+Break	Stop execution.

Loading the following shortcuts can greatly increase your productivity with the Visual Studio development environment.

Shortcut	Action
ESC	Close a menu or dialog box, cancel an operation in progress, or place focus in the current document window.
CTRL+SHIFT+N	Create a new file.
CTRL+N	Create a new project.
CTRL+F6 or CTRL+TAB	Cycle through the MDI child windows one window at a time.
CTRL+ALT+A	Display the auto window and move the cursor into it.
CTRL+ALT+C	Display the call stack window and move the cursor into it.
CTRL+ALT+T	Display the document outline window and move the cursor into it.
CTRL+H	Display the find window.
CTRL+F	Display the find window. If there is no current Find criteria, put the word under your cursor in the find box.
CTRL+ALT+I	Display the immediate window and move the cursor into it. Not available if you are in the text editor window.
CTRL+ALT+L	Display the locals window and move the cursor into it.
CTRL+ALT+O	Display the output window and move the cursor into it
CTRL+ALT+J	Display the project explorer window and move the cursor into it.
CTRL+ALT+P	Display the properties window and move the cursor into it.
CTRL+SHIFT+O	Open a file.
CTRL+O	Open a project.
CTRL+P	Print all or part of the document.
CTRL+SHIFT+S	Save all of the files, projects, or documents.
CTRL+S	Select all.
CTRL+A	Save the current document or selected item or items.

Navigating through online help topics

Right-click a blank area of a toolbar to display a list of all the available toolbars. The Infoviewer toolbar contains up and down arrows that allow you to cycle through help topics in the order in which they appear in the table of contents. The left and right arrows cycle through help topics in the order that you visited them.

IMPORTING ARCGIS TYPE LIBRARIES

In order to reference ArcGIS interfaces, types and objects you will need to import the definitions into VisualC++ types. The *#import* command automates the creation of the necessary files required by the compiler. The *#import* was developed to support Direct-To-Com). When importing ArcGIS type libraries there are a number of parameters that must be passed.

```
#pragma warning(push)
#pragma warning(disable : 4192)        /* Ignore warnings for types that are
                                       duplicated in win32 header files */
#pragma warning(disable : 4146)        /* Ignore warnings for use of minus on
                                       unsigned types */

#import "\Program Files\ArcGIS\com\esriSystem.olb"
                    /* Type library to generate C++ wrappers */ \
  raw_interfaces_only,       /* Don't add raw_ to method names      */ \
  raw_native_types,          /* Don't map to DTC smart types        */ \
  no_namespace,              /* Don't wrap with C++ name space      */ \
  named_guids,      /* Named guids and declspecs           */ \
  exclude("OLE_COLOR", "OLE_HANDLE", "VARTYPE")
                    /* Exclude conflicting types           */
#pragma warning(pop)
```

The main use of #import is to create C++ code for interface definitions, GUID constants (LIBID, CLSID and IID) and define smart pointers. The exclude ("OLE_COLOR", "OLE_HANDLE", "VARTYPE") is required because Win dows defines these to be unsigned longs, which conflicts with the ArcGIS definition of long—this was required to support Visual Basic as a client of ArcObjects, since Visual Basic has no support for unsigned types. There are no issues with excluding these.

You can view the code generated by #import in the ".tlh" files (type library header file - similar format to a .h). You may also find a ".tli" file (type library implementation - corresponds to .cpp). These files can be large but are only regenerated when the type libraries change.

As there are many type libraries at ArcGIS 9 for different functional areas you can start by just importing those that contain the definitions that you require. However #import does not automatically include all other definitions that the imported type library requires. For example when importing the type library esriGeometry it will contain references to types that are defined in esriSystem, so esriSystem must be imported before esriGeometry.

A complete list of library dependencies can be found in the Overview topic for each library.

Choosing the minimum set of type libraries helps reduce compilation time, although this is not always significant. Here are some steps to help determine the minimum number type libraries required:

1. Do a compilation and look at the "missing type definition" errors generated from code, e.g. "ICommand not found".

2. Place a #import statement for the library you need a reference for into your stdafx.h file. Use the LibraryLocator utility or component help to assist in this task.

3. Compile the project a second time.

4. The compiler will issue errors for types it cannot resolve in the imported type libraries, these are typically type definitions like WKSPoint or interfaces that are inherited into other interfaces. For example if working with geometry objects like points, start by importing esriGeometry, the compiler will issue various errors like :
   c:\temp\sample\debug\esrigeometry.tlh(869) : error C2061: syntax error : identifier 'WKSPoint'
   Looking up the definition of WKSPoint you see it is defined in esriSystem. Therefore importing esriSystem before esriGeometry will resolve all these issues.

   Below is a typical list of imports for working with the ActiveX controls.

```
#pragma warning(push)
#pragma warning(disable : 4192) /* Ignore warnings for types that are
                                   duplicated in win32 header files */
#pragma warning(disable : 4146) /* Ignore warnings for use of minus on
                                   unsigned types */

  #import "\Program Files\ArcGIS\com\esriSystem.olb" raw_interfaces_only,
raw_native_types, no_namespace, named_guids, exclude("OLE_COLOR",
"OLE_HANDLE", "VARTYPE")
  #import "\Program Files\ArcGIS\com\esriSystemUI.olb" raw_interfaces_only,
raw_native_types, no_namespace, named_guids
```

```
  #import "\Program Files\ArcGIS\com\esriGeometry.olb" raw_interfaces_only,
raw_native_types, no_namespace, named_guids
  #import "\Program Files\ArcGIS\com\esriDisplay.olb" raw_interfaces_only,
raw_native_types, no_namespace, named_guids
  #import "\Program Files\ArcGIS\com\esriOutput.olb" raw_interfaces_only,
raw_native_types, no_namespace, named_guids
  #import "\Program Files\ArcGIS\com\esriGeoDatabase.olb"
raw_interfaces_only, raw_native_types, no_namespace, named_guids
  #import "\Program Files\ArcGIS\com\esriCarto.olb" raw_interfaces_only,
raw_native_types, no_namespace, named_guids

  // Some of the Engine controls
  #import "\Program Files\ArcGIS\bin\TOCControl.ocx" raw_interfaces_only,
raw_native_types, no_namespace, named_guids
  #import "\Program Files\ArcGIS\bin\ToolbarControl.ocx"
raw_interfaces_only, raw_native_types, no_namespace, named_guids
  #import "\Program Files\ArcGIS\bin\MapControl.ocx" raw_interfaces_only,
raw_native_types, no_namespace, named_guids
  #import "\Program Files\ArcGIS\bin\PageLayoutControl.ocx"
raw_interfaces_only, raw_native_types, no_namespace, named_guids

  // additionally for 3D controls
 #import "\Program Files\ArcGIS\com\esri3DAnalyst.olb" raw_interfaces_only,
 raw_native_types, no_namespace, named_guids
 #import "\Program Files\ArcGIS\com\esriGlobeCore.olb" raw_interfaces_only,
 raw_native_types, no_namespace, named_guids
 #import "\Program Files\ArcGIS\bin\SceneControl.ocx" raw_interfaces_only,
 raw_native_types, no_namespace, named_guids
 #import "\Program Files\ArcGIS\bin\GlobeControl.ocx" raw_interfaces_only,
 raw_native_types, no_namespace, named_guids
```

A similar issue arises when writing IDL that contains definitions from other type libraries. In this situation use importlib just after the library definition. For example writing an external command for ArcMap would require you to create a COM object implementing ICommand. This definition is in esriSystemUI and is imported into the IDL as follows:

```
library WALKTHROUGH1CPPLib
{
  importlib("stdole32.tlb");
  importlib("stdole2.tlb");
  importlib("\Program Files\ArcGIS\com\esriSystemUI.olb");

  coclass ZoomIn
  {
    [default] interface IUnknown;
    interface ICommand;
  }
};
```

ATL AND THE ACTIVEX CONTROLS

For a general discussion of ATL, see the earlier section 'ATL in Brief'.

This section covers how to use ATL to add controls to a dialog. Although ATL is focused on providing COM support it also supplies some useful windows programming wrapper classes. One of the most useful is CWindow, this is a wrapper around a window handle (HWND). The method names on CWindow correspond to the win32 api functions. For example:

```
HWND buttonHWnd = GetDlgItem( IDC_BUTTON1 );      // Get window handle of
                                                   button
CWindow myButtonWindow( buttonHWnd );             // Attach window handle
                                                   to CWindow class
myButtonWindow.SetWindowText(_T("Button Title")); // Win32 function to
                                                  change button caption
```

CWindow is a generic wrapper for all window handles, so for specific windows messages to window common controls, like buttons, tree views or edit boxes, one approach is to send window messages direct to the window, for example:

```
// Set button to be checked (pushed-in or ticked, depending on button style)
myButtonWindow.SendMessage(BM_SETCHECK, BST_CHECKED);
```

However there are some wrapper classes for these standard window common controls in a header file *atlcontrols.h*. This is available as part of an ATL sample ATLCON supplied in MSDN. See the following article "*HOWTO: Using Class Wrappers to Access Windows Common Controls in ATL*". This header file is an early version of WTL (Windows Template Libraries), available for download from Microsoft.

Visual Studio Resource editor can be used to design and position windows common controls and ActiveX® controls on a dialog. To create and manipulate the dialog a C++ class is typically created that inherits from CAxDialogImpl. This class provides the plumbing to create and manage the ActiveX control on a window. The ATL wizard can be used to supply the majority of the boiler plate code. The steps to create a dialog and add an ActiveX control in an ATL project are discussed below.

1. Select the menu option "Insert / New ATL Object".

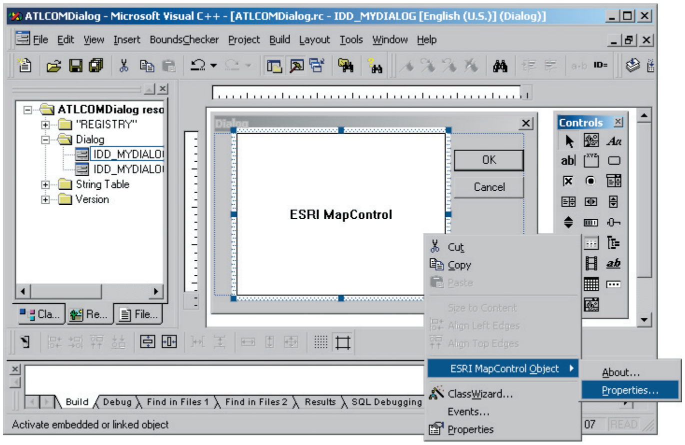
2. Select the "Miscellaneous" category and then select the "Dialog" object.

3. A dialog resource and a class inheriting from CAxDialogImpl will be added to your project.

4. Right click on the dialog in resource view and select "Insert ActiveX Control", this will display a list of available ActiveX control.

5. Double click on a control in the list to add the control to the dialog.

6. Right click on the control and select “Properties” to set the controls design time properties.



Make sure dialogs that host ActiveX controls inherit from CAxDialogImpl and not CDialogImpl. If this mistake is made the DoModal method of the dialog simply exits with no obvious cause.

Make sure applications that use windows common controls, like treeview, correctly call InitCommonControlsEx *to load the window class, otherwise the class will not function correctly.*

Accessing a control on a dialog through a COM interface

To retrieve a handle to the control that is hosted on a form, use *GetDlgControl* ATL method that is inherited from CAxDialogImpl to take a resource ID and return the underlying control pointer:

```
ITOCControlPtr ipTOCControl;
GetDlgControl(IDC_TOCCONTROL1, IID_ITOCControl, (void**) &ipTOCControl);
ipTOCControl->AboutBox();
```

Make sure applications using COM objects call CoInitialize*. This initialises COM in the application, without this call any CoCreate calls will fail.*

Listening to events from a control

The simplest way to add events is use the class wizard. Simply right-click on the control and select “Events...”. Next select the resource ID of the control and then select the event (e.g. OnMouseDown). Now click “Add Handler”. Finally ensure the dialog begins listening to events by adding AtlAdviseSinkMap(this,TRUE) to the OnInitDialog. To finish listening to events add a message handler for OnDestroy and add a call to AtlAdviseSinkMap(this, FALSE).

For a detailed discussion on handling events in ATL, see the section ‘Handling COM events in ATL’.

Creating a control at run time

The CAxWindow class provides a mechanism to create and host ActiveX controls in a similar manner to any other window class. This may be desirable if the parent window of the control is also created at runtime.

```
AtlAxWinInit();
CAxWindow wnd;
//m_hWnd is the parent window handle
//rect is the size of ActiveX control in client coordinates
//IDC_MYCTL is a unique id to identify the controls window
RECT rect = {10,10,400,300};
wnd.Create(m_hWnd, rect, _T(“esriReaderControl.ReaderControl”),
WS_CHILD|WS_VISIBLE, 0, IDC_MYCTL);
```

Setting the buddy control property

The ToolbarControl and TOCControl need to be associated with a "buddy" control on the dialog. This is typically performed in the OnInitDialog windows message handler of a dialog.

```
LRESULT CEngineControlsDlg::OnInitDialog(UINT uMsg, WPARAM wParam, LPARAM
lParam, BOOL& bHandled)
{
  // Get the Control's interfaces into class member variables
  GetDlgControl(IDC_TOOLBARCONTROL, IID_IToolbarControl, (void **)
&m_ipToolbarControl);
  GetDlgControl(IDC_TOCCONTROL, IID_ITOCControl, (void **) &m_ipTOCControl);
  GetDlgControl(IDC_PAGELAYOUTCONTROL, IID_IPageLayoutControl, (void **)
&m_ipPageLayoutControl);

  // Connect to the controls
  AtlAdviseSinkMap(this, TRUE);

  // Set buddy controls
  m_ipTOCControl->SetBuddyControl(m_ipPageLayoutControl);
  m_ipToolbarControl->SetBuddyControl(m_ipPageLayoutControl);

  return TRUE;
}
```

Known limitations of Visual Studio C++ Resource Editor and ArcGIS ActiveX controls

Buddy property on property page is disabled

In Visual Studio C++ you cannot set the "Buddy" property of the TOCControl and the ToolbarControl through the "General" property page. Visual C++ does not support controls finding other controls at design time. However this step can be performed in code in OnInitDialog method.

ToolbarControl does not resize to the height of one button

In other environments (Visual Basic 6, .Net) the ToolbarControl will automatically resize to be one button high. However in Visual Studio C++ 6 it can be any size. In MFC and ATL the ActiveX® host classes do not allow controls to determine their own size.

Design time property pages disappear when display context sensitive help

When viewing the controls property page at design time, using right click and selecting "whats' This?" will result in the help tip to display, however the property pages will then close. This is a limitation of the Visual Studio floating windows combined with the floating tip window from HTML help. Clicking the "Help" button provides the same text for the whole property page.

MFC AND THE ACTIVEX CONTROLS

There are many choices in how to work with ArcGIS ActiveX Controls in VC++, firstly which framework is used to host the controls (for example ATL or

MFC), secondly what the control will be hosted on (Dialog, MDI app etc). This section discusses MFC and hosting the control on a dialog.

Creating an MFC dialog-based application

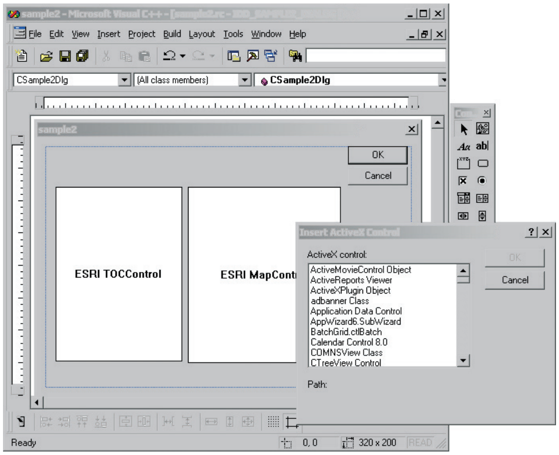
If you do not have a dialog in your application or component here are the steps to create an MFC dialog application.

1. Launch Visual Studio C++ 6 and select “New”
2. Select Projects Tab and select “MFC AppWizard (exe)” Enter Project name and location for project, click OK
3. For wizard step 1 - From the radio buttons change the application type to “Dialog Based”, Click Next For wizard step 2 - The default project features are fine, although you can uncheck AboutBox to simplify the application. Ensure support for “ActiveX Controls” is checked.
4. Click Next For wizard step 3 - The default settings in this page are fine, the MFC dll is shared, click Next.
5. For wizard step 4 - This shows you what the wizard will generate, click “Finish”

You should now have a simple dialog based application - in the resource view you will see “TODO: Place Dialog Controls Here”. You can place buttons, list boxes etc in this dialog, also the dialog can host an ActiveX controls, there are two approaches to doing this as discussed below. You can also compile and run this application.

Hosting controls on an MFC dialog and access using with using *IDispatch*

Inserting ActiveX controls on a dialog in Visual Studio C++ Design time. The TOCControl, MapControl have been added to the dialog, the ToolbarControl is next.



1. Right click on the MFC dialog and select “Insert ActiveX control”
2. Double click on a control from the list box and the control appears on the dialog with a default size.
3. Size and position the control as required.
4. Repeat steps 1 to 3 for each control.
5. You can right click on the control and select “Properties” to set the controls design time properties.
6. To access the control in code, you will need ArcGIS interface definitions for IMapControl etc. To do this use the #import in you stdafx.h file. See section Importing ArcGIS type libraries on how to do this.
7. MFC provides control hosting on a dialog, this will translate windows messages like WM_SIZE into appropriate control method calls. However to be able to make calls on a control there are a few steps to go from a resource ID to a controls interface. The following code illustrates setting the TOCControl’s Buddy to be the MapControl:

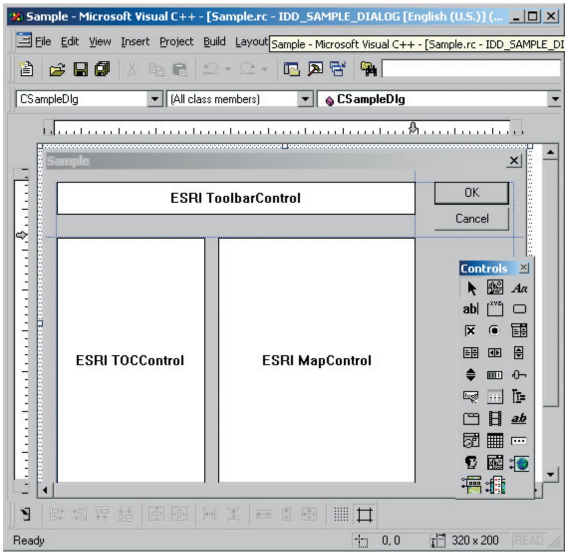
```
// Code to set the Buddy property of the TOCControl to be the MapControl
//
// Get a pointer to the PageLayoutControl and TOCControl
IPageLayoutControlPtr ipPageLayoutControl;
GetDlgControl(IDC_PAGELAYOUTCONTROL1, IID_IPageLayoutControl, (void**)
          &ipPageLayoutControl);
ITOCControlPtr ipTOCControl;
GetDlgControl(IDC_TOCCONTROL1, IID_ITOCControl, (void**) &ipTOCControl);

// Get the IDispatch of the PageLayoutControl
IDispatchPtr ipBuddyDisp = ipPageLayoutControl;

// Set the TOCControls Buddy to the map control
ipTOCControl->putref_Buddy(ipBuddyDisp);
```

8. To catch events from the controls, simply double click on the control on the form and supply the name of a method to be called. By default the wizard will add an extra word “On” to the beginning of the event handler. Remove this to avoid the event handler name becoming “OnOnMouseDownMapcontrol1”. The wizard will then automatically generate the necessary MFC sink map macros to listen to events.

Adding controls to an MFC dialog using *IDispatch* wrappers

As all ActiveX controls support IDispatch, this is the typical approach to add an ActiveX control to an MFC project:

1. Select Project / Add / Components and Controls
2. Select “Registered ActiveX Controls”
3. Double click to select control(e.g. ESRI TOCControl), then click “OK” to “Insert Component?” then click “OK” to generate wrappers. This will add an icon for the control to the “Controls” toolbar in Visual Studio
4. Additional source files are added to your project (e.g. toccontrol.cpp and toccontol.h). These files contain a wrapper class (e.g. CTOCControl) to provide methods and properties to access the control. This class will invoke the control through the IDispatch calling mechanism. Note that IDispatch does incur some performance overhead to package parameters when making method and property calls. The wrapper class inherits from a MFC CWnd class that hosts an ActiveX control.
5. Repeat steps 1 to 4 to add each control to the projects “Controls” toolbar.
6. Select a control from the “Controls” toolbar and drag it onto the dialog.


Sample - Microsoft Visual C++ - [Sample.rc - IDD_SAMPLE_DIALOG [English (U.S.)] (...
File Edit View Insert Project Build Layout
Sample - Microsoft Visual C++ - [Sample.rc - IDD_SAMPLE_DI
CSampleDlg
(All class members)
CSampleDlg
Sample
ESRI ToolbarControl
OK
Cancel
Controls
ESRI TOCControl
ESRI MapControl
Ready
0, 0
320 x 200
READ

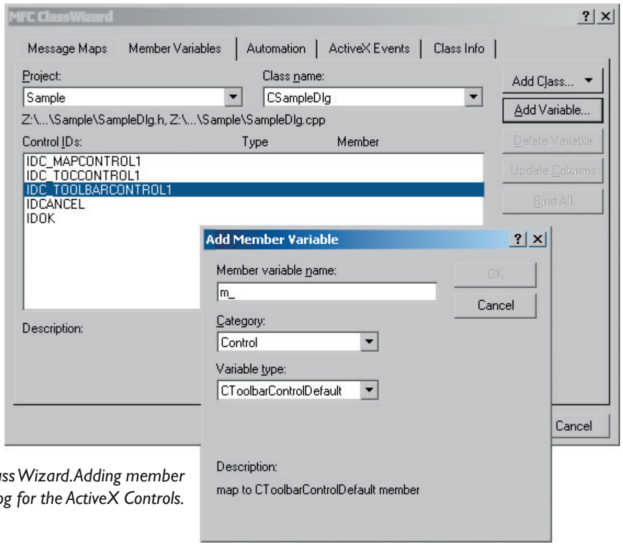
Addin Control wrappers in Visual Studio C++ Design time. The TOCControl, MapControl and ToolbarControl have been added to the “Controls” toolbar and to the dialog.

7. You can right click on the control, select properties and this will allow design time properties to be set on the control. Note. in Visual Studio C++ you cannot set the “Buddy” property of the TOCControl and the ToolbarControl.

This environment does not support controls finding other controls at design time. However this step can be performed in code in on OnInitDialog method.

```
// Note no addref performed with GetControlUnknown, so no need to release
    this pointer
LPUNKNOWN pUnk = m_mapcontrol.GetControlUnknown();
LPDISPATCH pDisp =
0;pUnk->QueryInterface(IID_IDispatch, (void **) &pDisp);

// Set TOCControls buddy to be MapControl
m_toccontrol.SetRefBuddy(pDisp);
pDisp->Release();
```

8. Right click on the control and select “Class Wizard” to launch the class wizard. Select “Member Variables tab and click on the resource ID corresponding to the control to give the control member variable name. The dialog class member variable can now used to invoke methods and properties on the control.



Project: Sample

Class name: CSampleDlg

Z:\...\Sample\SampleDlg.h, Z:\...\Sample\SampleDlg.cpp

Control IDs: Type Member

IDC_MAPCONTROL1
IDC_TOCCONTROL1
IDC_TOOLBARCONTROL1
IDCANCEL
IDOK

Description:

Member variable name: m_

Category: Control

Variable type: CToolbarControlDefault

Description:
map to CToolbarControlDefault member

Visual Studio C++ Class Wizard.Adding member variables to the dialog for the ActiveX Controls.

Do not use the method GetIDispatch *(inherited from MFC's CCmdTarget) on the wrapper classes, it is intended for objects which are implementing IDispatch and not the wrapper classes that are calling IDispatch. Instead to get a controls IDispatch use m_mapcontrol.GetControlUnknown() and then QueryInterface to IDispatch (see the above example of setting the Buddy property).*

9. To catch control events, select the “Message Maps” tab of the class wizard and select the resource ID of the control. In the list of messages select the event to catch e.g. OnBeginLabelEdit. Double click on this event and a handler for this event will be added to your dialog class. By default the wizard will add an extra word “On” to the beginning of the event handler. Remove this to avoid the event handler name becoming “OnOnBeginLabelEditToccontrol1”.

HANDLING COM EVENTS IN ATL

Here is a summary of terminology used here when discussing COM events in Visual C++ and ATL.

Inbound interface—This is the normal case where a COM object implements a predefined interface.

Outbound interface—This is an interface of methods which a COM object will fire at various times. For example the Map CoClass will *fire* an event on the IActiveViewEvents in response to changes in the map.

Event source—The source COM object will *fire events* to an outbound interface when certain actions occur. For example the Map CoClass is a source of IActiveViewEvents and will fire the IActiveViewEvents::ItemAdded event when a new layer is added to the map. The source object can have any number of clients (or *event sink objects)* listening to events. Also a source object may have more than one outbound interface, for example the MapCoClass also fires events on an IMapEvents interface. An event source will typically declare its outbound interfaces in idl with the *[source]* tag.

Event sink—A COM object that listens to events is said to be a sink for events. The sink object implements the outbound interface, this is not always advertised in the type libraries as the sink may listen to events internally. An event sink typically uses the *connection point* mechanism to register its interest in the events of a source object.

Connection point—COM objects which are the source of events typically use the connection point mechanism to allow sinks to hook up to a source. The connection point interfaces are standard COM interfaces *IConnectionPointContainer* and *IConnectionPoint*.

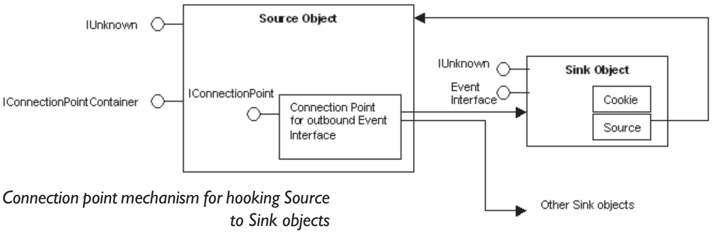
Fire event—When a source object needs to inform all the sinks of a particular action, the source is said to fire an event. This results in the source iterating all the sinks and making the same method call on each. For example when a layer is added to a map, The *Map* CoClass is said to fire the *ItemAdded* event. So all the objects listening to the Map's outbound *IActiveViewEvents* interface will be called on their implementation of the *ItemAdded* method.

Advise and unadvise events—To begin receiving events a sink object is said to advise a source object that it needs to receive events. When events are no longer required the sink will unadvise the source.

The ConnectionPoint mechanism

The source object implements *IConnectionPointContainer* interface allowing sinks to query a source for a specific outbound interface. The following steps are performed to begin listening to an event. ATL implements this with the AtlAdvise method.

1. The sink will QIs to the source object's *IConnectionPointContainer* and call *FindConnectionPoint* suppling an interface ID for an outbound interface. To be able to receive events the sink object must implement this interface.
2. The source may implement many outbound interfaces and will return a pointer to a specific connection point object implementing *IConnectionPoint* representing one outbound interface.
3. The sink calls *IConnectionPoint::Advise* passing a pointer to its own *IUnknown* implementation. The source will store this with any other sinks that may be listening to events. If the call to Advise was successful then the sink will be given an identifier (a simple unsigned long - called a cookie) to give back to the source at a later point when it no longer needs to listen to events.

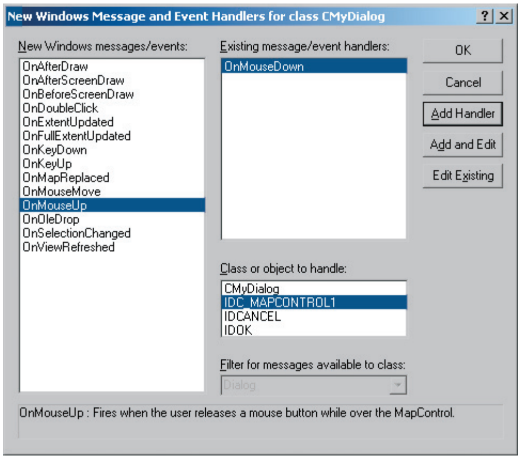


Connection point mechanism for hooking Source to Sink objects

The connection is now complete, methods will be called on any listening sinks by the source. The sink will typically hold onto an interface pointer to the source, so when a sink has finished listening it can be released from the source object by calling IConnectionPoint::Unadvise. This is implemented with AtlUnadvise.

IDispatch events versus pure COM events

An outbound interface can be a pure dispatch interface. This means instead of the source calling directly onto a method in a sink, the call is made via the IDispatch::Invoke mechanism. The IDispatch mechanism has a performance overhead to package parameters compared to a pure vtable COM call. However there are some situations where this must be used. ActiveX controls must implement their default outbound interface as a pure IDispatch interface, for example IMapControlEvents2 is a pure dispatch interface. Secondly Microsoft VisualBasic 6 can only be a source of pure IDispatch events. The connection point mechanism is the same as for pure COM mechanism, the main difference being in how the events are fired.


New Windows Message and Event Handlers for class CMyDialog

New Windows messages/events:

OnAfterDraw
OnAfterScreenDraw
OnBeforeScreenDraw
OnDoubleClick
OnExtentUpdated
OnFullExtentUpdated
OnKeyDown
OnKeyUp
OnMapReplaced
OnMouseMove
OnMouseUp
OnOleDrop
OnSelectionChanged
OnViewRefreshed

Existing message/event handlers:

OnMouseDown

Class or object to handle:

CMyDialog
IDC_MAPCONTROL1
IDCANCEL
IDOK

Filter for messages available to class:

Dialog

OnMouseUp : Fires when the user releases a mouse button while over the MapControl.

Visual Studio C++ ClassWizard.Adding event handler to an ActiveX control on a dialog.

ATL provides some macros to assist with listening to IDispatch events, this is discussed in MSDN under "*Event Handling and ATL*". There are two templates available IDispEventImpl and IDispEventSimpleImpl which are discussed in the following sections.

Using *IDispEventImpl* to listen to events

The ATL template IDispEventImpl will use a type library to "crack" the IDispatch calls and process the arguments into C++ method calls. The Visual Studio class wizard can provide this mechanism automatically when adding an ActiveX controls to a dialog. Simply right click on the Control and select "Events". In the class wizard select the resource ID of the control and then choose the event, then select "Add Handler".

The code below illustrates the event handling code added by the wizard, with some modifications to ensure advise and unadvise are performed.

```
#pragma once

#include "resource.h"      // main symbols
#include <atlhost.h>

/////////////////////////////////////////////////////////////////////////////
// CMyDialog
class CMyDialog :
  public CAxDialogImpl<CMyDialog>,
  public IDispEventImpl<IDC_MAPCONTROL1, CMyDialog>
{
```

There is a bug in the wizard as it does not add the advise and unadvise code to the dialog.To fix this issue add a message handler for OnDestroy. Then in the OnInitDialog handler call AtlAdviseSinkMap with a TRUE second parameter to begin listening to events. Place a corresponding call to AtlAdviseSinkMap (with FALSE as the Second parameter) in the OnDestroy handler.This is discussed further in MSDN article "BUG:ActiveX Control Events Are Not Fired in ATL Dialog (Q190530)".

```
 public
  enum { IDD = IDD_MYDIALOG };

BEGIN_MSG_MAP(CMyDialog)
 MESSAGE_HANDLER(WM_INITDIALOG, OnInitDialog)

 // Add a handler to ensure event unadvise occurs
 MESSAGE_HANDLER(WM_DESTROY, OnDestroy)

 COMMAND_ID_HANDLER(IDOK, OnOK)
 COMMAND_ID_HANDLER(IDCANCEL, OnCancel)
END_MSG_MAP()

 LRESULT OnInitDialog(UINT uMsg, WPARAM wParam, LPARAM lParam, BOOL&
bHandled)
 {
  // Calls IConnectionPoint::Advise() for each control on the dialog box
with sink map entry.
  AtlAdviseSinkMap(this, TRUE);
  return 1;  // Let the system set the focus
 }

 LRESULT OnDestroy(UINT uMsg, WPARAM wParam, LPARAM lParam, BOOL& bHandled)
 {
  // Calls IConnectionPoint::Unadvise() for each control on the dialog box
with sink map entry.
  AtlAdviseSinkMap(this, FALSE);
  return 0;
 }

 LRESULT OnOK(WORD wNotifyCode, WORD wID, HWND hWndCtl, BOOL& bHandled)
 {
  EndDialog(wID);
  return 0;
 }

 LRESULT OnCancel(WORD wNotifyCode, WORD wID, HWND hWndCtl, BOOL& bHandled)
 {
  EndDialog(wID);
  return 0;
 }

 // ATL callback from SinkMap entry
 VOID __stdcall OnMouseDownMapcontrol1(LONG button, LONG shift, LONG x,
LONG y, DOUBLE mapX, DOUBLE mapY)
 {
  MessageBox(_T(“MouseDown!”));
 }

BEGIN_SINK_MAP(CMyDialog)
 //Make sure the Event Handlers have __stdcall calling convention
```

The following issues with events are documented in MSDN Knowledge Base when using IDispEventImpl. Fixes to ATL code are shown in MSDN for these issues, however it is not always desirable to modify or copy ATL header files. In this case the IDispEventSimpleImpl can be used instead.

BUG: Events Fail in ATL Containers when Enum Used as Event Parameter (Q237771)

BUG: IDispEventImpl Event Handlers May Give Strange Values for Parameters (Q241810)

```
  // The 0x1 is the Dispatch ID of the OnMouseDown method
  SINK_ENTRY(IDC_MAPCONTROL1, 0x1, OnMouseDownMapcontrol1)
END_SINK_MAP()
};
```

Using *IDispEventSimpleImpl* to listen to events

As the name of this template suggests it is a simpler version of IDispEventImpl. The type library is no longer used to turn the IDispatch arguments into a C++ method call. While this may be a simpler implementation, it does now require the developer to supply a pointer to a structure describing the format of the event parameters. This structure is typically placed in the .cpp, for example here is the structure describing the parameters of a OnMouseDown event for the MapControl:

```
_ATL_FUNC_INFO g_ParamInfo_MapControl_OnMouseDown =
{
  CC_STDCALL,                              // Calling convention.
  VT_EMPTY,                                 // Return type.
  6,                                       // Number of arguments.
  {VT_I4, VT_I4, VT_I4, VT_I4, VT_R8, VT_R8}  // VariantArgument types.
};
```

See the 'Importing ArcGIS type libraries' section for more explanation of #import.

The header file now inherits from IDispEventSimpleImpl and uses a different macro SINK_ENTRY_INFO in the SINK_MAP. Also the events interface ID is required, #import can be used to define this symbol. Note that a dispatch interface is normally prefixed with DIID instead of IID.

```
#pragma once

#include "resource.h"      // main symbols
#include <atlhost.h>

// reference to structure defining event parameters
extern _ATL_FUNC_INFO g_ParamInfo_MapControl_OnMouseDown;

//////////////////////////////////////////////////////////////////////////
// CMyDialog2
class CMyDialog2 :
  public CAxDialogImpl<CMyDialog2>,
  public IDispEventSimpleImpl<IDC_MAPCONTROL1, CMyDialog2,
&DIID_IMapControlEvents2>
{
public:

// Message handler code removed, it is the same as CMyDialog using
IDispEventSimple

BEGIN_SINK_MAP(CMyDialog2)
  // Make sure the Event Handlers have __stdcall calling convention
  // The 0x1 is the Dispatch ID of the OnMouseDown method
  SINK_ENTRY_INFO(IDC_MAPCONTROL1,      // ID of event source
        DIID_IMapControlEvents2,        // interface to listen to
        0x1,                            // dispatch ID of MouseDown
```

```
    OnMapControlMouseDown,          // method to call when event arrives
    &g_ParamInfo_MapControl_OnMouseDown) // parameter info for method call

END_SINK_MAP()
};
```

Listening to more than one *IDispatch* event interface on a COM object

If a single COM object needs to receive events from more than one IDispatch source, then this can cause compiler issues with ambiguous definitions of the DispEventAdvise method. This is not normally a problem in a dialog as AtlAdviseSinkMap will handle all the connections. The ambiguity can be avoided by introducing different typedefs each time IDispEventSimpleImpl is inherited. The following example illustrates a COM object called CListen which is a sink for dispatch events from a MapControl and a PageLayoutControl.

```
#pragma once

#include “resource.h”      // main symbols

// This is the parameter information
extern _ATL_FUNC_INFO g_ParamInfo_MapControl_OnMouseDown;
extern _ATL_FUNC_INFO g_ParamInfo_PageLayoutControl_OnMouseDown;

//
// Define some typedefs of the dispatch template
//
class CListen; // forward definition

typedef IDispEventSimpleImpl<0, CListen, &DIID_IMapControlEvents2>
    IDispEventSimpleImpl_MapControl;

typedef IDispEventSimpleImpl<1, CListen, &DIID_IPageLayoutControlEvents>
    IDispEventSimpleImpl_PageLayoutControl;

/////////////////////////////////////////////////////////////////////////////
// CListen

class ATL_NO_VTABLE CListen :
  public CComObjectRootEx<CComSingleThreadModel>,
  public CComCoClass<CListen,&CLSID_Listen>,
  public IDispEventSimpleImpl_MapControl,
  public IDispEventSimpleImpl_PageLayoutControl,
  public IListen
{
public:
  CListen()
  {
  }

DECLARE_REGISTRY_RESOURCEID(IDR_LISTEN)
DECLARE_PROTECT_FINAL_CONSTRUCT()

BEGIN_COM_MAP(CListen)
  COM_INTERFACE_ENTRY(IListen)
END_COM_MAP()

// Associated source and dispatchID to a method call
BEGIN_SINK_MAP(CListen)
   SINK_ENTRY_INFO(0,                     // ID of event source
               DIID_IMapControlEvents2,   // interface to listen to
               0x1,                       // dispatch ID to receive
            OnMapControlMouseDown,        // method to call when event arrives
            &g_ParamInfo_MapControl_OnMouseDown) // parameter info for
                                                   method call

   SINK_ENTRY_INFO(1,
                    DIID_IPageLayoutControlEvents,
                       0x1,
                    OnPageLayoutControlMouseDown,
                    &g_ParamInfo_PageLayoutControl_OnMouseDown)
END_SINK_MAP()

// IListen
public:
  STDMETHOD(SetControls)(IUnknown* pMapControl, IUnknown*
pPageLayoutControl);
  STDMETHOD(Clear)();

private:
  void __stdcall OnMapControlMouseDown(long button, long shift, long x, long
y, double mapX, double mapY);
  void __stdcall OnPageLayoutControlMouseDown(long button, long shift, long
x, long y, double pageX, double pageY);

  IUnknownPtr m_ipUnkMapControl;
  IUnknownPtr m_ipUnkPageLayoutControl;
};
```

The implementaion of CListen contains the following code to start listening to the controls, the typdef avoids the ambiguity of the DispEventAdvise implementation.

```
 // Start listening to the MapControl
  IUnknownPtr ipUnk = pMapControl;
  HRESULT hr = IDispEventSimpleImpl_MapControl::DispEventAdvise(ipUnk);
  if (SUCCEEDED(hr))
    m_ipUnkMapControl = ipUnk;   // Store pointer to MapControl for Unadvise

  // Start listening to the PageLayoutControl
  ipUnk = pPageLayoutControl;
  hr = IDispEventSimpleImpl_PageLayoutControl::DispEventAdvise(ipUnk);
  if (SUCCEEDED(hr))
    m_ipUnkPageLayoutControl = ipUnk; // Store pointer to PageLayoutControl
                                         for Unadvise
```

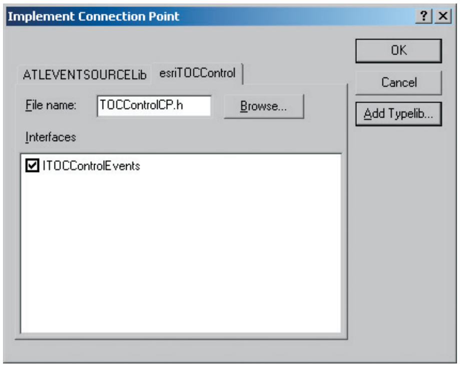
The implementation of CListen also contains the following code to UnAdvise and stop listening to the controls.

```
// Stop listening to the MapControl
if (m_ipUnkMapControl!=0)
  IDispEventSimpleImpl_MapControl::DispEventUnadvise(m_ipUnkMapControl);
m_ipUnkMapControl = 0;

if (m_ipUnkPageLayoutControl!=0)
 IDispEventSimpleImpl_PageLayoutControl::DispEventUnadvise(m_ipUnkPageLayoutControl);
m_ipUnkPageLayoutControl= 0;
```

Creating a COM events source

In order for an object to be a source of events it will need to provide an implementation of IConnectionPointContainer and a mechanism to track which sinks are listening to which IConnectionPoint interfaces. ATL provides this through the *IConnectionPointContainerImpl* template. Additionally ATL provides a wizard to generate code to fire IDispatch events for all members of a given dispatch events interface. Below are the steps to modify an ATL COM CoClass to support a connection point:

1. First make sure your ATL CoClass has been compiled at least once. This will allow the wizard to find an initial type library.
2. In Class View, Right click on the COM object and select "Implement Connection Point..."
3. Either use a definition of events from the idl in the project, or select "Add Type Lib" to browse to another definition.
4. Check the outbound interface to be implemented in the CoClass.



File name: TOCControlCP.h

☑ ITOCControlEvents

Generating Event Firing Code

5. Clicking OK will modify ur ATL class and generate the proxy classes in a header file (with a name ending in CP) for firing events.

If the wizard fails to run then use the following example that illustrates a CoClass that is a source of ITOCControlEvents. ITOCControlEvents is a pure dispatch interface.

```
#pragma once

#include "resource.h"      // main symbols
#include "TOCControlCP.h"   // Include generated connection point class
                           for firing events

////////////////////////////////////////////////////////////////////////////
// CMyEventSource
class ATL_NO_VTABLE CMyEventSource :
   public CComObjectRootEx<CComSingleThreadModel>,
   public CComCoClass<CMyEventSource,&CLSID_MyEventSource>,
   public IMyEventSource,
   public CProxyITOCControlEvents< CMyEventSource >,    // Generated
                                       ConnectionPoint class
   public IConnectionPointContainerImpl< CMyEventSource > // Implementation
                                 of Connection point Container
{
public:
   CMyEventSource()
   {
   }

DECLARE_REGISTRY_RESOURCEID(IDR_MYEVENTSOURCE)

DECLARE_PROTECT_FINAL_CONSTRUCT()

BEGIN_COM_MAP(CMyEventSource)
   COM_INTERFACE_ENTRY(IMyEventSource)
   COM_INTERFACE_ENTRY(IConnectionPointContainer) // Allow QI to this
                                                      interface
END_COM_MAP()

// List of available connection points
BEGIN_CONNECTION_POINT_MAP(CMyEventSource)
   CONNECTION_POINT_ENTRY(DIID_ITOCControlEvents)
END_CONNECTION_POINT_MAP()
};
```

The connection point class (TOCControlEventsCP.h in the above example) contains code to fire an event to all sink objects on a connection point.

There is one method on the class for each event beginning "Fire_...". Each method will build a parameter list of variants to pass as an argument to the dispatch Invoke method. Each sink is iterated, a pointer to the sink is stored in a vector m_vec member variable inherited from IConnectionPointContainerImpl.

Note that m_vec can contain pointers to 0 and this must be checked for before firing the event.

```
template <class T>
class CProxyITOCControlEvents : public IConnectionPointImpl<T,
&DIID_ITOCControlEvents, CComDynamicUnkArray>
{
public:
  VOID Fire_OnMouseDown(LONG button, LONG shift, LONG x, LONG y)
  {
   // Package each of the parameters into an IDispatch argument list
   T* pT = static_cast<T*>(this);
   int nConnectionIndex;
   CComVariant* pvars = new CComVariant[4];
   int nConnections = m_vec.GetSize();

   // Iterate each sink object
   for (nConnectionIndex =  0; nConnectionIndex < nConnections;
nConnectionIndex++)
    {
     pT->Lock();
     CComPtr<IUnknown> sp = m_vec.GetAt(nConnectionIndex);
     pT->Unlock();
     IDispatch* pDispatch = reinterpret_cast<IDispatch*>(sp.p);

     // Note m_vec can contain 0 entries so it is important to check for
          this
     if (pDispatch != NULL)
      {
       // Build up the argument list
       pvars[3] = button;
       pvars[2] = shift;
       pvars[1] = x;
       pvars[0] = y;
       DISPPARAMS disp = { pvars, NULL, 4, 0 };

      // Fire the dispatch method, 0x1 is the DispatchId for MouseDown
      pDispatch->Invoke(0x1, IID_NULL, LOCALE_USER_DEFAULT,
DISPATCH_METHOD, &disp, NULL, NULL, NULL);
      }
    }
   delete[] pvars; // clean up the parameter list

  }
  VOID Fire_OnMouseUp(LONG button, LONG shift, LONG x, LONG y)
  {
   // ... Other events
```

To fire an event from the source, simply call the Fire_OnMouseDown when required.

A similar approach can be used for firing events to a pure COM (non IDispatch) interface. The wizard will not generate the connection point class so this must be written by hand, the following example illustrates a class that will fire an ITOCBuddyEvents::ActiveViewReplaced event, ITOCBuddyEvents is a pure COM, non-IDispatch interface. The key difference is that there is no need to package the parameters, a direct method call can be made.

```
template < class T >
class CProxyTOCBuddyEvents : public IConnectionPointImpl< T,
&IID_ITOCBuddyEvents, CComDynamicUnkArray >
{
  // This class based on the ATL generated connection point class - but this
       is not an IDispatch based events
public:
  void Fire_ActiveViewReplaced(IActiveView* pNewActiveView)
   {
    T* pT = static_cast< T* >(this);
    int nConnectionIndex;
    int nConnections = this->m_vec.GetSize();
    for (nConnectionIndex =  0;  nConnectionIndex < nConnections;
nConnectionIndex++)
     {
      pT->Lock();
      CComPtr< IUnknown > sp=this->m_vec.GetAt(nConnectionIndex);
      pT->Unlock();
      ITOCBuddyEvents* pTOCBuddyEvents = reinterpret_cast< ITOCBuddyEvents*
>(sp.p);
      if (pTOCBuddyEvents)
        pTOCBuddyEvents->ActiveViewReplaced(pNewActiveView);
     }
   }
};
```

IDL declarations for an object that supports events

When an object is exported to a type library the event interfaces are declared by using the *[source]* tag against the interface name. For example an object that fires ITOCBuddyEvents declares

```
[source] interface ITOCBuddyEvents;
```

If the outbound interface is a dispatch events interface *dispinterface* is used instead of *interface* . Additionally a CoClass can have a default outbound interface and this is specified with the *[default]* tag. Default interfaces are identified by some design environments (e.g. VisualBasic 6). For example here is the declaration for the default outbound events interface:

```
[default, source] dispinterface IMyEvents2;
```

Event circular reference issues

After a sink has performed an Advise on the source there is typically a COM circular reference. This occurs because the source has an interface pointer to a sink in order to fire events, this keeps the sink alive. Similarly a sink object has a pointer back to the source so it can perform the unadvise at a later point. This keeps the source alive. Therefore these two objects will never be release and may cause substantial memory leaks. There are a number of ways to tackle this issue:

1. Ensure the advise and unadvise is made on a method or windows message that are guaranteed to happen in pairs and are independent of an objects life cycle. For example in a CoClass that is also receiving windows messages use theWindows messages OnCreate (WM_CREATE) and OnDestroy (WM_DESTROY) to advise and unadvise.

2. If a ATL dialog class needs to listen to events then one approach is to make the dialog a private COM class and implement the events interface directly on the dialog. ATL allows this without much extra coding. This approach is illustrated below. The dialog class creates a CustomizeDialog CoClass and listens to ICustomizeDialogEvents. The OnInitDialog and OnDestroy methods (corresponding to window messages) are used to Advise and Unadvise on the CustomizeDialog.

```
class CEngineControlsDlg :
  public CAxDialogImpl<CEngineControlsDlg>,
  public CComObjectRoot, // Make Dialog Class a COM Object aswell
  public ICustomizeDialogEvents  // Implement this interface directly on
                                    this object

CEngineControlsDlg() : m_dwCustDlgCookie(0) {} // initialise cookie for
                                                  event listening

// ... Event handlers and other standard dialog code has been removed ...

BEGIN_COM_MAP(CEngineControlsDlg)
  COM_INTERFACE_ENTRY(ICustomizeDialogEvents) // Make sure QI works for
                                                 this event interface
END_COM_MAP()

  // ICustomizeDialogEvents implementation to receive events on this
     dialog
  STDMETHOD(OnStartDialog)();
  STDMETHOD(OnCloseDialog)();

  ICustomizeDialogPtr      m_ipCustomizeDialog;  // The source of events
  DWORD                   m_dwCustDlgCookie;    // Cookie for
                                                   CustomizeDialogEvents
}
```

The dialog needs to be created like a noncreateble COM object rather than on the stack as a local variable, this allocates the object on the heap and allows it to be released through the COM reference counting mechanism.

```
// Create dialog class on the heap using ATL CComObject template
CComObject<CEngineControlsDlg> *myDlg;
CComObject<CEngineControlsDlg>::CreateInstance(&myDlg);

myDlg->AddRef();   // Keep dialog alive until we're done with it
myDlg->DoModal();  // Launch the dialog, when method returns dialog has
                      exited
myDlg->Release();  // typically the refcount now goes to 0 and frees the
                      dialog object
```

3. Implement an intermediate COM object for use by the sink, this is sometimes called a listener or event helper object. This object typically contains no implementation but simply uses C++ method calls to forward on events to the sink object. The listener has its reference count incremented by the source, but the sinks reference count is unaffected. This breaks the cycle, allowing the sink's reference count to reach 0, when all other references are released. As the sink executes its destuctor code it instucts the listener to unadvise and release the source.

An alternative to using C++ pointers to communicate between listener and sink, is to use an interface pointer which is a weak reference. That is the listener contains a COM pointer to the sink but does not increment the sinks reference count. It is the responsibility of the sink to ensure this pointer is not accessed after the sink object has been released.

WHAT IS THE .NET FRAMEWORK?

This section, 'What is the .NET Framework?' summarizes the Microsoft overview of the .NET Framework available online as part of MSDN Library. The complete text is available at http://www.msdn.microsoft.com/library/default.asp?url=/library/en-us/cpguide/html/cpovrintroductiontonetframeworksdk.asp

The .NET Framework is an integral Windows component that supports building and running the next generation of applications and XML Web services. The .NET Framework is designed to fulfill the following objectives:

- To provide a consistent object-oriented programming environment whether object code is stored and executed locally, executed locally but Internet-distributed, or executed remotely.
- To provide a code-execution environment that minimizes software deployment and versioning conflicts.
- To provide a code-execution environment that guarantees safe execution of code, including code created by an unknown or semi-trusted third party.
- To provide a code-execution environment that eliminates the performance problems of scripted or interpreted environments.
- To make the developer experience consistent across widely varying types of applications, such as Windows-based applications and Web-based applications.
- To build all communication on industry standards to ensure that code based on the .NET Framework can integrate with any other code.

The .NET Framework has two main components: the common language runtime and the .NET Framework class library. The common language runtime is the foundation of the .NET Framework. You can think of the runtime as an agent that manages code at execution time, providing core services such as memory management, thread management, and remoting, while also enforcing strict type safety and other forms of code accuracy that ensure security and robustness. In fact, the concept of code management is a fundamental principle of the runtime. Code that targets the runtime is known as managed code, while code that does not target the runtime is known as unmanaged code. The class library, the other main component of the .NET Framework, is a comprehensive, object-oriented collection of reusable types that you can use to develop applications ranging from traditional command-line or graphical user interface (GUI) applications to applications based on the latest innovations provided by ASP.NET, such as Web Forms and XML Web services.

The .NET Framework can be hosted by unmanaged components that load the common language runtime into their processes and initiate the execution of managed code, thereby creating a software environment that can exploit both managed and unmanaged features. The .NET Framework not only provides several runtime hosts, but also supports the development of third-party runtime hosts.

For example, ASP.NET hosts the runtime to provide a scalable, server-side environment for managed code. ASP.NET works directly with the runtime to enable ASP.NET applications and XML Web services, both of which are discussed later in this topic.

Internet Explorer is an example of an unmanaged application that hosts the runtime (in the form of a MIME type extension). Using Internet Explorer to host the runtime enables you to embed managed components or Windows Forms controls in HTML documents. Hosting the runtime in this way makes managed mobile code (similar to Microsoft ActiveX controls) possible, but with significant improvements that only managed code can offer, such as semi-trusted execution and secure isolated file storage.

The following sections describe the main components and features of the .NET Framework in greater detail.

Features of the Common Language Runtime

The common language runtime manages memory, thread execution, code execution, code safety verification, compilation, and other system services. These features are intrinsic to the managed code that runs on the common language runtime.

With regards to security, managed components are awarded varying degrees of trust, depending on a number of factors that include their origin (such as the Internet, enterprise network, or local computer). This means that a managed component might or might not be able to perform file-access operations, registry-access operations, or other sensitive functions, even if it is being used in the same active application.

The runtime enforces code access security. For example, users can trust that an executable embedded in a Web page can play an animation on screen or sing a song, but cannot access their personal data, file system, or network. The security features of the runtime thus enable legitimate Internet-deployed software to be exceptionally feature rich.

The runtime also enforces code robustness by implementing a strict type-and-code-verification infrastructure called the common type system (CTS). The CTS ensures that all managed code is self-describing. The various Microsoft and third-party language compilers generate managed code that conforms to the CTS. This means that managed code can consume other managed types and instances, while strictly enforcing type fidelity and type safety.

In addition, the managed environment of the runtime eliminates many common software issues. For example, the runtime automatically handles object layout and manages references to objects, releasing them when they are no longer being used. This automatic memory management resolves the two most common application errors, memory leaks and invalid memory references.

The runtime also accelerates developer productivity. For example, programmers can write applications in their development language of choice, yet take full advantage of the runtime, the class library, and components written in other languages by other developers. Any compiler vendor who chooses to target the runtime can do so. Language compilers that target the .NET Framework make the features of the .NET Framework available to existing code written in that language, greatly easing the migration process for existing applications.

While the runtime is designed for the software of the future, it also supports software of today and yesterday. Interoperability between managed and unmanaged code enables developers to continue to use necessary COM components and DLLs.

The runtime is designed to enhance performance. Although the common language runtime provides many standard runtime services, managed code is never interpreted. A feature called just-in-time (JIT) compiling enables all managed code to run in the native machine language of the system on which it is executing. Meanwhile, the memory manager removes the possibilities of fragmented memory and increases memory locality-of-reference to further increase performance.

Finally, the runtime can be hosted by high-performance, server-side applications, such as Microsoft SQL Server and Internet Information Services (IIS). This infrastructure enables you to use managed code to write your business logic, while still enjoying the superior performance of the industry's best enterprise servers that support runtime hosting.

.NET Framework class library

The .NET Framework class library is a collection of reusable types that tightly integrate with the common language runtime. The class library is object oriented, providing types from which your own managed code can derive functionality. This not only makes the .NET Framework types easy to use, but also reduces the time associated with learning new features of the .NET Framework. In addition, third-party components can integrate seamlessly with classes in the .NET Framework.

For example, the .NET Framework collection classes implement a set of interfaces that you can use to develop your own collection classes. Your collection classes will blend seamlessly with the classes in the .NET Framework.

As you would expect from an object-oriented class library, the .NET Framework types enable you to accomplish a range of common programming tasks, including tasks such as string management, data collection, database connectivity, and file access. In addition to these common tasks, the class library includes types that support a variety of specialized development scenarios. For example, you can use the .NET Framework to develop the following types of applications and services:

- Console applications.
- Windows GUI applications (Windows Forms).
- ASP.NET applications.
- XML Web services.
- Windows services.

For example, the Windows Forms classes are a comprehensive set of reusable types that vastly simplify Windows GUI development. If you write an ASP.NET Web Form application, you can use the Web Forms classes.

Client application development

Client applications are the closest to a traditional style of application in Windows-based programming. These are the types of applications that display windows or forms on the desktop, enabling a user to perform a task. Client applications include applications such as word processors and spreadsheets, as well as custom business applications such as data-entry tools, reporting tools, and so on. Client applications usually employ windows, menus, buttons, and other GUI elements, and they likely access local resources such as the file system and peripherals such as printers.

Another kind of client application is the traditional ActiveX control (now replaced by the managed Windows Forms control) deployed over the Internet as a Web page. This application is much like other client applications: it is executed natively, has access to local resources, and includes graphical elements.

In the past, developers created such applications using C/C++ in conjunction with the Microsoft Foundation Classes (MFC) or with a rapid application development (RAD) environment such as Microsoft Visual Basic. The .NET Framework incorporates aspects of these existing products into a single, consistent development environment that drastically simplifies the development of client applications.

The Windows Forms classes contained in the .NET Framework are designed to be used for GUI development. You can easily create command windows, buttons, menus, toolbars, and other screen elements with the flexibility necessary to accommodate shifting business needs.

For example, the .NET Framework provides simple properties to adjust visual attributes associated with forms. In some cases the underlying operating system does not support changing these attributes directly, and in these cases the .NET Framework automatically recreates the forms. This is one of many ways in which the .NET Framework integrates the developer interface, making coding simpler and more consistent.

Unlike ActiveX controls, Windows Forms controls have semi-trusted access to a user’s computer. This means that binary or natively executing code can access some of the resources on the user’s system (such as GUI elements and limited file access) without being able to access or compromise other resources. Because of code access security, many applications that once needed to be installed on a user’s system can now be safely deployed through the Web. Your applications can implement the features of a local application while being deployed like a Web page.

Server application development

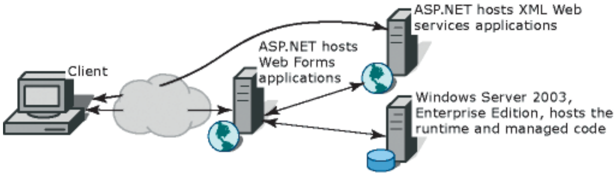
Server-side applications in the managed world are implemented through runtime hosts. Unmanaged applications host the common language runtime, which allows your custom managed code to control the behavior of the server. This model provides you with all the features of the common language runtime and class library while gaining the performance and scalability of the host server.

Server-side managed code

ASP.NET is the hosting environment that enables developers to use the .NET Framework to target Web-based applications. However, ASP.NET is more than just a runtime host; it is a complete architecture for developing Web sites and Internet-distributed objects using managed code. Both Web Forms and XML Web services use IIS and ASP.NET as the publishing mechanism for applications, and both have a collection of supporting classes in the .NET Framework.

XML Web services, an important evolution in Web-based technology, are distributed, server-side application components similar to common Web sites. However, unlike Web-based applications, XML Web services components have no UI and are not targeted for browsers such as Internet Explorer and Netscape Navigator. Instead, XML Web services consist of reusable software components designed to be consumed by other applications, such as traditional client applications, Web-based applications, or even other XML Web services. As a result, XML Web services technology is rapidly moving application development and deployment into the highly distributed environment of the Internet.

If you have used earlier versions of ASP technology, you will immediately notice the improvements that ASP.NET and Web Forms offer. For example, you can develop Web Forms pages in any language that supports the .NET Framework. In addition, your code no longer needs to share the same file with your HTTP text (although it can continue to do so if you prefer). Web Forms pages execute in native machine language because, like any other managed application, they take full advantage of the runtime. In contrast, unmanaged ASP pages are always scripted and interpreted. ASP.NET pages are faster, more functional, and easier to develop than unmanaged ASP pages because they interact with the runtime like any managed application.


Client
ASP.NET hosts Web Forms applications
ASP.NET hosts XML Web services applications
Windows Server 2003, Enterprise Edition, hosts the runtime and managed code

This diagram illustrates a basic network schema with managed code running in different server environments. Servers such as IIS and SQL Server can perform standard operations while your application logic executes through the managed code.

The .NET Framework also provides a collection of classes and tools to aid in development and consumption of XML Web services applications. XML Web services are built on standards such as SOAP (a remote procedure-call protocol), XML (an extensible data format), and WSDL (the Web Services Description Language). The .NET Framework is built on these standards to promote interoperability with non-Microsoft solutions.

For example, the Web Services Description Language tool included with the .NET Framework SDK can query an XML Web service published on the Web, parse its WSDL description, and produce C# or Visual Basic source code that your application can use to become a client of the XML Web service. The source code can create classes derived from classes in the class library that handle all the underlying communication using SOAP and XML parsing. Although you can use the class library to consume XML Web services directly, the Web Services Description Language tool and the other tools contained in the SDK facilitate your development efforts with the .NET Framework.

If you develop and publish your own XML Web service, the .NET Framework provides a set of classes that conform to all the underlying communication standards, such as SOAP, WSDL, and XML. Using those classes enables you to focus on the logic of your service, without concerning yourself with the communications infrastructure required by distributed software development.

Finally, like Web Forms pages in the managed environment, your XML Web service will run with the speed of native machine language using the scalable communication of IIS.

Interoperating with COM

Code running under the .NET Framework's control is called managed code; conversely, code executing outside the .NET Framework is termed unmanaged code. COM is one example of unmanaged code. The framework promotes interaction with unmanaged code and for COM, the technology that bridges this gap is COM Interop, which is part of the .NET Framework itself.

For COM Interop to work, the CLR requires metadata for all the COM types. This means that the COM type definitions normally stored in the type libraries need to be converted to .NET metadata. This is easily accomplished with the Type Library Importer utility (tlbimp.exe) which ships with the .NET Framework SDK. This utility generates interop assemblies containing the metadata for all the COM definitions in a type library. Once metadata is available, .NET clients can seamlessly create instances of COM types and call its methods as though they were native .NET instances.

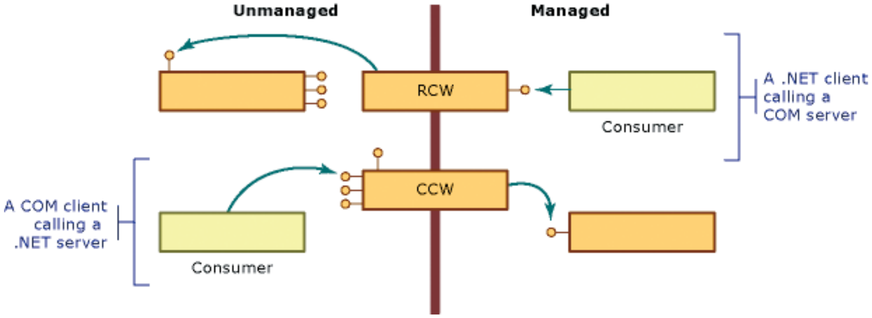
Primary interop assemblies

Primary interop assemblies (PIAs) are the official, vendor-supplied, .NET type definitions for interoperating with underlying COM types. Primary interop assemblies are strongly named by the COM library publisher to guarantee uniqueness.

ESRI provides primary interop assemblies for all the ArcObjects type libraries that are implemented with COM. ArcGIS .NET developers should only use these primary interop assemblies which are installed in the Global Assembly Cache (GAC) during install if version 1.1 of the .NET Framework is detected. ESRI only supports the interop assemblies that ship with ArcGIS. You can identify a valid ESRI assembly by its public key (8FC3CC631E44AD86).

COM wrappers

The .NET runtime provides wrapper classes to make both managed and unmanaged clients believe they are communicating with objects within their respective environment. When managed clients call a method on a COM object, the runtime creates a runtime callable wrapper (RCW) which handles the marshaling between the two environments. Similarly, the .NET runtime creates COM callable wrappers for the reverse case, COM clients communicating with .NET components.


Unmanaged
Managed
RCW
Consumer
A .NET client calling a COM server
A COM client calling a .NET server
CCW
Consumer

Exposing .NET components to COM

When creating .NET components that COM clients will make use of, follow the guidelines listed below to ensure interoperability.

- Avoid using parameterized constructors.
- Avoid using static methods.
- Define event-source interfaces in managed code.
- Include HRESULTs in user-defined exceptions.

- Supply Globally Unique Identifiers (GUIDs) for types that require them.
- Expect inheritance differences

For more information review 'Interoperating with Unmanaged Code' in the MSDN help collection.

Performance considerations

COM Interop clearly adds a new layer of overhead to applications but the overall cost of interoperating between COM and .NET is very small and often unnoticeable. However, the cost creating wrappers and having them marshal between environments does add up; if you suspect COM Interop is the bottleneck in your application's performance, try creating a COM worker class that wraps all the chatty COM calls into one function which managed code can invoke. This improves performance by limiting the marshaling between the two environments.

COM to .NET type conversion

Generally speaking, the type library importer imports types with the same name they originally had in COM. All imported types are additionally added to a namespace which have the following naming convention: ESRI.ArcGIS plus the name of library. For example, the namespace for the Geometry library is ESRI.ArcGIS.Geometry. All types are identified by their complete namespace and type name.

Classes, Interfaces, and Members

All COM coclasses are converted to managed classes; the managed classes have the same name as the original with 'Class' appended. For example, the Point coclass is PointClass.

All classes additionally have an interface with the same name as the coclass that corresponds to the default interface for the coclass. For example, the PointClass has a Point interface. The type library importer adds this interface so clients can register as event sinks.

The .NET classes additionally have class members which .NET supports but COM does not. Each member of each interface the class implements is added as a class member. Any property or method a class implants can be accessed directly from the class rather than having to cast to a specific interface. Since interface member names are not unique, name conflicts are resolved by prefixing the interface name and an underscore to the name of each conflicting member. When member names conflict, the first interface listed with the coclass remains unchanged.

Properties in C# that have by-reference or multiple parameters are not supported with the regular property syntax. In these cases, it is necessary to use the accessor methods instead. The following code excerpt shows an example.

```
    ILayer layer = mapControl.get_Layer(0);
  MessageBox.Show(layer.Name);
```

Events

The type library importer creates several types that enable managed applications to sink to events fired by COM classes. The first type is a delegate that is named after the event interface plus an underscore followed by the event name, and then the word EventHandler. For example, the SelectionChanged event defined on the IActiveViewEvents interface has the following delegate defined: IActiveViewEvents_SelectionChangedEventHandler. The importer additionally creates an event interface with a '_Event' suffix added to the end of the original interface name. For example, IActiveViewEvents generates IActiveViewEvents_Event. Use the event interfaces to set up event sinks.

Non-Ole Automation Compliant Types

COM types that are not OLE automation compliant generally do not work in .NET. ArcGIS contains a few non-compliant methods and these are unfortunately not usable in .NET. However, in most cases, supplemental interfaces have been added which have the offending members rewritten compliantly. For example, when defining an envelope via a point array you can't use IEnvelope::DefineFromPoints; instead, you must use IEnvelopeGEN::DefineFromPoints.

```
[VB.NET]
  Dim pointArray(1) As IPoint
  pointArray(0) = New PointClass
  pointArray(1) = New PointClass
  pointArray(0).PutCoords(0, 0)
  pointArray(1).PutCoords(100, 100)

  Dim env As IEnvelope
  Dim envGEN As IEnvelopeGEN
  env = New EnvelopeClass
  envGEN = New EnvelopeClass

  'Won't compile
  'env.DefineFromPoints(2, pointArray)

  'Doesn't work
  env.DefineFromPoints(2, pointArray(0))

  'Works
  envGEN.DefineFromPoints(pointArray)

[C#]
  IPoint[] pointArray = new IPoint[2];
  pointArray[0] = new PointClass();
  pointArray[1] = new PointClass();
  pointArray[0].PutCoords(0,0);
  pointArray[1].PutCoords(100,100);

  IEnvelope env = new EnvelopeClass();
  IEnvelopeGEN envGEN = new EnvelopeClass();

  //Won't compile
  //env.DefineFromPoints(3, ref pointArray);
```

```
//Doesn't work
env.DefineFromPoints(3, ref pointArray[0]);

//Works
envGEN.DefineFromPoints(ref pointArray);
```

.NET PROGRAMMING TECHNIQUES AND CONSIDERATIONS

This section contains several programming tips and techniques to help developers who are moving to .NET.

Casting between interfaces (QueryInterface)

.NET uses casting to jump from one interface to another interface on the same class. In COM this is called QueryInterface. VB.NET and C# cast differently.

VB.NET

There are two types of casts, implicit and explicit. Implicit casts require not additional syntax whereas explicit cast require cast operators.

```
geometry = point                        'Implicit cast
geometry = CType(point, IGeometry)      'Explicit cast
```

When casting between interfaces it perfectly acceptable to use implicit casts as there is no chance of data loss as there is when casting between numeric types. However, when casts fail an exception (System.InvalidCastException) is thrown; to avoid handling unnecessary exceptions, it's best to test if the object implements both interfaces beforehand. The recommend technique is to use the TypeOf keyword which is a comparison clause that tests whether an object is derived from or implements a particular type, such as an interface. The example below performs an implicit conversion from an IPoint to an IGeometry only if at runtime it is determined that the Point class implements IGeometry.

```
Dim point As New PointClass
Dim geometry As IGeometry
If (TypeOf point Is IGeometry) Then
   geometry = point
End If
```

If you prefer using the Option Strict On statement to restrict implicit conversions, use the CType function to make the cast explicit. The example below adds an explicit cast to the code sample above.

```
Dim point As New PointClass
Dim geometry As IGeometry
If (TypeOf point Is IGeometry) Then
   geometry = CType(point, IGeometry)
End If
```

C#

In C#, the best method for casting between interfaces is to use the **as** operator. Using the as operator is a better coding strategy than a straight cast because it yields a null on a conversion failure rather than raising an exception.

The first line of code below is a straight cast. This is acceptable practice if you are absolutely certain the object in question implements both interfaces; if the object does not implement the interface you are attempting to get a handle to, .NET will throw an exception. A safer model is to use the as operator which returns a null if the object cannot return a reference to the desired interface.

```
  IGeometry geometry = point;            // straight cast
  IGeometry geometry = point as IGeometry;  // as operator
```

The example below shows how to handle to possibility of a returned null interface handle.

```
  IPoint point = new PointClass();
  IGeometry geometry = point;
  IGeometry geometry = point as IGeometry;
  if (geometry != null)
  {
    Console.WriteLine(geometry.GeometryType.ToString());
  }
```

Binary compatibility

Most existing ArcGIS Visual Basic 6 developers are familiar with the notion of binary compatibility. This compiler flag in Visual Basic ensures that components maintain the same GUID each time they are compiled. When this flag is not set, a new GUID is generated for each class every time the project is compiled. This has the adverse side-effect of having to then re-register the components in their appropriate component categories.

To keep from having the same problem in .NET, you can use the GUIDAtrribute class to manually specify a GUID for a class. Explicitly specifying a GUID guarantees that it will never change. If you do not specify a GUID, the type library exporter will automatically generate one when you first export your components to COM and although the exporter is meant to keep using the same GUIDs on subsequent exports, it's not guaranteed to do so.

The example below shows a GUID attribute being applied to a class.

```
[VB.NET]
  <GuidAttribute("9ED54F84-A89D-4fcd-A854-44251E925F09")> _
  Public Class SampleClass
    '
  End Class

[C#]
  [GuidAttribute("9ED54F84-A89D-4fcd-A854-44251E925F09")]
  Public class SampleClass
  {
  //
  }
```

Events

An event is a message sent by an object to signal the occurrence of an action. The action could be caused by user interaction, such as a mouse click, or it could be triggered by some other program logic. The object that raises (triggers) the event is called the event sender. The object that captures the event and responds to it is called the event receiver.

In event communication, the event sender class does not know which object or method will receive (handle) the events it raises. What is needed is an intermediary (or pointer-like mechanism) between the source and the receiver. The .NET Framework defines a special type (<Delegate>) that provides the functionality of a function pointer.

A delegate is a class that can hold a reference to a method. Unlike other classes, a delegate class has a signature, and it can hold references only to methods that match its signature. A delegate is thus equivalent to a type-safe function pointer or a callback.

To consume an event in an application, you must provide an event handler (an event-handling method) that executes program logic in response to the event and register the event handler with the event source and it must have the same signature as the event delegate. This process is referred to as event wiring.

The ArcObjects code excerpt below shows a custom ArcMap command wiring up to the Map object’s selection changed event. For simplicity, the event is wired up in the OnClick event.

```
[VB.NET]
  ‘Can’t use WithEvents because the outbound interface is not the
  ‘default interface

  ‘IActiveViewEvents is the sink event interface
  ‘SelectionChanged is the name of the event
  ‘IActiveViewEvents_SelectionChangedEventHandler is the delegate name

  ‘Declare the delegate
  Private SelectionChanged As IActiveViewEvents_SelectionChangedEventHandler

  Private m_mxDoc As IMxDocument

  Public Overloads Overrides Sub OnCreate(ByVal hook As Object)
    Dim app As IApplication
    app = hook
    m_mxDoc = app.Document
  End Sub

  Public Overrides Sub OnClick()
    Dim map As Map
    map = m_mxDoc.FocusMap

    ‘Create an instance of the delegate, add it to SelectionChanged event
    SelectionChanged = New
IActiveViewEvents_SelectionChangedEventHandler(AddressOf OnSelectionChanged)
    AddHandler map.SelectionChanged, SelectionChanged

  End Sub

  ‘Event handler
  Private Sub OnSelectionChanged()
    MessageBox.Show(“Selection Changed”)
  End Sub
```

```
[C#]
  // IActiveViewEvents is the sink event interface
  // SelectionChanged is the name of the event
  // IActiveViewEvents_SelectionChangedEventHandler is the delegate name
  IActiveViewEvents_SelectionChangedEventHandler m_selectionChanged;
  private ESRI.ArcGIS.ArcMapUI.IMxDocument m_mxDoc;

  public override void OnCreate(object hook)
    {
      IApplication app = hook as IApplication;
      m_mxDoc = app.Document as IMxDocument;

    }

    public override void OnClick()
    {
      IMap map = m_mxDoc.FocusMap;

      // Create a delegate instance and add it to SelectionChanged event
      m_selectionChanged = new
IActiveViewEvents_SelectionChangedEventHandler(SelectionChanged);
    ((IActiveViewEvents_Event)map).SelectionChanged += m_selectionChanged;
  }
  // Event hanlder
  private void SelectionChanged()
  {
    MessageBox.Show("Selection changed");
  }
```

Error handling

The error handling construct in Visual Studio .NET is known as structured exception handling. The constructs used may be new to Visual Basic users, but should be familiar to users of C++ or Java.

Structured exception handling is straightforward to implement, and the same concepts are applicable to either VB.NET or C#. VB.NET allows backward compatibility by also providing unstructured exception handling, via the familiar On Error GoTo statement and Err object, although this model is not discussed in this section.

Exceptions

Exceptions are used to handle error conditions in Visual Studio .NET. They provide information about the error condition.

An exception is an instance of a class which inherits from the System.Exception base class. Many different types of exception class are provided by the .NET Framework, and it is also possible to create your own exception classes. Each type extends the basic functionality of the System.Exception class by allowing further access to information about the specific type of error that has occurred.

An instance of an Exception class is created and thrown when the .NET Framework encounters an error condition. You can deal with exceptions by using the Try, Catch Finally construct.

Try, Catch, Finally

This construct allows you to catch errors that are thrown within your code. An example of this construct is shown below. An attempt is made to rotate an envelope, which throws an error.

```
[VB.NET]
  Dim env As IEnvelope = New EnvelopeClass()
  env.PutCoords(0D, 0D, 10D, 10D)
  Dim trans As ITransform2D = env
  trans.Rotate(env.LowerLeft, 1D)
Catch ex As System.Exception
  MessageBox.Show("Error: " + ex.Message)

  ' Perform any tidy up code.
End Try

[C#]
{
  IEnvelope env = new EnvelopeClass();
  env.PutCoords(0D, 0D, 10D, 10D);
  ITransform2D trans = (ITransform2D) env;
  trans.Rotate(env.LowerLeft, 1D);
}
catch (System.Exception ex)
{
  MessageBox.Show("Error: " + ex.Message);
}

{
  // Perform any tidy up code.
}
```

You place a try block is placed around the code which may fail. If the application throws an error within the Try block, the point of execution will switch to the first Catch block.

The Catch block handles a thrown error. The application executes the Catch block when the Type of a thrown error matches the Type of error specified by the Catch block. You can have more than one Catch block to handle different kinds of errors. The code shown below checks first if the exception thrown is a DivideByZeroException.

```
[VB.NET]
  ...
  Catch divEx As DivideByZeroException
    ' Perform divide by zero error handling.
  Catch ex As System.Exception
    ' Perform general error handling.
  ...

[C#]
  ...
  catch (DivideByZeroException divEx)
  {
    // Perform divide by zero error handling.
  }
  catch (System.Exception ex)
  {
    // Perform general error handling.
  }
  ...
```

If you do have more than one Catch block, note that the more specific exception Types should precede the general System.Exception, which will always succeed the type check.

The application always executes the Finally block, either after the Try block completes, or after a Catch block, if an error was thrown. The Finally block should therefore contain code which must always be executed, for example to clean up resources like file handles or database connections.

If you do not have any cleanup code, you do not need to include a Finally block.

Code without exception handling

If a line of code not contained in a Try block throws an error, the .NET runtime searches for a Catch block in the calling function, continuing up the call stack until a Catch block is found.

If no Catch block is specified in the call stack at all, the exact outcome may depend on the location of the executed code and the configuration of the .NET runtime. It is therefore advisable to at least include a Try, Catch, Finally construct for all entry points to a program.

Errors from COM components

The structured exception handling model differs from the HRESULT model used by COM. C++ developers can easily ignore an error condition in an HRESULT if they wished; in Visual Basic 6 however, an error condition in an HRESULT populates the Err object and raises an error.

The .NET runtime's handling of errors from COM components is somewhat similar to the way COM errors were handled at VB 6. If a .NET program calls a function in a COM component (through the COM interop services) and returns an error condition as the HRESULT, the HRESULT is used to populate an instance of the COMException class. This is then thrown by the .NET runtime, where you can handle it in the usual way, by using a Try, Catch Finally block.

It is advisable therefore to enclose all code that may raise an error in a COM component within a Try block with a corresponding Catch block to catch a COMException. Below is the first example rewritten to check for an error from a COM component.

```
[VB.NET]
  Dim env As IEnvelope = New EnvelopeClass()
  env.PutCoords(0D, 0D, 10D, 10D)
  Dim trans As ITransform2D = env
  trans.Rotate(env.LowerLeft, 1D)
Catch COMex As COMException
  If (COMex.ErrorCode = -2147220984) Then
```

```
    MessageBox.Show(“You cannot rotate an Envelope”)

    MessageBox.Show _
      (“Error “ + COMex.ErrorCode.ToString() + “: “ + COMex.Message)
  End If
Catch ex As System.Exception
  MessageBox.Show(“Error: “ + ex.Message)
...

[C#]
{
  IEnvelope env = new EnvelopeClass();
  env.PutCoords(0D, 0D, 10D, 10D);
  ITransform2D trans = (ITransform2D) env;
  trans.Rotate(env.LowerLeft, 1D);
}
catch (COMException COMex)
{
  if (COMex.ErrorCode == -2147220984)
    MessageBox.Show(“You cannot rotate an Envelope”);

    MessageBox.Show (“Error “ + COMex.ErrorCode.ToString() + “: “ +
COMex.Message);
}
catch (System.Exception ex)
{
  MessageBox.Show(“Error: “ + ex.Message);
}
...
```

The COMException class belongs to the System.Runtime.InteropServices namespace. It provides access to the value of the original HRESULT via the ErrorCode property which you can test to find out which error condition occurred.

Throwing errors and the exception hierarchy

If you are coding a user interface, you may wish to attempt to correct the error condition in code and try the call again. Alternatively you may wish to report the error to the user to let them decide which course of action to take; here you can make use of the Message property of the Exception class to identify the problem.

However, if you are writing a function that is only called from other code, you may wish to deal with an error by creating a specific error condition and propagating this error to the caller. You can do exactly this by using the Throw keyword.

To simply throw the existing error to the caller function, write your error handler simply by using the Throw keyword, as shown below.

```
[VB.NET]
Catch ex As System.Exception
...
```

```
[C#]
catch (System.Exception ex)
{
  throw;
}
...
```

If you wish to propagate a different or more specific error back to the caller, you should create a new instance of an Exception class, populate it appropriately, and throw this exception back to the caller. The example shown below uses the ApplicationException constructor to set the Message property.

```
[VB.NET]
Catch ex As System.Exception
  Throw New ApplicationException _
    ("You had an error in your application")
...

[C#]
catch (System.Exception ex)
{
  throw new ApplicationException("You had an error in your application");
}
...
```

If you do this however, the original exception is lost. In order to allow complete error information to be propagated, the Exception class includes the InnerException property. This property should be set to equal the caught exception, before the new exception is thrown. This creates an error hierarchy. Again, the example shown below uses the ApplicationException constructor to set the InnerException and Message properties.

```
[VB.NET]
Catch ex As System.Exception
   Dim appEx As System.ApplicationException = _
      New ApplicationException("You had an error in your application", ex)
   Throw appEx
...

[C#]
catch (System.Exception ex)
{
   System.ApplicationException appEx =
      new ApplicationException("You had an error in your application", ex);
   throw appEx;
}
...
```

In this way, the function that eventually deals with the error condition can access all the information about the cause of the condition and its context.

If you throw an error, the application will execute the current function's Finally clause before control is returned to the calling function.

Working with resources

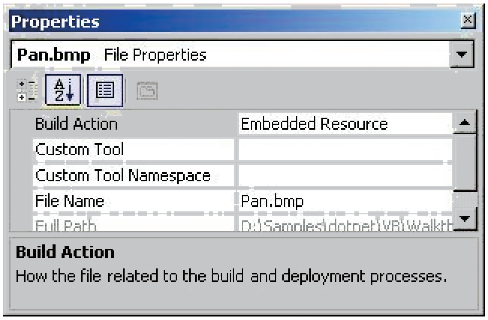
Using strings and embedded images directly (no localization)

If your customization does not support localization now, and you do not intend for it to support localization later, you can use strings and images directly without the need for resource files. For example, strings can be specified and used directly in your code:

```
[VB.NET]
Me.TextBox1.Text = “My String”

[C#]
this.textBox1.Text = “My String”;
```

Image files (bitmaps, jpegs, pngs, etc.) can be simply embedded in your assembly as follows:

1. Right click the Project in the Solution Explorer, choose Add, and then choose Add Existing Item.
2. In the Add Existing Item dialog box, browse to your image file and click Open.
3. In Solution Explorer, select the image file you just added, and then press F4 to display its Properties.
4. Set the Build Action property to Embedded Resource.



Properties

Pan.bmp File Properties

Build Action	Embedded Resource
Custom Tool	
Custom Tool Namespace	
File Name	Pan.bmp
Full Path	D:\Samples\dotnet\VB\Walkth

Build Action
How the file related to the build and deployment processes.

Now you can reference the image in your code. For example, the following code creates a Bitmap object from the first embedded resource in the assembly:

```
[VB.NET]
Dim res() As String = GetType(Form1).Assembly.GetManifestResourceNames()
If (res.GetLength(0) > 0)
  Dim bmp As System.Drawing.Bitmap = New System.Drawing.Bitmap( _
    GetType(Form1).Assembly.GetManifestResourceStream(res(0)))
  ...

[C#]
string[] res = GetType().Assembly.GetManifestResourceNames();
if (res.GetLength(0) > 0)
{
  System.Drawing.Bitmap bmp = new System.Drawing.Bitmap(
    GetType().Assembly.GetManifestResourceStream(res[0]));
...
```

Creating resources files

Before attempting to provide localized resources, you should ensure you are familiar with the process of creating resources files for your .NET projects. Even if you do not intend to localize your resources, you can still use resources files, instead of using images and strings directly as described above.

Visual Studio .NET projects use an XML-based file format to contain managed resources. These XML files have the extension .resx and can contain any kind of data (images, cursors, etc.) providing the data is converted to ASCII format. Resx files are compiled to .resources files which are binary representations of the resource data. Binary .resources files can be embedded by the compiler either into the main project assembly, or into a separate satellite assembly which contains only resources.

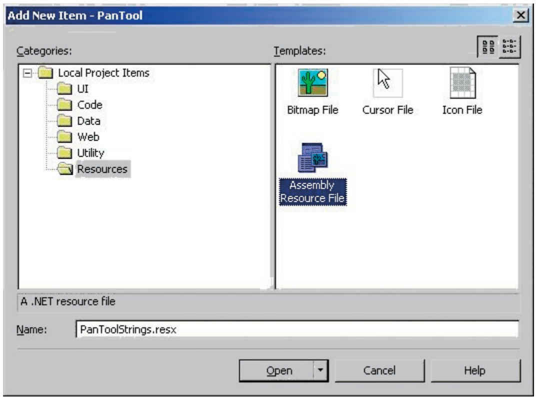
The following options are available to create your resources files. Each is discussed below.

- Creating a .resx file for string resources
- Creating resources files for image resources
- Compiling a .resx file into a .resources file

Creating a .resx file for string resources

If all you need to localize is strings (not images or cursors), you can use Visual Studio.NET to create a new .resx file which will be automatically compiled into a .resources module embedded in the main assembly.

1. Right click the Project name in the Solution Explorer, select Add, and then select Add New Item.
2. In the Add New item dialog select Assembly Resource File.

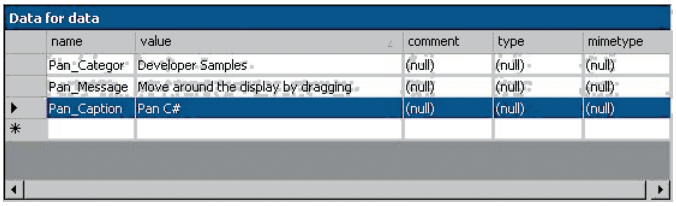


Add New Item - PanTool

A .NET resource file

Name: PanToolStrings.resx

3. Open the new .resx file in Visual Studio, and add name-value pairs for the culture-specific strings in your application.



Data for data

	name	value	comment	type	mimetype
	Pan_Categor	Developer Samples	(null)	(null)	(null)
	Pan_Message	Move around the display by dragging	(null)	(null)	(null)
▶	Pan_Caption	Pan C#	(null)	(null)	(null)
*					

4. When you compile your project, the .resx file will be compiled to a .resources module inside your main assembly.

Creating resources files for image resources

The process of adding images, icons, or cursors to a resources file in .NET is more complex than creating a file containing only string values, because the tools currently available in the Visual Studio .NET IDE can only be used to add string resources.

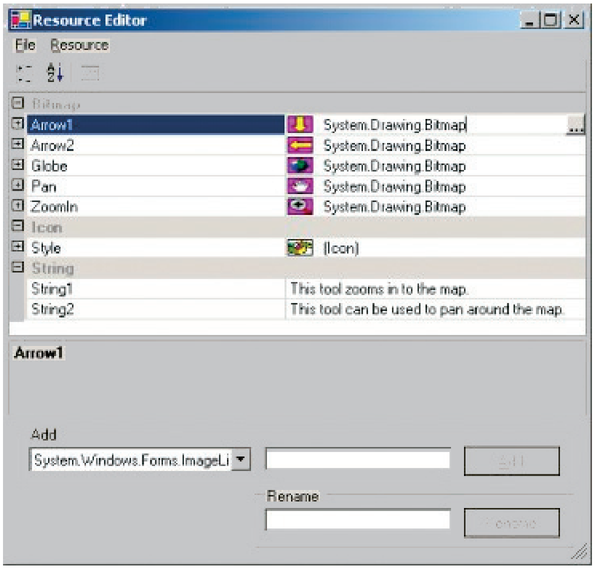
A list of tools useful for working with resources can be found in the Microsoft .NET Framework documentation at http://msdn.microsoft.com/library/default.asp?url=/library/en-us/cptutorials/html/appendix_b__resource_tools.asp.

However, a number of sample projects are available with Visual Studio .NET Framework SDK which can help you work with resource files. One such sample is the Resource Editor (ResEditor).

Additional information on theResEditor sample can be found at http://msdn.microsoft.com/library/default.asp?url=/library/en-us/cptutorials/html/resource_editor__reseditor_.asp.

The ResEditor sample can be used to add images, icons, imagelists and strings to a resources file. The tool cannot be used to add cursor resources. Files can be saved as either .resx or .resource files.

The ResEditor sample is provided by Microsoft as source code.You must build the sample first if you wish to create resource files using this tool. You can find information on building the SDK samples under the SDK sub-directory of your Visual Studio .NET installation.



Bitmap	
Arrow1	System.Drawing.Bitmap
Arrow2	System.Drawing.Bitmap
Globe	System.Drawing.Bitmap
Pan	System.Drawing.Bitmap
ZoomIn	System.Drawing.Bitmap
Icon	
Style	(Icon)
String	
String1	This tool zooms in to the map.
String2	This tool can be used to pan around the map.

Arrow1

Add: System.Windows.Forms.ImageLi

Creating resources files programmatically

You can create XML .resx files containing resources programmatically by using the *ResXResourceWriter* class (part of the .NET framework). You can create binary .resources files programmatically by using the *ResourceWriter* class (also part of the .NET framework). These classes will allow more flexibility to add the kind of resources you require.

These classes may be particularly useful if you wish to add resources which cannot be handled by the .NET Framework SDK samples and tools, for example, cursors. The basic usage of the two classes is very similar, first create a new resource writer class specifying the file name, then add resources individually by using the *AddResource* method.

The code below demonstrates how you could create a new .resx file using the *ResXResourceWriter* class, and add a bitmap and a cursor to the file.

```
[VB.NET]
Dim img As System.Drawing.Image = CType(New
System.Drawing.Bitmap("ABitmap.bmp"), System.Drawing.Image)
Dim cur As New System.Windows.Forms.Cursor("Pencil.cur")

Dim rsxw As New System.Resources.ResXResourceWriter("en-AU.resx")
rsxw.AddResource("MyBmp_jpg", img)
rsxw.AddResource("Mycursor_cur", cur)
rsxw.Close()
[C#]
System.Drawing.Image img = (System.Drawing.Bitmap) new
System.Drawing.Bitmap("ABitmap.bmp");
System.Windows.Forms.Cursor cur = new
System.Windows.Forms.Cursor("Pencil.cur");

System.Resources.ResXResourceWriter rsxw = new
System.Resources.ResXResourceWriter("en-GB.resx");
rsxw.AddResource("MyBmp_jpg", img);
rsxw.AddResource("Mycursor_cur", cur);
rsxw.Close();
```

The PanTool developer sample (Samples\Map Analysis\Tools) includes a script called "MakeResources" that shows you how to use the *ResXResourceWriter* class to write bitmap, cursor files and strings into an .resx file. It also shows you how to read from a .resx file using the *ResXResourceReader* class. The sample includes a .resx file that holds a bitmap, two cursors, and three strings.

Compiling a .resx file into a .resources file

XML-based .resx files can be compiled to binary .resources files either by using the Visual Studio IDE, or by using the ResX Generator (ResXGen) sample in the tutorial.

More information on the ResXGen can be found at http://msdn.microsoft.com/library/default.asp?url=/library/en-us/cptutorials/html/ResX_Generator__RESXGEN_.asp.

- Any .resx file included in a Visual Studio project will be compiled to a .resources module when the project is built. See the 'Using resources with localization' section below for more information on how multiple resource files are used for localization.
- If you wish you can convert a .resx file into a .resources file independently of the build process, using the .NET Framework SDK command resgen, for example:

```
resgen PanToolCS.resx PanToolCS.resources
```

Using resources with localization

This section explains how you can localize resources for your customizations.

How to use resources with localization

In .NET, a combination of a specific Language and Country/Region is called a *culture* . For example, the American dialect of English is indicated by the string “en-US”, and the Swiss dialect of French is indicated by “fr-CH”.

If you wish your project to support various cultures (languages and dialects), you should construct a separate .resources file containing culture-specific strings and images for each culture.

When you build a .NET project which uses resources, .NET embeds the default .resources file in the main assembly. Culture-specific .resources files are compiled into satellite assemblies (using the naming convention <Main Assembly Name>.resources.dll), and placed in sub-directories of the main build directory. The sub-directories are named after the culture of the satellite assembly they contain, for example Swiss-French resources would be contained in a sub-directory called “fr-CH”.

When an application runs, it automatically uses the resources contained in the satellite assembly with the appropriate culture. The appropriate culture is determined from the Windows settings. If a satellite assembly for the appropriate culture cannot be found, the default resources (those embedded in the main assembly) will be used instead.

The following sections give more information on creating your own .resx and .resources files.

The Visual Basic .NET and C# flavors of the Pan Tool developer sample illustrate how to localize resources for German language environments. The sample can be found in the Developer Samples\ArcMap\Commands and Tools\Pan Tool folder. Strictly speaking, the sample only requires localized strings, but the images have been changed for the “de” culture as well, to serve as illustration.

A batch file named buildResources.bat has been provided in the Pan Tool sample to create the default .resources files and the culture-specific satellite assemblies.

Embedding a default .Resources file in your project

1. Right click the Project name in the Solution Explorer, click Add, and then click Add Existing Item to browse to your .resx or .resources file.
2. In Solution Explorer, select the file you just added, and then click F4 to display its Properties.
3. Set the Build Action property to Embedded Resource.

   This will ensure that your application always has a set of resources to fall back on if there isn’t a resource dll for the culture your application runs in.

Creating .Resources.dll files for cultures supported by your project

1. First ensure you have a default .resx or .resources file in your project.
2. Take the default .resx or .resources file and create a separate localized file for each culture you want to support.
   - Each file should contain resources with the same Names - the Value of each resource in the file should contain the localized value.
   - Localized resource files should be named according to their culture, e.g. <BaseName>.<Culture>.resx or <BaseName>.<Culture>.resources.
3. Add the new resources files to the project, ensuring each one has its Build Action set to Embedded Resource.

4. Build the project

   The compiler and linker will create a separate satellite assembly for each culture. The satellite assemblies will be placed in sub-directories under the directory holding your main assembly. The sub-directories will be named by culture, which allows the .NET runtime to locate the resources appropriate to the culture the application runs in.

   The main (default) resources file will be embedded in the main assembly.

Assembly versioning and redirection

Applications that are built using a specific version of a strongly named assembly require the same assembly at run time. For example, if you create an application that uses ESRI.ArcGIS.System version 9.0.452, you will not be able to run this application on a system that has a newer version of ESRI.ArcGIS.System (e.g. 9.0.0.692) installed. This may be the case if someone has installed a newer version of ArcGIS; however, using configuration files you can redirect an application to use a newer version of an assembly.

You have two choices for redirecting assemblies:

- Application configuration files
- Machine configuration files

Application configuration files

Application configuration files contain settings specific to an application. This file contains configuration settings that the common language runtime reads (such as assembly binding policy, remoting objects, and so on), and settings that the application can read.

The name and location of the application configuration file depend on the application's host, which can be one of the following:

- Executable–hosted application—The configuration file for an application hosted by the executable host is in the same directory as the application. The name of the configuration file is the name of the application with a .config extension. For example, an application called myApp.exe can be associated with a configuration file called myApp.exe.config.
- ASP.NET-hosted application—ASP.NET configuration files are called Web.config. Configuration files in ASP.NET applications inherit the settings of configuration files in the URL path. For example, given the URL www.esri.com/aaa/bbb, where www.esri.com/aaa is the Web application, the configuration file associated with the application is located at www.esri.com/aaa. ASP.NET pages that are in the subdirectory /bbb use both the settings that are in the configuration file at the application level and the settings in the configuration file that is in /bbb.
- Internet Explorer-hosted application—If an application hosted in Internet Explorer has a configuration file, the location of this file is specified in a <link> tag with the following syntax:

```
<link rel="ConfigurationFileName" href="location">
```

In this tag, *location* is a URL to the configuration file. This sets the application base. The configuration file must be located on the same Web site as the application.

Machine configuration files

The machine configuration file, Machine.config, contains settings that apply to an entire computer. This file is located in the %runtime install path%\Config directory. Machine.config contains configuration settings for machine-wide assembly binding, built-in remoting channels, and ASP.NET.

The configuration system first looks in the machine configuration file for the <appSettings> element and other configuration sections that a developer might define. It then looks in the application configuration file. To keep the machine configuration file manageable, it is best to put these settings in the application configuration file. However, putting the settings in the machine configuration file can make your system more maintainable. For example, if you have a third-party component that both your client and server application use, it is easier to put the settings for that component in one place. In this case, the machine configuration file is the appropriate place for the settings, so you don't have the same settings in two different files.

Deploying an application using XCOPY will not copy the settings in the machine configuration file.

The configuration file below shows how to bind to an assembly and then redirect it to a newer version.

```
<configuration>
  <runtime>
    <assemblyBinding xmlns=”urn:schemas-microsoft-com:asm.v1">
      <dependentAssembly>
        <assemblyIdentity name=”ESRI.ArcGIS.System”
                 publicKeyToken=”8fc3cc631e44ad86"
                 culture=”neutral” />
        <!– Assembly versions can be redirected in application, publisher
policy, or machine configuration files. –>
        <bindingRedirect oldVersion=”9.0.0.452" newVersion=”9.0.0.692"/>
      </dependentAssembly>
    </assemblyBinding>
  </runtime>
</configuration>
```

ArcGIS DEVELOPMENT USING .NET

Using .NET you can customize the ArcGIS applications, create standalone applications that use ESRI's types, and extend ESRI's types. For example, you can create a custom tool for ArcMap, create a standalone application that uses the MapControl, or create a custom layer. This section discusses several key issues related to developing with ArcGIS and .NET.

Registering .NET components with COM

Extending the ArcGIS applications with custom .NET components requires registering the components in the COM registry and exporting the .NET assemblies to a type library (tlb). When developing a component, there are two ways to perform this task, you can use the RegAsm utility that ships with the .NET

Framework SDK, or use Visual Studio.NET which has a 'Register for COM Interop' compiler flag.

The example below shows an EditTools assembly being register with COM. The /tlb parameter specifies that a type library should additionally be generated and the /codebase option indicates that the path to the assembly should be included in the registry settings. Both of these parameters are required when extending the ArcGIS applications with .NET components.

```
regasm EditTools.dll /tlb:EditTools.tlb /codebase
```

Visual Studio.NET performs this same operation automatically if you set the 'Register for COM Interop' compiler flag – this is the simplest way to perform the registration on a development machine. To check a project's settings, select 'Project Properties' from the 'Project' menu and then look at the Build property under 'Configuration Properties'. The very last item is 'Register for COM Interop', set this property to True.

Registering .NET classes in COM component categories

Much of ArcGIS's extensibility relies on COM component categories. In fact, most custom ArcGIS components must be registered in component categories appropriate to their intended context and function in order for the host application to make use of their functionality. For example, all ArcMap commands and tools must be registered in the *ESRI Mx Commands* component category. There are a few different ways you can register a .NET component in a particular category but before doing so, the .NET components must be registered with COM. See the 'Registering .NET Components with COM' section above for details.

Customize dialog

Custom .NET ArcGIS commands and tools can quickly be added to toolbars via the 'Add From File…" button on the Customize dialog. In this case, you simply have to browse for the tlb and open it. The ArcGIS framework will automatically added the classes you select in the type library to the appropriate component category.

Categories utility

Another option is to use the Component Categories Manager (Categories.exe). In this case you select the desired component category in the utility and then browse for your type library and select the appropriate class.

COMRegisterFunction

The final and recommended solution is to add code to your .NET classes that will automatically register them in a particular component category whenever the component is registered with COM. The .NET Framework contains two attribute classes (ComRegisterFunctionAttribute and ComUnregisterFunctionAttribute) that allow you to specify methods which will be called whenever your component is being registered or unregistered. Both methods are passed the CLSID of the class currently being registered and with this information you can you can write code inside the methods to make the appropriate registry entries or deletions. Registering a component in a component category requires that you also know the component category's unique ID (CATID).

The code excerpt below shows a custom ArcMap command that automatically registers itself in the MxCommands component category whenever the .NET assembly in which it resides is registered with COM.

```
  public sealed class AngleAngleTool: BaseTool
  {

     [ComRegisterFunction()]
     static void Reg(String regKey)
     {
       Microsoft.Win32.Registry.ClassesRoot.CreateSubKey(regKey.
Substring(18)+ “\\Implemented Categories\\” + “{B56A7C42-83D4-11D2-A2E9-
080009B6F22B}”);
    }
   [ComUnregisterFunction()]
   static void Unreg(String regKey)
    {
     Microsoft.Win32.Registry.ClassesRoot.DeleteSubKey(regKey.Substring(18)+
“\\Implemented Categories\\” + “{B56A7C42-83D4-11D2-A2E9-080009B6F22B}”);
    }
```

To simplify this process, ESRI provides classes for each component category ArcGIS exposes with static functions to register and unregister components. Each class knows the GUID of the component category it represents so registering custom components becomes greatly simplified. For more details on using these classes see the 'Working with the ESRI .NET Component Category Classes' section below.

Simplify your code using the ESRI.ArcGIS.Utility assembly

Part of the ArcGIS Developer Kit includes a number of .NET utility classes that facilitate .NET development by taking advantage of a few .NET capabilities including object inheritance and static functions.

Working with the ESRI .NET Base Classes

ESRI provides two abstract base classes (BaseCommand and BaseTool) to help you create new custom commands and tools for ArcGIS. The classes are abstract classes (marked as MustInherit in Visual Basic .NET), which means that although the class may contain some implementation code, it cannot itself be instantiated directly and can only be used by being inherited by another class. Both base classes are defined in the ESRI.ArcGIS.Utility assembly and belong to the ESRI.ArcGIS.Utility.BaseClasses namespace.

These base classes simplify the creation of custom commands and tools by providing a default implementation for each of the members of ICommand and ITool. Instead of stubbing out each member and providing implementation code, you only have to override the members that your custom command or tool requires. The exception is ICommand::OnCreate; this member must be overridden in your derived class.

Using these base classes is the recommended way to create commands and tools for ArcGIS applications in .NET languages. You can create similar COM classes from first principles; however, you should find the base class technique to be a quicker, simpler, less error-prone method of creating commands and tools.

Syntax

Both base classes additionally have an overloaded constructor, allowing you to quickly set many of the properties of a command or tool (such as Name and Category) via constructor parameters.

The overloaded BaseCommand constructor has the following signature:

```
[VB.NET]
Public Sub New( _
  ByVal bitmap As System.Drawing.Bitmap _
  ByVal caption As String _
  ByVal category As String _
  ByVal helpContextId As Integer _
  ByVal helpFile As String _
  ByVal message As String _
  ByVal name As String _
  ByVal tooltip As String)

[C#]
  public BaseCommand(
    System.Drawing.Bitmap bitmap,
    string caption,
    string category,
    int helpContextId,
    string helpFile,
    string message,
    string name,
    string toolTip,
  );
```

The overloaded BaseTool constructor has the following signature:

```
[VB.NET]
Public Sub New( _
  ByVal bitmap As System.Drawing.Bitmap _
  ByVal caption As String _
  ByVal category As String _
  ByVal cursor As System.Windows.Forms.Cursor _
  ByVal helpContextId As Integer _
  ByVal helpFile As String _
  ByVal message As String _
  ByVal name As String _
  ByVal tooltip As String _
)

[C#]
  public BaseTool(
    System.Drawing.Bitmap bitmap,
    string caption,
    string category,
    System.Windows.Forms.Cursor cursor,
    int helpContextId,
    string helpFile,
    string message,
    string name,
    string toolTip,
  );
```

Inheriting the base classes

You can use these parameterized constructors when you write your new classes, for example as shown below for a new class called PanTool that inherits the BaseTool class.

```
[VB.NET]
Public Sub New()
  MyBase.New( Nothing, “Pan”, “My Custom Tools”, _
    System.Windows.Forms.Cursors.Cross,     0, “”, “Pans the map.”,
“PanTool”, “Pan”)
End Sub

[C#]
public PanTool() : base ( null,”Pan”, “My Custom Tools”,
  System.Windows.Forms.Cursors.Cross, 0, “”,”Pans the map.”, “PanTool”,
“Pan”)
{
  ...
}
```

Setting base class members directly

As an alternative to using the parameterized constructors, you can alternatively set the members of the base class directly.

The base classes expose their internal member variables to the inheritor class, one per property, so that you can directly access them in your derived class. For example, instead of using the constructor to set the ‘Caption’ or overriding the Caption function, you can set the ‘m_caption’ class member variable declared in the base class.

```
[VB.NET]
Public Sub New()
  MyBase.New()
  MyBase..m_bitmap = New
System.Drawing.Bitmap([GetType]().Assembly.GetManifestResourceStream(“Namespace.Pan.bmp”))
  MyBase..m_cursor = System.Windows.Forms.Cursors.Cross
  MyBase..m_category = “My Custom Tools”
  MyBase..m_caption = “Pan”
  MyBase..m_message = “Pans the map.”
  MyBase..m_name = “PanTool”
  MyBase..m_toolTip = “Pan”
End Sub
```

```
[C#]
public PanTool()
{
  base.m_bitmap = new
System.Drawing.Bitmap(GetType().Assembly.GetManifestResourceStream("Namespace.Pan.bmp"));
  base.m_cursor = System.Windows.Forms.Cursors.Cross;
  base.m_category = "My Custom Tools";
  base.m_caption = "Pan";
  base.m_message = "Pans the map.";
  base.m_name = "PanTool";
  base.m_toolTip = "Pan";
}
```

Overriding members

When you create custom commands and tools that inherit a base class, you will more than likely need to override a few members. When you override a member in your class, the implementation code that you provide for that member will be executed instead of the default member implementation inherited from the base class. For example, the OnClick method in the BaseCommand has no implementation code at all (OnClick will not do anything by default), which may be suitable for a tool but probably not for a command.

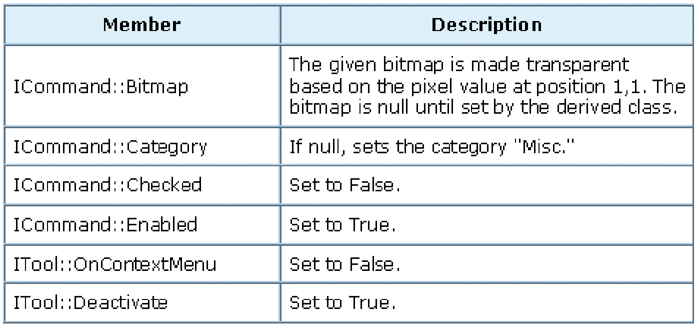
To override any member, you can right-click the member of the base class in the Solution Explorer Window then choose Add, Override to stub out the member as overridden. (Note that if you right-click on the member of the underlying interface (ICommand or ITool) instead of the base class member, the overridden member will not include the overrides keyword, and the method will instead be shadowed).

```
[VB.NET]
Public Overrides Sub OnClick()
  ' Your OnClick
End Sub

[C#]
public override void OnClick()
{
  // Your OnClick
}
```

Alternatively, to override a member of the base class, choose (Overrides) from the right hand drop-down list in the code window wizard bar, and then choose the member you wish to override from the left hand drop-down list. This will stub out the member as overridden.

What do the base classes do by default?

The table below shows the base class members that have a significant base class implementation, along with a description of that implementation. Override these members when the base class behavior is not consistent with your customization. For example, Enabled is set to True by default, if you want your custom command enabled only when a specific set of criteria has been met, you must override this property in your derived class.



Member	Description
ICommand::Bitmap	The given bitmap is made transparent based on the pixel value at position 1,1. The bitmap is null until set by the derived class.
ICommand::Category	If null, sets the category "Misc."
ICommand::Checked	Set to False.
ICommand::Enabled	Set to True.
ITool::OnContextMenu	Set to False.
ITool::Deactivate	Set to True.

Working with the ESRI .NET component category classes

To help the register .NET components in COM component categories, ESRI provides the ESRI.ArcGIS.Utility.CATIDs namespace which has classes that represent each of the ArcGIS component categories. Each class knows its CATID and exposes static methods (Register and Unregister) for adding and removing components. Registering your component becomes as easy as adding COM registration methods with the appropriate attributes, and passing the received CLSID to the appropriate static method.

The example below shows a custom Pan tool that registers itself in the ESRI Mx Commands component category. Notice in this example we are using MxCommands.Register and MxCommands.Unregister instead of Microsoft.Win32.Registry.ClassesRoot.CreateSubKey and Microsoft.Win32.Registry.ClassesRoot.DeleteSubKey.

```
[VB.NET]
Public NotInheritable Class PanTool
  Inherits BaseTool

  <ComRegisterFunction()> _
  Public Shared Sub Reg(ByVal regKey As [String])
    MxCommands.Register(regKey)
  End

  <ComUnregisterFunction()> _
  Public Shared Sub Unreg(ByVal regKey As [String])
    MxCommands.Unregister(regKey)
  End Sub

[C#]
public sealed class PanTool : BaseTool
{
  [ComRegisterFunction()]
  static void Reg(string regKey)
  {
    MxCommands.Register(regKey);
  }
```

```
[ComUnregisterFunction()]
static void Unreg(string regKey)
{
  MxCommands.Unregister(regKey);
}
```

Extending the Server

When using .NET to create a COM object for use in the GIS server, there are some specific guidelines you need to follow to ensure that you can use your object in a server context, and to ensure that it will perform well in that environment. The guidelines below apply specifically to COM objects you create to run within the server.

- You must explicitly create an interface that your COM class implements. Unlike Visual Basic 6, .NET will not create an implicit interface for your COM class that you can use when creating the object in a server context.
- Your COM class should be marshaled using the Automation marshaller. You specify this by adding the AutomationProxyAttribute attribute to your class with a value of true.
- Your COM class should generate a dual class interface. You specify this by adding the ClassInterfaceAttribute attribute to your class with a value of ClassInterfaceType.AutoDual.
- To ensure that your COM object performs well in the server, it must inherit from ServicedComponent which is in the System.EnterpriseServices assembly. This is necessary due to the current COM interop implementation of the .NET framework.

For more details and an example of a custom Server COM object written in .NET, see Chapter 4, 'Developing ArcGIS Server applications' in the *ArcGIS Server Administrator and Developer Guide.*

Releasing COM references

ArcGIS Engine and ArcGIS Desktop applications

An unexpected crash may occur when a standalone application attempts to shutdown. For example, an application hosting a MapControl with a loaded map document will crash on exit. The crashes result from COM objects hanging around longer than expected. To stop the crash, all COM references must be unloaded prior to shutdown. To help unload COM references, a static Shutdown function has been added to the ESRI.ArcGIS.Utility assembly. The following code excerpt shows the function in use.

```
[VB.NET]
Private Sub Form1_Closing(ByVal sender As Object, ByVal e As
System.ComponentModel.CancelEventArgs) Handles MyBase.Closing
   ESRI.ArcGIS.Utility.COMSupport.AOUninitialize.Shutdown()
End Sub

[C#]
private void Form1_Closing(object sender, CancelEventArgs e)
{
  ESRI.ArcGIS.Utility.COMSupport.AOUninitialize.Shutdown();
}
```

The *AOUninitalize.Shutdown* function handles most of the shut down problems in standalone applications but you may still experience problems as there are COM objects that additionally require explicit releasing; in these cases, call *System.Runtime.InteropServices.Marshal.ReleaseComObject()* to decrement the reference count which allows the application to terminate cleanly. The StyleGallery is one such object and the following example documents how to handle references to this class.

```
[VB.NET]
  Dim styleGallery As IStyleGallery
  styleGallery = New StyleGalleryClass
  MessageBox.Show(styleGallery.ClassCount)
  Marshal.ReleaseComObject(styleGallery)
[C#]
  IStyleGallery sg = new StyleGalleryClass() as IStyleGallery;
  MessageBox.Show(sg.ClassCount.ToString());
  Marshal.ReleaseComObject(sg);
```

Working with geodatabase cursors in ArcGIS Server

Some objects that you can create in a server context may lock or use resources that the object frees only in its destructor. For example, a geodatabase cursor may acquire a shared schema lock on a file-based feature class or table on which it is based, or may hold onto an SDE stream.

While the shared schema lock is in place, other applications can continue to query or update the rows in the table, but they cannot delete the feature class or modify its schema. In the case of file-based data sources, such as shapefiles, update cursors acquire an exclusive write lock on the file, which will prevent other applications from accessing the file for read or write. The effect of these locks is that the data may be unavailable to other applications until all of the references on the cursor object are released.

In the case of SDE data sources, the cursor holds onto an SDE stream, and if the application has multiple clients, each may get and hold onto an SDE stream, eventually exhausting the maximum allowable streams. The effect of the number of SDE streams exceeding the maximum is that other clients will fail to open their own cursors to query the database.

Because of the above reasons, it's important to ensure that your reference to any cursor your application opens is released in a timely manner. In .NET, your reference on the cursor (or any other COM object) will not be released until garbage collection kicks in. In a web application or web service, servicing multiple concurrent sessions and requests, relying on garbage collection to release references on objects will result in cursors and their resources not being released in a timely manner.

To ensure a COM object is released when it goes out of scope, the *WebControls* assembly contains a helper object called *WebObject*. Use the *ManageLifetime* method to add your COM object to the set of objects that will be explicitly released when the *WebObject* is disposed. You must scope the use of *WebObject* within a *Using* block. When you scope the use of *WebObject* within a using block, any object (including your cursor) that you have added to the *WebObject* using the *ManageLifetime* method will be explicitly released at the end of the using block.

The following example demonstrates this coding pattern:

[VB.NET]

```
  Private Sub doSomething_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles doSomething.Click
    Dim webobj As WebObject = New WebObject
    Dim ctx As IServerContext = Nothing
    Try
      Dim serverConn As ServerConnection = New ServerConnection(“doug”,
True)
      Dim som As IServerObjectManager = serverConn.ServerObjectManager

      ctx = som.CreateServerContext(“Yellowstone”, “MapServer”)
      Dim mapsrv As IMapServer = ctx.ServerObject
      Dim mapo As IMapServerObjects = mapsrv
      Dim map As IMap = mapo.Map(mapsrv.DefaultMapName)

      Dim flayer As IFeatureLayer = map.Layer(0)
      Dim fClass As IFeatureClass = flayer.FeatureClass

      Dim fcursor As IFeatureCursor = fClass.Search(Nothing, True)
      webobj.ManageLifetime(fcursor)

      Dim f As IFeature = fcursor.NextFeature()
      Do Until f Is Nothing
        ‘ do something with the feature
        f = fcursor.NextFeature()
      Loop

    Finally
      ctx.ReleaseContext()
      webobj.Dispose()
    End Try
  End Sub
```

[C#]

```
  private void doSomthing_Click(object sender, System.EventArgs e)
  {
    using (WebObject webobj = new WebObject())
    {
      ServerConnection serverConn = new ServerConnection(“doug”,true);
      IServerObjectManager som = serverConn.ServerObjectManager;

      IServerContext ctx =
som.CreateServerContext(“Yellowstone”,”MapServer”);
      IMapServer mapsrv = ctx.ServerObject as IMapServer;
      IMapServerObjects mapo = mapsrv as IMapServerObjects;
      IMap map = mapo.get_Map(mapsrv.DefaultMapName);

      IFeatureLayer flayer = map.get_Layer(0) as IFeatureLayer;
      IFeatureClass fclass = flayer.FeatureClass;
      IFeatureCursor fcursor = fclass.Search(null, true);
      webobj.ManageLifetime(fcursor);

      IFeature f = null;
      while ((f = fcursor.NextFeature()) != null)
      {
        // do something with the feature
      }

      ctx.ReleaseContext();
    }
  }
```

The *WebMap*, *WebGeocode* and *WebPageLayout* objects also have a *ManageLifetime* method. If you are using, for example, a *WebMap* and scope your code in a using block, you can rely on these objects to explicitly release objects you add with *ManageLifetime* at the end of the using block.

Deploying .NET ArcGIS customizations

All ArcGIS Engine and Desktop customizations require an ArcGIS installation on all client machines. The ArcGIS installation must include the ESRI primary interop assemblies which the setup program installs in the global assembly cache. For example, deploying a standalone GIS application that only requires an ArcGIS Engine license requires an ArcGIS Engine installation on all target machines.

Standalone applications

Deploying standalone applications to either ArcGIS Engine or Desktop clients involves copying over the executable to the client machine. Copying over the executable can be as simple as using xcopy or more involved such as creating a custom install/setup program. Note, aside from the ArcGIS primary interop assemblies and the .NET Framework assemblies, all dependencies must additionally be packaged and deployed.

ArcGIS components

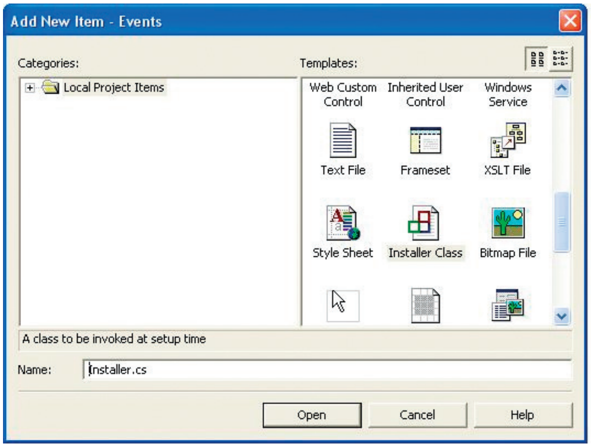
Components that extend the ArcGIS applications are trickier to deploy than standalone applications because they must be registered with COM and in specific component categories. As discussed earlier, implementing COMRegisterFunction and COMUnregisterFunctions facilitates deployment by providing self category registration but this only occurs when the components are registered.

There are two techniques for registering components with COM. One option is to run the register assembly utility (RegAsm.exe) that ships with the .NET Framework SDK. This is typically not a viable solution as client machines may or may not have this utility and it's difficult to automate. The second and recommended approach is to add an automatic registration step to a custom setup/install program.

The key to creating a custom install program that both deploys and registers components is the System.Runtime.InteropServices.RegistrationServices class. This class has the members: RegisterAssembly and UnregisterAssembly, which register and unregister managed classes with COM. These are the same functions the RegAsm utility uses. Using these functions inside a custom installer class along with a setup program is the complete solution.

The basic steps below outline the creation of a deployable solution. Note, the steps assume you are starting with a solution that already contains a project with at least one COM enabled class.

1. In Visual Studio.NET, add a new Installer Class and name it accordingly.



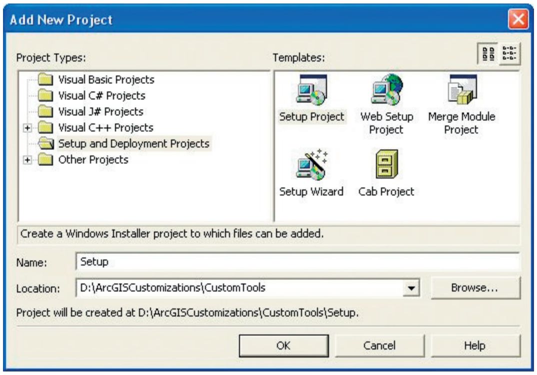
A class to be invoked at setup time

Name: Installer.cs

Override the Install and Uninstall functions that are implemented in the Installer base class and use the RegistrationServices class's RegisterAssembly and UnregisterAssembly methods to register the components. Make sure you use the SetCodeBase flag; this indicates that the code base key for the assembly should be set in the registry.

```
[VB.NET]
Public Overrides Sub Install(ByVal stateSaver As
System.Collections.IDictionary)
   MyBase.Install(stateSaver)
   Dim regsrv As New RegistrationServices
   regsrv.RegisterAssembly(MyBase.GetType().Assembly,
AssemblyRegistrationFlags.SetCodeBase)
End Sub

Public Overrides Sub Uninstall(ByVal savedState As
System.Collections.IDictionary)
   MyBase.Uninstall(savedState)
   Dim regsrv As New RegistrationServices
   regsrv.UnregisterAssembly(MyBase.GetType().Assembly)
  End Sub
End Class

[C#]
public override void Install(IDictionary stateSaver)
{
  base.Install (stateSaver);
  RegistrationServices regSrv = new RegistrationServices();
  regSrv.RegisterAssembly(base.GetType().Assembly,
AssemblyRegistrationFlags.SetCodeBase);
}

public override void Uninstall(IDictionary savedState)
{
  base.Uninstall (savedState);
  RegistrationServices regSrv = new RegistrationServices();
  regSrv.UnregisterAssembly(base.GetType().Assembly);
}
```

2. Add a Setup program to your solution.



Create a Windows Installer project to which files can be added.

Name: Setup

Location: D:\ArcGISCustomizations\CustomTools

Project will be created at D:\ArcGISCustomizations\CustomTools\Setup.

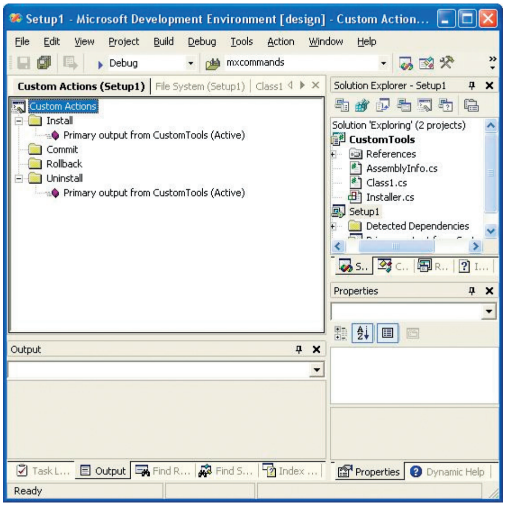
a. In the Solution Explorer, right-click on the new project and select 'Add -> Project Output'. Select the project you wish to deploy and choose 'Primary output'.

b. From the list of detected dependencies that is regenerated, remove all references to ESRI primary interop assemblies (e.g. ESRI.ArcGIS.System) and stdole.dll. The only items typically left in the list are your tlb and 'Primary output from <AssemblyName><Version>' which represents the dll or exe you are compiling.

c. The final steps involve associating the custom installation steps configured in the new installer class with the Setup project. To do this, right-click on the Setup project in the Solution Explorer and choose 'View Custom Actions'.

d. In the resulting view, right-click on 'Install' folder and choose 'Add Custom Action'. Double-click the 'Application' folder, then double-click the 'Primary output from <AssemblyName><Version>' item. This steps associates the custom install function created earlier with the setup's custom install action.

e. Repeat the last step for the setup's uninstall.



3. Finally, rebuild the entire solution to generate the setup executable. Running the executable on a target machine installs the components and registers them with COM. The COMRegisterFunction routines then register the components in the appropriate component categories.

ArcGIS Server deployments

To deploy web applications developed on a development server to product production servers, use the built-in Visual Studio.NET tools.

1. In the Solution Explorer, click on your project.

2. Click the *Project* menu, then select *Copy Project*.

3. In the Copy Project dialog, specify the deployment location.

4. Click OK.

In addition to copying the project, you must copy and register any related dlls containing custom COM objects onto your web server and all the GIS server's server object container machines.

The ArcGIS version 9.0 Java API is a programming interface which interoperates with ArcObjects and is specifically designed to target Java developers. Java technology is both a platform and an object-oriented programming language developed by Sun Microsystems which comes in three versions and consists of two components:

Versions:

- Java 2 platform, Standard Edition (J2SE)
- Java 2 platform, Enterprise Edition (J2EE)
- Java 2 platform, Micro Edition (J2ME)

Components:

- Java Virtual Machine (JVM) - Java runtime and client/server compilers.
- Java Application Programming Interface (API) - Suite of core, integration, and user interface toolkits.

The Java language is important because it is an open standard. All implementations of the programming language must meet the standards provided by the JVM. This enables applications to run on any hardware platforms that host the JVM.

PLATFORM CONFIGURATION

This section will describe all the necessary configurations needed to be productive with the Java API including classpath and environment settings.

Java Developer Kit

In order to develop with ArcObjects using the Java API, you must have the Java 2 Platform Standard Software Developer Kit (J2SDK) installed. All of your J2SDK tools are located in the install directory. You can either explicitly invoke them from that directory or add it to your PATH environment variable. Adding the directory to your PATH variable involves two steps:

Setting the JAVA_HOME variable is not absolutely necessary; however, some Java IDEs and Java tools require it be set.

1. Create a new environment variable named JAVA_HOME.

```
JAVA_HOME=[path to JDK install directory]
```

For example:

```
        JAVA_HOME=c:\j2sdk
```

Adding the PATH variable allows you to run executables (javac, java, javadoc, etc.) from any directory on your system.

2. Edit the PATH variable to include the bin directory of JAVA_HOME.

```
PATH=..;%JAVA_HOME%\bin
```

In order to compile server-based applications like servlets and EJBs, you will also need to install and include the Java 2 Enterprise Edition toolkit in your classpath. Java application servers generally provide this or you can get the reference implementation provided by Sun Microsystems.

ArcGIS Engine

The ArcGIS Engine developer kit uses standard Java Native Interface (JNI) to access core ArcObjects components. This requires some native libraries to be in the developer's path when compiling and running applications. You must be sure to include the correct paths to invoke interoperabilty into native ArcObjects.

The native *.dlls are located in the following location:

```
..\ArcGIS\bin
```

ESRI recommends setting an ARCENGINEHOME environment variable and added the bin directory to your PATH variable. Although this is not a requirement to use the ArcGIS Engine developer kit, the developer samples all use this variable to ensure your classpath setting are accurate.

Setting the ARCENGINEHOME variable:

```
ARCENGINEHOME=[path to ArcGIS install directory]
```

For example:

```
ARCENGINEHOME=c:\ArcGIS
```

Editing the PATH variable enables your system to use the native resource libraries that ship with the ArcGIS Engine Runtime. Edit the PATH directory to include the jre\bin directory of ARCENGINEHOME.

```
PATH=..%ARCENGINEHOME%\java\jre\bin
```

Classpath

The Java API provides Java Archive (JAR) files (*.jar) for ArcObjects and the native runtime libraries. These JAR files are located on disk at %ARCENGINEHOME%\java

All Java applications built with any of the ArcGIS developer kits must have the following JARs referenced in the respective application's classpath:

- arcobjects.jar—contains all of the non-UI classes in one complete archive
- jintegra.jar—contains all classes for the runtime library that handles interop to COM

In addition, individual arcgis_xxx.jar files should be added to your classpath as needed. For example, applications that leverage the Java visual beans included with ArcObjects, require that the arcgis_visualbeans.jar file be added in the classpath.

JRE

The ArcGIS Engine and Server developer kits include a version of the Java Runtime Environment (JRE). This enables you to run any ArcGIS Java application as long as all the necessary settings described above are local to the runtime. You will notice the necessary *.dlls in the bin directory and the necessary *.jars in the library extension directory. All you need to do to get started with this runtime environment is ensure that the bin directory is added to your PATH environment variable:

```
PATH=..\ArcGIS\java\jre\bin
```

JAVA PROGRAMMING TECHNIQUES

This section provides you with some fundamental concepts of the Java programming language. It assumes you understand general programming concepts but are relatively new to Java.

Features of the Java Virtual Machine

The Java Virtual Machine (JVM) specification provides a platform independent, abstract computer for executing code. The JVM knows nothing about the Java language, instead it understands a particular binary format, the class file that contains instructions in the form of bytecodes. The Java Virtual Machine specification provides an environment that both compiles and interprets programs. The compiler creates a .java file, produces a series of bytecodes, and stores them in a .class file, and the Java interpreter executes the bytecodes stored in the .class file.

Each implementation of the JVM is platform specific as it actually interacts with the specific operating system. The JVM handles things such as memory allocation, garbage collection, and security monitoring.

Java Native Interfaces

Even though Java programs are designed to run on multiple platforms, there may be times where the standard Java class library doesn't support platform-dependent features needed by a particular application or a Java program needs to implement a lower-level program and have the Java program call it. The JNI is a standard cross-platform programming interface provided by the Java language. It enables you to write Java programs that can operate with applications and libraries written in other programming languages, such as C or C++. This is the technology used to bridge native ArcObjects with the Java API in ArcGIS.

In order to initialize your Java environment for native usage of ArcObjects, every ArcGIS Engine Java application must call the static *initializeEngine()* method on the *EngineInitializer* class. This should be the first call you do, even before *AoInitialize*.

To see how the initializeEngine *method is used as the first call, refer to the Java developer samples in* ArcGIS Developer Help.

```
public static void main(String[] args){
  /* always initialize ArcGIS Engine for native usage */
  EngineInitializer.initializeEngine();
  ...
}
```

ArcGIS DEVELOPMENT USING JAVA

This section is intended for developers using the Java SDK for ArcGIS Engine. The SDK provides interoperability with ArcObjects, allowing a developer to access ArcObjects as though they were Java objects. The API is not limited to any specific Java Virtual Machine or platform and uses standard Java Native Interface to access ArcObjects. The API exposes the complete functionality of ArcObjects via Java classes and interfaces, which allows Java developers to write once, run anywhere, and also benefit from ArcObjects component reuse. The Java API provides "Proxy" classes that are generated from ArcObjects components type libraries (TLBs) which allow interoperability with all of the underlying components. These proxy classes expose ArcObjects properties, methods, and events via their Java equivalents.

Interfaces

Native ArcObjects uses an interface-based programming model. The concept of an interface is fundamental to ArcObjects and emphasizes four points:

1. An interface is not a class.
2. An interface is not an object.
3. Interfaces are strongly typed.
4. Interfaces are immutable.

ArcObjects interfaces are abstract, meaning there is no implementation associated with an interface. Objects use type inheritance; the code associated with an interface comes from the class implementation.

This model shares some features of the Java interface model, which was introduced to enhance Java's single-inheritance model. An interface in the Java language is a specification of methods that an object declares it implements. A Java interface does not include instance variables or implementation code.

The Java API has two objects for every ArcObjects interface: a corresponding interface and an interface proxy class. The interface is named in the ArcObjects style, prefixed with an I. The interface proxy class appends the term proxy to the name. An example of this mapping is provided below:

```
interface IArea : IUnknown   public interface IArea{}
       public class IAreaProxy implements IArea{}
```

The proxy classes are used internally by the Java API to provide implementation to respective interfaces. An application developer should never use the default constructor of these classes as it holds no implementation. ArcObjects requires developers to go through an interface to access objects. The Java language does not use this model; subsequently, the Java API to ArcObjects has two ways of accessing objects—by interface or by class.

```
/* use the class implementing IPoint */
IPoint iPoint = new Point();

/* access object through class */
Point cPoint = new Point();
```

You cannot access objects through the default interface proxy class:

```
IPointProxy proxyPoint = new IPointProxy(); // incorrect usage
```

This will be discussed in more depth in subsequent sections.

ArcObjects interfaces are immutable and subsequently never versioned. An interface is never changed once it is defined and published. When an interface requires additional methods, the API defines a new interface by the same name with a version number appended to it as described in the following table.

```
interface IGeometry : IUnknown        public interface IGeometry{}
interface IGeometry2 : IGeometry      public interface IGeometry2 extends
                                                           IGeometry{}
interface IGeometry3 : IGeometry2     public interface IGeometry3 extends
                                                           IGeometry2{}
                                      public interface IGeometry4 extends
                                                           IGeometry3{}
```

Classes

In the ArcObjects model, classes provide the implementation of the defined interfaces. ArcObjects provides three types of classes: *abstract classes*, *classes*, and *coclasses*. These classes can be distinguished through the object model diagrams provided in ArcGIS Developer Help. It is important to be familiar with them before you begin to use the three class types.

In ArcObjects, an *abstract class* cannot be used to create new objects and are absent in the Java API. These classes are specifications in ArcObjects for instances of subclasses through type inheritance. An abstract class enumerates what interfaces are to be implemented by the implementing subclass but does not provide an implementation to those interfaces. For each abstract class in ArcObjects there are subclasses that provide the implementation.

A *class* cannot be publicly created in ArcObjects; however, objects of this class can be created as a property of another class or instantiated by objects from another class. In the Java API, the default constructor normally used to create a class is undefined for ArcObjects classes.

```
/* the constructor for FeatureClass() is unsupported*/
FeatureClass fc = new FeatureClass();
```

The following example illustrates this behavior while stepping you through the process of opening a feature class.

```
IWorkspaceFactory wf = new ShapefileWorkspaceFactory();
IFeatureWorkspace fw = new
IFeatureWorkspaceProxy(wf.openFromFile("\path\to\data", 0));
/* create a Feature Class from FeatureWorkspace */
IFeatureClass fc = fw.openFeatureClass("featureclass name");
```

In ArcObjects, a *CoClass* is a publicly creatable class. This means that you can create your own objects merely by declaring a new object as shown below.

```
/* create an Envelope from the Envelope CoClass */
Envelope env = new Envelope();
```

Structs

A structure defines a new data type made up of elements called members. Java does not have structures as complex data types. The Java language provides this functionality though classes; you can simply declare a class with the appropriate instance variables. For each structure in ArcObjects, there is a representative Java class with publicly declared instance variables matching the structure members as outlined below.

```
struct WKSPointZ             public class _WKSPointZ ... {
  double X         public double  x;
  double Y         public double  y;
  double Z         public double  z;
       }
```

You can work with these classes like any other class in Java:

```
_WKSPointZ pt = new _WKSPointZ();
pt.x = 2.23;
pt.y = -23.14;
pt.z = 4.85;

System.out.println(pt.x + " " + pt.y + " " + pt.z);
```

Enumerations

Java does not have enum types. To emulate enumerations in Java, a class or interface must be created that holds constants. For each enumeration in native ArcObjects, there is a Java interface with publicly declared static integers representing the enumeration value.

```
enum esri3DAxis                public interface esri3DAxis {
  esriXAxis = 0                public static final int esriXAxis = 0;
  esriYAxis = 1                public static final int esriYAxis = 1;
  esriZAxis = 2                public static final int esriZAxis = 2;
      }
```

You can now refer to the *esriXAxis* constant using the following notation:

```
esri3DAxis.esriXAxis;
```

Variants

The variant data type can contain a wide array of subtypes. With variants all types can be contained within a single type variant. Everything in the Java programming language is an object. Even primitive data types can be encapsulated inside objects if required. Every class in Java extends *java.lang.Object;* consequently, methods in ArcObjects that take variants as parameters can be passed any object type in the Java API.

Calling methods with "variant" objects as parameters

For methods that take variants as parameters, any object types can be passed, as all objects derive from *java.lang.Object*. As this is considered a "widening cast", an explicit cast to *Object* is not needed. If you want to pass primitives as parameters to methods, when variants are required, the corresponding primitive wrapper class can be used.

Using methods that return variants

When using variant objects returned by methods, explicitly "downcast" those objects to the corresponding wrapper object. For example, if expecting a *String*, downcast to *java.lang.String*; if expecting a *short*, downcast to short's wrapper class, that is, *java.lang.Short*, as shown in the code below.

```
ICursor spCursor = spTable.ITable_search(spQueryFilter, false);
/*Iterate over the rows*/
IRow spRow = spCursor.nextRow();
while (spRow != null) {
  Short ID = (Short) (spRow.getValue(1));
  String name = (String) (spRow.getValue(2));
  Short baseID =   (Short) (spRow.getValue(3));

    System.out.println("ID="+ ID +"\t name="+ name +"\tbaseID="+ baseID);
  /* Move to the next row.*/
  spRow = spCursor.nextRow();
}
```

IRowBuffer is a superinterface of *IRow* and defines the *getValue(int)* method as:

```
public Object getValue(int index)
              throws IOException,
                     AutomationException
The value of the field with the specified index.
Parameters:
index - The index (in)
Returns:
return value. A Variant
```

The return value is an *Object*, specified by the javadoc as "variant". Therefore, the value can be downcasted to *String* or *Short*, depending upon their type in the geodatabase being queried.

Casting

ArcObjects follows an interface-based programming style. Many methods use interface types as parameters and have interfaces as return value. When the return value of a method is an interface type, the method returns an object implementing that interface. When a method takes an interface type as parameter, it can take in any object implementing that interface. This style of programming has the advantage that the same method can work with many different object types, provided they all implement the said interface.

For example, *IFeature.getShape()* method returns an object implementing *IGeometry*. The object returned could potentially be any one of the following classes that implement *IGeometry*: BezierCurve, CircularArc, EllipticArc, Envelope, GeometryBag, Line, MultiPatch, Multipoint, Path, Point, Polygon, Polyline, Ray, Ring, Sphere, TriangleFan, Triangles, or TriangleStrip.

Casting is used to convert between types. There are three types of potential casts you, as a developer, may be tempted to use with the Java API:

1. Interface to concrete class casting
2. Interface cross-casting
3. Interface downcasting

It is important to understand that objects returned from methods within ArcObjects can behave differently than objects implicitly defined because the object reference is not held in the JVM.

If you have a method, *doSomeProcessingOnPolygon(Polygon p)*, that operates only on *Polygon* objects, and you want to pass the object obtained as a result of *IFeature.getShape()*, you need a way to convert the "type" of the object from *IGeometry* to *Polygon*. In Java, this is done using a class cast operation:

```
/* incorrect usage: will give ClassCastException */
Polygon poly = (Polygon)geom;
```

However, use the same code with the ArcObjects Java API, and you will get a *ClassCastException*. The reason for the exception is that the "geom" object reference is actually a reference to the native ArcObjects component. As a consequence of the interoperability between Java and the native ArcObjects components, the logic of casting this object reference to the *Polygon* object resides in the constructor of the *Polygon* object, and not in the JVM.

Every class in the Java API has a constructor that takes in a single object as parameter. This constructor can create the corresponding object using the reference to the ArcObjects component. Therefore, to achieve the equivalent of a class casting when using the Java API, use the "object constructor" of the class being casted to.

```
Polygon poly = new Polygon(geom);
```

The following code illustrates the object constructor being used to "cast" the geom object to a *Polygon*:

```
IFeature feature = featureClass.getFeature(i);
IGeometry geom = feature.getShape();
if (geom.getGeometryType() == esriGeometryType.esriGeometryPolygon){
  /*Note: "Polygon p = (Polygon) geom;" will give ClassCastException*/
   Polygon poly = new Polygon(geom);
   doSomeProcessingOnPolygon(poly);
}
```

The *Polygon* object thus constructed will implement all interfaces implemented by the *Polygon* class. Consequently, you can call methods belonging to any of the implemented interfaces on the *poly* object.

You could write all your code using the object constructors alone, but there are times when it might be better to cast an object implementing a particular interface, not to a class type, but to another interface implemented by that object.

Continuing the previous example, suppose you want to use the *doSomeProcessingOnPolygon(Polygon p)* method not only on *Polygon* objects but on other objects implementing *IArea*, such as *Envelope* and *Ring*. You could write a generic *doSomeProcessingOnArea(IArea area)* method that works on all objects implementing *IArea*. As *Polygon*, *Envelope*, and *Ring* objects all implement the *IArea* interface, you could pass in those objects to this generic method, thereby preventing the need to write additional methods for each object type, such as *doSomeProcessingOnEnvelope(Envelope env)* and *doSomeProcessingOnRing(Ring ring)*. To accomplish this, you would need to cast from the *IGeometry* type to the *IArea* type. In Java, this is typically done using interface cross-casting.

```
/* incorrect usage: will give ClassCastException */
IArea area = (IArea) geom ;
```

However, for the same reason noted in the class-cast above, such a cast would fail with a *ClassCastException*. To be able to cast to the ArcObjects interface, you will need to use the interface proxy classes discussed earlier in this section. In the Java API, you achieve the equivalent of an interface cross-casting by using the *InterfaceProxy* of the interface being casted to.

```
IArea area = new IAreaProxy(geom);
```

The following code shows the use of an *InterfaceProxy* class to cross-cast the geom object to *IArea*:

```
IFeature feature = featureClass.getFeature(i);
IGeometry geom = feature.getShape();
/*Note: "IArea area = (IArea) geom;" will give ClassCastException*/
IArea area = new IAreaProxy(geom);
doSomeProcessingOnArea(area);
```

Using the *IAreaProxy* class as shown in the code above allows you to access the object through its *IArea* interface so that it can then be passed to a method that takes an argument of type *IArea*. Thus, in this particular example, one method can deal with three different object types. However, only methods belonging to the *IArea* interface will be valid for the area object. To call other methods of the object, you will need to either class-cast to the appropriate object type using its object constructor or get a reference to the other interfaces using the *InterfaceProxy* classes.

Instanceof

The *instanceof* operator in Java allows a developer to determine if its first operand is an instance of its second.

```
operand1 instanceof operand2
```

You can use instanceof in ArcObjects—Java when the logic behind the type is held in Java. You cannot use instanceof when the type is held in ArcObjects as the logic of determining whether an object is an instance of a specified type resides in the constructors of that object type and not the JVM.

```
 Point point = new Point();
point.putCoords(0, 0);
point.putCoords(10, 10);
point.putCoords(20, 20);

if(point instanceof IGeometry){
   System.out.println("  point is a IGeometry");
    geom = point;
 }
if(point instanceof IClone){
   System.out.println("  point is a IClone");
 }
```

The above code works since the type information is held in Java for *Point*. When you construct a *Point* object, a proxy class for each implementing interface is also constructed. This allows you to use instanceof on any of these types. Developers would have access to any methods on *Poin*t implementing the *IGeometry* or *IClone* interfaces.

This is backwards compatible as well:

```
if(geom instanceof Polyline){
   System.out.println("  geom is a Polyline");
 }
else if(geom instanceof Point){
   System.out.println("  geom is a Point");
   pnt = (IPoint)geom;  // allowable cast as the type is held in JVM
 }
```

Since a direct cast of the *geom* object into *Point* was created, the *geom* object is of type *Point* and instanceof can be used to check this information. However, since the type information was known before it was checked above, it is not extremely useful. What would be useful is to apply the above logic on methods that return objects of super interfaces.

Consider the *IWorkspaceFactory.openFromFile()* method which returns an *IWorkspace*. Since the object returned is a Java object which implements *IWorkspace*, you cannot check if the returned object is of any of the known implementing classes that implement *IWorkspace*. In this case, to check for type information, you should call a method on the returned object which is expected. If the method does not throw an exception, it is of that type. This occurs because the logic on this object is declared at runtime and is held inside the underlying ArcObjects component.

```
RasterWorkspaceFactory rasterWkspFactory = new RasterWorkspaceFactory();
IWorkspace wksp = rasterWkspFactory.openFromFile( aPath, 0 );

if(wksp instanceof RasterWorkspace){
   /*code does not execute as logic is in ArcObjects*/
   System.out.println("  wksp is a RasterWorkspace");
   rasWksp = (RasterWorkspace)wksp;
 }
else{
    try{
      rasWksp = (RasterWorkspace)wksp;
      rasWksp.openRasterDataset( aRaster );

   }catch(Exception e){
      /*code executes if wksp is not a RasterWorkspace*/
      System.out.println("  wksp is not a RasterWorkspace");
     }
 }
```

Methods that take out parameters

ArcObjects provides many methods that return more than one value. The Java API requires sending single element arrays as paramters to such methods. Basically, you pass in single element arrays of the object that you want to be returned, and ArcObjects fills in the first elements of those arrays with the return value. Upon returning from the method call, the first element of the array contains the value that has been set during the method call. One such method, that you will be using in this section is the *toMapPoint* of *IARMap* interface. Take a look at the javadoc of this method:

```
public void toMapPoint(int x,
                         int y,
                       double[] xCoord,
                       double[] yCoord)
      throws IOException,
                       AutomationException
```

```
Converts a point in device coordinates (typically pixels) to coordinates in
map units.
Converts the x and y screen coordinates supplied in pixels to x and y map
coordinates. The returned map coordinates will be in MapUnits.
```

Parameters:

x - The x (in)

```
y - The y (in)
xCoord - The xCoord (in/out: use single element array)
yCoord - The yCoord (in/out: use single element array)
```

Notice that the parameters *xCoord* and *yCoord* are marked as “in/out: use single element array”. To use this method, the first two parameters are the *x* and *y* coordinates in pixel units. The next two parameters are actually used to get return values from the method call. You pass in single dimensional single element double arrays:

```
   double [] dXcoord = {0.0};
   double [] dYcoord = {0.0};
```

When the method call completes, you can query the values of *dXcoord[0]* and *dYcoord[0]*. These values will be modified by the method and will actually refer to the x and y coordinates in map units. A practical example of this method call is to update the status bar with the current map coordinates as the mouse moves over the control.

```
public void updateStatusBar(IARControlEventsOnMouseMoveEvent params)
   throws IOException {
   /*create single dimensional array of doubles
    *the values of the first element of the arrays will be filled in
    *by the arMap.toMapPoint(...) method
    */
   double [] dXcoord = {0.0};
   double [] dYcoord = {0.0};
   int screenX = params.esri_getX();
   int screenY = params.esri_getY();
   IARMap arMap = arControl.getARPageLayout().getFocusARMap();
   arMap.toMapPoint(screenX, screenY, dXcoord, dYcoord);
   /*set the statusLabel*/
   statusLabel.setText(“Map x,y: “ + dXcoord[0]+”, “+dYcoord[0]);
}
```

The Java API will not allow developers to populate an array with a superclass type, even when it has been cast to a superclass type. Consider the following Java example:

```
Integer[] integers = { new Integer(0), new Integer(1), new Integer(2)};
Object[] integersAsObjects = (Object[])integers;
integersAsObjects[0] = new Object();
```

The above is not allowed and will cause an *ArrayStoreException*. Consider the following ArcObjects example:

```
Polyline[] polyline = {new Polyline()};
tin.interpolateShape( breakline, polyline, null );
Polyline firstPolyLine = polyline[0];
```

The above is not allowed and will cause the same *ArrayStoreException* as the earlier example. Take a look at the *interpolateShape()* method of *ISurface* and analyze what is going on here.

```
public void interpolateShape(IGeometry pShape,
        IGeometry[] ppOutShape,
    Object pStepSize)
  throws IOException,
           AutomationException
```

```
Parameters:
   pShape - A reference to a com.esri.arcgis.geometry.IGeometry (in)
   ppOutShape - A reference to a com.esri.arcgis.geometry.IGeometry
                                     (out: use single element array)
   pStepSize - A Variant (in, optional, pass null if not required)
Throws:
   IOException - If there are communications problems.
   AutomationException - If the remote server throws an exception.
```

IGeometry is a super-interface to *IPolyline*, and the *Polyline* class implements both interfaces. In the first attempt youe tried to send a single element *Polyline* array into a method which requires an in/out *IGeometry* parameter. This causes an *ArrayStoreException* as ArcObjects is attempting to populate an *IPolyline* array with an *IGeometry* object, attempting to place a super-class type into a subclass array. The correct way to use this method is outlined below:

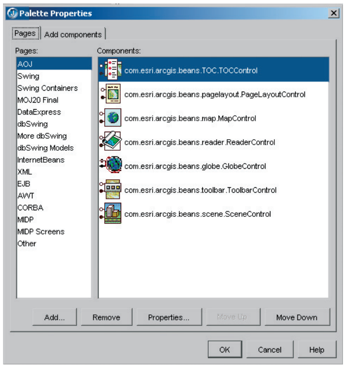
```
/*Set up the array and call the method*/
IGeometry[] geoArray = {new Polyline()};
tin.interpolateShape( breakline, geoArray, null );
/* "Cast" the first array element as a Polyline - this is
 * the equivalent of calling QueryInterface on IGeometry
 */
IPolyline firstPolyLine = new IPolylineProxy(geoArray[0]);
```

Non-OLE Automation Compliant Types

ArcObjects types that are not OLE automation compliant types do not work within the Java API. ArcObjects contains a few of these methods and unfortunately such methods are not usable in the Java API. However, in most cases, supplemental interfaces have been added which have the offending methods overwritten as automation compatible. These new interfaces are appended with the letters “GEN”, implying that they are generic for all supported APIs.

```
IPoint[] points = new Point[2];
points[0] = new Point();
points[1] = new Point();

points[0].putCoords(0, 0);
points[1].putCoords(10, 10);

IEnvelope env = new Envelope();
IEnvelopeGEN envGEN = new Envelope();
/*not automation compatible - throws exception*/
env.defineFromPoints(2, points[0]);
/*automation compatible*/
envGEN.defineFromPoints(points);
```

Using visual beans



Pages:

AOJ
Swing
Swing Containers
MOJ20 Final
DataExpress
dbSwing
More dbSwing
dbSwing Models
InternetBeans
XML
EJB
AWT
CORBA
MIDP
MIDP Screens
Other

Components:

com.esri.arcgis.beans.TOC.TOCControl
com.esri.arcgis.beans.pagelayout.PageLayoutControl
com.esri.arcgis.beans.map.MapControl
com.esri.arcgis.beans.reader.ReaderControl
com.esri.arcgis.beans.globe.GlobeControl
com.esri.arcgis.beans.toolbar.ToolbarControl
com.esri.arcgis.beans.scene.SceneControl

The Java API provides a set of reusable components as prebuilt pieces of software code designed to provide graphical functions. As a visual beans developer, you only need to write code to “buddy” them into your ArcGIS Engine application. The use of beans creates a bridge between Java and the ActiveX controls provided by ArcGIS. These visual components are heavyweight AWT components and conform to the JavaBeans™ component architecture, allowing them to be used as drag and drop components for designing Java GUIs in JavaBean-compatible IDEs.

Mixing heavyweight and lightweight components

One of the primary goals of the Swing architecture was that it be based on the existing AWT architecture. This allows developers to mix both kinds of components in the same application. When using the ArcObjects JavaBeans with Swing components, care should be taken while mixing the heavyweight and lightweight components. For guidelines, refer to the article ‘Mixing heavy and light components’ at *http://java.sun.com/products/jfc/tsc/articles/mixing/*.

If using Swing components, disable lightweight popups where the option is available, using code similar to:

```
jComboBox.setLightWeightPopupEnabled(false);
jPopupMenu.setLightWeightPopupEnabled(false);
```

Listening to events

All ArcObjects JavaBean are capable of firing events. For instance, the *ARControl* bean fires the following events:

```
void onAction(IARControlEventsOnActionEvent theEvent)
void onAfterScreenDraw(IARControlEventsOnAfterScreenDrawEvent theEvent)
void onBeforeScreenDraw(IARControlEventsOnBeforeScreenDrawEvent theEvent)
void onCurrentViewChanged(IARControlEventsOnCurrentViewChangedEvent
theEvent)
void onDocumentLoaded(IARControlEventsOnDocumentLoadedEvent theEvent)
void onDocumentUnloaded(IARControlEventsOnDocumentUnloadedEvent
theEvent)
void onDoubleClick(IARControlEventsOnDoubleClickEvent theEvent)
void onFocusARMapChanged(IARControlEventsOnFocusARMapChangedEvent
theEvent)
void onKeyDown(IARControlEventsOnKeyDownEvent theEvent)
void onKeyUp(IARControlEventsOnKeyUpEvent theEvent)
void onMouseDown(IARControlEventsOnMouseDownEvent theEvent)
void onMouseMove(IARControlEventsOnMouseMoveEvent theEvent)
void onMouseUp(IARControlEventsOnMouseUpEvent theEvent)
```

To add and remove listeners for the events, the beans have methods of the form *addXYZEventListener* and *removeXYZEventListener*. Adapter classes are provided as a convenience for creating listener objects.

```
  public void addIARControlEventsListener(IARControlEvents theListener)
      throws IOException

  public void removeIARControlEventsListener(IARControlEvents
        theListener)
      throws IOException
```

The following code shows using an anonymous inner class with the *IARControlEventsAdapter* to add event listeners for *onDocumentLoaded* and *onDocumentUnloaded* events to the *arControl* object:

```
arControl = new ARControl();
...

/*wire up the events for arControl*/
arControl.addIARControlEventsListener(new IARControlEventsAdapter(){
  public void onDocumentLoaded(IARControlEventsOnDocumentLoadedEvent evt)
    throws IOException{
        /*set the statusbar text to point to the currently loaded docu-
ment*/
        statusLabel.setText(" Document filename: "+
arControl.getDocumentFilename());
        /*Determine whether permission to toggle TOC visibility*/
        if (arControl.hasDocumentPermission(
esriARDocumentPermissions.esriARDocumentPermissionsViewTOC))
          {
           tocVisibilityCheckBox.setEnabled(true);
           tocVisibilityCheckBox.setSelected(arControl.isTOCVisible());
          } else {
           JOptionPane.showMessageDialog((Component)arg0.getSource(),
"You do not have permission toggle TOC visibility");
          }
  }

  public void onDocumentUnloaded(IARControlEventsOnDocumentUnloadedEvent
evt)
    throws IOException{
        /*set the statusbar text to empty string*/
        statusLabel.setText("");
  }
});
```

It is worthwhile to note that the events fired by the beans are custom events for which the listeners are provided as part of the Java API. Adding listeners from the *java.awt.event* package (such *MouseListener*) to the beans will not be helpful as the JavaBeans do not fire those events. Instead, you could use similar events, such as *onMouseDown*, *onMouseUp*, and *onMouseMove*, provided by the corresponding event listener, which in the case of *ARControl* is *IARControlEvents*.

C++ is an object-oriented programming language which evolved in the mid 1980s from its predecessor, C. C++ is endowed with many features which give the language an unrivaled expressive power, such as object orientation with inheritance, operator overloading, virtual functions, templates, and a library of useful and often necessary functions called the Standard Template Library (STL). The C++ language has been standardized by the International Organization for Standardization (ISO) and several influential national standards organizations.

Developers may consider using the ArcGIS C++ API, as opposed to one of the other APIs, for the following reasons:

- Execution speed—C++ code typically executes faster than the equivalent Java, Visual Basic, and C# code.
- Cross-platform compatibility—Visual Basic and C# are currently used primarily on the Windows platform. C++ and Java are inherently more cross-platform.
- Prior familiarity—If the developers in your organization already have a good deal of experience using the language, then C++ is a logical choice.

This section is intended to serve two main purposes:

1. To familiarize you with general C++ coding style and debugging.
2. To provide an introduction to the ArcGIS C++ API, detailing specific usage requirements and recommendations for working with the ArcObjects programming platform.

C++ DEVELOPMENT TECHNIQUES

Naming conventions

Type names

All type names (*class, struct, enum,* and *typedef*) begin with an uppercase letter and use mixed case for the rest of the name:

```
class Foo : public CObject { . . .};
struct Bar { . . .};
enum ShapeType { . . . };
typedef int* FooInt;
```

Typedefs for function pointers (callbacks) append Proc to the end of their names.

```
typedef void (*FooProgressProc)(int step);
```

Enumeration values all begin with a lowercase string that identifies the project; in the case of ArcObjects this is esri, and each string occurs on separate lines:

```
typedef enum esriQuuxness
 {
  esriQLow,
  esriQMedium,
  esriQHigh
} esriQuuxness;
```

Here are some suggestions for a naming convention.These help identify the variables usage and type and so reduce coding errors.This is an abridged Hungarian notation:

[<scope>_]<type><name>

Prefix	Variable scope
m	Instance class members
c	Static class member (including constants)
g	Globally static variable
<empty>	local variable or struct or public class member

<type>

Prefix	Data Type
b	Boolean
by	byte or unsigned char
cx / cy	short used as size
d	double
dw	DWORD, double word or unsigned long
f	float
fn	function
h	handle
i	int (integer)
ip	smart pointer
l	long
p	a pointer
s	string
sz	ASCIIZ null-terminated string
w	WORD unsigned int
x, y	short used as coordinates

<name> describes how the variable is used or what it contains.The <scope> and <type> portions should always be lowercase, and the <name> should use mixed case:

Variable Name	Description
m_hWnd	a handle to a HWND
ipEnvelope	a smart pointer to a COM interface
m_pUnkOuter	a pointer to an object
c_isLoaded	a static class member
g_pWindowList	a global pointer to an object

Function names

Name functions using the following conventions:

For simple accessor and mutator functions, use Get<Property> and Set<Property>:

```
int GetSize();
void SetSize(int size);
```

If the client is providing storage for the result, use Query<Property>:

```
void QuerySize(int& size);
```

For state functions, use Set<State and Is<State> or Can<State>:

```
bool IsFileDirty();
void SetFileDirty(bool dirty);
bool CanConnect();
```

Where the semantics of an operation are obvious from the types of arguments, leave type names out of the function names.

Instead of:

```
AddDatabase(Database& db);
```

consider using:

```
Add(Database& db);
```

Instead of:

```
ConvertFoo2Bar(Foo* foo, Bar* bar);
```

consider using:

```
Convert(Foo* foo, Bar* bar)
```

If a client relinquishes ownership of some data to an object, use Give<Property>. If an object relinquishes ownership of some data to a client, use Take<Property>:

```
void GiveGraphic(Graphic* graphic);
Graphic* TakeGraphic(int itemNum);
```

Use function overloading when a particular operation works with different argument types:

```
void Append(const CString& text);
void Append(int number);
```

Argument names

Use descriptive argument names in function declarations. The argument name should clearly indicate what purpose the argument serves:

```
bool Send(int messageID, const char* address, const char* message);
```

Smart types

Smart types are objects that behave like types. They are C++ class implementations that encapsulate a data type, wrapping it with operators and functions that make working with the underlying type easier and less error prone. When these smart types encapsulate an interface pointer, they are referred to as *smart pointers*. Smart pointers work with the *IUnknown* interface to ensure that resource allocation and deallocation is correctly managed. They accomplish this by various functions, construct and destruct methods, and overloaded operators.

Smart types can make the task of working with COM interfaces and data types easier, since many of the API calls are moved into a class implementation; however they must be used with caution, and never without a clear understanding of how they are interacting with the encapsulated data type.

The smart types supplied with the C++ API consist of:

- *_com_ptr_t* - this class encapsulates a COM interface pointer, creating a smart pointer.
- *CComBSTR* - class encapsulates the *BSTR* data type.
- *CComVariant* - class encapsulates the *VARIANT* data type

To define a smart pointer for an interface you can use the macro *_COM_SMARTPTR_TYPEDEF* like this:

```
_COM_SMARTPTR_TYPEDEF(IFoo, __uuidof(IFoo));
```

The compiler expands this as follows:

```
typedef _com_ptr_t< _com_IIID<IFoo, __uuidof(IFoo)> > IFooPtr;
```

Once declared, it is simply a matter of declaring a variable as the type of the interface and appending *Ptr* to the end of the interface. Below are some common uses of this smart pointer that you will see in the numerous C++ samples.

```
// Get a CLSID GUID constant
extern “C” const GUID __declspec(selectany) CLSID_Foo = \
  {0x2f3b470c,0xb01f,0x11d3,{0x83,0x8e,0x00,0x00,0x00,0x00,0x00,0x00}};

// Declare Smart Pointers for IFoo, IBar and IGak interfaces
_COM_SMARTPTR_TYPEDEF(IFoo, __uuidof(IFoo));
_COM_SMARTPTR_TYPEDEF(IBar, __uuidof(IBar));
_COM_SMARTPTR_TYPEDEF(IGak, __uuidof(IGak));

HRESULT SomeClass::Do()
{
  // Create Instance of Foo class and QueryInterface (QI) for IFoo inter-
face
  IFooPtr ipFoo;
  HRESULT hr = ipFoo.CreateInstance(CLSID_Foo);
  if (FAILED(hr)) return hr;

  // Call method on IFoo to get IBar
  IBarPtr ipBar;
  hr = ipFoo->get_Bar(&ipBar);
  if (FAILED(hr)) return hr;

  // QI IBar interface for IGak interface
  IGakPtr ipGak(ipBar);

  // Call method on IGak
  hr = ipGak->DoSomething();
  if (FAILED(hr)) return hr;
```

```
    // Explicitly call Release()
    ipGak = 0;
    ipBar = 0;

    // Let destructor call IFoo's Release
    return S_OK;
  }
```

If you have used smart pointers before, you might have seen differences in the implementation of the equality ("== ") operator for smart pointer comparisons.The COM specification states object indentity is performed by comparing the pointer values of IUnknown. Our smart pointers will perform necessary QI and comparison when using the "==" operator.*

When working with *CComBSTR*, use the text mapping L"" to declare constant *OLECHAR* strings. To display a *CComBSTR* at the command line, use wcerr. You will need to include iostream to use wcerr.

```
CComBSTR bsName(L"Matt");
std::wcerr << L"The name is " << (BSTR) bsName << std::endl;
```

CComVariant derives directly from the VARIANT data type, meaning that there is no overloading with its implementation, which in turn simplifies it use. It has a rich set of constructors and functions that make working with *VARIANTs* straightforward; there are even methods for reading and writing from streams. Be sure to call the *Clear* method before reusing the variable.

CComVariant(VARIANT_TRUE) will create a short integer variant (type VT_I2) and not a boolean variant (type VT_BOOL) as expected. You can use CComVariant(true) to create a boolean variant.

```
ipFoo->put_Name(CComBSTR(L"NewName"));
if FAILED(hr)) return hr;

// Create a VT_I4 variant (signed long)
CComVariant vValue(12);

// Change its data type to a string
hr = vValue.ChangeType(VT_BSTR);
if (FAILED(hr)) return hr;
```

Some method calls in idl are marked as being optional and take a variant parameter. However in VC++ these parameters still have to be supplied. To signify that a parameter value is not supplied a variant is passed specifying an error code or type DISP_E_PARAMNOTFOUND:

```
CComBSTR documentFilename(L"World.mxd");

CComVariant noPassword;
noPassword.vt = VT_ERROR;
noPassword.scode = DISP_E_PARAMNOTFOUND;
HRESULT hr = ipMapControl->LoadMxFile(documentFilename, noPassword);
```

CComVariant myVar(ipSmartPointer) will result in a variant type of boolean (VT_BOOL) and not a variant with an object reference (VT_UNKNOWN) as expected. It is better to pass unambiguous types to constructors, i.e. types which are not themselves smart types with overloaded cast operators.

```
// Perform QI it IUnknown
IUnknownPtr ipUnk = ipSmartPointer;
// Ensure we use IUnknown* constructor of CComVariant
CComVariant myVar2(ipUnk.GetInterfacePtr());
```

However, if you do have a value that you wish to pass in for the variant, use the smart type, CComVariant.

```
int val = 1;
CComVariant smartVal(val);
ipRowBuffer->put_Value(2, smartVal);
```

When working with *CComBSTR* and *CComVariant*, the *Detach()* function releases the underlying data type from the smart type and can be used when passing a result as an [out] parameter of a method. The use of the Detach method with *CComBSTR* is shown below:

```
HRESULT CFoo::get_Name(BSTR* name)
{
  if (name==0) return E_POINTER;
  CComBSTR bsName(L"FooBar");
  *name = bsName.Detach();
}
```

A common practice with smart pointers is to use *Detach()* to return an object from a method call. When returning an interface pointer the COM standard is to increment reference count of the [out] parameter inside the method implementation. It is the callers responsibility to call Release when the pointer is no longer required. Consequently care must be taken to avoid calling *Detach()* directly on a member variable, a typical pattern is show below:

```
HRESULT CFoo::get_Bar(IBar **pVal)
{
  if (pVal==0) return E_POINTER;

  // Constructing a local smart pointer using another smart pointer
  // results in an AddRef (if pointer is not 0).
  IBarPtr ipBar(m_ipBar);

  // Detach will clear the local smart pointer and the
  // interface is written into the output parameter.
  *pVal = ipBar.Detach();

  // This can be combined into one line
  // *pVal = IBarPtr(m_ipBar).Detach();

  return S_OK;
}
```

The above pattern has the same result as the following code, note that a conditional test for a 0 pointer is required before AddRef can be called, calling AddRef (or any method) on a 0 pointer will result in an access violation exception and typically crash the application:

```
HRESULT CFoo::get_Bar(IBar **pVal)
{
  if (pVal==0) return E_POINTER;

  // copy the interface pointer (no AddRef) into the output parameter
  *pVal = m_ipBar;

  // Make sure interface pointer is non 0 before calling AddRef
  if (*pVal)
    *pVal->AddRef();

  return S_OK;
}
```

When using a smart pointer to receive an object from from an [out] parameter on a method, use the smart pointer “&” de-reference operator. This will cause the previous interface pointer in the smart pointer to be released. The smart pointer is then populated with the new [out] value. The implementation of the method will have already incremented the object reference count. This will be released when the smart pointer goes out of scope:

```
 {
  IFooPtr ipFoo1, ipFoo2;
  ipFoo1.CreateInstance(CLSID_Foo);
  ipFoo2.CreateInstance(CLSID_Foo);

  // Initalise ipBar Smart pointer from Foo1
  IBarPtr ipBar;
  ipFoo1->get_Bar(&ipBar);

  // The “&” de-reference will call Release on ipBar
  // ipBar is then repopulate with a new instance of IBar
  ipFoo2->get_Bar(&ipBar);
 }
// ipBar goes out of scope and the smart pointer destructor calls Release
```

Debugger

Visual C++ comes with a feature-rich debugger. These tips will help you get the most from your debugging session.

Backing up after failure

When a function call has failed and you’d like to know why (by stepping into it), you don’t have to restart the application. Use the Set Next Statement command to reposition the program cursor back to the statement that failed (right-click on the statement to bring up the debugging context menu). Then, just step into the function.

Edit and Continue

Visual Studio 6 allows changes to source code to be made during a debugging session. The changes can be recompiled and incorporated into the executing code without stopping the debugger. There are some limitations to the type of changes that can be made, in this case the debug session must be restarted. This feature is enabled by default, the settings are available in “Settings” of the project menu then Select “C/C++ tab. Select “General” from the “Category” group box. In the Debug info group box, select “Program Database for Edit and Continue.”

Unicode string display

Set your debugger options to display Unicode strings (click the Tools menu, click Options, click Debug, then check the Display Unicode Strings check box).

Variable value display

Pause the cursor over a variable name in the source code to see its current value. If it is a structure, click it and bring up the QuickWatch dialog box (the Eyeglasses icon or Shift+F9) or drag and drop it into the Watch window.

Undocking windows

If the Output window (or any docked window, for that matter) seems too small to you, try undocking it to make it a real window. Just right-click it and toggle the Docking View item.

Conditional break points

Use conditional break points when you need to stop at a break point only once some condition is reached (a for-loop reaching a particular counter value). To do so, set the break point normally, then bring up the Breakpoints window (Ctrl+B or Alt+F9). Select the specific break point you just set and then click the Condition button to display a dialog in which you specify the break point condition.

Preloading DLLs

You can preload DLLs that you wish to debug before executing the program. This allows you to set break points up front rather than wait until the DLL has been loaded during program execution. (Click Project, click Settings, click Debug, click Category, then click Additional DLLs.) Then, click in the list area below to add any DLLs you wish to have preloaded.

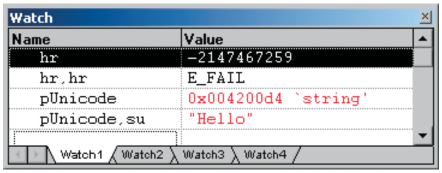
Changing display formats

You can change the display format of variables in the QuickWatch dialog box or in the Watch window using the formatting symbols in the following table.

Symbol	Format	Value	Displays
d, i	signed decimal integer	0xF000F065	-268373915
u	unsigned decimal integer	0x0065	101
o	unsigned octal integer	0xF065	0170145
x, X	hexadecimal integer	61541	0x0000F065
l, h	long or short prefix for d, I, u, o, x, X	00406042, hx	0x0C22
f	signed floating-point	3./2.	1.500000
e	signed scientific notation	3./2.	1.500000e+00
g	e or f, whichever is shorter	3./2.	1.5
c	single character	0x0065	'e'
s	string	0x0012FDE8	"Hello"
su	Unicode string		"Hello"
hr	string	0	S_OK

To use a formatting symbol, type the variable name followed by a comma and the appropriate symbol. For example, if var has a value of 0x0065, and you want to see the value in character form, type var,c in the Name column on the tab of the

Watch window. When you press ENTER, the character-format value appears: var,c = ‘e’. Likewise, assuming that *hr* is a variable holding *HRESULTS*, view a human-readable form of the *HRESULT* by typing “hr,hr” in the Name column.



Watch

Name	Value
hr	-2147467259
hr,hr	E_FAIL
pUnicode	0x004200d4 `string'
pUnicode,su	"Hello"

You can use the formatting symbols shown in the following table to format the contents of memory locations.

You can apply formatting symbols to structures, arrays, pointers, and objects as unexpanded variables only. If you expand the variable, the specified formatting affects all members.You cannot apply formatting symbols to individual members.

Symbol	Format	Value
ma	64 ASCII characters	0x0012ffac .4...0...".0W&.. ......1W&.0.:W..1 ....."..1.JO&.1.2 .."..1...0y....1
m	16 bytes in hex, followed by 16 ASCII characters	0x0012ffac B3 34 CB 00 84 30 94 80 FF 22 8A 30 57 26 00 00 .4...0....".0W&..
mb	16 bytes in hex, followed by 16 ASCII characters	0x0012ffac B3 34 CB 00 84 30 94 80 FF 22 8A 30 57 26 00 00 .4...0...".0W&..
mw	8 words	0x0012ffac 34B3 00CB 3084 8094 22FF 308A 2657 0000
md	4 double-words	0x0012ffac 00CB34B3 80943084 308A22FF 00002657
mu	2-byte characters (Unicode)	0x0012fc60 8478 77f4 ffff ffff 0000 0000 0000 0000

With the memory location formatting symbols, you can type any value or expression that evaluates to a location. To display the value of a character array as a string, precede the array name with an ampersand, *&yourname*. A formatting character can also follow an expression:

- *rep+1,x*
- *alps[0],mb*
- *xloc,g*
- *count,d*

To watch the value at an address or the value pointed to by a register, use the *BY, WO,* or *DW* operator:

- *BY* returns the contents of the byte pointed at.
- *WO* returns the contents of the word pointed at.
- *DW* returns the contents of the doubleword pointed at.

Follow the operator with a variable, register, or constant. If the *BY, WO,* or *DW* operator is followed by a variable, then the environment watches the byte, word, or doubleword at the address contained in the variable.

You can also use the context operator { } to display the contents of any location.

To display a Unicode string in the Watch window or the QuickWatch dialog box, use the su format specifier. To display data bytes with Unicode characters in the Watch window or the QuickWatch dialog box, use the mu format specifier.

Keyboard shortcuts

There are numerous keyboard shortcuts that make working with the Visual Studio editor faster. Some of the more useful keyboard shortcuts follow.

The text editor uses many of the standard shortcut keys used by Windows applications, such as Word. Some specific source code editing shortcuts are listed below.

Shortcut	Action
Alt+F8	Correctly indent selected code based on surrounding lines.
Ctrl+]	Find the matching brace.
Ctrl+J	Display list of members.
Ctrl+Spacebar	Complete the word, once the number of letters entered allows the editor to recognize it. Use full when completing function and variable names.
Tab	Indents selection one tab stop to the right.
Shift+Tab	Indents selection one tab to the left.

Below is a table of common keyboard shortcuts used in the debugger.

Shortcut	Action
F9	Add or remove breakpoint from current line.
Ctrl+Shift+F9	Remove all breakpoints.
Ctrl+F9	Disable breakpoints.
Ctrl+Alt+A	Display auto window and move cursor into it.
Ctrl+Alt+C	Display call stack window and move cursor into it.
Ctrl+Alt+L	Display locals window and move cursor into it.
Ctrl+Alt+A	Display auto window and move cursor into it.
Shift+F5	End debugging session.
F11	Execute code one statement at a time, stepping into functions.
F10	Execute code one statement at a time, stepping over functions.
Ctrl+Shift+F5	Restart a debugging session.
Ctrl+F10	Resume execution from current statement to selected statement.
F5	Run the application.
Ctrl+F5	Run the application without the debugger.
Ctrl+Shift+F10	Set the next statement.
Ctrl+Break	Stop execution.

Loading the following shortcuts can greatly increase your productivity with the Visual Studio development environment.

Shortcut	Action
ESC	Close a menu or dialog box, cancel an operation in progress, or place focus in the current document window.
CTRL+SHIFT+N	Create a new file.
CTRL+N	Create a new project.
CTRL+F6 or CTRL+TAB	Cycle through the MDI child windows one window at a time.
CTRL+ALT+A	Display the auto window and move the cursor into it.
CTRL+ALT+C	Display the call stack window and move the cursor into it.
CTRL+ALT+T	Display the document outline window and move the cursor into it.
CTRL+H	Display the find window.
CTRL+F	Display the find window. If there is no current Find criteria, put the word under your cursor in the find box.
CTRL+ALT+I	Display the immediate window and move the cursor into it. Not available if you are in the text editor window.
CTRL+ALT+L	Display the locals window and move the cursor into it.
CTRL+ALT+O	Display the output window and move the cursor into it
CTRL+ALT+J	Display the project explorer window and move the cursor into it.
CTRL+ALT+P	Display the properties window and move the cursor into it.
CTRL+SHIFT+O	Open a file.
CTRL+O	Open a project.
CTRL+P	Print all or part of the document.
CTRL+SHIFT+S	Save all of the files, projects, or documents.
CTRL+S	Select all.
CTRL+A	Save the current document or selected item or items.

Navigating through online Help topics

Right-click a blank area of a toolbar to display a list of all the available toolbars. The Infoviewer toolbar contains up and down arrows that allow you to cycle through help topics in the order in which they appear in the table of contents. The left and right arrows cycle through help topics in the order that you visited them.

ARCGIS DEVELOPMENT USING C++

The upcoming sections are intended for developers using C++ to develop with ArcGIS Engine. The ArcGIS Engine Developer Kit gives developers access to ArcObjects and is designed to satisfy requirements for C++ development projects that require ArcObjects without the ArcGIS Desktop applications.

Building an understanding of COM and ArcObjects

The C++ API for ArcGIS gives the C++ developer access to the ArcObjects components used to build the ArcGIS family of products; these components are themselves written in C++ using the COM framework. While it is not necessary to write any COM code in order to use the C++ API, a basic understanding of how COM objects work is necessary in order to use ArcObjects. If you are unfamiliar with the ArcObjects framework, the initial two sections of this chapter, ‘The Microsoft Component object model’ and ‘Developing with ArcObjects’, are recommended reading.

Getting help

In addition to the Start Menu -> Programs shortcut, the C++ version of ArcGIS Developer Help can also be accessed from your ArcGIS Engine installation directory. Open it by navigating to <ArcGIS install directory>\DeveloperKit\Help\COM and double-clicking on ArcGISDevHelpVC.chm.

For help on the objects used in this API (interfaces, classes, and so on), refer to the ArcGIS Developer Help for C++ that installed with ArcGIS Engine. If you accepted the default installation options, it can be accessed from Start Menu -> Programs -> ArcGIS -> Developer Help -> C++ Help.

Supported compilers

The compilers in Visual Studio 6.0 and Visual Studio .NET 2003 are supported by the ArcGIS C++ API.

Development environments

As an ArcGIS C++ API developer, you can choose any development environment. However, if you wish to use an integrated development environment (IDE), either Visual Studio 6.0 or Visual Studio .NET 2003 (7.1) is recommended. If you don't wish to use an IDE, you can write your code in any text editor and then compile it from the command line. If you choose this option, we recommend using the nmake utility from a Windows command prompt. Your choice of development environment depends entirely on your personal preference and the tools available. The following sections focus on each of these three options.

You are also able to mix and match by coding in one development environment and compile in another. For example, you can write code in Visual Studio but compile and build it via a script utilizing the command line tools.

ARCGIS DEVELOPMENT USING C++ IN VISUAL STUDIO 6.0

Set up your application

To begin creating your ArcGIS Engine application in Visual Studio 6.0, start Microsoft Visual C++ and use the Win32 Console Application wizard to create an empty project. Choose File > New, select Win32 Console Application, and type in the name of the project and choose its location.

Next, set the required project options. On the C/C++ tab of the Project Menu -> Settings dialog, select 'Preprocessor' from the Category combobox. In the Additional Include Directories text box, type the path to ArcSDK.h. In addition, type in the location of the ArcGIS Engine olb files. Don't forget to separate the two paths with a semi-colon.

The ArcSDK.h and ArcGIS Engine olb files are installed, by default, in the <ArcGIS install directory>\include\CPPAPI and <ArcGIS install directory>\com folders respectively.

Next, go to the Preprocessor Definitions text box and type in 'ESRI_WINDOWS' to define the ESRI_WINDOWS symbol. Similar to the include directories, symbols need to be separated with commas.

It is recommended that you use the /GX and /NOLOGO compiler flags. /GX enables synchronous exception handling and /NOLOGO prevents display of a compiler startup banner and informational compiler messages. Activate /GX by opening the project Property Pages, navigating to C/C++ -> C++ Language, and checking 'Enable exception handling'. Set /NOLOGO by checking the 'Suppress Startup Banner and Information Messages' option from the menu bar, Project -> Settings -> C/C++ tab -> Customize category.

Finally, go to Project > Add to Project > New in order to add some files to the project; these will eventually contain your code. Select the type of file you wish to add and give it a name. Add as many files as you need for your application. You are ready to write your code. Don't forget to start by including ArcSDK.h!

Compile your application

To compile an ArcGIS Engine application in Visual Studio 6.0, type F7 or choose Build -> Build YourApplicationName.exe.

Run your application

Before you can run an ArcGIS Engine command line application from within Visual Studio 6.0, you need to set up the arguments. Arguments are added to your program by customizing your project settings;: go to the Project menu -> Settings -> Debug tab and add any arguments to the Program arguments text box.

Once the arguments are added, run the application by choosing Build -> Execute YourApplicationName.exe or by typing CONTROL-F5. To run the application in debug mode, choose Build -> Start Debug -> Go, or type F5.

ARCGIS DEVELOPMENT USING C++ IN VISUAL STUDIO .NET

Set up your application

The easiest way to program with the C++ API in .NET is to use the C/C++ Console Application wizard. This wizard is not the same as the Win32 Console Project and Console Application (.NET) wizards. To access the C/C++ Console Application wizard, you must install Academic Tools for VS .NET 2003 from http://www.msdnaa.net/Resources/display.aspx?ResID=1911. Once you have installed the wizard, choose New > Project, click the Visual C++ Projects folder, and select C/C++ Console Application.

The ArcSDK.h and ArcGIS Engine olb files are installed, by default, in the <ArcGIS install directory>\include\CPPAPI and <ArcGIS install directory>\com folders respectively.

Now you are ready to proceed with your project options. First, add some additional include directories. Do this by selecting Project Menu -> Properties, clicking on the C/C++ folder, and clicking General. In the Additional Include Directories text box, type the path to ArcSDK.h. In addition, type in the location of the ArcGIS Engine olb files. Remember to separate the paths with a semi-colon! You can also click on the ellipses to add new directories.

Next, click on Preprocessor and in the Preprocessor Definitions text box type in ‘ESRI_WINDOWS’ to define the ESRI_WINDOWS symbol. You can also click on the ellipses to define symbols.

Now you are ready to write your code. Don’t forget to start by including ArcSDK.h!

Compile your application

To compile an ArcGIS Engine application in Visual Studio .NET 2003, choose Build -> Build Solution.

Run your application

Before you can run an ArcGIS Engine command line application from within Visual Studio .NET 2003, you need to set up the arguments. Arguments are added to your program by customizing your project settings; go to the Project menu -> Properties -> Debugging item and add any arguments to ‘Command Arguments’. Make sure the configuration you are working on is selected in the configuration combobox.

Finally, run the application by choosing Debug -> Start Without Debugging or by typing CONTROL-F5. If you wish to run the application in debug mode, choose Debug -> Start, or type F5.

ArcGIS DEVELOPMENT WITH NMAKE AND THE WINDOWS COMMAND PROMPT

Set up a compiler for use from the command prompt

From the command prompt you have your choice of supported compilers; your first step will be to select one and prepare it for use. The command line build tools of Visual Studio are not available by default, so you need to use a provided batch file, vcvars32.bat, to configure one of the Visual Studio compilers for command line compilation and execution.

Access the Visual Studio 6.0 compiler from the command line

The command line build tools of Visual Studio are not available by default. However a batch file, called vcvars32.bat, is provided to make them available. The vcvars32.bat file must be run each time you open a new command prompt. Alternatively you can create your own batch file that runs vcvars32.bat and opens a command prompt that is ready for development. Each process is described below.

- Run vcvars32.bat from a command prompt

The vcvars32.bat file's default location is Program Files\Microsoft Visual Studio\VC98\Bin (Visual Studio 6.0).

1. Open a command prompt and cd to the directory containing vcvars32.bat.
2. Type vcvars32.bat to run the batch file.
3. For development, cd to the directory containing your code and begin. The Visual Studio command line build tools will be available from your command prompt.
4. For execution, run your exe with any necessary parameters.

- Create a batch file to run vcvars32.bat for you

1. Navigate to the directory in which you wish to store the batch file
2. Right-click in the directory and choose New -> Text Document
3. Change the name of the file to end in .bat (cmdpromptdevel.bat, for example) and click yes to confirm the name change.
4. Right-click the file and choose Edit

The vcvars32.bat file's default location is Program Files\Microsoft Visual Studio\VC98\Bin (Visual Studio 6.0).

5. Find vcvars32.bat, right-click the file and choose Edit.
6. Copy all of the text in vcvars32.bat into the batch file you created and opened above and close vcvars32.bat
7. Add the following line to your batch file:

```
%SystemRoot%\system32\cmd.exe
```

   This line opens a command prompt.

8. When you wish to develop on the command line, double-click your batch file. A command prompt will open with the necessary environment already set up for you. You can also create a shortcut to your batch file and add it to the Start Menu or a toolbar.

9. For development, cd to the directory containing your code and begin. The Visual Studio command line build tools will be available from your command prompt.
10. For execution, run your exe with any necessary parameters.

Access the Visual Studio .NET 2003 compiler from the command line

The command line build tools of Visual Studio are not available by default. However, Visual Studio .NET 2003 includes a command prompt that makes the tools available. To open the command prompt and access these tools, go to the Start Menu -> All Programs -> Microsoft Visual Studio .NET 2003 -> Visual Studio .NET Tools -> Visual Studio .NET 2003 Command Prompt.

When opened, the prompt automatically runs a batch file, vcvars32.bat, that makes the build tools available. The vcvars32.bat file's default location is Program Files\Microsoft Visual Studio .NET 2003\Vc7\bin.

Set up your application

Open your favorite text editor and begin writing your code. Use a Makefile to set the following include directories, and compiler options.

If desired, you can utilize the sample Makefile.Windows provided with ArcGIS Engine. Refer to the next section for details on this sample file.

If you didn't install to the default location, find and use your install location to add the \include\CPPAPI and \Com folders as include directories.

1. Use the /I compiler option to add Program Files\ArcGIS\include\CPPAPI and Program Files\ArcGIS\Com as additional include directories.
2. Use the /D compiler option to define the ESRI_WINDOWS symbol to direct the compiler to read the Windows support headers from within ArcSDK.h.
3. Use the /GX compiler flag to enable synchronous exception handling.
4. Use the /NOLOGO compiler flag to prevent display of a compiler startup banner and informational compiler messages.

Customizing the sample Makefile

As a convenience, an example Makefile, named Makefile.Windows, is included with ArcGIS Engine for your use. The following steps highlight the specific areas of the file that must be customized for it to be used in your development process. The modifications shown are based on an application that is written in a single file, my_application.cpp, which takes in a single file.

The template, Maekfile.Windows, can be found in ArcGIS Developer Help under Development Environments > C++ > Makefiles.

1. Throughout the makefile, update the program name, currently 'basic_sample', to reflect your application name.

```
# Set up the program name
PROGRAM = my_application.exe
…
# Program name updates – source and object file lists
CPPSOURCES = my_application.cpp
CPPOBJECTS = my_application.obj
…
# Program name updates -- dependencies list
my_application.obj: my_application.cpp my_application.h
```

The comment text used here to describe the code of the Makefile has been modified from the actual comments within the file to reflect the steps being taken.

2. The compiler options outlined in Steps 2 through 4 above have been set for you, however you need to complete Step 1 yourself in order to prepare the template for use in your applications.

Although the compiler options have already been set in the template, the line is included here to illustrate the use of the built-in CPPFLAGS macro.

```
…
# Setting up the include directories
INCLUDEDIRS =  \
  /I "C:\Program Files\ArcGIS\include\CPPAPI"  \
  /I "C:\Program Files\ArcGIS\Com"
…
# Setting up the compiler options
CPPFLAGS = /DESRI_WINDOWS $(INCLUDEDIRS) /nologo /GX
```

3. Provide dependencies lists for your application.

This line was also shown in Step 1to illustrate the update of the program name.

```
…
# Program name updates -- dependencies list
my_application.obj: my_application.cpp my_application.h
```

With your Makefile prepared, you are ready to write your code. Don't forget to start by including ArcSDK.h!

Compile your application

Once Makefile.Windows is ready to compile your application, you can compile from the command line by typing 'nmake /f Makefile.Windows'.

Run your application

Before you can run an ArcGIS Engine command line application the command line parameters must be set up. Update your makefile to include variables for each input parameter and a run target.

An example of these modifications is shown below:

```
# Setting up the program argument
INPUT = C:\Data\inputfile
…

# Setting up a run target
run:
  $(PROGRAM) $(INPUT)
```

Once Makefile.Windows is ready for use with your application, you will be able to run from the command line by typing 'nmake /f Makefile.Windows run'.

ARCOBJECTS C++ PRACTICES

The following are some of our recommendations for programming with the ArcGIS C++ API.

Recommended function usage

The C++ API provides it's own implementation of certain functions:

- *AoInitialize*—used where CoInitialize would be used in COM programming.

  ```
  extern "C" HRESULT AoInitialize(LPVOID pvReserved);
  ```

  This function initializes ArcGIS Engine and COM. The initialization must be done prior to any ArcObjects being used and in addition to the use of the *IAoInitialize* interface, which handles licensing for the application.

For the sake of simplicity the code snippets given don't always check HRESULTs, although as a developer you should always do so.

- *AoUninitialize*—used where CoUninitialize would be used in COM programming

  ```
  extern "C" void AoUninitialize(void);
  ```

  This function uninitializes ArcGIS Engine and COM.

- *AoExit*—used where exit would be used in non-ArcObjects code, as well as return would be used in main().

  ```
  extern "C" VOID AoExit (int number);
  ```

  AoExit must be called before an application is exited. This allows portability to supported operating systems that require *AoExit* to correctly clean up various ArcGIS Engine and COM elements.

The following example illustrates how the three functions discussed above should be used within an application:

Note that IAoInitialize is scoped so that it will be out of scope before AoUninitialize is called.

```
int main (int argc, char* argv[])
{
  // Initialize ArcGIS Engine and COM
  ::AoInitialize(NULL);

  // ArcGIS Engine licensing
  {
    IAoInitialize ipInit(CLSID_AoInitialize);
    esriLicenseStatus status;
    ipInit->Initialize(esriLicenseProductCodeEngine, &status);

    // ArcObjects Code here

    ipInit->Shutdown();
  }

  // Uninitialize ArcGIS Engine and COM
  ::AoUninitialize();

  // Exit the application
  AoExit(0);
}
```

- *AoCreateObject*—used where CoCreateInstance would be used in COM programming

```
extern "C" HRESULT AoCreateObject(REFCLSID rclsid,
                                  LPUNKNOWN pUnkOuter,
                                  DWORD dwClsContext,
                                  REFIID riid,
                                  LPVOID *ppv);
```

  When using smart pointers, this function will not be needed. However, you can create an instance of an object without smart pointers by using this function, as shown in the following code:

```
// Create a Workspace Factory without using smart pointers
IWorkspaceFactory *pWorkspaceFactory;
hr = ::AoCreateInstance(CLSID_ShapefileWorkspaceFactory, 0,
                    CLSCTX_INPROC_SERVER,
                    IID_IWorkspaceFactory,
                    (void **)&pWorkspaceFactory);
```

- *AoAllocBSTR* replaces SysAllocString

```
extern "C" BSTR AoAllocBSTR(const OLECHAR *sz);
```

- *AoFreeBSTR* replaces SysFreeString

```
extern "C" void AoFreeBSTR(BSTR bstr);
```

When using the smart type CComBSTR, the above two functions will not be needed. However, you can create and free BSTRs with them, as illustrated in the following example:

```
// display the feature type as "simple" or "other"
BSTR bsFeatureType;
esriFeatureType featType;
pFeatureClass->get_FeatureType(&featType);
switch (featType)
{
case esriFTSimple :
  bsFeatureType = ::AoAllocBSTR(L"simple");
  break;
default:
  bsFeatureType = ::AoAllocBSTR(L"other");
}
std::wcerr << L"Feature Type  : " << (BSTR) bsFeatureType << std::endl;
::AoFreeBSTR(bsFeatureType);
```

Get a feature's Field Values from type VARIANT

A feature's field values are passed back as type VARIANT, requiring you to do some processing to get the actual values. The following example loops through all of a feature's fields and prints out the feature's value for each field. Only certain field types are handled by the code shown here (for example, 2-byte integers, 4-byte integers, and BSTR strings); however, you could choose to handle other types as determined by the needs of your application.

```
// ipFeature is of type IFeaturePtr, and we assume it has already
// been declared and instantiated above
IFieldsPtr ipFields;
```

```
hr = ipFeature->get_Fields(&ipFields);
long fieldCount;
hr = ipFields->get_FieldCount(&fieldCount);
IFieldPtr ipField;
CComVariant fieldValue;

for (long i=0; i<fieldCount; i++)
{
  hr = ipFields->get_Field(i, &ipField);
  hr = ipFeature->get_Value(i, &fieldValue);

  // Get field's value based on its type
  switch (fieldValue.vt)
  {
  case VT_I2:
    std::cerr << fieldValue.iVal << std::endl;
    break;

  case VT_I4:
    std::cerr << fieldValue.lVal << std::endl;
    break;

  case VT_R4:
    std::cerr << fieldValue.fltVal << std::endl;
    break;

  case VT_R8:
    std::cerr << fieldValue.dblVal << std::endl;
    break;

  case VT_BSTR:
    std::wcerr << fieldValue.bstrVal << std::endl;
    break;

  default:
    std::wcerr << "Field type not supported.\n";
    break;
  }
}
```

Cocreate an object with a smart pointer reference after the smart pointer is declared:

A class is often cocreated at the same time that a smart pointer is declared:

```
IFooPtr ipFoo(CLSID_Foo);
```

However, in certain cases, it may be necessary for you to declare the smart pointer first and cocreate the class later. This can be accomplished in the manner below. Notice the use of the 'dot' member selection operator, as opposed to the arrow member selection operator that is usually used with smart pointer types:

```
IFooPtr ipFoo;
// more code would be here
ipFoo.CreateInstance(CLSID_Foo);
```

Inline Query Interface (QI)

You can use the built-in smart types to QI to other supported interfaces of a coclass on which you have an interface:

```
// ipTin is of type ITinPtr and is an interface to an instance of
//   the Tin coclass
// The Tin coclass supports both the ITin and ITinSurface interfaces
// GetVolume is a method on the ITinSurface interface
((ITinSurfacePtr) ipTin)->GetVolume(…);
```

Raw pointers in function signatures

Rather than having a smart pointer in the function signature, consider using a raw pointer to save the overhead of a call to the smart pointer constructor/destructor upon invocation of the function. You can still pass smart pointer objects to functions since each smart pointer has an overloaded pointer operator that returns the underlying raw pointer. The following example illustrates this:

```
HRESULT DoRasterOp(IRaster* pRaster);  // Function dec: raw pointer
IRasterPtr ipRaster;

HRESULT hr = DoRasterOp(ipRaster);     // Pass in smart pointer
```

Return ArcObjects from functions

This tip builds on the previous one. In this case, raw pointers are used in the function declaration and a double indirection is used for the object that will be returned. This allows you to alter what the pointer you are passed points to. Next, initialize a smart pointer object with the value you wish to return and assign it to the pointer you were passed in.

Notice the call to AddRef. *This is required to ensure that resources are managed properly.*

```
  HRESULT GetTinWorkspace(char* path, ITinWorkspace** ppTinWorkspace)
  {
    HRESULT hr = S_OK;
    IWorkspaceFactoryPtr ipWorkspaceFactory(CLSID_TinWorkspaceFactory);
    IWorkspacePtr ipWork;
    hr = ipWorkspaceFactory->OpenFromFile(CComBSTR(path), 0, &ipWork);
    if (FAILED(hr) || ipWork == 0)
      return E_FAIL;
    // Initialize ipTinWorkspace with ipWork
    ITinWorkspacePtr ipTinWorkspace(ipWork);
    *ppTinWorkspace = ipTinWorkspace;
    // AddRef() if the assignment worked
    if (*ppTinWorkspace)
      (*ppTinWorkspace)->AddRef();
    return hr;
  }
```

IUID

For further details on GUIDs, see 'The Microsoft Component Object Model' section earlier in this chapter.

There are several methods in ArcObjects that take an IUID object as a parameter. An IUID object is a globally unique identifier object. It can be either a GUID, as shown in the example below, or a ProgID.

```
IUIDPtr ipUID(CLSID_UID);
IEnumLayerPtr ipEnumLayer;

// use IGeoFeatureLayer's GUID
hr = ipUID->put_Value(CComVariant(L"{E156D7E5-22AF-11D3-9F99-
  00C04F6BC78E}"));
hr = ipMap->get_Layers(ipUID, VARIANT_TRUE, &ipEnumLayer);
```

Replicate the functionality of instanceof (Java) or TypeOf (Visual Basic)

It is common to have an interface pointer that could point to one of several coclasses. You can find out more information about the coclass by attempting to QI to other interfaces using if/else logic. For example, both the *RasterDataset* and *FeatureDataset* coclasses implement *IDataset*. If you are passed *IDataset* as a function parameter you can determine which coclass the *IDataset* references as follows:

```
void Foo (IDataset *pDataset)
{
   IFeatureDatasetPtr ipFeatureDataset(ipDataset);
   if (ipFeatureDataset != 0)
    {
      // use IFeatureDataset methods
    }
   else
    {
      IRasterDataset2Ptr ipRasterDataset(ipDataset);
      if (ipRasterDataset!= 0)
       {
         // use IRasterDataset2 methods
       }
    }
}
```

ESRI System interfaces

The System library within ArcGIS Engine, which is included with ArcSDK.h, contains a number of interfaces that simplify programming. It contains components that expose services used by the other ArcGIS libraries.

GROUPS OF OBJECTS IN INTERFACES WITHOUT STL

ArcObjects contains several interfaces for managing groups of objects. Although they are not all discussed here, the examples given illustrate some that can greatly simplify your work with COM objects in C++. Standard Template Library (STL) types exist which are similar to some of these; however, these interfaces are often simpler than their STL counterparts and are already set up to be used with COM objects. For additional details on any of the interfaces below, see ArcGIS Developer Help for C++.

IArray

The *IArray* interface provides access to members that control a simple array of objects. There are multiple related interfaces, such as *IDoubleArray* and *IVariantArray*. The following code snippet shows how to add the geometries of the features in a *FeatureCursor* to an array.

```
// ipFeatureLayer is of type IFeatureLayerPtr and ipQueryFilter is of
//  type IQueryFilterPtr. Both have already been declared and
//  instantiated.
IArrayPtr ipCacheArray (CLSID_Array);
IFeatureCursorPtr  ipFeatureCursor;
ipFeatureLayer->Search(ipQueryFilter, VARIANT_FALSE, &ipFeatureCursor);

IFeaturePtr ipFeature;
while (ipFeatureCursor->NextFeature(&ipFeature) == S_OK)
{
 IGeometryPtr ipGeom;
 ipFeature->get_ShapeCopy(&ipGeom);
 ipCacheArray->Add((IUnknownPtr) ipGeom);
}
```

ISet

The *ISet* interface provides access to members that control a simple set of unique objects. For example, the following code snippet cycles through a map's layers and attempts to add all of the unique feature class workspaces that aren't being edited to a set.

```
// ipMap is of type IMapPtr and was previously
// declared and instantiated
ISetPtr ipSet(CLSID_Set);
ILayerPtr ipLayer;
IFeatureLayerPtr ipFeatLayer;
IFeatureClassPtr ipFeatClass;
IDatasetPtr ipDataset;
IWorkspacePtr ipWorkspace;
IWorkspaceEditPtr ipWorkspaceEdit;
long layerCount;
hr = ipMap->get_LayerCount(&layerCount);
for (long i=0; i<layerCount; i++)
{
  hr = ipMap->get_Layer(i, &ipLayer);
  ipFeatLayer = ipLayer;
  // layer might not be a feature layer
  if (ipFeatLayer == 0 || FAILED(hr)) continue;
  hr = ipFeatLayer->get_FeatureClass(&ipFeatClass);
  // layer could reference bogus data
  if (ipFeatClass == 0 || FAILED(hr)) continue;
  ipDataset = ipFeatClass;
  hr = ipDataset->get_Workspace (&ipWorkspace);
  ipWorkspaceEdit = ipWorkspace;
  // some data are not editable
  if (ipWorkspaceEdit == 0 || FAILED(hr)) continue;
  VARIANT_BOOL beingEdited;
  hr = ipWorkspaceEdit->IsBeingEdited(&beingEdited);
  if (!beingEdited)
  {
  // only adds unique workspaces
  hr = ipSet->Add(ipWorkspace);
  }
}
```

Copy objects

The *IClone* interface is helpful when comparing and copying objects, saving time and computing resources. Many coclasses support the *IClone* interface. See the documentation for *IClone* in ArcGIS Developer Help for details. The following code snippet clones a *Point* object:

```
// ipMouseClickPoint is of type IPointPtr and was previously declared //
and instantiated.
IClonePtr ipClone (ipMouseClickPoint);
IClonePtr ipCloned;
ipClone->Clone(&ipCloned);
```

Error handling

COM methods return an *HRESULT* to signify the success or failure of a call, as discussed in the 'Developing with ArcObjects' section early in this chapter. When you are programming with the C++ API you should check the returned *HRESULT* of all calls to COM objects.

There are a few common *HRESULT*s that can be returned.

- *S_OK* signifies success
- *E_FAIL* indicates a failure.
- E_NOTIMPL indicates a method is not implemented

There are some macros that can be used to test the returned *HRESULT*.

- bool FAILED(HRESULT)
  For example, if the opening of a workspace in which you want to process data fails, you will not be able to use the data. At that point, you should exit the application to avoid a crash later on.

```
// Open the workspace
IWorkspaceFactoryPtr ipWorkspaceFactory(CLSID_RasterWorkspaceFactory);
IWorkspacePtr ipWorkspace;
HRESULT hr = ipWorkspaceFactory->OpenFromFile(inPath, 0, &ipWorkspace);
if (FAILED(hr) || ipWorkspace == 0)
{
  std::cerr << “Could not open the workspace.” << std::endl;
  return E_FAIL;
}
```

You might notice that the samples and scenarios do not follow the Good Error Handling practiced outlined here. This is done simply to increase code readability since error checking is not the focus of those bits of code.

- bool SUCCEEDED(HRESULT)
  For example, if you are going to create a new raster dataset, you must first know that no dataset already exists with the desired name. To find out if such a dataset exists, try to open it. If it succeeds, you know that you cannot create a new dataset with that name.

```
  // Check for existence of a dataset with the desired output name.
  //  If such exists, we can't create a new one with the name.
  IRasterDatasetPtr ipExistsCheck;
  hr = ipRastWork->OpenRasterDataset(outFile, &ipExistsCheck);
  if (SUCCEEDED(hr))
  {
    std::cerr << "A dataset with the output name already exists!" <<
std::endl;
    return E_FAIL;
  }
```

Troubleshooting

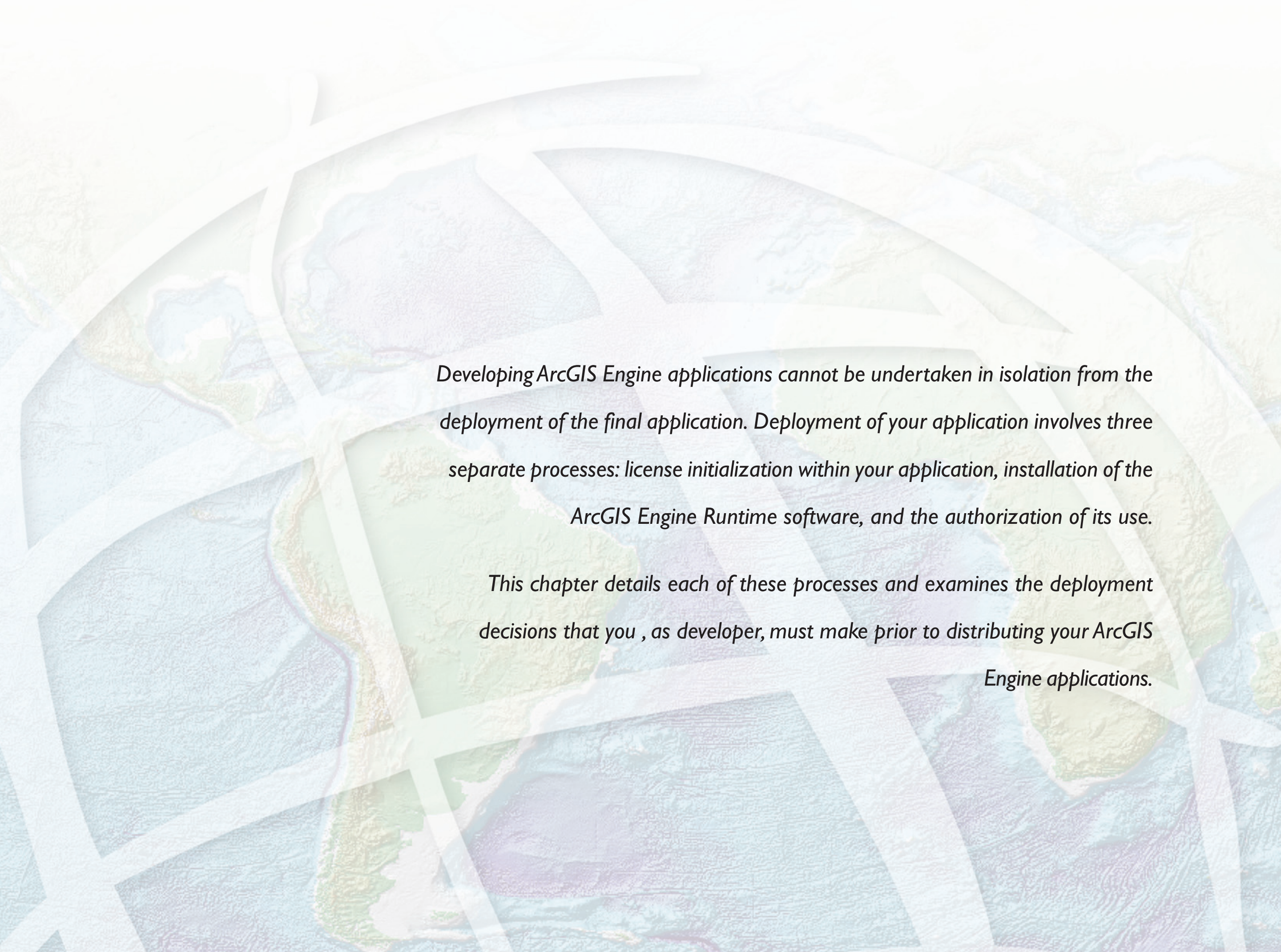
- *Cannot open include file "ArcSDK.h"*—If your sample is not compiling because it cannot open ArcSDK.h, make sure that you have the correct argument for the include directory: ArcGIS\include\CPPAPI. If it is correct and it is still not working, make sure the file is in that directory.
- *Cannot open type library file "esriSystem.olb"* —If your sample is not compiling because it cannot open an ESRI OLB file, make sure that you have the correct argument for the include directory: ArcGIS\Com. If it is correct and it is still not working, make sure the file is in that directory.
- *Code is compiling but will not run. Sometimes I get an odd "abnormal program termination" error.*—If samples or your own code compiles but fails to run, make sure you registered your ArcGIS Engine. Use the SoftwareAuthorization tool, found in Start > All Programs > ArcGIS and check current configuration. If the configuration does not list "standardengine", run the tool and follow its prompts.
- *Error: Please define either ESRI_WINDOWS or ESRI_UNIX*—You need to inform the compiler which set of header files to use. You forgot to define the ESRI_WINDOWS symbol.

Limitations

When using the C++ API, only Windows C++ command line applications are supported. However, GUI applications can be built with the COM API (including Visual C++), and several ActiveX controls provide GIS functionality to standalone GUI applications. For details, see the 'Visual C++' section earlier in this chapter.

5

Licensing and deployment



Developing ArcGIS Engine applications cannot be undertaken in isolation from the deployment of the final application. Deployment of your application involves three separate processes: license initialization within your application, installation of the ArcGIS Engine Runtime software, and the authorization of its use.

This chapter details each of these processes and examines the deployment decisions that you , as developer, must make prior to distributing your ArcGIS Engine applications.

One of the most important decisions you make as an ArcGIS Engine developer is to determine what the functional requirements of your prospective application are and the minimum licensing level needed to meet those needs. In most cases, it is inefficient for you to develop an application that requires an ArcGIS Engine Runtime with 3D, Spatial, and GeoDatabase options, just because you didn't assess the actual needs of your organization or client and plan accordingly.

This section of the chapter details the myriad of licensing options available.

One of the most important decisions you make as an ArcGIS Engine developer is to determine what the functional requirements of your prospective application are and the minimum licensing level needed to meet those needs. In most cases, it is inefficient for you to develop an application that requires an ArcGIS Engine Runtime with 3D, Spatial, and GeoDatabase options, just because you didn't assess the actual needs of your organization or client and plan accordingly.

Each of the sample applications that you build in Chapter 6, 'Developer scenarios', also illustrates the license initialization process.

This section of the chapter details the myriad of options available for intializing your application with multiple levels of licensing. It first dicusses them in detail and then provides a number of sample license initialization situations.

LICENSE INITIALIZATION

Each stand-alone application developed using ArcObjects must initialize itself with a suitable license to ensure it runs successfully on any machine to which it is deployed. License initialization must be performed by an application, at application start time, before any ArcObjects are accessed. Failure to do so will result in application errors.

While every standalone application must be initialized, the following examples illustrate situations that do not qualify as standalone and therefore don't need to be initialized as described:

• The application is a DLLthat will be incorporated into an application that will itself perform the license configuration.

• The application is an extension to ArcMap or another third party application.The extension is responsible for license management.

There are two types of licenses to consider when initializing an application: product licenses and, if an application uses any of the ArcGIS extension features, extension licenses. Each of these types of licenses are made available in certain license flavors—Engine Single Use, Desktop Concurrent Use, and Desktop Single Use.

- Engine Single Use—provides access to either the ArcGIS Engine or Engine with GeoDatabase Editing licenses. Each single use license is only available to the machine on which it is installed.
- Desktop Concurrent Use—FlexLM technology is used to provide concurrent access to the ArcGIS Desktop products, ArcView, ArcEditor, ArcInfo, and its extensions. The licenses can be available to multiple machines; they are stored on a license manager and checked out when being used.
- Desktop Single Use—provides access to Single Use ArcView, ArcEditor, and ArcInfo licenses. Like the Engine Single Use licenses, each one is available only to the machine on which it is installed. Even though this is significantly different than the Desktop Concurrent Use licensing, they actually utilize the same technology. This means that there is no mechanism for you, as an ArcGIS Engine developer, to differentiate between a Single Use and a Desktop Concurrent license, and hence, they should be treated as the same.

CONSIDERATIONS FOR APPLICATIONS NOT USING ARCGIS RUNTIME OPTIONS

Once an application has been initialized with a license it cannot be re-initialized; an application is initialized with a license for the duration of its life. When initializing an application with a license the following must be considered:

- The types of product license that the application can run with. For example, an enterprise geodatabase editing application will not be able to run with an ArcGIS Engine license or an ArcView license. However, it will be able to run with an ArcGIS Engine with GeoDatabase Editing license, an ArcEditor license or an ArcInfo license.

- The types of product license available to the application. For example, an application that can be run with an ArcGIS Engine license will also run with an ArcView, ArcEditor and ArcInfo license. However, you may not want to consume an ArcInfo license with such an application.

Using an ArcView license on an ArcGIS Engine application will give you access to all the functionality available to a standard Engine license. Likewise, using an ArcEditor license on an ArcGIS Engine application with GeoDatabase Editing will give you access to all the functionality available to an Engine license with GeoDatabase Editing.

Additional considerations for applications using ArcGIS extensions

When an application is initialized with a particular product license, a connection is made to a license server. All subsequent calls to check extensions out and in are made to the same license server. As such, you cannot use a combination of licenses from difference license servers or Engine Single Use.

- If an application is initialized with a Desktop Concurrent license, the application will subsequently only be able to access that Desktop Concurrent license server and its extension licenses.
- If an application is initialized with a Desktop Single Use license, the application will subsequently only be able to access that single use license server and its extension licenses.
- If an application is initialized with the Engine Single Use license on your machine, the application will subsequently only be able to access the Engine Single Use extension licenses.

It is possible before initialization has been performed to query the license servers (Desktop Concurrent or Single Use) and Engine Single Use to see if the licenses you require are available. If all the licenses you require are available using Engine Single Use then we recommend you use it in preference to the Desktop Concurrent and Desktop Single Use licenses. This means you will not limit the Desktop Concurrent licenses available to any other users.

The following extensions are available with Engine Single Use licenses:

- 3D Analyst
- Spatial Analyst
- StreetMap

For applications developed using the ArcGIS Controls, Table 1 describes each Controls runtime license requirements.

	ArcGIS Engine Single Use licence	ArcGIS Engine Single Use licence with Engine Singe Use 3D Analyst extension	ArcView, ArcEditor, ArcInfo Single Use or Desktop Cocurrent licence	ArcView, ArcEditor, ArcInfo Single Use or Desktop Cocurrent licence with 3D Analyst Extension	ArcReader
ArcReaderControl	✓				✓
MapControl	✓		✓		
PageLayoutControl	✓		✓		
ReaderControl	✓		✓		
ToolbarControl	✓		✓		
TOCControl	✓		✓		
GlobeControl		✓		✓	
SceneControl		✓		✓	

For applications that are developed using the ArcGIS Controls,Table 2 describes each Controls design-time license requirements.

ENGINE SINGLE USE DESIGNER OPTION

The Engine Single Use designer extension is required when developing applications with ArcObjects. This license works differently to the other extensions in that:

- The extension is only needed when designing and developing applications; it is never needed at run-time.
- The extension does not need to be checked out; it is automatically checked out for you.

	ArcGIS Engine Single Use licence	ArcGIS Engine Single Use licence with Engine Singe Use designer extension	ArcGIS Engine Single Use licence with Engine Singe Use designer extension and 3D Analyst extension	ArcView, ArcEditor, ArcInfo Single Use or Desktop Cocurrent licence	ArcView, ArcEditor, ArcInfo Single Use or Desktop Cocurrent licence with Publisher Extension
ArcReaderControl					✓
MapControl	✓			✓	
PageLayoutControl	✓			✓	
ReaderControl		✓			
ToolbarControl		✓			
TOCControl		✓			
GlobeControl			✓		
SceneControl			✓		

INITIALIZING AN APPLICATION WITH A LICENSE

The initialization of an application with a license must be performed in the following order:

1. Check the product license is available.
2. Check extension licenses are available (if required).
3. Initialize the application with the product license.
4. As required, perform extension check outs and check ins.
5. Shutdown the application.

CHECKING PRODUCT LICENSE AVAILABILITY

The product license that is chosen determines the functionality the application will be able to access. Once the product license has been initialized it cannot be changed for the duration of the applications life.

- If the product you require is not licensed you may optionally initialize the application with a higher product license.
- If there are no appropriate product licenses available the application should inform the user of the issue, and either allow the user to resolve the issue or exit the application.

CHECKING EXTENSION LICENSE AVAILABILITY

If an application has been designed to use extension functionality, it may check for the availability of extension licenses before the application is initialized. Checking the availability of an extension license must be done in conjunction with the product license that the application will ultimately be initialized with, as not every extension license is available with every product license.

- If an extension required by the application for it to run successfully is not available, the application should inform the user of the issue, and exit the application.
- If the extension functionality is not necessary for the application to function, and the extension license is unavailable, the application should disable to the user the functionality dependant upon the extension.

INITIALIZING THE APPLICATION

Once it has been established that the appropriate product and extension licenses are available the application should be initialized with the product license. Once initialized it is not possible to re-initialize the application.

CHECKING EXTENSIONS IN AND OUT

Extensions can either be checked out as and when an application requires the extension functionality, and checked in once the application has finished with the functionality; or the extension can be checked out directly after the application is initialized and checked back in before shutdown. The way that the extensions are checked in and out will depend on the type of product license the application was initialized with.

- If the application was initialized with either of the Engine Single Use licenses, any extensions used by the application will also be Engine Single Use. As such any extensions can be checked out directly after the application is initialized and checked back in before shutdown.
- If the application was initialized with a license server and the extensions are required by the application for it to run successfully, the extensions should be checked out directly after the application is initialized and checked back in before shutdown.

- If the application was initialized with a license server and the extension functionality is not necessary for the application to function, the extensions can either be checked out directly after the application is initialized or checked out as and when the extension functionality is required. When the extension is checked in the functionality should be disabled.

SHUTDOWN

Before an application is shut down the *AoInitialize* object must be shut down. This ensures that any ESRI libraries that have been used are unloaded.

LICENSE INITIALIZATION FAILURE

If a product or extensions fails to check out the license status indicates the reason for the failure. Licenses can fail to check out for the following reasons:

- A product is not licensed.
- A license is unavailable because it is already being used (Desktop Concurrent licenses only).
- An unexpected license failure due to system administration problems.
- The license is already initialized. An application is initialized with a product license for the duration of it life. It is possible to check what product license an application has been initialized with. For example, if an application containing some enterprise GeoDatabase editing has been initialized with an Engine Single Use with GeoDatabase Editing or an ArcEditor or ArcInfo license, the editing functionality can be enabled. If however the application has been initialized with an Engine Single Use or ArcView license the editing functionality must be disabled.

EXAMPLE ONE

The application requires a minimum of an ArcGIS Engine license. If an ArcGIS Engine license is not available, the application can run with an ArcView or ArcEditor license but not an ArcInfo license (ArcInfo will provide all the functionality we require but we do not wish to consume an ArcInfo license for our simple application). The application also requires 3D Analyst and Spatial Analyst extension functionality for it to run successfully, so both of these extensions need to be checked out for the duration of the application. The following steps are taken to initialize the application with a license.

Can the application be initialized with the ArcGIS Engine product license?

1. Check whether an ArcGIS Engine product license is available.
2. Check whether a 3D Analyst extension license is available with the ArcGIS Engine product license.
3. Check whether a Spatial Analyst extension license is available with the ArcGIS Engine product license.

If any of these licenses are not available then the application cannot be initialized with an ArcGIS Engine license. Can the application be initialized with an ArcView product license instead?

If all of these licenses are available, initialize the application.

4. Check out the ArcGIS Engine product license by initializing the application.

If the license failed to check out, can the application be initialized with an ArcView product license instead?

If the license checked out, check the extension licenses out.

5. Check out the 3D Analyst extension.

6. Check out the Spatial Analyst extension.

If either of the extension licenses failed to check out, the application cannot run.

If the extension licenses are checked out, the application has been successfully configured with licenses.

Can the application be initialized with the ArcView product license?

1. Check whether an ArcView product license is available.
2. Check whether a 3D Analyst extension license is available with the ArcView product license.
3. Check whether a Spatial Analyst extension license is available with the ArcView product license.

If any of these licenses are not available then the application cannot be initialized with an ArcView license. Can the application be initialized with an ArcEditor product license instead?

If all of these licenses are available, initialize the application.

4. Check out the ArcView product license by initializing the application.

If the license failed to check out, can the application be initialized with an ArcEditor product license instead?

If the license checked out, check the extension licenses out.

5. Check out the 3D Analyst extension.

6. Check out the Spatial Analyst extension.

If either of the extension licenses failed to check out, the application cannot run. Checking out Desktop Concurrent licenses may fail here as they may have been checked out by another application since this application checked their availability.

If the extension licenses are checked out, the application has been successfully configured with licenses.

Can the application be initialized with the ArcEditor product license?

1. Check whether an ArcEditor product license is available.
2. Check whether a 3D Analyst extension license is available with the ArcEditor product license.
3. Check whether a Spatial Analyst extension license is available with the ArcEditor product license.

If any of these licenses are not available then the application cannot be initialized with an ArcEditor license and the application cannot run.

If all of these licenses are available, initialize the application.

4. Check out the ArcEditor product license by initializing the application.

If the license failed to check out, the application cannot run.

If the license checked out, check the extension licenses out.

5. Check out the 3D Analyst extension.

6. Check out the Spatial Analyst extension.

If either of the extension licenses failed to check out, the application cannot run. Checking out Desktop Concurrent licenses may fail here as they may have been checked out by another application since this application checked their availability.

If the extension licenses are checked out, the application has been successfully configured with licenses.

EXAMPLE 2

The application is an enterprise GeoDatabase editing application so requires a minimum of an ArcGIS Engine with GeoDatabase Editing license. If an ArcGIS Engine with GeoDatabase Editing license is not available, the application can run with an ArcEditor or ArcInfo license. The application requires the Spatial Analyst extension functionality for it to run successfully, so the extension needs to be checked out for the duration of the application. The following steps are taken to initialize the application with a license.

Can the application be initialized with the ArcGIS Engine with geodatabase editing product license?

1. Check whether an ArcGIS Engine with GeoDatabase Editing product license is available.

2. Check whether a Spatial Analyst extension license is available with the ArcGIS Engine with GeoDatabase Editing product license.

If any of these licenses are not available then the application cannot be initialized with an ArcGIS Engine with GeoDatabase Editing license. Can the application be initialized with an ArcEditor product license instead?

If all of these licenses are available, initialize the application.

3. Check out the ArcGIS Engine with GeoDatabase Editing product license by initializing the application.

If the license failed to check out, can the application be initialized with an ArcEditor product license instead?

If the license checked out, check the extension license out.

4. Check out the Spatial Analyst extension.

If the extension licenses failed to check out, the application cannot run. Checking out Desktop Concurrent licenses may fail here as they may have been checked out by another application since this application checked their availability.

If the extension license checked out, the application has been successfully configured with licenses.

Can the application be initialized with the ArcEditor product license?

1. Check whether an ArcEditor product license is available.
2. Check whether a Spatial Analyst extension license is available with the ArcEditor product license.

If any of these licenses are not available then the application cannot be initialized with an ArcEditor license. Can the application be initialized with an ArcInfo product license instead?

If all of these licenses are available, initialize the application.

3. Check out the ArcEditor product license by initializing the application.

If the license failed to check out, can the application be initialized with an ArcInfo product license instead?

If the license checked out, check the extension license out.

4. Check out the Spatial Analyst extension.

If the extension license failed to check out, the application cannot run. Checking out Desktop Concurrent licenses may fail here as they may have been checked out by another application since this application checked their availability.

If the extension license checked out, the application has been successfully configured with licenses.

Can the application be initialized with the ArcInfo product license?

1. Check whether an ArcInfo product license is available.
2. Check whether a Spatial Analyst extension license is available with the ArcInfo product license.

If any of these licenses are not available then the application cannot be initialized with an ArcInfo license and the application cannot run.

If all of these licenses are available, initialize the application.

3. Check out the ArcInfo product license by initializing the application.

If the license failed to check out, the application cannot run.

If the license checked out, check the extension license out.

4. Check out the Spatial Analyst extension.

If the extension license failed to check out, the application cannot run. Checking out Desktop Concurrent licenses may fail here as they may have been checked out by another application since this application checked their availability.

If the extension license checked out, the application has been successfully configured with licenses.

EXAMPLE 3

The application requires a minimum of an ArcGIS Engine license. If an ArcGIS Engine license is not available, the application should be able to run with an ArcView, ArcEditor or ArcInfo license. The application requires 3D Analyst functionality, but not for the application to run successfully, so the extension will be checked out as needed.

Can the application be initialized with the ArcGIS product license?

1. Check whether an ArcGIS Engine product license is available.
2. Check whether a 3D Analyst extension license is available with the ArcGIS Engine product license.

If either of these licenses are not available then the application cannot be initialized with an ArcGIS Engine license. Can the application be initialized with an ArcView product license instead?

If both these licenses are available, initialize the application.

3. Check out the ArcGIS Engine product license by initializing the application.

If the license failed to check out then the application cannot run. Can the application be initialized with an ArcView product license instead?

If the license is checked out, the application has been successfully configured with a license, but the functionality using the 3D Analyst extension should be disabled.

Can the application be initialized with the ArcView product license?

1. Check whether an ArcView product license is available.
2. Check whether a 3D Analyst extension license is available with the ArcView product license.

If either of these licenses are not available then the application cannot be initialized with an ArcView license. Can the application be initialized with an ArcEditor product license instead?

If both these licenses are available, initialize the application.

3. Check out the ArcView product license by initializing the application.

If the license failed to check out then the application cannot run. Can the application be initialized with an ArcEditor product license instead?

If the license is checked out, the application has been successfully configured with a license, but the functionality using the 3D Analyst extension should be disabled.

Can the application be initialized with the ArcEditor product license?

1. Check whether an ArcEditor product license is available.
2. Check whether a 3D Analyst extension license is available with the ArcEditor product license.

If either of these licenses are not available then the application cannot be initialized with an ArcInfo license. Can the application be initialized with an ArcInfo product license instead?

If both these licenses are available, initialize the application.

3. Check out the ArcEditor product license by initializing the application.

If the license failed to check out then the application cannot run. Can the application be initialized with an ArcInfo product license instead?

If the license is checked out, the application has been successfully configured with a license, but the functionality using the 3D Analyst extension should be disabled.

Can the application be initialized with the ArcInfo product license?

1. Check whether an ArcInfo product license is available.
2. Check whether a 3D Analyst extension license is available with the ArcInfo product license.

If either of these licenses are not available then the application cannot be initialized with an ArcInfo license.

If both these licenses are available, initialize the application.

3. Check out the ArcInfo product license by initializing the application.

If the license failed to check out then the application cannot run.

If the license is checked out, the application has been successfully configured with a license, but the functionality using the 3D Analyst extension should be disabled.

Using the 3D Analyst functionality

1. Check whether the 3D Analyst extension is already checked out

If the license is checked out the application can use the 3D Analyst functionality.

If the license is not checked out, check the license out.

2. Check out the 3Dl Analyst extension.

If the license failed to check out then the application cannot use the 3D Analyst functionality. Checking out a Desktop Concurrent license may fail here as it may have been checked out by another application since this application checked its availability.

If the license is checked out the application can use the 3D Analyst.

The ArcGIS Engine Runtime setup program is available for deployment on Microsoft Windows operating systems.

The following includes a brief introduction to the technology used to create the setup program and discusses various deployment methods for the ArcGIS Engine Runtime setup on Windows.

WHAT IS THE ARCGIS ENGINE RUNTIME SETUP?

The ArcGIS Engine Runtime setup was created using Microsoft Windows Installer technology. This technology uses a package file (.msi) and a client-side installer service (msiexec.exe). The Windows Installer is a service that runs on your operating system. This service enables the operating system to manage the installation and uses the information contained within the package file to install the software.

The Msiexec.exe program is a component of Windows Installer. Msiexec.exe uses a dynamic link library, Msi.dll, to read the package files (.msi), apply transforms (.mst), and incorporate command-line options.

More information on Windows Installer can be found in the Windows Installer Software Development Kit (SDK).

http://www.microsoft.com/msdownload/platformSDK/sdkupdate/

DEPLOYING ARCGIS ENGINE RUNTIME

An ArcGIS Engine Runtime developed application requires ArcGIS Engine Runtime to be installed on the end-user's machine. To ensure that ArcGIS Engine Runtime is on the machine, you can:

Have the user run the ArcGIS Engine Runtime setup directly

1. The user can launch the setup from an ArcGIS Engine Runtime CD
2. Before proceeding with the setup, the user will need to check if ArcGIS Engine Runtime has already been installed on their machine. To check if ArcGIS Engine Runtime is on the machine:

   i. Go to Control Panel > Add/Remove Programs.

   ii. In the programs list, check if ArcGIS Engine Runtime is listed. If it is, ArcGIS Engine Runtime has been installed on this machine. Click Support Information to verify the product version number is 9.0.

3. If ArcGIS Engine Runtime is installed on the machine, the user can proceed with installing your developed application.
4. If the ArcGIS Engine Runtime setup is launched, and ArcGIS Engine Runtime already exists on the machine the setup will execute as a maintenance installation.
5. Before proceeding with the setup, the user must check that no ArcGIS products of version less than 9.0 are installed on the machine. This includes: ArcIMS, ArcGIS Desktop, ArcGIS Workstation and ArcReader. To check if any of these products are on the machine:

   i. Go to Control Panel > Add/Remove Programs.

ii. In the programs list, check to see if any ArcGIS Products are installed on the machine. If an ArcGIS product is listed, click Support Information to determine the product version number. If the version is less than 9.0, the product should be removed before installing ArcGIS Engine Runtime 9.0.

6. If ArcGIS Engine Runtime is not installed on the machine, the user must launch the ArcGIS Engine Runtime Setup.exe before installing your developed application.
7. If your application requires the Java or .NET installation feature, the user must check that these features are installed. Java and .NET are optional installation features and are not installed with a typical ArcGIS Engine Runtime installation. To check if these features are installed:

i. Go to Control Panel > Add/Remove Programs

ii. In the programs list, check if ArcGIS Engine Runtime is listed. If it is, ArcGIS Engine Runtime has been installed on this machine.

iii. Click Change on the ArcGIS Engine Runtime feature, then Modify. The Select Features dialog box shows the current state of the ArcGIS Engine Runtime installation. A hard drive icon illustrates the features installed on the machine. A red cross illustrates the features not installed on the machine, but available for installation.

iv. To install a feature, the user must select the feature, right click and select Will be installed on local hard drive.

Include the ArcGIS Engine Runtime setup in your application's setup program

1. As a developer, you can include the ArcGIS Engine Runtime setup in your application's setup using a number of methods. The methods for deploying ArcGIS Engine Runtime are discussed later in this chapter.
2. Before your setup launches the ArcGIS Engine Runtime setup, you need to determine if ArcGIS Engine Runtime already exists on the user's machine by performing a system check. Performing a system check is discussed later in this chapter.
3. If ArcGIS Engine Runtime is not installed on the machine, you can install it by using one of the deployment options discussed below.
4. If ArcGIS Engine Runtime is already installed on the machine, you do not need to install ArcGIS Engine Runtime and you can instead proceed with installing your application only.
5. If your application requires the Java or .NET installation feature, you must check that these features are installed (discussed later in this chapter). Java and .NET are optional installation features and are not installed with a typical ArcGIS Engine Runtime installation. If these features are not installed and your application requires one or both of them, you will need to install the feature(s) required.

ARCGIS ENGINE RUNTIME SETUP FEATURES

An .msi setup consists of features. Features are a group of components a user can install. The ArcGIS Engine Runtime setup consists of the following installation features:

Feature	Descriptive Feature Name	Description
ArcEngine	ArcGIS Engine	ArcGIS Engine
JavaRuntime	ArcGIS Engine Java Runtime	Java Archives
DotNetRuntime	ArcGIS Engine .Net Runtime	.Net Assemblies

SYSTEM REQUIREMENTS FOR ARCGIS ENGINE RUNTIME

ArcGIS Engine Runtime is supported on Windows NT SP 6a, Windows 2000, Windows XP Professional and Windows 2003 Server.

For additional or updated information regarding ArcGIS Engine Runtime system requirements, visit: http://support.esri.com.

ARCGIS ENGINE RUNTIME INSTALLATION LOCATION

The ArcGIS 9.0 products, ArcGIS Engine Runtime, ArcGIS Engine Developer Kit, ArcGIS Desktop, ArcReader standalone and ArcGIS Server will install to the same installation directory. The first ArcGIS 9.0 product installed will determine the installation location for all subsequent ArcGIS 9.0 products.

NOTE: If you have ArcIMS® ArcMap Server 9.0 already installed, ArcGIS 9.0 products will default to the ArcIMS installation location.

For example, if ArcGIS Desktop is installed to C:\Desktop, the installation location for ArcGIS 9.0 will be C:\Desktop\ArcGIS. If you install ArcGIS Engine Runtime next, you will not be provided with the opportunity to browse to an installation location. The ArcGIS Desktop installation has predetermined the installation location for all ArcGIS 9.0 products. Therefore, in this example, ArcGIS Engine Runtime will also be installed to C:\Desktop\ArcGIS.

If you run out of disk space while installing an ArcGIS 9.0 product, you will need to uninstall all ArcGIS 9.0 products (listed above) and reinstall them to a location where more disk space is available. ArcGIS 9.0 products (excluding ArcSDE, ArcIMS and ArcINFO Workstation) cannot be installed to different locations.

DEPLOYMENT METHODS

There are two recommended methods for deploying ArcGIS Engine Runtime.

1. The user installs ArcGIS Engine Runtime setup directly from the CD
   The ArcGIS Engine Runtime setup can be redistributed on CD. You may copy the contents of ArcGIS Engine Runtime CD image and create additional CD's, or you may contact ESRI to obtain additional ArcGIS Engine Runtime CD's.

Additional requirements for running the setup directly from the CD:

- The user will need to check if ArcGIS Engine Runtime has already been installed on their machine. If ArcGIS Engine Runtime is already installed on the machine, the user can proceed with installing your developed application. If the ArcGIS Engine Runtime setup is launched, and ArcGIS Engine Runtime already exists on the machine the setup will execute as a maintenance installation.

- ArcGIS Engine Runtime 9.0 cannot be installed on a machine with any ArcGIS Products less than version 9.0. ArcGIS Products less than version 9.0 must be uninstalled before installing ArcGIS Engine Runtime 9.0.
- If any of the ArcGIS Engine Runtime features required for your application are not installed, the user should install them. If the required ArcGIS Engine Runtime features are on the machine they do not have to run the ArcGIS Engine Runtime setup.

For additional ArcGIS Engine Runtime CD's contact ESRI Customer Service at www.esri.com, or in the U.S. call 888-377-4575, or contact your local ESRI Regional Office.

2. Incorporate the ArcGIS Engine Runtime setup to run within your application's setup program
   ArcGIS Engine can be installed without a user interface by running the setup using Windows Installer command line parameters.

The ArcGIS Engine Runtime setup can be incorporated using the following options:

a. At the end of an msi-based setup

b. Within a batch file

c. Within a scripted setup

Examples of these options are provided in this chapter.

Additional requirements for incorporating ArcGIS Engine Runtime setup:

If any of the ArcGIS Engine Runtime features required for your application are not installed, your setup should add them. If the required ArcGIS Engine Runtime features are on the machine, you do not have to run the ArcGIS Engine Runtime setup.

When ArcGIS Engine Runtime is installed, a registry entry is created for it at the following location:

```
HKEY_LOCAL_MACHINE\Software\ESRI\ArcGIS Engine Runtime
```

The RealVersion value of this registry key will be 9.0

You can use this key to check whether ArcGIS Engine Runtime is installed on the user's machine.

1. Perform a system check for ArcGIS Engine Runtime optional installation features
   If your application requires ArcGIS Engine Runtime .NET or Java features to be installed, you must check the user's system for the presence of the appropriate .NET or Java installation feature as well as ArcGIS Engine Runtime (see Perform a system check for ArcGIS Engine Runtime).

The following registry entry will determine the installation state of ArcGIS Engine Runtime features:

```
[HKEY_CLASSES_ROOT\Installer\Features\7A1A3A9178A2BC74EB114EA6B5DB1C1B]
"Registry"=""
"ArcEngine"=""
"DotNetRuntime"="ArcEngine" or "?ArcEngine"
"JavaRuntime"="ArcEngine" or "?ArcEngine"
```

To determine if the .NET feature is installed:
The .NET ArcGIS Engine installation feature requires .NET Framework 1.1 to be installed on the machine. The following registry key can be used to determine whether the .NET ArcGIS Engine feature is installed:

```
HKEY_CLASSES_ROOT\Installer\Features\7A1A3A9178A2BC74EB114EA6B5DB1C1B
```

The DotNetRuntime string value represents the .NET ArcGIS Engine installation feature.

If a DotNetRuntime string value under this registry key is not displayed, .NET Framework 1.1 was not installed on the machine at the time the ArcGIS Engine Runtime setup was run.

If the DotNetRuntime feature IS NOT installed then

```
"DotNetRuntime"="?ArcEngine"
```

If DotNetRuntime IS installed then

```
"DotNetRuntime"="ArcEngine"
```

To determine if the Java feature is installed:
The following registry key can be used to determine whether the .Java ArcGIS Engine feature is installed:

```
HKEY_CLASSES_ROOT\Installer\Features\7A1A3A9178A2BC74EB114EA6B5DB1C1B
```

The JavaRuntime string value represents the .Java ArcGIS Engine installation feature.
If JavaRuntime IS NOT installed then

```
"JavaRuntime"="?ArcEngine"
```

If JavaRuntime IS installed then

```
"JavaRuntime"="ArcEngine"
```

If you detect that a required installation feature is not installed, you will need to install only that feature using the ADDLOCAL Windows Installer command within your script. For example, ADDLOCAL=DotNetRuntime or ADDLOCAL=JavaRuntime.

If you choose to install only the DotNetRuntime or JavaRuntime feature, and the ArcEngine feature (main ArcGIS Engine installation feature) has not been installed, the setup will include the ArcEngine feature during installation.

If .NET Framework 1.1 is not detected on the machine, the DotNetRuntime feature will be hidden, and will not be installed.

The following are requirements that need to be taken into consideration when incorporating the ArcGIS Engine Runtime setup within your application's setup program.

2. The ArcGIS Engine Runtime setup uses Windows Installer 2.0.

- The ArcGIS Engine Runtime setup uses Windows Installer 2.0. Windows Installer version 2.0 must be installed and running on the target machine prior to running the Engine Runtime setup using command line parameters. To check the version of Windows Installer on the machine:

    i. Locate msiexec.exe in the system32 folder.

    ii. Right click on msiexec.exe and select Properties.

    iii.On the Properties dialog box, select the Version tab to check the file version.

- If you are including the ArcGIS Engine Runtime setup in a non-msi-based setup, the end-user will require Microsoft Installer version 2.0 to be installed on their machine. The ArcGIS Engine Runtime setup uses the Windows Installer technology.

- If you will be launching the Engine Runtime setup at the end of an msi-based setup, you must create your msi setup using Windows Installer version 2.0 or higher, to be compatible with the ArcGIS Engine Runtime setup.

- The ArcGIS Engine Runtime msi cannot be nested within an msi. Each product, including ArcGIS Engine Runtime must be listed individually in Add/ Remove Programs.

- The Windows installer version 2.0 setup is available from <ArcGIS Engine Runtime CD>\Support\MSI\instmsiw.exe

3. Perform a system check for ArcGIS Engine Runtime

   If you are including the ArcGIS Engine Runtime setup, in your application's setup program, you must perform a system check on the users machine to detect whether ArcGIS Engine Runtime has already been installed. If the ArcGIS Engine Runtime setup is launched, and ArcGIS Engine Runtime already exists on the machine the setup will execute as a maintenance installation.

Sample Scripts:

The following is an example of the command line parameters implemented within a batch file that could be used to install only the ArcGIS Engine Runtime .NET feature:

```
REM ###########################
REM Set variables
SET MSI_PATH=\\CDROM\Setup.msi

REM ###########################

REM Launch MSI Silently - NO UI
msiexec.exe /i "%MSI_PATH%" /qn ADDLOCAL=DotNetRuntime
```

Note: In the above example, if .NET Framework 1.1 is not detected on the machine, the DotNetRuntime feature will not be installed.

The following is an example of the command line parameters implemented within a batch file that could be used to install only the ArcGIS Engine Runtime Java feature:

```
REM #############################
REM Set variables
SET MSI_PATH=\\CDROM\Setup.msi

REM #############################

REM Launch MSI Silently - NO UI
msiexec.exe /i “%MSI_PATH%” /qn ADDLOCAL=JavaRuntime
```

DEPLOYMENT OPTION EXAMPLES

The following are examples of deployment options for the ArcGIS Engine Runtime setup.

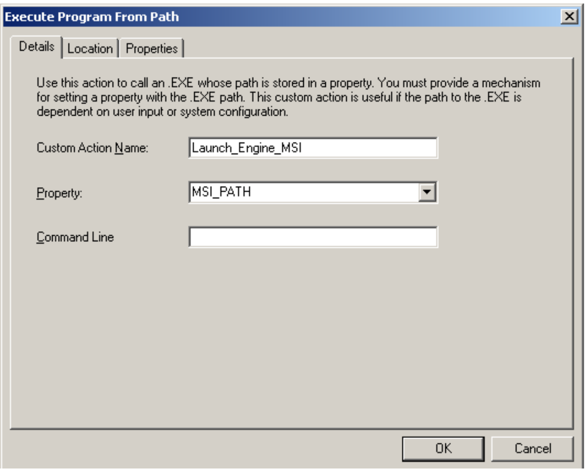
At the end of an MSI-based setup

The following example uses Wise for Windows Installer MSI Authoring program to launch the ArcGIS Engine Runtime setup at the end of another Msi-based setup. The ArcGIS Engine Runtime setup will be launched after clicking the Finish button. The example below suggests custom actions behind the Finish button.

This example assumes that the ArcGIS Engine Runtime setup resides in the same location as your application’s setup program. In this case, Setup.exe resides in a folder named ArcEngine. To launch ArcGIS Engine Runtime Setup.exe located in an ArcEngine folder on your application’s media:

1. Create Properties MSI_PATH and ArcEngineExists and initialize them to 1 in the Property table.
2. Perform a system check for ArcGIS Engine Runtime at the beginning of your setup program. The system check should search for the following registry key and set the property ArcEngineExists to True if the registry value returned is 9.0. HKEY_LOCAL_MACHINE\Software\ESRI\ArcGIS Engine Runtime and the RealVersion value will be 9.0.
3. Create a Custom action called Launch_Engine_MSI. Use the “Execute Program from Path” type of custom action. Set the Property in this custom action to MSI_PATH. The path to execute the program from is the path

specified in the Property field, in this case MSI_PATH. No Command Line is needed as setup.exe is being launched.


Execute Program From Path

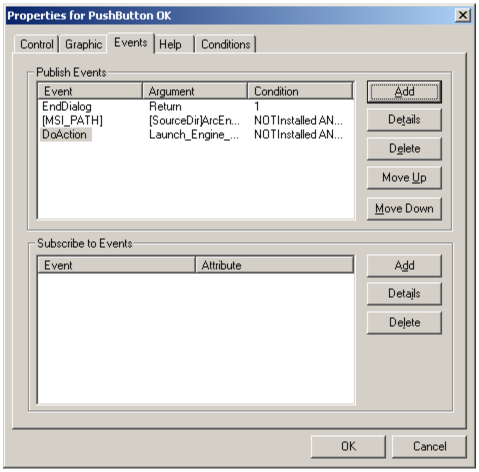
Use this action to call an .EXE whose path is stored in a property. You must provide a mechanism for setting a property with the .EXE path. This custom action is useful if the path to the .EXE is dependent on user input or system configuration.

Custom Action Name: Launch_Engine_MSI

Property: MSI_PATH

Command Line

4. In the Exit Dialog of your application's setup program, add two actions behind the Finish button control.


Properties for PushButton OK

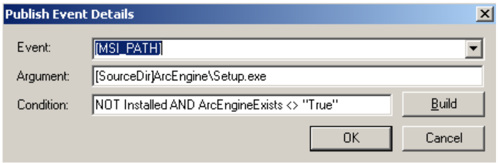
Publish Events

Event	Argument	Condition
EndDialog	Return	1
[MSI_PATH]	[SourceDir]ArcEn...	NOTInstalled AN...
DoAction	Launch_Engine_...	NOTInstalled AN...

Subscribe to Events

Event	Attribute

The first action sets the MSI_PATH property to [SourceDir]ArcEngine\Setup.exe. This will change depending on where the ArcGIS Engine Runtime setup is located on the media. In this example, Setup.exe is located on the CD in a folder named ArcEngine.

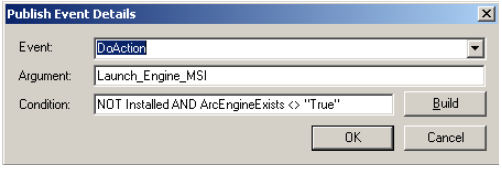

Publish Event Details

Event: [MSI_PATH]

Argument: [SourceDir]ArcEngine\Setup.exe

Condition: NOT Installed AND ArcEngineExists <> "True"

The second action calls the Launch_Engine_MSI that you previously created.



Event: DoAction

Argument: Launch_Engine_MSI

Condition: NOT Installed AND ArcEngineExists <> "True"

Both these custom actions should execute only if the property ArcEngineExists does not equal True.

Within a scripted setup

A scripted setup can be used to install ArcGIS Engine Runtime using command line parameters (the example below uses the Wise Install Master Setup Authoring Software):

Note: In the example below MSI_PATH will change depending on where the ArcGIS Engine Runtime setup is located on the media. In this example, Setup.msi is located on the CD in a folder named ArcEngine

```
Rem Set variable
Set Variable MSI_PATH to \\CDROM\ArcEngine\Setup.msi
Rem Launch ArcGIS Runtime setup program silently – No UI
Execute %SYS32%\msiexec.exe /i %MSI_PATH% /qn (Wait)
Rem Launch ArcGIS Runtime setup program silently – No UI except for a modal
dialog box displayed at the end
Execute %SYS32%\msiexec.exe /i %MSI_PATH% /qn+ (Wait)
```

Within a batch file

The following are examples of command line parameters implemented within a batch file that could be used to install ArcGIS Engine Runtime:

Note: In the example below MSI_PATH will change depending on where the ArcGIS Engine Runtime setup is located on the media. In this example, Setup.msi is located on the CD in a folder named ArcEngine.

```
REM ############################
REM Set variables
SET MSI_PATH=\\CDROM\ArcEngine\Setup.msi

REM ############################

REM Launch MSI Silently - NO UI
msiexec.exe /i “%MSI_PATH%” /qn

REM Launch MSI Silently - Reduced UI
msiexec.exe /i “%MSI_PATH%” /qb

REM Launch MSI Silently - No UI except for a modal dialog box displayed at
the end.
msiexec.exe /i “%MSI_PATH%” /qn+
```

Launch the setup from the ArcGIS Engine Runtime CD

The ArcGIS Engine Runtime setup can be launched manually from the CD using setup.exe.

1. Copy the contents of the ArcGIS Engine Runtime CD image and create additional CD's, or include the ArcGIS Engine Runtime CD image on the same CD as your application's setup program
2. Inform your user of the following:

   a. Check to see if ArcGIS Engine Runtime 9.0 is already installed on the machine

   b. Check that no ArcGIS Products of version less than 9.0 are installed on the machine. If they are, they must be uninstalled before installing ArcGIS Engine Runtime 9.0.

   c. If ArcGIS Engine Runtime 9.0 is already installed, proceed with the developed application's setup program.

   d. If ArcGIS Engine Runtime 9.0 is not installed, launch Setup.exe from the ArcGIS Engine Runtime CD image location. Once installation of ArcGIS Engine Runtime is complete, launch the developed application's setup program.

HOW TO USE INSTALLATION COMMAND LINE PARAMETERS

The ArcGIS Engine Runtime setup can be installed using the .msi file and client-side installer service (msiexec.exe) command line parameters. The table below illustrates some available msiexec.exe command line parameters:

(SOURCE: http://msdn.microsoft.com/library/default.asp?url=/library/en-us/msi/setup/command_line_options.asp)

Option	Parameters	Description
/I	**Package | ProductCode**	Installs or configures a product.
/a	**Package**	Administrative installation option. Installs a product on the network.
/x	**Package | ProductCode**	Uninstalls a product.
/q	**n | b | r | f**	Sets user interface level. **q** No UI **qn** No UI **qb** Basic UI. Use qb! to hide the Cancel button. **qr** Reduces UI with no modal dialog box displayed at the end of the installation. **qf** Full UI and any authored FatalError, UserExit, or Exit modal dialog boxes at the end. **qn+** No UI except for a modal dialog box displayed at the end. **qb+** Basic UI with a modal dialog box displayed at the end. The modal box is not displayed in the user cancels the installation. Use **qb+!** or **qb!+** to hide the Cancel button. **qb-** Basic UI with no modal dialog boxes. Please note that /qb+- is not a supported UI level. Use **qb-!** or **qb!-** to hide the Cancel button. ***Note that the* ! *option is available with Windows Installer version 2.0 and works only with basic UI. It is not valid with full UI.***
/? *or* **/h**		Displays copyright information for Windows Installer.

For more Windows Installer command parameters see http://msdn.microsoft.com/library/default.asp?url=/library/en-us/msi/setup/command_line_options.asp

Examples:

To perform a typical installation without a user interface and to a non-default installation location, the following command can be used:

```
Msiexec.exe /i <setup location>\setup.msi /qn InstallDir=C:\Mysetup
```

To perform a typical installation with a basic user interface (progress bar) and to a non-default installation location, the following commandn can be used:

```
Msiexec.exe /i <setup location>/setup.msi /qb InstallDir=C:\Mysetup
```

To perform a complete installation without a user interface and to the default installation location, the following command can be used:

```
Msiexec.exe /i <setup location>\setup.msi /qn ADDLOCAL=All
```

To perform a custom installation without a user interface and to the default installation location, the following command can be used:

```
Msiexec.exe /i <setup location>\setup.msi /qn
ADDLOCAL=<feature1>,<feature2>,....
```

The features available to install are specified with the ADDLOCAL parameter. ArcGIS Engine has the following installation features:

Feature	Descriptive Feature Name	Description
ArcEngine	ArcGIS Engine	ArcGIS Engine
JavaRuntime	ArcGIS Engine Java Runtime	Java Assemblies
DotNetRuntime	ArcGIS Engine .Net Runtime	.Net Assemblies

To perform a custom installation without a user interface consisting of the ArcGIS Engine and .NET installation features, the following command can be used:

```
Msiexec.exe /i <setup location>\setup.msi /qn
ADDLOCAL=ArcEngine,DotNetRuntime
```

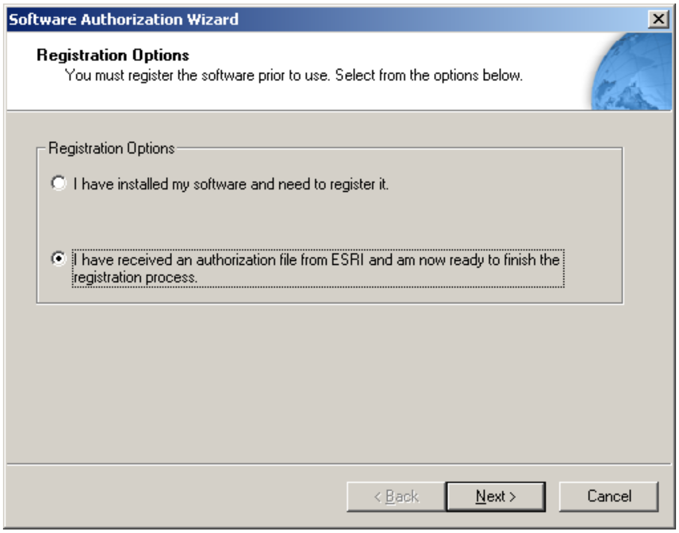
DEPLOYMENT GUIDELINES

- You must not uninstall ArcGIS Engine Runtime during the uninstallation of the your developed application.
- You must recommend that the user manually uninstall ArcGIS Engine Runtime, only if the user knows there are no third-party applications using it. NOTE: ArcGIS Engine Runtime should be uninstalled using the Control Panel, not by deleting files from disk.
- The ArcGIS Engine Runtime cannot be included in an authored MSI file (nested MSI setup).
- Your application setup program should not launch the ArcGIS Engine Runtime setup if it is already installed on the machine.
- You cannot redistribute individual Engine Runtime files; the deployment methods discussed in this chapter are the only means of deploying ArcGIS Engine Runtime files.

The final step in developing and deploying ArcGIS Engine applications is to ensure that all client machines have the correct license configuration to support your ArcGIS Engine application. This section details the various ways end users and developers can “authorize” the ArcGIS Engine Runtime components on client systems.

License initialization must be built into your application. For more information, see the earlier section 'Standalone executable license initialization'.

Software Authorization is the process of unlocking the underlying ArcGIS Engine Runtime software components. As a developer, you did this yourself when you installed and set up the ArcGIS Engine Developer Kit. Once you had installed the software, a Software Authorization wizard opened. It asked that you navigate to the authorization file (.ecp) that had been issued to you when you registered the product. Only after the authorization file was read and accepted, were you able to design and run applications that use ArcGIS Engine components. All deployed applications must be authorized in a similar manner, although there a number of different ways to achieve "authorization."



Software Authorization Wizard

Registration Options

You must register the software prior to use. Select from the options below.

Registration Options

○ I have installed my software and need to register it.

● I have received an authorization file from ESRI and am now ready to finish the registration process.

The Software Authorization Wizard opens after installing ArcGIS Engine Developer Kit. However, installations of ArcGIS Engine Runtime do not trigger the Software Authorization Wizard to start automatically.

As discussed earlier in this chapter, every application you build and deploy must first initialize itself with a suitable license. The store of 'suitable' licenses that your application initializes itself against are contained within the software authorization or keycode file, whichever is applicable, on the client machine or network. If your application attempts to initialize against a license that is not contained in the authorization file or, if all instances of the needed license have been checked out, then your application will not be able to run.

You, as the developer, must think in advance about how your clients will acquire and access an authorization or keycode file suitable to run your application. Your clients may fall into three categories:

- Licensed ArcGIS Desktop users who have access to the license features that your application uses.
- Those that will acquire the ArcGIS Engine Runtime software and/or its authorizations directly from ESRI.
- Those who will receive the ArcGIS Engine Runtime software and authorizations packaged within your application and have no direct contact with ESRI.

The following sections discuss the software authorization process for each of these three user types.

ARCGIS DESKTOP USERS

If your client is a licensed ArcGIS Desktop user, you and your client would go through the following process to install and run an application that you built:

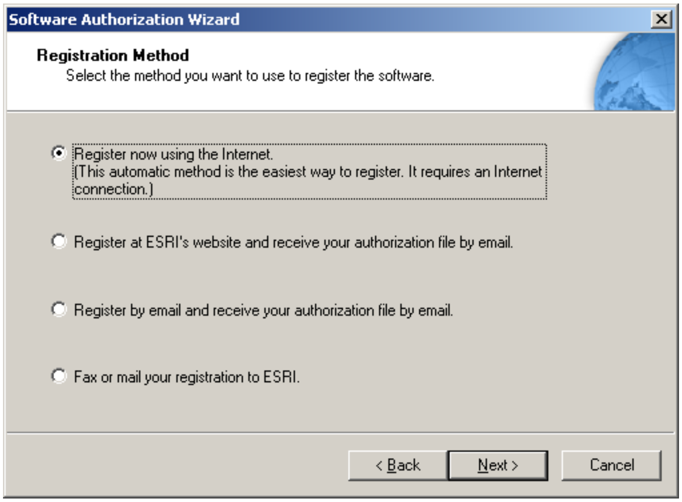
1. You review and confirm licensing requirements of your application—ArcView, ArcEditor, or ArcInfo (single use or concurrent) along with any necessary extensions.
2. Your client confirms that they have the applicable ArcGIS Desktop authorization or keycode files available for use with your application, as determined in the previous step.
3. You or your client installs your custom ArcGIS Engine application.
4. Upon application start up, it initializes and checks out an available license from the client's previously existing authorization or keycode file.

END USER LICENSES ARCGIS RUNTIME WITH ESRI

The second type of end user purchases and/or authorizes the ArcGIS Engine Runtime software themselves. You and your client would go through the following process to install and run an application that you built:

1. You review and confirm licensing required by your application.
2. Your client purchases ArcGIS Engine Runtime and any needed options (3D, GeoDatabase, Spatial, StreetMap, etcetera), as determined in the previous step.
3. Your client registers the ArcGIS Engine product, and options if necessary, with ESRI (http://www.service.esri.com).
4. Your client receives an authorization file (.ecp) from ESRI and saves it to their computer.
5. Your client installs the ArcGIS Engine Runtime software.
6. Once the installation is complete, your client navigates to the \ArcGIS\bin folder and runs the SoftwareAuthorization.exe it contains.
7. When asked by the Software Authorization wizard, your client navigates to the location of their authorization file.
8. You or your client installs your custom ArcGIS Engine application.
9. Upon application start up, it initializes and checks out an available license from the client's authorization file.

If your client does not register and receive their authorization file in advance, the Software Authorization wizard directs them to do so.



Software Authorization Wizard

Registration Method

Select the method you want to use to register the software.

● Register now using the Internet.
(This automatic method is the easiest way to register. It requires an Internet connection.)

○ Register at ESRI's website and receive your authorization file by email.

○ Register by email and receive your authorization file by email.

○ Fax or mail your registration to ESRI.

Although the redistribution of authorization files within your application is documented here, there are restrictions upon its use:

• If your application will be used solely within your organization, you can redistribute in this manner. However, you cannot distribute it in excess of the number of licenses you have purchased.

• If the application will be used or sold to a third party, use of a 'redistributable' authorization file violates the standard ESRI Software Master License Agreement and an individual contract must be negotiated. Contact the ESRI Business Partner Group or your International Distributor for information on such licensing.

For additional information on your right to use and deploy ArcGIS Engine applications, see footnotes 12 and 23 of the ESRI Software Master License Agreement.

To run the SoftwareAuthorization tool from within your application's installation program, use the following arguments:
/LIF <filename> /S

The /S triggers the tool to run silently with no user interface displaying.

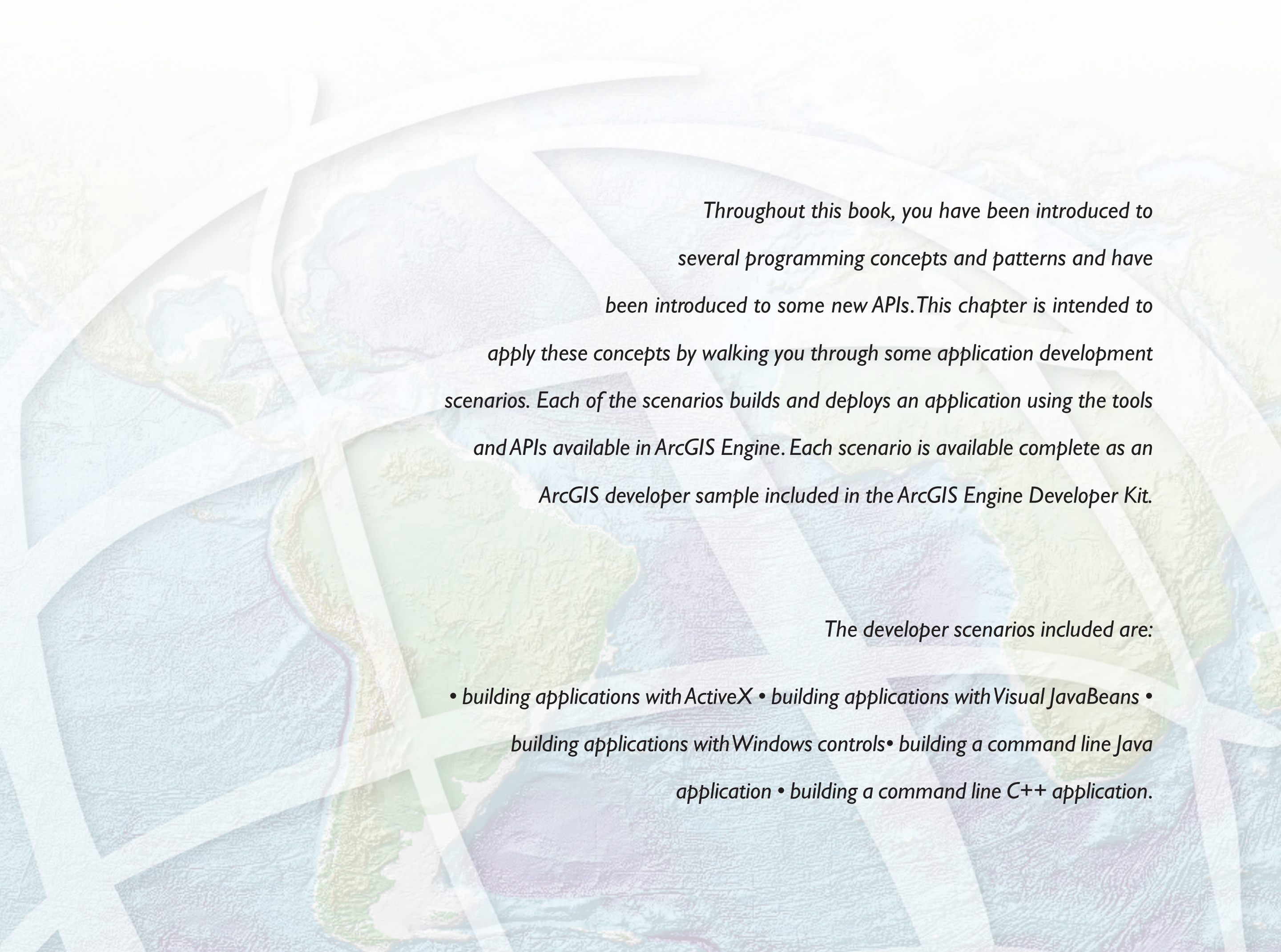
See the developer help for more information on the IAuthorizeLicense *interface.*

The final type of end user has no direct contact with ESRI. Instead your application calls the SoftwareAuthorization.exe or the *IAuthorizeLicense* object, contained in your installation program or application, to unlock the functionality of ArcGIS Engine. This would require that you 'hardcode' the authorization key code into your program. The advantage of this method is that the software will be authorized silently and does not require prompting your user for any registration information. In this case, you and your client would go through the following process to install and run an application that you built:

1. You review and confirm licensing required by your application.
2. You purchase the necessary redistributable ArcGIS Engine Runtime product and any needed options (3D, GeoDatabase, Spatial, etcetera), as determined in the previous step.
3. You register the ArcGIS Engine product, and options if necessary, with ESRI (http://www.service.esri.com).
4. You receive a redistributable authoriztion file (.ecp) and add its features into the code for your application.
5. Your client installs your custom-built ArcGIS Engine application. This
   a. installs ArcGIS Engine Runtime software, and
   b. automatically runs the SoftwareAuthorization.exe in \ArcGIS\bin\ or uses the *IAuthorizeLicense* object.
6. Upon application start up, it initializes and checks out an available license from the client's authorization file.

6

Developer scenarios



Throughout this book, you have been introduced to several programming concepts and patterns and have been introduced to some new APIs.This chapter is intended to apply these concepts by walking you through some application development scenarios. Each of the scenarios builds and deploys an application using the tools and APIs available in ArcGIS Engine. Each scenario is available complete as an ArcGIS developer sample included in the ArcGIS Engine Developer Kit.

The developer scenarios included are:

• building applications with ActiveX • building applications with Visual JavaBeans • building applications with Windows controls• building a command line Java application • building a command line C++ application.

This walkthrough is for developers who want to build and deploy an application using ActiveX. It describes the process of building and deploying an application using the ArcGIS Controls.

Rather than walk through this scenario, you can get the completed application from the samples installation location. The sample is installed as part of the ArcGIS developer samples.

The ArcGIS developer samples are not included in the 'typical' installation of the ArcGIS Engine Developer Kit. If you do not have them installed, rerun the Developer Kit install wizard, choose Custom or Modify, and select the samples feature under Software Developer Kit.

You can find this sample in

```
<install_location>\DeveloperKit\Samples\Developer_Guide_Scenarios\
ArcGIS_Engine\Building_an_ArcGIS_Control_Application\Map_Viewer
```

PROJECT DESCRIPTION

The goal of this scenario is to demonstrate and familiarize you with the steps required to develop and deploy a GIS application using the standard ArcGIS Controls within a COM API. This scenario uses the *MapControl, PageLayoutControl, TOCControl,* and *ToolbarControl* as ActiveX Controls within the Microsoft Visual Basic 6.0 development environment. C++, Java, and .NET programmers should refer to the following scenarios available later in this chapter: Building a command line C++ application, Building applications with visual JavaBeans, Building a command line Java application, and Building applications with Windows Controls.

This scenario demonstrates the steps required to create a GIS application for viewing map documents pre-authored using ArcMap, an ArcGIS Desktop application. The scenario covers the following techniques:

- Loading and embedding the ArcGIS Controls in Microsoft Visual Basic 6.0.
- Loading pre-authored map documents into the *MapControl* and *PageLayoutControl.*
- Setting *ToolbarControl* and *TOCControl* buddy controls.
- Handling form resize.
- Adding Control commands and tools to the *ToolbarControl.*
- Creating popup menus.
- Managing label editing in the *TOCControl.*
- Drawing shapes on the *MapControl.*
- Creating a custom tool to work with the *MapControl, PageLayoutControl,* and *ToolbarControl.*
- Customizing the *ToolbarControl.*
- Deploying the application onto a Windows operating system.

CONCEPTS

ActiveX is another term for a Microsoft Component Object Model (COM) object. All of ArcObjects is based on COM, and the ArcGIS Controls are COM objects.

This scenario is implemented using the Microsoft Visual Basic 6.0 development environment and uses the ArcGIS Controls as ActiveX components. ActiveX refers to a set of technologies that enables software components written in different languages to work together in a networked environment. Each ActiveX ArcGIS control has events, properties, and methods that can be accessed once the control is embedded within an ActiveX container, like a Visual Basic form. The objects and functionality within each control can be combined with other ESRI ArcObjects and custom controls to create customized end user applications.

The scenario could have been written in any other COM development environment that fully supports ActiveX, including Microsoft Visual C++, Borland Delphi, Sybase PowerBuilder, and Microsoft Visual Basic for Applications (VBA). Visual Basic, while not providing all the functionality of a development environment such as Visual C++, was chosen because it appeals to a wider audience. Whichever development environment you use, your future success with the ArcGIS Controls depends on your skill in both the programming environment and ArcObjects.

The *MapControl, PageLayoutControl, TOCControl,* and *ToolbarControl* are used in this scenario to provide the user interface of the application. The ArcGIS Controls are used in conjunction with other ArcObjects and Control commands by the developer to create a GIS viewing application.

DESIGN

The scenario has been designed firstly to highlight how the ArcGIS Controls interact with each other and secondly to expose a part of each ArcGIS Controls object model to the developer.

Each ActiveX ArcGIS control has a set of property pages that can be accessed once the control is embedded within an ActiveX container. These property pages provide shortcuts to a selection of a control's properties and methods and allow a developer to build an application without writing any code. This scenario does not use the property pages, but rather builds up the application programmatically. For further information about the property pages, refer to *ArcGIS Developer Help*.

REQUIREMENTS

To successfully follow this scenario you need the following (the requirements for deployment are covered later in the Deployment section):

- An installation of the ArcGIS Engine Developer Kit with an authorization file enabling it for development use.
- An installation of the Microsoft Visual Basic 6.0 development environment and an appropriate license.
- Familiarity with Microsoft Windows operating systems and a working knowledge of Microsoft Visual Basic 6.0. While the scenario provides some information about how to use the ArcGIS Controls in Microsoft Visual Basic 6.0, it is not a substitute for training in the development environment.
- While no experience with other ESRI software is required, previous experience with ArcObjects and a basic understanding of ArcGIS applications, such as ArcMap and ArcCatalog, is advantageous.
- Access to the sample data and code that comes with this scenario. This is located at

  `<install_location>\DeveloperKit\Samples\Developer_Guide_Scenarios\ArcGIS_Engine\Building_an_ArcGIS_Control_Application\Map_Viewer`

The ArcGIS developer samples are not included in the 'typical' installation of the ArcGIS Engine Developer Kit. If you do not have them installed, rerun the Developer Kit install wizard, choose Custom or Modify, and select the samples feature under Software Developer Kit.

The controls and libraries used in this scenario are as follows:

- MapControl
- PageLayoutControl
- Carto Object Library
- Display Object Library
- Geometry Object Library
- TOCControl
- ToolbarControl
- System Object Library
- SystemUI Object Library

In Visual Basic, these control and library names are prefixed with 'esri'.

IMPLEMENTATION

The implementation below provides you with all the code you will need to successfully complete the scenario. It does not provide step-by-step instructions to develop applications in Visual Basic 6.0, as it assumes that you have a working knowledge of the development environment already.

Loading the ArcGIS Controls

Before you start to program your application, the ArcGIS Controls and the other ArcGIS Engine library references that the application will use should be loaded into the development environment.

1. Start Visual Basic and create a new ‘Standard EXE’ project from the New project dialog box.
2. Click the Project menu and choose Components.
3. In the Components dialog box, check ‘ESRI MapControl’, ‘ESRI PageLayoutControl’, ‘ESRI TOCControl’, and ‘ESRI ToolbarControl’. Click OK.

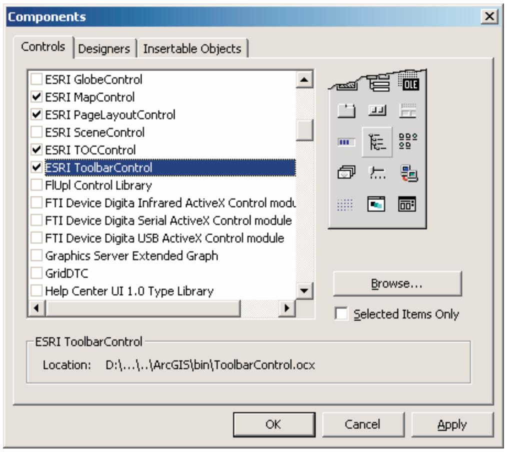
The ‘ESRI Automatic’ References Visual Basic Add-In can be used to quickly select and reference the ArcGIS Controls and other ArcGIS Engine libraries that you frequently use in Visual Basic 6.0. To load the Add-In select ‘Add-In Manager…’ from the ‘Add-Ins’ menu, select ‘ESRI Automatic References’ and check the load behaviour check boxes. To then display the Add-In select ‘ESRI Automatic References’ from the ‘Add-Ins’ menu.

ESRIMapControl

ESRIPageLayoutControl

ESRITOCControl

ESRIToolbarControl



Components

☐ ESRI GlobeControl
☑ ESRI MapControl
☑ ESRI PageLayoutControl
☐ ESRI SceneControl
☑ ESRI TOCControl
☑ ESRI ToolbarControl
☐ FlUpl Control Library
☐ FTI Device Digita Infrared ActiveX Control modu
☐ FTI Device Digita Serial ActiveX Control module
☐ FTI Device Digita USB ActiveX Control module
☐ Graphics Server Extended Graph
☐ GridDTC
☐ Help Center UI 1.0 Type Library

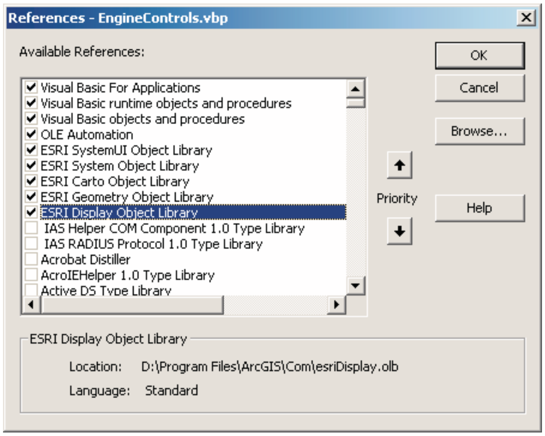
☐ Selected Items Only

ESRI ToolbarControl

Location: D:\...\..\ArcGIS\bin\ToolbarControl.ocx

The controls will now appear in the Visual Basic toolbox.

4. Click on the Project menu again and choose References.
5. In the References dialog box, check 'ESRI Carto Object Library', 'ESRI Display Object Library', 'ESRI Geometry Object Library', 'ESRI System Object Library', and 'ESRI SystemUI Object Library'. Click OK.


References - EngineControls.vbp

Available References:

☑ Visual Basic For Applications
☑ Visual Basic runtime objects and procedures
☑ Visual Basic objects and procedures
☑ OLE Automation
☑ ESRI SystemUI Object Library
☑ ESRI System Object Library
☑ ESRI Carto Object Library
☑ ESRI Geometry Object Library
☑ ESRI Display Object Library
□ IAS Helper COM Component 1.0 Type Library
□ IAS RADIUS Protocol 1.0 Type Library
□ Acrobat Distiller
□ AcroIEHelper 1.0 Type Library
□ Active DS Type Library

ESRI Display Object Library

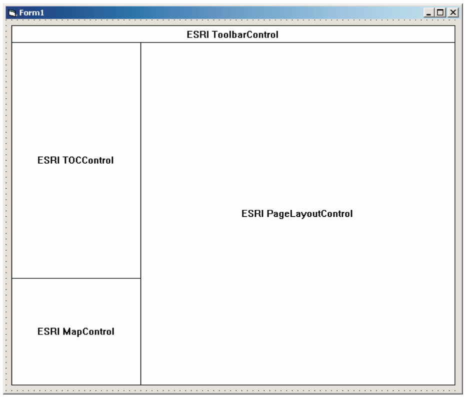
Location: D:\Program Files\ArcGIS\Com\esriDisplay.olb

Language: Standard

Embedding the ArcGIS Controls in a container

Before you can access each control's properties, methods, and events, each control needs embedding within an ActiveX container. Once the controls are embedded within the form, they will shape the application's user interface.

1. Open the Visual Basic Form.
2. Double-click the MapControl icon in the Visual Basic toolbox to add a MapControl onto a form.
3. Repeat to add the *PageLayoutControl*, *TOCControl*, and *ToolbarControl*.
4. Resize and reposition each control on the form as shown.



Loading Map Documents into the PageLayoutControl and MapControl

Individual data layers or map documents pre-authored using ArcMap, an ArcGIS Desktop application, can be loaded into the *MapControl* and *PageLayoutControl*. You can either load the sample map document provided or you can load in your own map document. Later you will add a dialog box to browse to a map document.

1. Double-click on the form to display the code window.
2. Select the Form_Load event and enter the following code (if you are using your own map document, substitute the filename).

```
Private Sub Form_Load()

  'Check and load a pre-authored map document into the PageLayoutControl
using relative paths.
  Dim sFileName As String
  sFileName = "..\..\..\..\..\..\Data\ArcGIS_Engine_Developer_Guide\Gulf
of St. Lawrence.mxd"
  If PageLayoutControl1.CheckMxFile(sFileName) Then
    PageLayoutControl1.LoadMxFile sFileName
  End If

End Sub
```

3. Select the PageLayoutControl_ OnPageLayoutReplaced event and enter the following code to load the same map document into the MapControl. The OnPageLayoutReplaced event will be triggered whenever a document is loaded into the *PageLayoutControl*.

```
Private Sub PageLayoutControl1_OnPageLayoutReplaced(ByVal newPageLayout
  As Variant)

  'Load the same pre-authored map document into the MapControl.
  MapControl1.LoadMxFile PageLayoutControl1.DocumentFilename
  'Set the extent of the MapControl to the full extent of the data.
  MapControl1.Extent = MapControl1.FullExtent

End Sub
```

Setting the TOCControl and ToolbarControl Buddy Controls

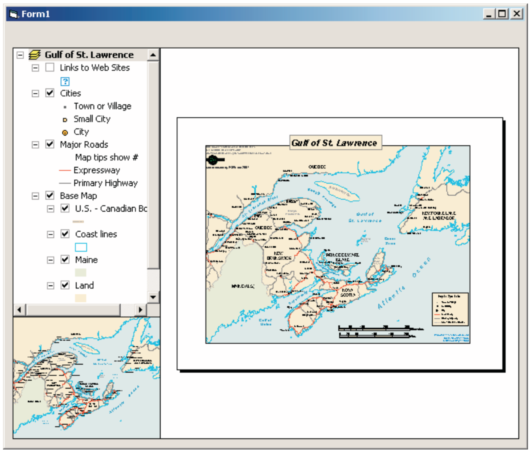
For the purpose of this application the *TOCControl* and *ToolbarControl* will work in conjunction with the *PageLayoutControl* rather than the *MapControl*. To do this the *PageLayoutControl* must be set as the buddy control. The *TOCControl* uses the buddy's ActiveView to populate itself with maps, layers, and symbols, while any command, tool, or menu items present on the *ToolbarControl* will interact with the buddy control's display.

1. Double-click on the form to display the code window.

2. Select the Form_Load event and enter the following after the load document code:

```
Private Sub Form_Load()

  'Check and load a pre-authored map document into the PageLayoutControl
using relative paths.
  Dim sFileName As String
  sFileName = "..\..\..\..\..\..\Data\ArcGIS_Engine_Developer_Guide\Gulf
of St. Lawrence.mxd"
  If PageLayoutControl1.CheckMxFile(sFileName) Then
    PageLayoutControl1.LoadMxFile sFileName
  End If

  'Set buddy controls.
  TOCControl1.SetBuddyControl PageLayoutControl1
  ToolbarControl1.SetBuddyControl PageLayoutControl1
End Sub
```

3. Run the application. The map document has been loaded into the *PageLayoutControl*, and the *TOCControl* lists the data layers in the map document. Use the *TOCControl* to toggle layer visibility by checking and unchecking the boxes. By default, the focus map of the map document is loaded into the *MapControl*. At this point the *ToolbarControl* is empty because no commands have been added to it. Try resizing the form, and note that the control's do not change size.



Handling Form Resize

When the form is resized at run time, the *PageLayoutControl* and *MapControl* do not automatically resize themselves. To resize the controls so that they always fill the extent of the form, you must respond to the Form_Resize event. If the *PageLayoutControl* or *MapControl* contain a lot of data, redrawing this data during the Form_Resize can be costly. To increase performance you can suppress the data redraw until the resizing is complete. During the resize a stretched bitmap will be drawn instead.

1. Double-click on the form to display the code window.
2. Select the Form_Resize event and enter the following code:

```
Private Sub Form_Resize()

  Dim dWidth As Double, dheight As Double, dMargin As Double

  'Set the margin size.
  dMargin = TOCControl1.Left

  'Resize the PageLayoutControl.
  dheight = Form1.ScaleHeight - PageLayoutControl1.Top - dMargin
  If dheight > 0 Then PageLayoutControl1.Height = dheight
  dWidth = Form1.ScaleWidth - TOCControl1.Width - (dMargin * 2)
  If dWidth > 0 Then PageLayoutControl1.Width = dWidth

  'Resize the MapControl.
  dheight = Form1.ScaleHeight - MapControl1.Top - dMargin
  If dheight > 0 Then MapControl1.Height = dheight

End Sub
```

3. Select the Form_Load event and add the following code at the end of the procedure:

```
Private Sub Form_Load()

  'Set buddy controls…

  'Suppress drawing while resizing
  MapControl1.SuppressResizeDrawing False, Form1.hWnd
  PageLayoutControl1.SuppressResizeDrawing False, Form1.hWnd

End Sub
```

4. Run the application and try resizing the form.

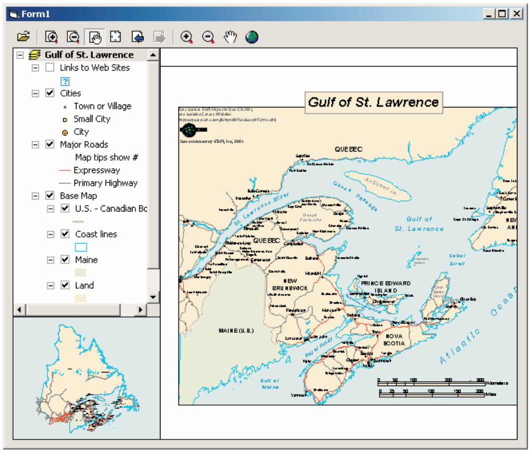
Adding Commands to the ToolbarControl

The ArcGIS Engine comes with more than 120 commands and tools that work with the *MapControl*, the *PageLayoutControl*, and the *ToolbarControl* directly. These commands and tools provide you with a lot of frequently used GIS functionality for map navigation, graphics management, and feature selection. You will now add some of these commands and tools to your application.

1. Double-click on the form to display the code window.
2. Select the Form_Load event and add the following code before the load document code:

```
Private Sub Form_Load()

  Dim sProgID As String

  'Add generic commands.
  sProgID = "esriControlTools.ControlsOpenDocCommand"
  ToolbarControl1.AddItem sProgID, , , False, , esriCommandStyleIconOnly
  'Add PageLayout navigation commands.
  sProgID = "esriControlTools.ControlsPageZoomInTool"
  ToolbarControl1.AddItem sProgID, , , True, , esriCommandStyleIconOnly
  sProgID = "esriControlTools.ControlsPageZoomOutTool"
  ToolbarControl1.AddItem sProgID, , , False, , esriCommandStyleIconOnly
  sProgID = "esriControlTools.ControlsPagePanTool"
  ToolbarControl1.AddItem sProgID, , , False, , esriCommandStyleIconOnly
  sProgID = "esriControlTools.ControlsPageZoomWholePageCommand"
  ToolbarControl1.AddItem sProgID, , , False, , esriCommandStyleIconOnly
  sProgID = "esriControlTools.ControlsPageZoomPageToLastExtentBackCommand"
  ToolbarControl1.AddItem sProgID, , , False, , esriCommandStyleIconOnly
  sProgID = "esriControlTools.ControlsPageZoomPageToLastExtentForwardCommand"
  ToolbarControl1.AddItem sProgID, , , False, , esriCommandStyleIconOnly
  'Add Map naviagtion commands.
  sProgID = "esriControlTools.ControlsMapZoomInTool"
  ToolbarControl1.AddItem sProgID, , , True, , esriCommandStyleIconOnly
  sProgID = "esriControlTools.ControlsMapZoomOutTool"
  ToolbarControl1.AddItem sProgID, , , False, , esriCommandStyleIconOnly
  sProgID = "esriControlTools.ControlsMapPanTool"
  ToolbarControl1.AddItem sProgID, , , False, , esriCommandStyleIconOnly
  sProgID = "esriControlTools.ControlsMapFullExtentCommand"
  ToolbarControl1.AddItem sProgID, , , False, , esriCommandStyleIconOnly

  'Load a pre-authored…

End Sub
```

3. Run the application. The *ToolbarControl* now contains ArcGIS Engine commands and tools that you can use to navigate the map document loaded into the *PageLayoutControl*. Use the page layout commands to navigate around the actual page layout and the map commands to navigate around the data present in the data frames. Use the open document command to browse and load other map documents.



Creating a Popup Menu for the PageLayoutControl

As well as adding Control commands to the *ToolbarControl* to work with the buddy control, as in the previous step, you can also create popup menus from the Control commands. You will add a popup menu that works with the *PageLayoutControl* to your application. The popup menu will display whenever the right mouse button is used on the display area of the *PageLayoutControl*.

1. Add the following code to the general declarations area of the form:

```
Option Explicit
Private m_pToolbarMenu As IToolbarMenu        'The popup menu
```

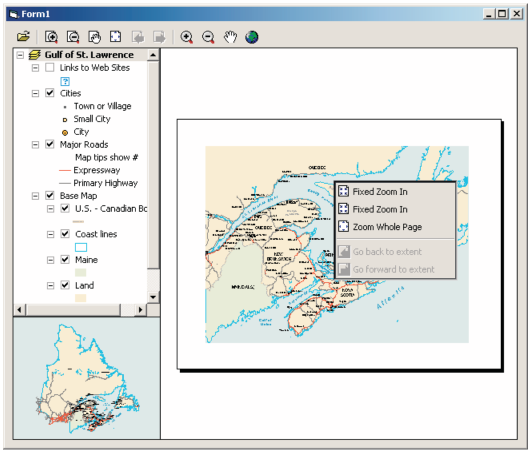
2. Add the following code to the Form_Load event after the code adding the commands to the *ToolbarControl* but before the load document code.

```
Private Sub Form_Load()

  'Add Map naviagtion commands…

  'Create a new ToolbarMenu.
  Set m_pToolbarMenu = New ToolbarMenu
  'Share the ToolbarControl's command pool.
  Set m_pToolbarMenu.CommandPool = ToolbarControl1.CommandPool
  'Add commands to the ToolbarMenu.
  sProgID = "esriControlTools.ControlsPageZoomInFixedCommand"
  m_pToolbarMenu.AddItem sProgID, , , False, esriCommandStyleIconAndText
  sProgID = "esriControlTools.ControlsPageZoomOutFixedCommand"
  m_pToolbarMenu.AddItem sProgID, , , False, esriCommandStyleIconAndText
  sProgID = "esriControlTools.ControlsPageZoomWholePageCommand"
  m_pToolbarMenu.AddItem sProgID, , , False, esriCommandStyleIconAndText
  sProgID = "esriControlTools.ControlsPageZoomPageToLastExtentBackCommand"
  m_pToolbarMenu.AddItem sProgID, , , True, esriCommandStyleIconAndText
  sProgID = "esriControlTools.ControlsPageZoomPageToLastExtentForwardCommand"
  m_pToolbarMenu.AddItem sProgID, , , False, esriCommandStyleIconAndText
  'Set the hook to the PageLayoutControl.
  m_pToolbarMenu.SetHook PageLayoutControl1

  'Load a pre-authored…

End Sub
```

3. Add the following code to the PageLayoutControl1_OnMouseDown event.

```
Private Sub PageLayoutControl1_OnMouseDown(ByVal button As Long, ByVal
  shift As Long, ByVal x As Long, ByVal y As Long, ByVal pageX As
  Double, ByVal pageY As Double)

  'Popup the ToolbarMenu
  If button = vbRightButton Then
    m_pToolbarMenu.PopupMenu x, y, PageLayoutControl1.hWnd
  End If

End Sub
```

4. Run the application. Right-click on the *PageLayoutControl's* display area to display the popup menu, and navigate around the page layout.



Controlling Label Editing in the TOCControl

By default, the *TOCControl* allows users to automatically toggle the visibility of layers and to change map and layer names as they appear in the table of contents. You will add code to prevent users from editing a name and replacing it with an empty string.

1. Add the following code to the beginning of the Form_Load event to trigger the *TOCControl* label editing events.

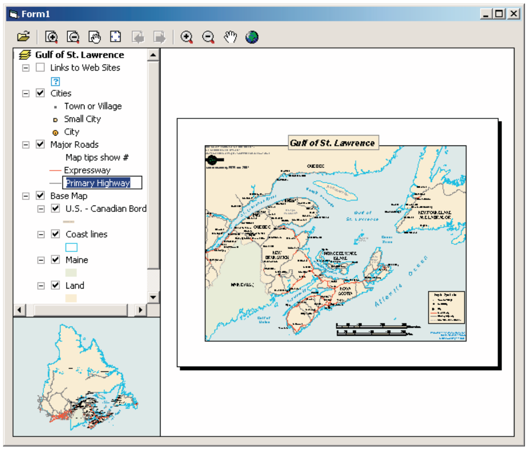
```
Private Sub Form_Load()

  'Set label editing to manual.
  TOCControl1.LabelEdit = esriTOCControlManual

  'Add generic commands…

End Sub
```

2. Add the following code to the TOCControl1_OnEndLableEdit event.

```
Private Sub TOCControl1_OnEndLabelEdit(ByVal x As Long, ByVal y As Long,
  ByVal newLabel As String, pcanEdit As Boolean)

  'If the new label is an empty string, then prevent the edit.
  If newLabel = "" Then pcanEdit = False

End Sub
```

3. Run the application. To edit a map, layer, heading, or legend class label in the *TOCControl*, click on it once, and click on it a second time to invoke label

editing. Try replacing the label with an empty string. You can use the ESC key on the keyboard at any time during the edit to cancel it.



Drawing Shapes on the MapControl

You will now use the *MapControl* as an overview window and draw the current extent of the focus map within the *PageLayoutControl* on its display. As you navigate around the data within the data frame of the *PageLayoutControl*, you will see the *MapControl* overview window update.

Navigating around the focus map using the map navigation tools will change the extent of the focus map in the PageLayoutControl and cause the MapControl to update. Navigating around the page layout with the page layout navigation tools will change the extent of the page layout (not the extent of the focus map in the PageLayoutControl), so the MapControl will not update.

1. Add the following code to the general declarations area of the form.

```
Option Explicit

Private m_pToolbarMenu As IToolbarMenu
Private m_pEnvelope As IEnvelope    'The envelope drawn on the MapControl
Private m_pFillSymbol As ISimpleFillSymbol   'The symbol used to draw the
            'envelope on the MapControl
Private WithEvents m_pTransformEvents As DisplayTransformation
'The PageLayoutControl's focus map events
```

2. Create a new public sub routine called *CreateOverviewSymbol*. This is where you will create the symbol used in the *MapControl* to represent the extent of the data in the focus map of the *PageLayoutControl*. Add the following code to the sub routine:

```
Private Sub CreateOverviewSymbol()

 'Get the IRGBColor interface.
 Dim pColor As IRgbColor
 Set pColor = New RgbColor
 'Set the color properties.
 pColor.RGB = RGB(255, 0, 0)

 'Get the ILine symbol interface.
 Dim pOutline As ILineSymbol
 Set pOutline = New SimpleLineSymbol
```

```
  'Set the line symbol properties.
  pOutline.Width = 1.5
  pOutline.Color = pColor

  'Get the IFillSymbol interface.
  Set m_pFillSymbol = New SimpleFillSymbol
  'Set the fill symbol properties.
  m_pFillSymbol.Outline = pOutline
  m_pFillSymbol.Style = esriSFSHollow

End Sub
```

3. Call the *CreateOverviewSymbol* sub routine from the Form_Load event before the *TOCControl* label editing code.

```
Private Sub Form_Load()

  'Create symbol used on the MapControl.
  CreateOverviewSymbol

  'Set label editing to manual…

End if
```

4. The default event interface of the *PageLayoutControl* is the *IPageLayoutControlEvents*. These events do not tell you when the extent of the map within the data frame changes. To do this you will use the *ITransformEvents* interface of the *PageLayoutControl*'s focus map. Add the following code to the PageLayoutControl_OnPageLayoutReplaced event directly above the load document code.

```
Private Sub PageLayoutControl1_OnPageLayoutReplaced(ByVal newPageLayout
  As Variant)

  'Get the IActiveView of the focus map in the PageLayoutControl.
  Dim pActiveView As IActiveView
  Set pActiveView = PageLayoutControl1.ActiveView.FocusMap
  'Trap the ITranformEvents of the PageLayoutCntrol's focus map.
  Set m_pTransformEvents = pActiveView.ScreenDisplay.DisplayTransformation
  'Get the extent of the focus map.
  Set m_pEnvelope = pActiveView.Extent

  'Load the same pre-authored map document into the MapControl.
  MapControl1.LoadMxFile PageLayoutControl1.DocumentFilename
  'Set the extent of the MapControl to the full extent of the data.
  MapControl1.Extent = MapControl1.FullExtent

End Sub
```

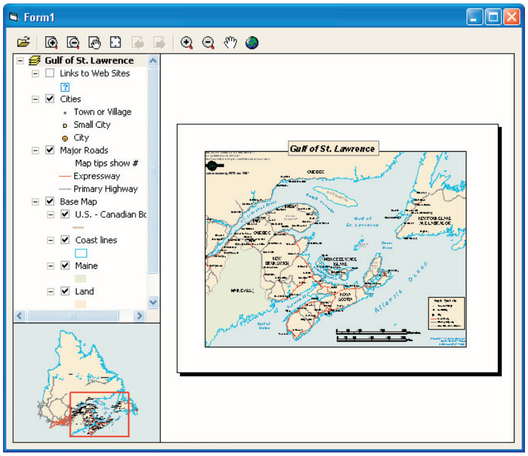
5. Add the following code to the m_pTransformEvents_VisibleBoundsUpdated event. This event is triggered whenever the extent of the map is changed and is used to set the envelope to the new visible bounds of the map. By refreshing the *MapControl* you force it to redraw the shape on its display.

```
Private Sub m_pTransformEvents_VisibleBoundsUpdated(ByVal sender As
  esriDisplay.IDisplayTransformation, ByVal sizeChanged As Boolean)

  'Set the extent to the new visible extent.
  Set m_pEnvelope = sender.VisibleBounds
  'Refresh the MapControl's foreground phase.
  MapControl1.Refresh esriViewForeground

End Sub
```

6. Add the following code to the MapControl_OnAfterDraw event to draw the envelope with the symbol you created earlier onto the *MapControl*'s display.

```
Private Sub MapControl1_OnAfterDraw(ByVal display As Variant, ByVal
  viewDrawPhase As Long)

  If m_pEnvelope Is Nothing Then Exit Sub

  'If the foreground phase has drawn
  Dim pViewDrawPhase As esriViewDrawPhase
  pViewDrawPhase = viewDrawPhase
  If pViewDrawPhase = esriViewForeground Then
    'Draw the shape on the MapControl.
    MapControl1.DrawShape m_pEnvelope, m_pFillSymbol
  End If

End Sub
```

7. Run the application. Use the map navigation tools that you added earlier to change the extent of the focus map in the *PageLayoutControl*. The new extent is drawn on the *MapControl*.



Creating a Custom Tool

Creating custom commands and tools to work with the *MapControl* and *PageLayoutControl* is very much like creating commands for the ESRI ArcMap application that you may have done previously. You will create a custom tool that adds a text element containing today's date to the *PageLayoutControl* at the location of a mouse click. You will, however, create the command to work with the *MapControl* and *ToolbarControl* as well as the *PageLayoutControl*.

The code for this custom tool is available with the rest of this scenario's source code. If you want to use the custom command directly, rather than creating it yourself, go directly to Step 12.

The command class is implemented in a separate ActiveX DLL project rather than inside the ActiveX exe project, because the command will not become a COM class unless it is in a DLL.

1. Start Visual Basic and create a new ActiveX DLL project from the New project dialog box.
2. Name the project EngineScenarioCommands.
3. Click the Project menu again and choose References.
4. In the References dialog box, check ESRI Carto Object Library, ESRI Control Commands Object Library, ESRI Display Object Library, ESRI Geometry Object Library, ESRI System Object Library, and ESRI SystemUI Object Library. Click OK.
5. Add a class module to the project and name it AddDateTool.

This scenario's source code is located at <install_location>\DeveloperKit\Samples\Developer_Guide_Scenarios\ArcGIS_Engine\Building_an_ArcGIS_Control_Application\Map_Viewer.

6. Add in the ControlCommands.res Visual Basic resource file from its location in this sample's source code. To do this you will need the VB6 Resource Editor Add-in.
7. Add the following code to the general declarations area of the AddDateTool class module.

```
Option Explicit

Implements ICommand
Implements ITool

Private m_pHookHelper As IHookHelper
Private m_pBitmap As IPictureDisp
```

8. Add the following code to the Class_Initialize and Class_Terminate methods.

```
Private Sub Class_Initialize()
  'Load resources
  Set m_pBitmap = LoadResPicture("Date", vbResBitmap)
  'Create a HookHelper
  Set m_pHookHelper = New HookHelper
End Sub

Private Sub Class_Terminate()
  'Clear variables
  Set m_pHookHelper = Nothing
  Set m_pBitmap = Nothing
End Sub
```

9. You now need to stub out all of the properties and events of the *ICommand* interface, even if you are not going to use some of these. Add the following code to the *ICommand* properties and methods.

```
Private Property Get ICommand_Bitmap() As esriSystem.OLE_HANDLE
  ICommand_Bitmap = m_pBitmap
End Property

Private Property Get ICommand_Caption() As String
  ICommand_Caption = "Add Date"
End Property

Private Property Get ICommand_Category() As String
  ICommand_Category = "CustomCommands"
End Property

Private Property Get ICommand_Checked() As Boolean
  ICommand_Checked = False
End Property

Private Property Get ICommand_Enabled() As Boolean
  If Not m_pHookHelper.ActiveView Is Nothing Then
    ICommand_Enabled = True
  Else
    ICommand_Enabled = False
  End If
End Property

Private Property Get ICommand_HelpContextID() As Long
  'Not implemented
End Property

Private Property Get ICommand_HelpFile() As String
  'Not implemented
End Property

Private Property Get ICommand_Message() As String
  ICommand_Message = "Adds a date element to the page layout"
End Property

Private Property Get ICommand_Name() As String
  ICommand_Name = "CustomCommands_Add Date"
End Property

Private Sub ICommand_OnClick()
  'Not implemented
End Sub

Private Sub ICommand_OnCreate(ByVal Hook As Object)
  Set m_pHookHelper.Hook = Hook
End Sub
```

```
Private Property Get ICommand_Tooltip() As String
  ICommand_Tooltip = "Add date"
End Property
```

The ICommand_OnCreate event is passed a handle or hook to the application that the command will work with. In this case it can be a MapControl, PageLayoutControl, or ToolbarControl. Rather than adding code into the OnCreate event to determine the type of hook that is being passed to the command, you will use the HookHelper to handle this. A command or tool needs to know how to handle the hook it gets passed, so a check is needed to determine the type of ArcGIS Control that has been passed. The HookHelper is used to hold the hook and return the ActiveView regardless of the type of hook (in this case either a MapControl, PageLayoutControl, or ToolbarControl).

10. You now need to stub out all of the properties and events of the *ITool* interface, even if you are not going to use some of these. Add the following code to the *ITool* properties and methods:

```
Private Property Get ITool_Cursor() As esriSystem.OLE_HANDLE
  'Not implemented
End Property

Private Function ITool_Deactivate() As Boolean
  ITool_Deactivate = True
End Function

Private Function ITool_OnContextMenu(ByVal x As Long, ByVal y As Long) As
  Boolean
  'Not implemented
End Function

Private Sub ITool_OnDblClick()
  'Not implemented
End Sub

Private Sub ITool_OnKeyDown(ByVal keyCode As Long, ByVal shift As Long)
  'Not implemented
End Sub

Private Sub ITool_OnKeyUp(ByVal keyCode As Long, ByVal shift As Long)
  'Not implemented
End Sub

Private Sub ITool_OnMouseDown(ByVal button As Long, ByVal shift As Long,
  ByVal x As Long, ByVal y As Long)

  'Get the active view.
  Dim pActiveView As IActiveView
  Set pActiveView = m_pHookHelper.ActiveView

  'Create a new text element.
  Dim pTextElement As ITextElement
  Set pTextElement = New TextElement
  'Create a text symbol.
  Dim pTextSymbol As ITextSymbol
  Set pTextSymbol = New TextSymbol

  'Create a font.
  Dim pFont As stdole.StdFont
  Set pFont = New stdole.StdFont
  pFont.Name = "Arial"
  pFont.Bold = True
  pFont.Size = 25

  'Set the symbol properties.
  pTextSymbol.Font = pFont
  'Set the text element properties
  pTextElement.Symbol = pTextSymbol
  pTextElement.Text = Date

  'QI for IElement
  Dim pElement As IElement
  Set pElement = pTextElement
  'Create a page point.
  Dim pPoint As IPoint
  Set pPoint = pActiveView.ScreenDisplay.DisplayTransformation.ToMapPoint(x, y)
  'Set the elements geometry.
  pElement.Geometry = pPoint

  'Add the element to the graphics container.
  pActiveView.GraphicsContainer.AddElement pTextElement, 0
  'Refresh the graphics
  pActiveView.PartialRefresh esriViewGraphics, Nothing, Nothing

End Sub

Private Sub ITool_OnMouseMove(ByVal button As Long, ByVal shift As Long,
  ByVal x As Long, ByVal y As Long)
  'Not implemented
End Sub

Private Sub ITool_OnMouseUp(ByVal button As Long, ByVal shift As Long,
  ByVal x As Long, ByVal y As Long)
  'Not implemented
End Sub

Private Sub ITool_Refresh(ByVal hdc As esriSystem.OLE_HANDLE)
  'Not implemented
End Sub
```

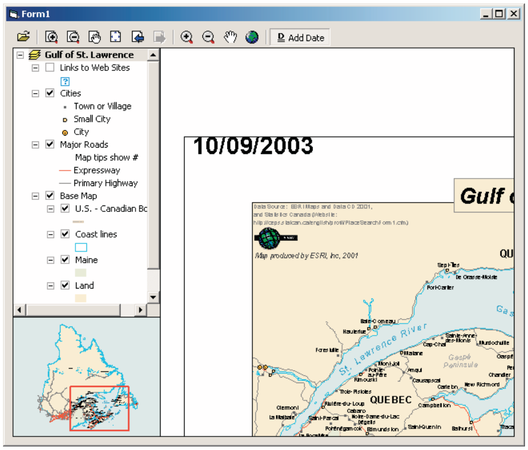
11. You now need to compile your project into an ActiveX DLL. Give it the name ControlCommands.dll.

12. Register the ControlCommands.dll.

13. In the Visual Basic Standard executable project that you created at the beginning of this scenario, select the Form_Load event and add the following code after the code to add the map navigation commands.

```
Private Sub Form_Load()

  'Add Map navigation commands…

  'Add custom date tool.
  sProgID = "EngineScenarioCommands.AddDateTool"
  ToolbarControl1.AddItem sProgID, , , True, , esriCommandStyleIconAndText

  'Create a new ToolbarMenu…

End Sub
```

14. Run the application and use the *AddDateTool* to add a text element to the *PageLayoutControl* containing today's date.



Customizing the ToolbarControl

As well as adding Controls commands and tools to the *ToolbarControl* in the Form_Load event, you can also add them by customizing the *ToolbarControl* and using the customize dialog box. To do this you will place the *ToolbarControl* in customize mode and display the customize dialog box.

1. Add the following code to the general declarations area of the form:

```
Option Explicit

Private m_pToolbarMenu As IToolbarMenu
Private m_pEnvelope As IEnvelope
Private m_pFillSymbol As IFillSymbol
Private WithEvents m_pTransformEvents As DisplayTransformation
Private WithEvents m_pCustomizeDialogEvents As CustomizeDialog
‘The customize dialog events
Private m_pCustomizeDialog As ICustomizeDialog
'The customize dialog box used by the ToolbarControl
```

2. Add a new sub routine called *CreateCustomizeDialog* and add the following code to it. This is where you will create the customize dialog box. Add the following code to the sub routine:

```
Private Sub CreateCustomizeDialog()

  Set m_pCustomizeDialog = New CustomizeDialog
  Set m_pCustomizeDialogEvents = m_pCustomizeDialog
  'Set the title.
  m_pCustomizeDialog.DialogTitle = "Customize ToolbarControl Items"
  'Show the 'Add from File' button.
  m_pCustomizeDialog.ShowAddFromFile = True
  'Set the ToolbarControl that new items will be added to.
  Set m_pCustomizeDialog.DoubleClickDestination = ToolbarControl1

End Sub
```

3. Call the *CreateCustomizeDialog* sub routine from the Form_Load event before the call to the *CreateOverviewSymbol* sub routine.

```
Private Sub Form_Load()

  'Create the customize dialog box for the ToolbarControl.
  CreateCustomizeDialog

  'Create symbol used on the MapControl…

End Sub
```

4. Add a check box to the Form and give it the name ‘chkCustomize’ and the caption ‘Customize’.

5. Add the following code to the chkCustomize_Click event.

```
Private Sub chkCustomize_Click()

  'Show or hide the customize dialog box.
  If chkCustomize.Value = 0 Then
    m_pCustomizeDialog.CloseDialog
  Else
    m_pCustomizeDialog.StartDialog ToolbarControl1.hWnd
  End If

End Sub
```

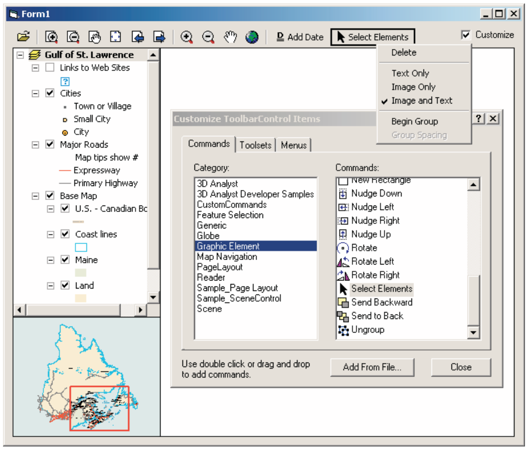
6. Add the following code to the m_pCustomizeDialogEvents_OnCloseDialog and m_pCustomizeDialogEvents_OnStartDialog events.

```
Private Sub m_pCustomizeDialogEvents_OnCloseDialog()

  ToolbarControl1.Customize = False
  chkCustomize.Value = 0

End Sub

Private Sub m_pCustomizeDialogEvents_OnStartDialog()

  ToolbarControl1.Customize = True

End Sub
```

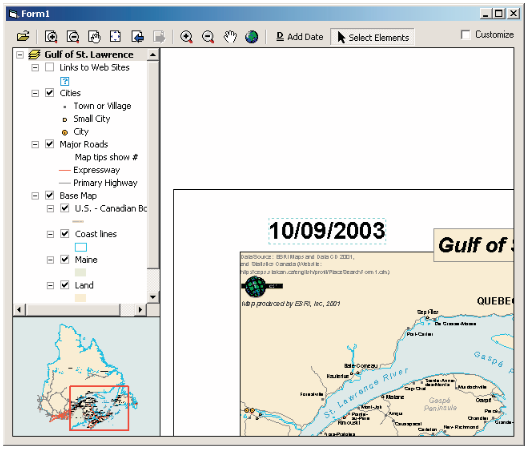
7. Run the application and check the customize box to put the *ToolbarControl* into customize mode and open the customize dialog box.

8. On the Commands tab select the Graphic Element category and double-click the Select Elements command to add it to the *ToolbarControl*. By right-clicking on an item on the *ToolbarControl*, you can adjust its appearance in terms of style and grouping.

In the customize dialog box the 'Add From File…' button could be used to browse to and select the Control Commands.dll file you compiled earlier. The AddDateTool will be added to the customize dialog box under the CustomCommands category.



9. Stop customizing the application. Use the select tool to move the text element containing today's date.



DEPLOYMENT

To successfully deploy this application onto another machine, the application must configure a license. First, it must check that the product license is available, and second it must initialize the license. If this license configuration fails, the application cannot run.

1. Add the following member variable to the general declarations area of the form:

```
Option Explicit

Private m_pAoInitialize As IAoInitialize      'The initialization object
```

2. Add the following code to the beginning of the Form_Load event:

```
Private Sub Form_Load()

  'Create a new AoInitialize object.
  Set m_pAoInitialize = New AoInitialize
  If m_pAoInitialize Is Nothing Then
    MsgBox "Unable to initialize. This application cannot run!"
    Unload Form1
    Exit Sub
  End If
  'Determine if the product is available.
  If m_pAoInitialize.IsProductCodeAvailable(esriLicenseProductCodeEngine)
  = esriLicenseAvailable Then
   If m_pAoInitialize.Initialize(esriLicenseProductCodeEngine) <>
  esriLicenseCheckedOut Then
      MsgBox "The initialization failed. This application cannot run!"
      Unload Form1
      Exit Sub
    End If
  Else
    MsgBox "The ArcGIS Engine product is unavailable. This application
  cannot run!"
    Unload Form1
    Exit Sub
  End If

End Sub
```

3. Add the following code to the Form_Unload event:

```
Private Sub Form_Unload(Cancel As Integer)

  'Shut down the AoInitilaize object
  m_pAoInitialize.Shutdown

End Sub
```

4. Compile the application into an executable.

When developing a stand-alone executable using ESRI ArcObjects, it is the responsibility of the application to check and configure the licensing options. The CoClass AoInitialize and the IAoInitialize interface it implements are designed to support license configuration. License initialization must be performed at application start time, before any ESRI ArcObjects functionality is accessed. Failure to do so will resolve in application errors.

To successfully deploy this application onto a user's machine:

- The application's executable and the DLL containing the custom command will need to be deployed onto the user's machine.
- The user's machine will need an installation of the ArcGIS Engine Runtime and a standard ArcGIS Engine license.

ADDITIONAL RESOURCES

The following resources may help you understand and apply the concepts and techniques presented in this scenario.

- Additional documentation available in the ArcGIS Engine Developer Kit including *ArcGIS Developer Help*, component help, object model diagrams, and samples to help you get started.
- ArcGIS Developer Online—Web site providing the most up-to-date information for ArcGIS Developers including updated samples and technical documents. Go to http://arcgisdeveloperonline.esri.com.
- ESRI online discussion forums—Web sites providing invaluable assistance from other ArcGIS developers. Go to http://support.esri.com and click the User Forums tab.
- Microsoft documentation on the Visual Basic 6 development environment.

Rather than walk through this scenario, you can get the completed application from the samples installation location. The sample is installed as part of the ArcGIS developer samples.

The ArcGIS developer samples are not included in the 'typical' installation of the ArcGIS Engine Developer Kit. If you do not have them installed, rerun the Developer Kit install wizard, choose Custom or Modify, and select the samples feature under Software Developer Kit.

This walkthrough is for developers who want to build and deploy an application using visual Java components. It describes the process of building and deploying an application using the visual JavaBeans available in the ArcGIS Engine Developer Kit.

You can find this sample in

```
<install_location>\DeveloperKit\Samples\Developer_Guide_Scenarios\
ArcGIS_Engine\Building_an_ArcGIS_Control_Application\Map_Viewer
```

PROJECT DESCRIPTION

This scenario demonstrates the steps required to create a GIS application for viewing map documents pre-authored using ArcMap, an ArcGIS Desktop application. The scenario covers the following techniques:

- Setting up the development environment
- Building a GUI using the visual components
- Loading map documents
- Adding commands to the toolbar
- 'Buddying up' the *ToolbarControl* and *TOCControl*
- Adding toolbar items to the *ToolbarControl*
- Creating popup menu using ToolbarMenu
- Controlling Label Editing in the *TOCControl* component
- Drawing an 'overview rectangle' on the *MapControl*
- Creating a custom tool
- Customizing the *ToolbarControl*
- Deploying using an executable JAR

CONCEPTS

The ArcGIS Engine Developer Kit provides reusable visual Java components corresponding to each ArcGIS Control. This developer scenario will show how these components can be embedded in a Java GUI to build a 'map viewer' application.

The visual components provided by the ArcGIS Engine Developer Kit are heavyweight AWT components that conform to the JavaBeans component architecture, allowing them to be used as drag and drop components for designing Java GUIs in JavaBean compatible IDEs. Each component has certain properties and methods and is capable of firing events. Internally, the Java components use JNI (Java Native Interface) to host the ArcGIS Controls, thereby providing nearly the same speed of execution as any native application built using the controls. By assembling the ArcGIS Engine visual components in a Java application and 'wiring them up' with each other and with other ArcObjects components, custom GIS applications can be rapidly built and deployed on supported ArcGIS Engine platforms.

DESIGN

In this application, the *MapControl*, *PageLayoutControl*, *TOCControl*, and *ToolbarControl* components are hosted inside a *javax.swing.JFrame* container and interact with each other and with other ArcGIS Engine objects to provide GIS viewing capability.

The scenario starts with building a GUI using the GridLayout layout manager to position the components. Once the components are added to the JFrame container, they are connected with each other using the *setBuddy* method. At this stage, the application is ready to function as a simple map viewer.

The scenario then extends the functionality of the simple map viewer by building custom tools and demonstrating event handling. To achieve this, it explores the API of the visual and other nonvisual ArcGIS Engine components further.

While the components can be used as 'drag and drop' JavaBeans in a Java IDE supporting visual GUI design, in this scenario, we will programmatically place the components; this gives us a better understanding of the code. For information on using the components as 'drag and drop' beans, refer to *ArcGIS Developer Help*.

REQUIREMENTS

In order to successfully follow this scenario you need the following (the requirements for deployment are covered later in the Deployment section):

The visual JavaBeans are not included in the 'typical' installation of the ArcGIS Engine Developer Kit. If you do not have them installed, rerun the Developer Kit install wizard, choose Custom or Modify, and select the Java feature under ArcGIS Engine. In addition, for access to the Javadoc and other Java-specific documentation, select the Java feature under Software Developer Kit.

The ArcGIS developer samples are not included in the 'typical' installation of the ArcGIS Engine Developer Kit. If you do not have them installed, rerun the Developer Kit install wizard, choose Custom or Modify, and select the samples feature under Software Developer Kit.

- An installation of the ArcGIS Engine Developer Kit (including Java) with an authorization file enabling it for development use.
- An installation of the Java 2 Platform, Standard Edition (J2SE) Software Development Kit (SDK), preferably 1.4.2 or later. If you do not already have one available, download it from the Java Website at http://java.sun.com/j2se/downloads.html
- A Java IDE of your choice or your favorite text editor.
- A beginner to intermediate knowledge of the Java programming language.
- While no experience with other ESRI software is required, previous experience with ArcObjects and a basic understanding of maps are advantageous.
- Access to the sample data and code that comes with this scenario.

```
<install_location>\DeveloperKit\Samples\Developer_Guide_Scenarios\
ArcGIS_Engine\Building_an_ArcGIS_Control_Application\Map_Viewer
```

To build this application, the following visual components from the ArcGIS Engine Developer Kit will be used:

- map.MapControl
- pagelayout.PageLayoutControl
- TOC.TOCControl
- toolbar.ToolbarControl
- toolbar.ToolbarMenu

In the Java API, these are prefixed by 'com.esri.arcgis.beans'.

In addition, objects from the following libraries will be used:

- Carto
- Geometry
- Display
- SystemUI

In the Java API, their package names se are prefixed by 'com.esri.arcgis'.

In order to reference the above mentioned packages, the following JAR files must be added to your classpath:

- arcobjects.jar, located at <install_location>\ArcGIS\java\opt
- arcgis_visualbeans.jar, located at <install_location>\ArcGIS\java\opt
- jintegra.jar, , located at <install_location>\ArcGIS\java

IMPLEMENTATION

To implement this scenario follow the steps below. While the scenario specifically uses the Gulf of St Lawrence map document located installed with the samples, you can use your own map document instead. The implementation below provides you with all the code you will need to successfully complete the scenario. It does not provide step-by-step instructions to develop applications in Java, as it assumes that you have a working knowledge of the development environment already.

Setting up the development environment

To compile and run applications using the ArcGIS Engine Developer Kit, the PATH environment variable should include:

- Path to ArcGIS/bin
- Path to J2SE SDK/bin
- Path to J2SE SDK/jre/bin

To set the PATH on Windows:

1 Right-click on the My computer icon and select the Properties item from the dropdown menu.

2. In the System Properties window, click the Advanced tab.

3. Click the Environment Variables button.

4. In the Environment Variables window, double-click on the Path variable in the System variables.

5. Add the path to ArcGIS\bin folder and the bin directories of your J2SE SDK and JRE to the Variable value text field. For example, if ArcGIS Developer Kit is installed in C:\Program Files\ArcGIS and J2SE SDK is installed in C:\j2sdk1.4.2, the following would be required to be added to the PATH environment variable:

```
C:\Program Files\ArcGIS\Bin;C:\j2sdk1.4.2\jre\bin;C:\j2sdk1.4.2\bin
```

Building a GUI using the visual components

Use your favorite text editor or IDE to write your source code.

1. Create a new Java class called MapViewerFrame.java. This class will provide the GUI and functionality of the map viewer application. Implement this class as a subclass of *javax.swing.JFrame*. Add an event listener for windowClosing event to exit the application when the frame is closed by clicking on the X button.

```
//MapViewerFrame.java
import javax.swing.JFrame;
import java.awt.event.WindowAdapter;
import java.awt.event.WindowEvent;

public class MapViewerFrame extends JFrame {
  //constructor
  public MapViewerFrame() {
    setTitle("MapViewer");
  }

  public void buildAndShow() throws IOException {
    this.show();
    addEventListeners();
  }

public void addEventListeners()throws IOException {
      addWindowListener(new WindowAdapter() {
           public void windowClosing(WindowEvent e) {
               System.exit(0);
              }
  });
}
}
```

2. Create a new Java class called MapViewer.java. This class will provide the main() method to construct the MapViewerFrame, give it an initial size, and launch it:

```
//MapViewer.java
import java.awt.Dimension;
import java.awt.Toolkit;
import java.io.IOException;

import javax.swing.UIManager;
import com.esri.arcgis.system.EngineInitializer;
public class MapViewer {

   static {
        try {
UIManager.setLookAndFeel("com.sun.java.swing.plaf.windows.WindowsLookAndFeel");
       } catch (Exception e) {
           //ignore
        }
    }
```

```
  public static void main(String[] args) throws IOException {
 EngineInitializer.initializeVisualBeans();
    MapViewerFrame mapViewerFrame = new MapViewerFrame();

    Dimension d = Toolkit.getDefaultToolkit().getScreenSize();
    int width  = d.width;
    int height = d.height;
    mapViewerFrame.setBounds(width/6, height/6, width*2/3, height*2/
3);

    mapViewerFrame.buildAndShow();
  }
}
```

Any application that uses visual JavaBean components of the ArcGIS Engine Developer Kit should include the following three JAR files in its classpath:

...\ArcGIS\java\opt\arcobjects.jar
...\ArcGIS\java\opt\arcgis_ visualbeans.jar
...\ArcGIS\java\jintegra.jar

These files provide the runtime libraries needed for accessing ArcObjects from Java.

When using a Java IDE, the classpath is typically set by adding the above mentioned JAR files as referenced libraries in the Java build path.

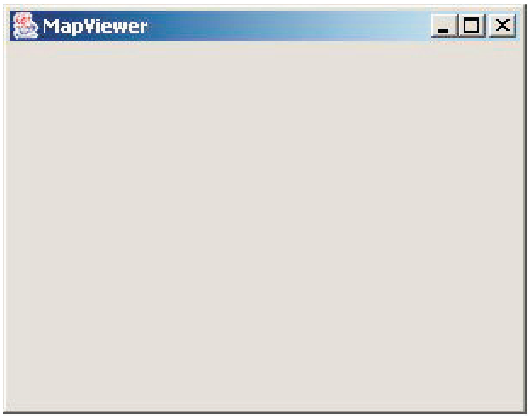
3. At this stage you should be able to compile both the MapViewerFrame and MapViewer Java files. To do so, the Java compiler needs to be told where to find the referenced Java classes. This is done by specifying the classpath.

   To compile using the command line, cd to the directory containing the MapViewerFrame and MapViewer Java code and give a command similar to:

   ```
   javac -classpath “C:\Program
   Files\ArcGIS\java\opt\arcgis_visualbeans.jar;
   C:\Program Files\ArcGIS\java\opt\arcgis_engine.jar;C:\Program
   Files\ArcGIS\java\jintegra.jar;.” *.java
   ```

This compiles the Java program and produces a MapViewerFrame.class and a MapViewer.class file. If you get a ‘NoClassDefFoundError’ error, double-check your classpath, and make sure it includes the required JAR files—arcobjects.jar, arcgis_ visualbeans.jar, and jintegra.jar.

4. Launch the MapViewer class, and make sure a blank Java frame comes up before proceeding to the next step.



To run the Java program, click the Run button in your IDE or give the following command from the command line:

java -classpath “C:\Program Files\ArcGIS\java\opt\arcgis_visualbeans.jar; C:\Program Files\ArcGIS\java\opt\ arcgis_engine.jar;C:\Program Files\ArcGIS\java\jintegra.jar;.” MapViewer

5. Next, add member variables for the components to be added to the MapViewerFrame class, and construct these components in the constructor.

   ```
   //MapViewerFrame.java
   import javax.swing.JFrame;
   import java.awt.event.WindowAdapter;
   ```

```
import java.awt.event.WindowEvent;

//add new imports for this step:
import javax.swing.JLabel;
import java.awt.BorderLayout;
import java.awt.Dimension;
import javax.swing.JPanel;
import com.esri.arcgis.beans.TOC.TOCControl;
import com.esri.arcgis.beans.map.MapControl;
import com.esri.arcgis.beans.pagelayout.PageLayoutControl;
import com.esri.arcgis.beans.toolbar.ToolbarControl;

public class MapViewerFrame extends JFrame {
   PageLayoutControl pageLayout;
   MapControl map;
   TOCControl toc;
   ToolbarControl toolbar;
   JLabel statusLabel;

     //constructor
  public MapViewerFrame() {
    setTitle(“MapViewer”);

      pageLayout = new PageLayoutControl();
      map = new MapControl();
      toc = new TOCControl();
      toolbar = new ToolbarControl();
      statusLabel = new JLabel(" ");
```

6. Set the size of the map and toolbar components in the constructor while allowing BorderLayout to manage the size of the remaining components.

```
      //set constraints on the size of map and toolbar
      //let the layout manager manage the size of all other components
      map.setSize(new Dimension(200,200));
      toolbar.setSize(new Dimension(900, 25));
  }
```

7. In the MapViewerFrame interface, the toolbar component will occupy BorderLayout.NORTH and the pageLayout component will occupy the main panel, such as the BorderLayout.CENTER position.

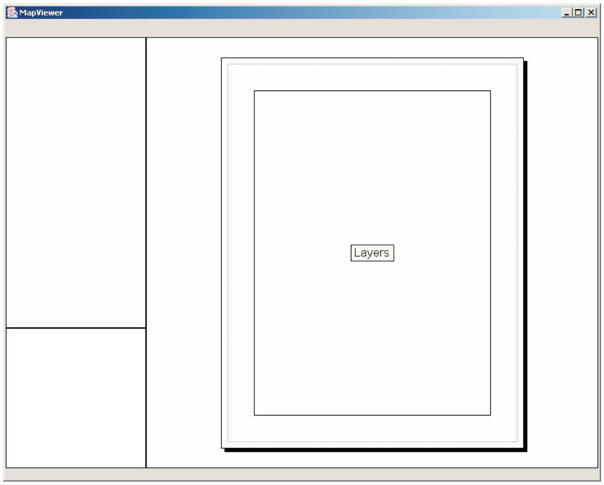
   The BorderLayout.WEST position will be occupied by both the TOC and map components. To achieve this, create a new method to build a JPanel containing the two controls with the desired layout:

```
   //new method to build the left panel
   private JPanel buildMapTOCPanel() {
      JPanel leftPanel = new JPanel();
      leftPanel.setLayout(new BorderLayout());
      leftPanel.add(toc, BorderLayout.CENTER);
      leftPanel.add(map, BorderLayout.SOUTH);
      return leftPanel;
    }
```

8. In the BorderLayout.SOUTH position of the MapViewerFrame, add a JLabel to act as a status bar. The buildAndShow() method should look like the following once you have updated it with all layout locations:

```
 public void buildAndShow()throws IOException {
     JPanel mapTOCPanel = buildMapTOCPanel();

     this.getContentPane().add(toolbar, BorderLayout.NORTH);
     this.getContentPane().add(pageLayout, BorderLayout.CENTER);
     this.getContentPane().add(mapTOCPanel, BorderLayout.WEST);
     this.getContentPane().add(statusLabel, BorderLayout.SOUTH);

  this.show();
     addEventListeners();
     }
```

9. Compile and run the application to confirm that the GUI has been laid out as shown:



As a standard Java layout manager (BorderLayout) has been used to position the components, it manages the layout and sizes of the components for you. You should be able to resize the frame and see that the components get placed appropriately without requiring any explicit resizing code.

Loading map documents

1. Now that the components have been added, you can load map documents into the controls. To do this, add the following code to the end of the buildAndShow() method, just after addEventListeners().

```
public void buildAndShow() throws IOException {
  . . .
  . . .
```

```
this.show();
addEventListeners();
    //load a pre-authored map document into the PageLayout control
    String documentPath = new java.io.File("../../../../../data/
ArcGIS_Engine_Developer_Guide/Gulf of St.
 Lawrence.mxd").getAbsolutePath();
 if (pageLayout.checkMxFile(documentPath))
        pageLayout.loadMxFile(documentPath,null);
```

2. When the document is loaded into the pageLayout control, an onPageLayoutReplaced event will be generated. You will add code to load the same map document in the map component in response to this event. In the addEventListeners() method, add the event listener to the pageLayout component for the onPageLayoutReplaced event, as an anonymous inner class, using IPageLayoutControlEventsAdapter.

```
   public void addEventListeners()throws IOException {
  . . .
pageLayout.addIPageLayoutControlEventsListener(
         new IPageLayoutControlEventsAdapter() {
              public void
onPageLayoutReplaced(IPageLayoutControlEventsOnPageLayoutReplacedEvent
 evt) throws IOException{
               map.loadMxFile(pageLayout.getDocumentFilename(),
 null, null);
               map.setExtent(map.getFullExtent());
                  });
               }
       });

}
```

As a general rule when using the Java API of ArcGIS Engine Developer Kit, all methods should be called on the visual components after they are displayed. The exception to this rule is when calling 'setXYZ()' methods—for example, to set the size of a component.

'Buddying up' the ToolbarControl and TOCControl

Although the components have been added to the JFrame, they do not yet 'know about each other'. For the controls to work in sync with each other, the TOC and toolbar components should know with which control they are associated. Otherwise, the toolbar component will not know which component it is 'controlling', and the TOC component will not know which component's table of contents it should display.

To set up this communication between the components, add the following code to the buildAndShow() method after the load document code.

```
public void buildAndShow() throws IOException {
   . . .
     //load a pre-authored map document on the pageLayout component
     String documentPath = new java.io.File("../../data/Gulf of St.
  Lawrence.mxd").getAbsolutePath();
     if (pageLayout.checkMxFile(documentPath))
        pageLayout.loadMxFile(documentPath,null);

     //set buddy controls to wire up the TOC and Toolbar Control
     //with the PageLayout Control
     toc.setBuddyControl(pageLayout);
 toolbar.setBuddyControl(pageLayout);
}
```

Adding commands to the toolbar

The toolbar control has been added to the user interface and, by default, this control is not populated with any tools. You will begin to add tools in the following steps.

The *ArcGIS Engine Developer Kit* comes with more than 120 commands and tools that work with the *MapControl*, the *PageLayoutControl*, and the *ToolbarControl*. These commands and tools provide a lot of frequently used GIS functionality for map navigation, graphics management, and feature selection. You will now add some of these commands and tools to your application.

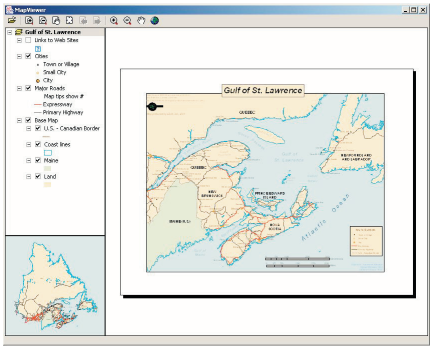
1. To add the prebuilt toolbar commands, add the following imports:

```
import com.esri.arcgis.controltools.ControlsOpenDocCommand;
import com.esri.arcgis.systemUI.esriCommandStyles;
```

2. In the buildAndShow() method, add the prebuilt commands to the toolbar after calling this.show() and before the addEventListeners() call:

```
  this.show();

  toolbar.addItem( new ControlsOpenDocCommand(), 0, -1, false, 0,
    esriCommandStyles.esriCommandStyleIconOnly );
  toolbar.addItem( new ControlsPageZoomInTool(), 0, -1, false, 0,
    esriCommandStyles.esriCommandStyleIconOnly );
  toolbar.addItem( new ControlsPageZoomOutTool(), 0, -1, false, 0,
    esriCommandStyles.esriCommandStyleIconOnly );
  toolbar.addItem( new ControlsPagePanTool(), 0, -1, false, 0,
  esriCommandStyles.esriCommandStyleIconOnly );
  toolbar.addItem( new ControlsPageZoomWholePageCommand(), 0, -1,
    false, 0, esriCommandStyles.esriCommandStyleIconOnly );
  toolbar.addItem( new ControlsPageZoomPageToLastExtentBackCommand(),
    0, -1, false, 0,
    esriCommandStyles.esriCommandStyleIconOnly );
  toolbar.addItem( new
  ControlsPageZoomPageToLastExtentForwardCommand(), 0,
  -1, false, 0,
  esriCommandStyles.esriCommandStyleIconOnly );
  toolbar.addItem( new ControlsMapZoomInTool(), 0, -1, false, 0,
  esriCommandStyles.esriCommandStyleIconOnly );
  toolbar.addItem( new ControlsMapZoomOutTool(), 0, -1, false, 0,
    esriCommandStyles.esriCommandStyleIconOnly );
  toolbar.addItem( new ControlsMapPanTool(), 0, -1, false, 0,
  esriCommandStyles.esriCommandStyleIconOnly );
  toolbar.addItem( new ControlsMapFullExtentCommand(), 0, -1, false,
    0, esriCommandStyles.esriCommandStyleIconOnly );

addEventListeners();
```

```
//load a pre-authored map document on the pageLayout component
```



3. Run the application.

   The map document has been loaded into the pageLayout component, and the toc component lists the data layers in the map document. Use the toc component to toggle layer visibility. By default the focus map of the map document is loaded into the map component. Also note that the commands have been added to the toolbar component.

Creating popup menu using ToolbarMenu

ArcGIS Engine provides a ToolbarMenu component that can be used to add popup menus to other components, such as the *MapControl* and *PageLayoutControl*. In this step, you will add a popup menu to the pageLayout component. The popup menu will contain prebuilt commands from the com.esri.arcgis.beans.toolbaritems package.

To display the popup menu, an event handler will be added to the pageLayout component for the onMouseDown event. If the right mouse button is clicked, the menu will be displayed.

1. Add the following imports to MapViewerFrame.java:

```
import com.esri.arcgis.beans.toolbar.ToolbarMenu;
import com.esri.arcgis.beans.pagelayout.IPageLayoutControlEventsAdapter;
import
com.esri.arcgis.beans.pagelayout.IPageLayoutControlEventsOnMouseDownEvent;
```

2. Construct a new ToolbarMenu object in the MapViewerFrame contstructor, as shown below:

```
public class MapViewerFrame extends JFrame{

  PageLayoutControl pageLayout;
  MapControl map;
  TOCControl toc;
  ToolbarControl toolbar;
  ToolbarMenu popupMenu;
  JLabel statusLabel;
```

3. Add a member variable for popupMenu and surround in a try/catch block:

```
public MapViewerFrame(){
      setTitle( "MapViewer" );
```

```
        try {
          pageLayout = new PageLayoutControl();
          map = new MapControl();
          toc = new TOCControl();
          toolbar = new ToolbarControl();
          popupMenu = new ToolbarMenu();
          statusLabel = new JLabel( " " );

          // set constraints on the size of the map and toolbar
          // let the layout manager
          map.setSize( new Dimension( 200, 200 ) );
          toolbar.setSize( new Dimension( 900, 25 ) );
       } catch (Exception e) {
          e.printStackTrace();
        }
}
```

4. Add prebuilt commands from the com.esri.arcgis.beans.toolbaritems package to the popupMenu component, after the call to this.show() in the buildAndShow() method:

```
this.show();

  // add popup menu items
  popupMenu.addItem( new ControlsPageZoomInFixedCommand(), 0, -1,
    false, esriCommandStyles.esriCommandStyleIconAndText );
  popupMenu.addItem( new ControlsPageZoomOutFixedCommand(), 0, -1,
    false, esriCommandStyles.esriCommandStyleIconAndText );
  popupMenu.addItem( new ControlsPageZoomWholePageCommand(), 0, -1,
    false, esriCommandStyles.esriCommandStyleIconAndText );
  popupMenu.addItem( new
  ControlsPageZoomPageToLastExtentBackCommand(), 0, -1,
   false, esriCommandStyles.esriCommandStyleIconAndText
     );
  popupMenu.addItem( new
  ControlsPageZoomPageToLastExtentForwardCommand(),
  0, -1, false,
    esriCommandStyles.esriCommandStyleIconAndText );


       //add Generic commands to the toolbar
    . . .
```

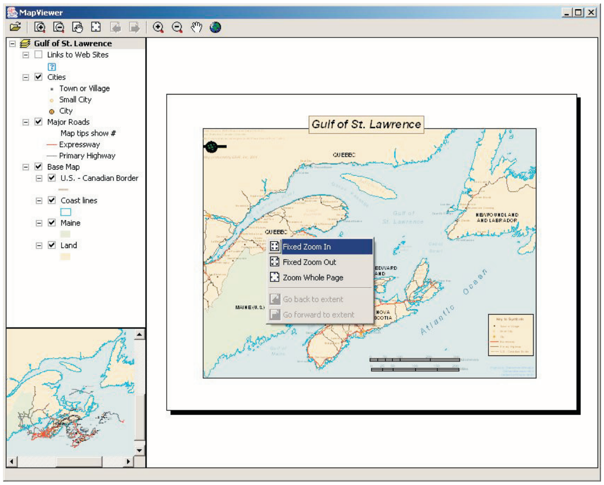
5. Associate the popupMenu with the pageLayout component, along with the code that sets up the buddy controls, in the buildAndShow() method:

```
//set buddy controls to wire up the TOC and ToolbarControl
//with the pageLayout object
toc.setBuddyControl(pageLayout);
toolbar.setBuddyControl(pageLayout);
popupMenu.setHookByRef(pageLayout);
}
```

6. In the addEventListeners() method, add the event listener to the pageLayout component for the onMouseDown event, as an anonymous inner class, using IPageLayoutControlEventsAdapter:

```
pageLayout.addIPageLayoutControlEventsListener(
  new IPageLayoutControlEventsAdapter(){
  public void onMouseDown(IPageLayoutControlEventsOnMouseDownEvent
    evt) throws IOException {
        //if Right mouse button(2) is pressed, display the popup menu
        if(evt.getButton() == 2)
              popupMenu.popupMenu(evt.getX(), evt.getY(),
    pageLayout.getHWnd());
  }
});
```

7. Run the application.



Right-click on the pageLayout component to display the popup menu, and navigate around the page layout.

Controlling Label Editing in the TOCControl

By default the *TOCControl* allows users to automatically toggle the visibility of layers and to change map and layer names as they appear in the table of contents. You will add code to prevent users from editing a name and replacing it with an empty string.

1. Add the following imports to MapViewerFrame.java for classes used in this step:

```
import com.esri.arcgis.beans.TOC.ITOCControlEventsAdapter;
import com.esri.arcgis.beans.TOC.ITOCControlEventsOnEndLabelEditEvent;
import com.esri.arcgis.beans.TOC.esriTOCControlEdit;
```

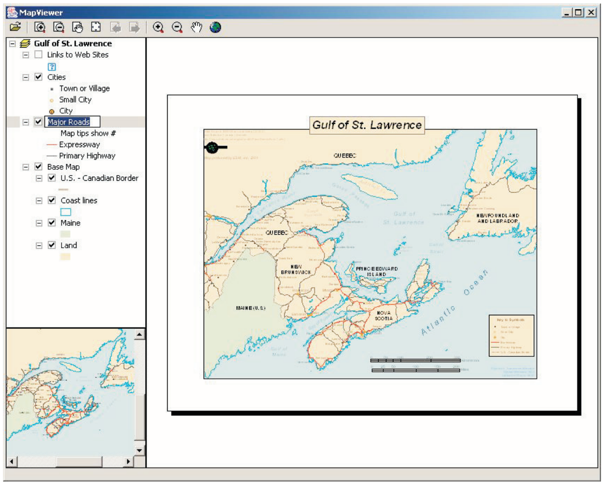
2. In the buildAndShow() method, set the labelEdit to manual:

```
. . .
this.show();

      //set label editing to manual
      toc.setLabelEdit(esriTOCControlEdit.esriTOCControlManual);

      //add popup menu items
   . . .
```

3. In the addEventListeners() method, add the event listener to the toc component for the onEndLabelEdit event, as an anonymous inner class, using ITOCControlEventsAdapter. After adding this, your addEventListeners() method should look like this:

```
private void addEventListeners() throws IOException {
   . . .
  toc.addITOCControlEventsListener(new ITOCControlEventsAdapter(){
     public void onEndLabelEdit(ITOCControlEventsOnEndLabelEditEvent
    labelEditEvt)
       throws IOException {
                String newLabel = labelEditEvt.getNewLabel();
                //if the new label is an empty string, prevent the
     //edit
                if (newLabel.equals(""))
                    labelEditEvt.setCanEdit(false);
    }
  });
}
```

4. Run the application. To edit a map, layer, heading, or legend class label in the toc component, click on it once, then click on it a second time to invoke label editing. Try replacing the label with an empty string. You will be able to replace the label with strings other than an empty string. You can use the Esc key on the keyboard at any time during the edit to cancel it.



Navigating around the focus map using the map navigation tools will change the extent of the focus map in the PageLayoutControl and cause the MapControl to update. Navigating around the page layout with the page layout navigation tools will change the extent of the page layout (not the extent of the focus map in the PageLayoutControl), so the MapControl will not update.

Drawing an 'overview rectangle' on the MapControl

You will now use the map component as an overview window and draw on its display the current extent of the focus map within the pageLayout component. As you navigate around the data within the data frame of the pageLayout component, you will see the map component's overview window update.

1. Add the following imports to the MapViewerFrame.java for classes used in this step:

```
import com.esri.arcgis.beans.map.IMapControlEvents2Adapter;
import com.esri.arcgis.beans.map.IMapControlEvents2OnAfterDrawEvent;

import com.esri.arcgis.carto.Map;
import com.esri.arcgis.carto.esriViewDrawPhase;
import com.esri.arcgis.display.DisplayTransformation;
import com.esri.arcgis.display.ITransformEventsAdapter;
import com.esri.arcgis.display.ITransformEventsVisibleBoundsUpdatedEvent;
import com.esri.arcgis.display.RgbColor;
import com.esri.arcgis.display.SimpleFillSymbol;
import com.esri.arcgis.display.SimpleLineSymbol;
import com.esri.arcgis.geometry.IEnvelope;
```

2. Add the following class members to MapViewerFrame:

```
public class MapViewerFrame extends JFrame {
   SimpleFillSymbol fillSymbol; //The symbol used to draw the envelope
   IEnvelope currentExtent;     //The envelope drawn on the MapControl
   Map focusMap;                //The PageLayoutControl's focus map

   PageLayoutControl pageLayout;
. . .
```

3. Create a new private method called createOverviewSymbol(). This is where you will create the symbol used in the map control to represent the extent of the data in the pageLayout control. Add the following code to the sub routine:

```
private void createOverviewSymbol() throws IOException{
   RgbColor color = new RgbColor();
   color.setRed(255);
   color.setGreen(0);
   color.setBlue(0);
   color.setTransparency((byte)255);

   SimpleLineSymbol outline = new SimpleLineSymbol();
   outline.setWidth(15);
   outline.setColor(color);

   fillSymbol = new SimpleFillSymbol();

   color.setTransparency((byte) 0);
   fillSymbol.setColor(color);
   fillSymbol.setOutline(outline);
}
```

4. Call the *createOverviewSymbol* from the buildAndShow() method, after the call setting label editing to manual:

```
//set label editing to manual
toc.setLabelEdit(esriTOCControlEdit.esriTOCControlManual);

//create symbol used to draw overview on the MapControl
createOverviewSymbol();

//add popup menu items
. . .
```

5. In the event listener for *IPageLayoutControlEvents* added in the addEventListeners() method earlier, get a reference to the pageLayout component's focus map. Store the extent of the focusMap in the currentExtent member variable. This extent will be used to draw the overview rectangle on the map component. To achieve this, add the lines of code (shown in bold) after loading the map documents and setting its extent in the onPageLayoutReplaced event handler:

```
pageLayout.addIPageLayoutControlEventsListener(
  new IPageLayoutControlEventsAdapter() {
    public void
            onPageLayoutReplaced(IPageLayoutControlEventsOnPageLayoutReplacedEvent
      evt) throws IOException{

map.loadMxFile(pageLayout.getDocumentFilename(), null,
    null);
     map.setExtent(map.getFullExtent());

     focusMap = new
  Map(pageLayout.getActiveView().getFocusMap());
     currentExtent = focusMap.getExtent();
  }
});
```

6. The default events of the *PageLayoutControl* are the IPageLayoutControlEvents. These events do not tell us when the extent of the map within the data frame changes. To do this you will trap the ITransformEvents of the *PageLayoutControl*'s focus map. Add the following event listeners to listen to the DisplayTransformation object's visibleBoundsUpdated event.

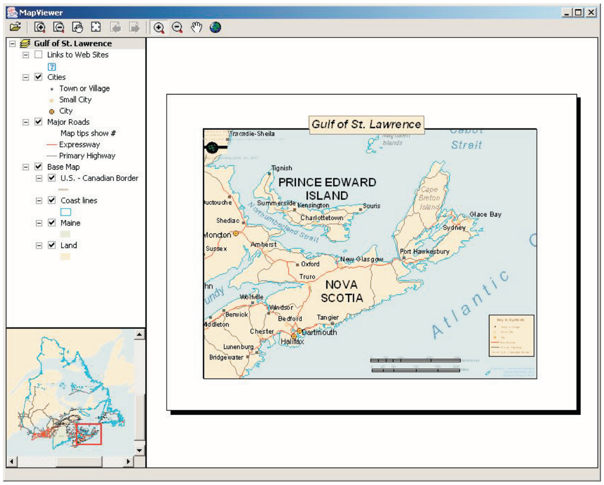
   The visibleBoundsUpdated event is triggered whenever the extent of the map is changed and is used to set the envelope to the new visible bounds of the map. By refreshing the *MapControl* you force it to redraw the shape on its display.

```
pageLayout.addIPageLayoutControlEventsListener(
  new IPageLayoutControlEventsAdapter() {
   public void
 onPageLayoutReplaced(IPageLayoutControlEventsOnPageLayoutReplacedEvent
     evt) throws IOException{
            map.loadMxFile(pageLayout.getDocumentFilename(), null,
     null);
      map.setExtent(map.getFullExtent());
   focusMap = new Map(pageLayout.getActiveView().getFocusMap());
   currentExtent = focusMap.getExtent();

   DisplayTransformation dt = new
      DisplayTransformation(focusMap.getScreenDisplay().getDisplayTransformation());
   dt.addITransformEventsListener(new ITransformEventsAdapter(){

public void
visibleBoundsUpdated(ITransformEventsVisibleBoundsUpdatedEvent evt)
 throws IOException {
   //set currentExtent to the new visible extent
   currentExtent = evt.getSender().getVisibleBounds();
   //refresh the map components foreground phase
 map.refresh(esriViewDrawPhase.esriViewForeground,null,null);
  }

   });

 }
});
```

7. Add an *IMapControlEvents2Listener* to the map component that draws the updated bounds, whenever a map refresh is triggered by the visibleBoundsUpdated event-handler, added in the last step. Add the following to the addEventListeners() method:

```
map.addIMapControlEvents2Listener(new IMapControlEvents2Adapter() {
 public void onAfterDraw(IMapControlEvents2OnAfterDrawEvent evt)
   throws IOException {
         if (evt.getViewDrawPhase() ==
esriViewDrawPhase.esriViewForeground){
  try{
    //draw the shape on the MapControl
    map.drawShape(currentExtent, fillSymbol);
  } catch (Exception e) {
    System.err.println("Error in drawing shape on
 MapControl");
    //e.printStackTrace();
  }
   }
 }
});
```

8. Run the application. Use the map navigation tools that you added earlier to change the extent of the focus map in the pageLayout component, and observe that the new extent is drawn on the map component, as a red rectangle.



Creating a custom tool

You will create a custom tool that can be added to the *ToolbarControl*. This tool will add today's date as a text element to the *PageLayoutControl*, at the location of a mouse click.

The tool will be built as a generic tool so it could be work with *MapControl* and *ToolbarControl* as well as the *PageLayoutControl*.

Custom tools can be built as Java classes that implement the following two interfaces:

```
import com.esri.arcgis.systemUI.ICommand;
import com.esri.arcgis.systemUI.ITool;
```

1. Create a new Java class AddDateCommand.java:

```
import com.esri.arcgis.systemUI.ICommand;
import com.esri.arcgis.systemUI.ITool;

public class AddDateCommand implements ICommand, ITool {

}
```

As this class implements the ICommand and ITool interfaces, it will compile only after all methods belonging to these interfaces have been implemented.

2. Add implementation for all methods defined in ICommand, as shown below. The implementation of onCreate and onClick methods will be done in the next step, so you can leave an 'empty' implementation for now.

```
import java.io.IOException;
import java.text.SimpleDateFormat;
import java.util.Date;
import com.esri.arcgis.carto.IActiveView;
import com.esri.arcgis.carto.TextElement;
import com.esri.arcgis.carto.esriViewDrawPhase;
import com.esri.arcgis.display.TextSymbol;
import com.esri.arcgis.geometry.IPoint;
import com.esri.arcgis.support.ms.stdole.StdFont;
import com.esri.arcgis.systemUI.ICommand;
import com.esri.arcgis.systemUI.ITool;

public class AddDateCommand implements ICommand, ITool {

  /**
  * @see com.esri.arcgis.systemUI.ICommand#isEnabled()
  */
  public boolean isEnabled() throws IOException, AutomationException {
     return true;
  }
  /**
   * @see com.esri.arcgis.systemUI.ICommand#isChecked()
   */
  public boolean isChecked() throws IOException, AutomationException {
     return false;
  }
  /**
   * @see com.esri.arcgis.systemUI.ICommand#getName()
   */
  public String getName() throws IOException, AutomationException {
     return “CustomCommands_Add Date”;
  }
  /**
   * @see com.esri.arcgis.systemUI.ICommand#getCaption()
   */
  public String getCaption() throws IOException, AutomationException {
     return "Add Date";
  }
  /**
   * @see com.esri.arcgis.systemUI.ICommand#getTooltip()
   */
  public String getTooltip() throws IOException, AutomationException {
     return "Add date";
  }
  /**
   * @see com.esri.arcgis.systemUI.ICommand#getMessage()
   */
  public String getMessage() throws IOException, AutomationException {
     return "Adds a date element to the page layout";
  }
  /**
   * @see com.esri.arcgis.systemUI.ICommand#getHelpFile()
   */
  public String getHelpFile() throws IOException, AutomationException {
    return null;
  }
  /**
   * @see com.esri.arcgis.systemUI.ICommand#getHelpContextID()
   */
  public int getHelpContextID() throws IOException, AutomationException {
    return 0;
  }
  /**
   * @see com.esri.arcgis.systemUI.ICommand#getBitmap()
   */
  public int getBitmap() throws IOException, AutomationException {
    return 0;//We rely on being displayed as text
  }
  /**
   * @see com.esri.arcgis.systemUI.ICommand#getCategory()
   */
  public String getCategory() throws IOException, AutomationException {
    return "CustomCommands";
  }
  /**
   * @see com.esri.arcgis.systemUI.ICommand#onCreate(java.lang.Object)
   */
                public void onCreate(Object obj) throws IOException,
AutomationException {
       //to be added later
  }
  /**
   * @see com.esri.arcgis.systemUI.ICommand#onClick()
   */
  public void onClick() throws IOException, AutomationException {
  }

}
```

3. The onCreate method is passed a reference of the application that the command works with. In this case, it can be a *MapControl*, *PageLayoutControl*, or *ToolbarControl*. Rather than adding logic in the onCreate method to determine the type of object passed to the hook method, you will use the HookHelper class to handle this. A command or tool needs to know how to handle the hook it gets passed, so a check is needed to determine the type of ArcGIS Control has been passed. The HookHelper is used to hold the hook and return the ActiveView regardless of the type of hook (in this case either a *MapControl*, *PageLayoutControl*, or *ToolbarControl*).

   Add the HookHelper member variable to the AddDateCommand class:

```
public class AddDateCommand implements ICommand, ITool{
HookHelper hookHelper;
```

4. In the onCreate method, construct the hookhelper and use the setHookByRef method to pass the controls object.

```
public void onCreate(Object obj) throws IOException, AutomationException
{
hookHelper = new HookHelper();
hookHelper.setHookByRef( obj );
}
```

5. Add implementation for all methods defined in ITool interface to the AddDateCommand class. Not all methods will be used, but they need to be implemented to be able to compile the class. In the following code, pay attention to the onClick() method, as this method creates a TextElement with the current date as its text and adds it to the graphics container of the hook object.

```
/**
   * @see com.esri.arcgis.systemUI.ITool#getCursor()
   */
  public int getCursor() throws IOException, AutomationException {
    return 0;
  }
  /**
   * @see com.esri.arcgis.systemUI.ITool#onMouseDown(int, int, int,
  int)
   */
  public void onMouseDown(int button, int shift, int x, int y)
    throws IOException, AutomationException {

       Date today = new Date();

       //format the date in the form "Wed 27 Aug, 2003"
       SimpleDateFormat formatter;
       formatter = new SimpleDateFormat("EEE d MMM, yyyy");
       String dateString = formatter.format(today);

  //create a font
       StdFont font = new StdFont();
       font.setName("Arial");
       font.setBold(true);

  //create a text symbol
       TextSymbol textSymbol = new TextSymbol();
       textSymbol.setFont(font);
       textSymbol.setSize(15);

  //create a text element to be added to the graphics container
  TextElement textElement = new TextElement();
       textElement.setSymbol(textSymbol);
       textElement.setScaleText(false);
       textElement.setText(dateString);
   //add the text element to the graphics container
       IActiveView activeView = hookHelper.getActiveView();
        IPoint pt =
          activeView.getScreenDisplay().getDisplayTransformation().toMapPoint(x,y);
       textElement.setGeometry(pt);

       activeView.getGraphicsContainer().addElement(textElement, 0);

  //refresh the view
  activeView.partialRefresh(esriViewDrawPhase.esriViewGraphics, null, null);


  }
  /**
   * @see com.esri.arcgis.systemUI.ITool#onMouseMove(int, int, int, int)
   */
  public void onMouseMove(int arg0, int arg1, int arg2, int arg3)
    throws IOException, AutomationException {
  }
  /**
                      * @see
com.esri.arcgis.systemUI.ITool#onMouseUp(int, int, int, int)
   */
  public void onMouseUp(int arg0, int arg1, int arg2, int arg3)
    throws IOException, AutomationException {
  }
  /**
   * @see com.esri.arcgis.systemUI.ITool#onDblClick()
   */
  public void onDblClick() throws IOException, AutomationException {
  }
  /**
   * @see com.esri.arcgis.systemUI.ITool#onKeyDown(int, int)
   */
  public void onKeyDown(int arg0, int arg1)
    throws IOException, AutomationException {
  }
  /**
   * @see com.esri.arcgis.systemUI.ITool#onKeyUp(int, int)
   */
  public void onKeyUp(int arg0, int arg1)
    throws IOException, AutomationException {
  }
  /**
   * @see com.esri.arcgis.systemUI.ITool#onContextMenu(int, int)
   */
  public boolean onContextMenu(int arg0, int arg1)
    throws IOException, AutomationException {
    return false;
  }
```

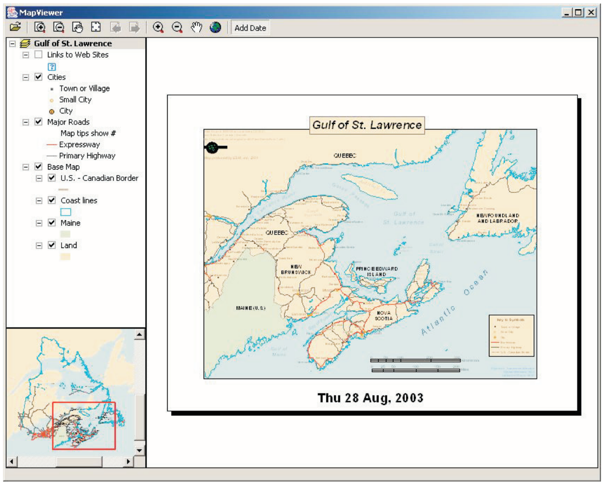
```
/**
 * @see com.esri.arcgis.systemUI.ITool#refresh(int)
 */
public void refresh(int arg0) throws IOException, AutomationException {
}
/**
 * @see com.esri.arcgis.systemUI.ITool#deactivate()
 */
public boolean deactivate() throws IOException, AutomationException {
  return true;
}
```

6. The AddDateCommand class is now complete. Compile it.

7. In MapViewerFrame, add an instance of the AddDateCommand to the toolbar component in the buildAndShow() method, after the lines of code that add prebuilt commands to the toolbar:

```
toolbar.addItem(new AddDateCommand(),
  0, -1, true, 0, esriCommandStyles.esriCommandStyleTextOnly);
```

8. Recompile and launch MapViewer class. A new Add date tool will come up on the toolbar. Select that tool and click on the page layout to add today's date at that point.



Customizing the ToolbarControl

In addition to adding ArcGIS Engine commands and tools to the *ToolbarControl* using addItem() as shown above, you can also add them by customizing the *ToolbarControl* using the customize dialog box. To do this, you will place the Toolbar component in customize mode and display the customize dialog box.

1. Add the following imports for this section:

```
import javax.swing.JCheckBox;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
```

Note that only tools and commands that are registered on the system as COM components can be added to the toolbar using the customize dialog box. Java commands and tools (like the one built in the previous step) do not appear in the Customize dialog box, as they are not registered as COM components in the system registry

```
import com.esri.arcgis.beans.toolbar.CustomizeDialog;
import com.esri.arcgis.beans.toolbar.ICustomizeDialogEvents;
import
com.esri.arcgis.beans.toolbar.ICustomizeDialogEventsOnCloseDialogEvent;
import
com.esri.arcgis.beans.toolbar.ICustomizeDialogEventsOnStartDialogEvent;
```

2. Add the class members.

```
. . .
public class MapViewerFrame extends JFrame {
       . . .
     . . .
   JCheckBox customizeCB; //JCheckbox to control toolbar customization
   CustomizeDialog customizeDialog; //The customize dialog box used by
           //the ToolbarControl constructor
  public MapViewerFrame() {
  . . .
. . .
```

3. Create a new method "createCustomizeDialog()" to instantiate the customize dialog box. Add the following code to the method:

```
/**
* Method createCustomizeDialog.
*/
private void createCustomizeDialog() throws IOException {
customizeDialog = new CustomizeDialog();
//set the title
customizeDialog.setDialogTitle("Customize Toolbar Items");
//show the 'Add From File' button
customizeDialog.setShowAddFromFile(true);
//set the toolbar component that the new items will be added to
customizeDialog.setDoubleClickDestination(toolbar);
}
```

4. In the buildandShow() method, call the createCustomizeDialog() method to instantiate the customize dialog box. Also instantiate the customizeCB JCheckBox and add event listeners to it to start and close the customize dialog box:

```
  public void buildAndShow() throws IOException {
      JPanel mapTOCPanel = buildMapTOCPanel();

  createCustomizeDialog();
  customizeCB = new JCheckBox("Customize");
       customizeCB.addActionListener(new ActionListener(){
     public void actionPerformed(ActionEvent e) {
                 try {
                  if (customizeCB.isSelected()) {
                     customizeDialog.startDialog(toolbar.getHWnd());
                     } else {
                      customizeDialog.closeDialog();
                     }
               } catch (Exception ee) {
                   ee.printStackTrace();
```

```
                }
        }});

           ...
```

5. To add the Customize check box to the GUI, modify the code to add the toolbar and check box to a new JPanel. Add this panel to the BorderLayout.NORTH position instead of adding just the toolbar in that position.

```
   JPanel topPanel = new JPanel();
      topPanel.setLayout(new BorderLayout());
      topPanel.add(toolbar, BorderLayout.CENTER);
      topPanel.add(customizeCB, BorderLayout.EAST);

      this.getContentPane().add(topPanel, BorderLayout.NORTH);
      ~~this.getContentPane().add(toolbar, BorderLayout.NORTH);//delete
this line~~
      this.getContentPane().add(pageLayout, BorderLayout.CENTER);
      this.getContentPane().add(mapTOCPanel, BorderLayout.WEST);
```

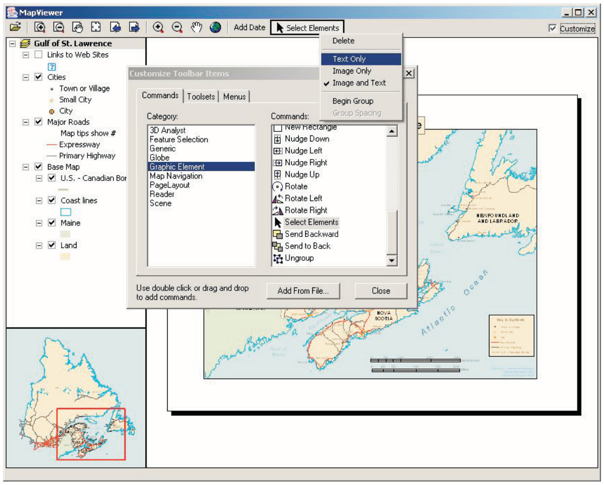
6. In the createCustomizeDialog() method, add event listener to the customize dialog box to put the toolbar in ‘customize’ state when the dialog box is started and in the normal state when it is closed:

```
  private void createCustomizeDialog() throws IOException {
      customizeDialog = new CustomizeDialog();
       //set the title
      customizeDialog.setDialogTitle("Customize Toolbar Items");
      //show the ‘Add From File’ button
      customizeDialog.setShowAddFromFile(true);
      //set the toolbar component that the new items will be added to
      customizeDialog.setDoubleClickDestinationByRef(toolbar);

      customizeDialog.addICustomizeDialogEventsListener(new
  ICustomizeDialogEvents(){
    public void onStartDialog(ICustomizeDialogEventsOnStartDialogEvent
arg0)
       throws IOException {
                 toolbar.setCustomize(true);
    }
    public void onCloseDialog(ICustomizeDialogEventsOnCloseDialogEvent
arg0)
       throws IOException {
                 toolbar.setCustomize(false);
                 customizeCB.setSelected(false);
    }});
  }
```

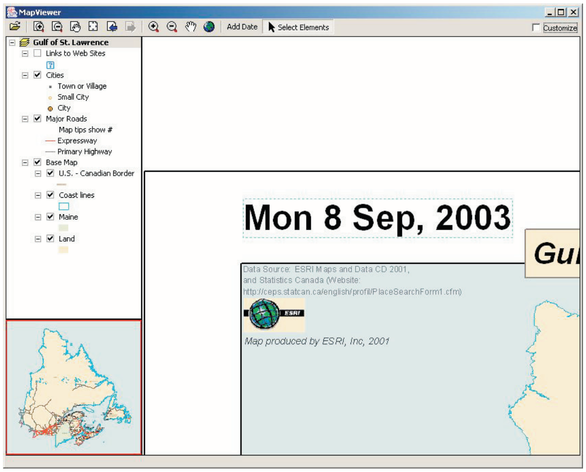
7. Run the application and check the ‘customize’ check box to put the toolbar into ‘customize’ mode and open the customize dialog box.

8. On the Commands tab, select the Graphic Element category, and drag the Select Elements command to the toolbar to add it there. By right-clicking on

an item on the toolbar, you can adjust the appearance in terms of style and grouping.



9. Stop customizing the application. Use the Select tool to move the text element containing today's date.



DEPLOYMENT

To successfully deploy this application onto a user's machine, you will create an executable JAR file. Users will then be able to launch the application by using the JRE installed as part of ArcGIS Engine Developer Kit or Runtime, by giving the following command:

```
java -jar mapviewer.jar
```

To create an executable JAR:

1. In the directory where the compiled Java class files are present, create a file called "manifest.txt".
2. Add the following single line to the manifest.txt file:

   ```
   Main-Class: com.esri.arcgis.scenario.beans.MapViewer
   ```
3. Make sure to hit enter at the end of the first line.
4. Save the file, and open a command window. cd to the directory containing the manifest.txt file.
5. Give the following command to create the executable JAR file:

   ```
   jar cmf manifest.txt mapviewer.jar *.class
   ```
6. A mapviewer.jar file will be created. This is the executable JAR that can be launched using the JRE included as part of the ArcGIS Engine Developer Kit, as follows:

   ```
   "C:\Program Files\ArcGIS\java\jre\bin\java" -jar mapviewer.jar
   ```

This command can be bundled in a batch file or shell script to provide a launch script.

ADDITIONAL RESOURCES

The following resources may help you understand and apply the concepts and techniques presented in this scenario.

- Additional documentation available in the ArcGIS Engine Developer Kit including *ArcGIS Developer Help*, javadoc, object model diagrams, and samples to help you get started.
- ArcGIS Developer Online—Web site providing the most up-to-date information for ArcGIS Developers including updated samples and technical documents. Go to http://arcgisdeveloperonline.esri.com.
- ESRI online discussion forums—Web sites providing invaluable assistance from other ArcGIS developers. Go to http://support.esri.com and click the User Forums tab.
- Sun's Java Tutorial at *http://java.sun.com/docs/books/tutorial/*
- Helpful Web sites for Java in general, such as Javaranch (h*ttp://www.javaranch.com/*)—a friendly place for Java greenhorns.

Rather than walk through this scenario, you can get the completed application from the samples installation location. The sample is installed as part of the ArcGIS developer samples.

The ArcGIS developer samples are not included in the 'typical' installation of the ArcGIS Engine Developer Kit. If you do not have them installed, rerun the Developer Kit install wizard, choose Custom or Modify, and select the samples feature under Software Developer Kit.

This walkthrough is for developers who want to build and deploy an application using .NET. It describes the process of building and deploying an application using the ArcGIS Controls.

You can find this sample in

```
<install_location>\DeveloperKit\Samples\Developer_Guide_Scenarios\
ArcGIS_Engine\Building_an_ArcGIS_Control_Application\Map_Viewer
```

PROJECT DESCRIPTION

The goal of the Building applications with Windows Controls scenario is to demonstrate and make you familiar with the steps required to develop and deploy a GIS application using the standard ArcGIS Controls within the Microsoft Visual Studio.NET API. The scenario uses the *MapControl*, *PageLayoutControl*, *TOCControl*, and *ToolbarControl* as Windows Controls within the Microsoft Visual Studio .NET development environment. COM, Java, and C++ programmers should refer to the following scenarios: Building applications with ActiveX, Building applications with visual JavaBeans, Building a command line Java application, and Building a command line C++ application.

The scenario demonstrates the steps required to create a GIS application for viewing map documents pre-authored using ArcMap, an ArcGIS Desktop application. The scenario covers the following techniques:

- Loading and embedding the ArcGIS Controls in Microsoft Visual Studio .NET.
- Loading pre-authored map documents into the *MapControl* and *PageLayoutControl*.
- Setting *ToolbarControl* and *TOCControl* buddy controls.
- Handling form resize.
- Adding ArcGIS Engine commands and tools to the *ToolbarControl*.
- Creating popup menus.
- Managing label editing in the *TOCControl*.
- Drawing shapes on the *MapControl*.
- Creating a custom tool to work with the *MapControl*, *PageLayoutControl*, and *ToolbarControl*.
- Customizing the *ToolbarControl*.
- Deploying the application onto a Windows operating system.

CONCEPTS

This scenario is implemented using the Microsoft Visual Studio .NET development environment and uses the ESRI interop assemblies to host the ArcGIS Controls inside .NET Windows Controls in a .NET form. These interoperability assemblies act as a bridge between the unmanaged code of COM and the managed .NET code. Any references to the members of the COM ArcGIS Controls are routed to the interop assemblies and forwarded to the actual COM object. Likewise, responses from the COM object are routed to the interop assembly and forwarded to the .NET application. Each ArcGIS Engine control has events, properties, and methods that can be accessed once embedded within a container such as a .NET form. The objects and functionality within each control can be combined with other ESRI ArcObjects and custom controls to create customized end user applications.

The scenario has been written in both C# and Visual Basic .NET, though the following implementation concentrates on the C# scenario. Many developers will feel more comfortable with Visual Basic .NET, as the code looks familiar to Visual Basic 6.0, while the syntax of the of the C# programming language will be familiar to Java and C++ programmers. Whichever development environment you use, your future success with the ArcGIS Controls depends on your skill in both the programming environment and ArcObjects.

The *MapControl, PageLayoutControl, TOCControl,* and *ToolbarControl* are used in this scenario to provide the user interface of the application. The ArcGIS Controls are used in conjunction with other ArcObjects and ArcGIS Engine commands by the developer to create a GIS viewing application.

DESIGN

The scenario has been designed firstly to highlight how the ArcGIS Controls interact with each other, and secondly to expose a part of each ArcGIS Controls object model to the developer.

Each .NET ArcGIS Engine control has a set of property pages that can be accessed once the control is embedded within a .NET Form. These property pages provide shortcuts to a selection of a control's properties and methods, and allow a developer to build an application without writing any code. This scenario does not use the property pages, but rather builds up the application programmatically. For further information about the property pages, refer to the *ArcGIS Developer Help*.

REQUIREMENTS

To successfully follow this scenario you need the following (the requirements for deployment are covered later in the Deployment section):

- An installation of the ArcGIS Engine Developer Kit with an authorization file enabling it for development use.
- An installation of the Microsoft Visual Studio .NET 2003 development environment and the Microsoft .NET Framework 1.1. and an appropriate license.
- Familiarity with Microsoft Windows operating systems and a working knowledge of Microsoft Visual Studio .NET and either the C# or Visual Basic .NET programming language. While the scenario provides some information about how to use the ArcGIS Controls in Microsoft Visual Studio .NET, it is not a substitute for training in the development environment.
- While no experience with other ESRI software is required, previous experience with ArcObjects and a basic understanding of ArcGIS applications, such as ArcMap and ArcCatalog, is advantageous.

The ArcGIS developer samples are not included in the 'typical' installation of the ArcGIS Engine Developer Kit. If you do not have them installed, rerun the Developer Kit install wizard, choose Custom or Modify, and select the samples feature under Software Developer Kit.

- Access to the sample data and code that comes with this scenario. This is located at

  `<install_location>\DeveloperKit\Samples\Developer_Guide_Scenarios\ArcGIS_Engine\Building_an_ArcGIS_Control_Application\Map_Viewer`

The controls and libraries used in this scenario are as follows:

- AxMapControl
- AxPageLayoutControl
- ESRI.ArcGIS.Carto
- ESRI.ArcGIS.Display
- ESRI.ArcGIS.Geometry
- esriMapControl
- esriPageLayoutControl
- AxTOCControl
- AxToolbarControl
- ESRI.ArcGIS.System
- ESRI.ArcGIS.SystemUI
- ESRI.ArcGIS.Utility
- esriTOCControl
- esriToolbarControl

The AxControlName .NET Framework components represent the control that is hosted within a .NET Form, whilst the esriControlName assemblies contain the object and interfaces from inside the control's type library.

IMPLEMENTATION

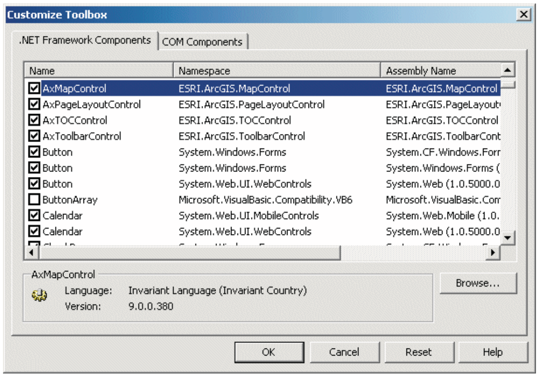
The implementation below provides you with all the code you will need to successfully complete the scenario. It does not provide step-by-step instructions to develop applications in Microsoft Visual Studio .NET, as it assumes that you have a working knowledge of the development environment already.

Loading the ArcGIS Controls

Before you start to program your application, the ArcGIS Controls and the other ArcGIS Engine library references that the application will use should be loaded into the development environment.

1. Start Visual Studio .NET and create a new Visual C# 'Windows Application' project from the New project dialog box.
2. Name the project 'Controls', and browse to a location to save the project.
3. Right-click in the Windows Forms tab of the Toolbox and select 'Add/Remove Items…' from the context menu.

4. In the Customize Toolbox dialog box, select .NET Framework components and check 'AxMapControl', 'AxPageLayoutControl', 'AxTOCControl', and 'AxToolbarControl'. Click OK. The controls will now appear in the Windows Forms tab of the toolbox.



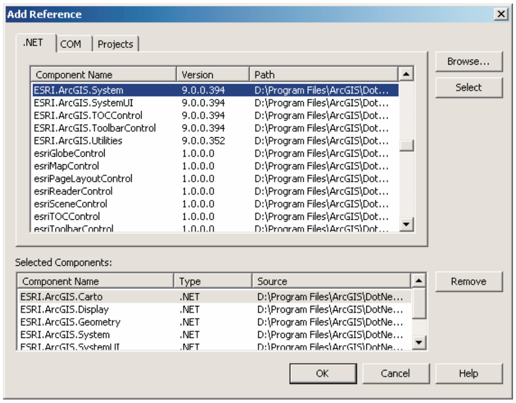
Name	Namespace	Assembly Name
☑ AxMapControl	ESRI.ArcGIS.MapControl	ESRI.ArcGIS.MapControl
☑ AxPageLayoutControl	ESRI.ArcGIS.PageLayoutControl	ESRI.ArcGIS.PageLayout
☑ AxTOCControl	ESRI.ArcGIS.TOCControl	ESRI.ArcGIS.TOCControl
☑ AxToolbarControl	ESRI.ArcGIS.ToolbarControl	ESRI.ArcGIS.ToolbarCont
☑ Button	System.Windows.Forms	System.CF.Windows.Forr
☑ Button	System.Windows.Forms	System.Windows.Forms (
☑ Button	System.Web.UI.WebControls	System.Web (1.0.5000.0
□ ButtonArray	Microsoft.VisualBasic.Compatibility.VB6	Microsoft.VisualBasic.Cor
☑ Calendar	System.Web.UI.MobileControls	System.Web.Mobile (1.0.
☑ Calendar	System.Web.UI.WebControls	System.Web (1.0.5000.0

AxMapControl
Language: Invariant Language (Invariant Country)
Version: 9.0.0.380

5. Click on the Project menu and choose Add Reference….

6. In the Add Reference dialog box, double click 'ESRI.ArcGIS.Carto', 'ESRI.ArcGIS.Display' 'ESRI.ArcGIS.Geometry', 'ESRI.ArcGIS.System', 'ESRI.ArcGIS.SystemUI' ESRI.ArcGIS.Utility. Click OK.

The ESRI .NET assemblies, will be used to instantiate and make calls on the objects in the ESRI object libraries from your C# project using the COM interoperability services provided by the .NET framework.



Component Name	Version	Path
ESRI.ArcGIS.System	9.0.0.394	D:\Program Files\ArcGIS\Dot…
ESRI.ArcGIS.SystemUI	9.0.0.394	D:\Program Files\ArcGIS\Dot…
ESRI.ArcGIS.TOCControl	9.0.0.394	D:\Program Files\ArcGIS\Dot…
ESRI.ArcGIS.ToolbarControl	9.0.0.394	D:\Program Files\ArcGIS\Dot…
ESRI.ArcGIS.Utilities	9.0.0.352	D:\Program Files\ArcGIS\Dot…
esriGlobeControl	1.0.0.0	D:\Program Files\ArcGIS\Dot…
esriMapControl	1.0.0.0	D:\Program Files\ArcGIS\Dot…
esriPageLayoutControl	1.0.0.0	D:\Program Files\ArcGIS\Dot…
esriReaderControl	1.0.0.0	D:\Program Files\ArcGIS\Dot…
esriSceneControl	1.0.0.0	D:\Program Files\ArcGIS\Dot…
esriTOCControl	1.0.0.0	D:\Program Files\ArcGIS\Dot…
esriToolbarControl	1.0.0.0	D:\Program Files\ArcGIS\Dot…

Selected Components:

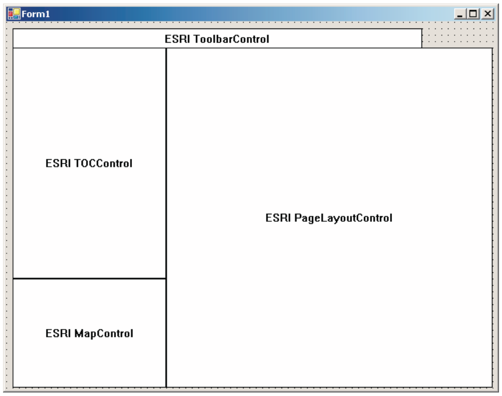
Component Name	Type	Source
ESRI.ArcGIS.Carto	.NET	D:\Program Files\ArcGIS\DotNe…
ESRI.ArcGIS.Display	.NET	D:\Program Files\ArcGIS\DotNe…
ESRI.ArcGIS.Geometry	.NET	D:\Program Files\ArcGIS\DotNe…
ESRI.ArcGIS.System	.NET	D:\Program Files\ArcGIS\DotNe…
ESRI.ArcGIS.SystemUI	.NET	D:\Program Files\ArcGIS\DotNe…

Embedding the ArcGIS Controls in a Container

Before you can access each control's properties, methods, and events, each control needs embedding within a .NET container. Once the controls are embedded within the form, they will shape the application's user interface.

In .NET a variable is fully qualified using a Namespace. Namespaces are a concept in .NET that allow objects to be organized hierarchically, regardless of the assembly they are defined in. To make code simpler and more readable, the directives act as shortcuts when referencing items specified in namespaces.

1. Open the .NET Form in design mode.
2. Double-click the AxMapControl in the Windows Forms tab of the toolbox to add a MapControl onto the form.
3. Repeat to add the AxPageLayoutControl, AxTOCControl, and AxToolbarControl.
4. Resize and reposition each control on the form as shown.



5. Double-click on the form to display the form's code window. Add the following 'using' directives to the top of the code window:

Remember that C# is case sensitive. If you start by typing "ESRI.", the auto completion feature of IntelliSense will allow you to complete the next section of code by pressing Tab.

```
using System;
using System.Windows.Forms;
using ESRI.ArcGIS.SystemUI;
using ESRI.ArcGIS.Carto;
using ESRI.ArcGIS.Display;
using ESRI.ArcGIS.Geometry;
using ESRI.ArcGIS.esriSystem;
using esriToolbarControl;
using esriTOCControl;
```

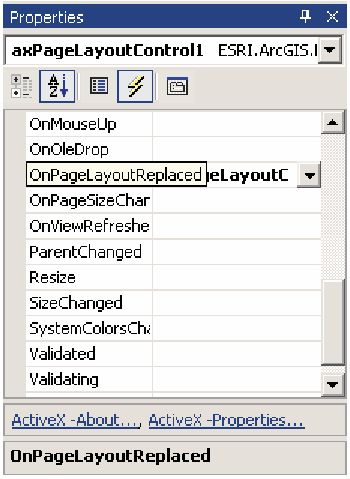
Loading Map Documents into the PageLayoutControl and MapControl

Individual data layers or map documents pre-authored using ArcMap, an ArcGIS Desktop application, can be loaded into the *MapControl* and *PageLayoutControl*. You can either load the sample map document provided or you can load in your own map document. Later you will add a dialog box to browse to a map document.

1. Select the Form_Load event and enter the following code (if you are using your own map document substitute the correct filename):

```
private void Form1_Load(object sender, System.EventArgs e)
{
//Load a pre-authored map document into the PageLayoutControl using
relative paths.
  string fileName =
@"..\..\..\..\..\..\..\Data\\ArcGIS_Engine_Developer_Guide\Gulf of St.
Lawrence.mxd";
  if (axPageLayoutControl1.CheckMxFile(fileName))
  {
    axPageLayoutControl1.LoadMxFile(fileName,"");
  }
}
```

2. Display the form in design mode and select axPageLayoutControl1 from the Properties window and display the axPageLayoutControl events. Double-click on the OnPageLayoutReplaced event to add an event handler to the code window.



3. In the axPageLayoutControl_OnPageLayoutReplaced event enter the following code to load the same map document into the *MapControl*. The

OnPageLayoutReplaced event will be triggered whenever a document is loaded into the *PageLayoutControl*.

```
private void axPageLayoutControl1_OnPageLayoutReplaced(object sender,
ESRI.ArcGIS.PageLayoutControl.IPageLayoutControlEvents_OnPageLayoutReplacedEvent
e)
{
  //Load the same pre-authored map document into the MapControl.
  axMapControl1.LoadMxFile(axPageLayoutControl1.DocumentFilename,null,
null);
  //Set the extent of the MapControl to the full extent of the data.
  axMapControl1.Extent = axMapControl1.FullExtent;
}
```

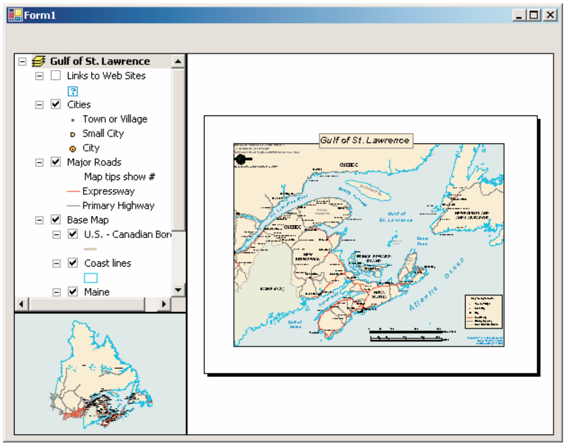
Setting the TOCControl and ToolbarControl Buddy Controls

For the purpose of this application the *TOCControl* and *ToolbarControl* will work in conjunction with the *PageLayoutControl*, rather than the *MapControl*. To do this the *PageLayoutControl* must be set as the buddy control. The *TOCControl* uses the buddy's ActiveView to populate itself with maps, layers, and symbols, while any command, tool, or menu items present on the *ToolbarControl* will interact with the buddy control's display.

1. In the Form_Load event enter the following after the load document code:

```
private void Form1_Load(object sender, System.EventArgs e)
{
  //Load a pre-authored map document into the PageLayoutControl using
relative paths.
  string fileName =
@"..\..\..\..\..\..\..\..\Data\\ArcGIS_Engine_Developer_Guide\Gulf of St.
Lawrence.mxd";
  if (axPageLayoutControl1.CheckMxFile(fileName))
  {
    axPageLayoutControl1.LoadMxFile(fileName,"");
  }

  //Set buddy controls.
  axTOCControl1.SetBuddyControl(axPageLayoutControl1);
  axToolbarControl1.SetBuddyControl(axPageLayoutControl1);
}
```

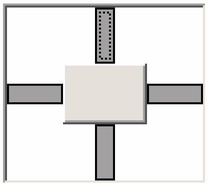
2. Build and run the application. The map document is loaded into the *PageLayoutControl*, and the *TOCControl* lists the data layers in the map document. Use the *TOCControl* to toggle layer visibility by checking and unchecking the boxes. By default, the focus map of the map document is loaded into the *MapControl*. At this point the *ToolbarControl* is empty because no commands have been added to it. Try resizing the form, and note that the controls do not change size.



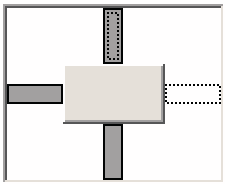
Handling Form Resize

When the form is resized at run time the *PageLayoutControl* and *MapControl* do not automatically resize themselves. To resize the controls so that they always fill the extent of the form, you must anchor the controls to the form. If the *PageLayoutControl* or *MapControl* contain a lot of data, redrawing this data during the Form_Resize can be costly. To increase performance you can suppress the data redraw until the resizing is complete. During the resize a stretched bitmap will be drawn instead.

1. Display the form in design mode and select axPageLayoutControl1 from the Properties window. Click on the anchor property and anchor the axPageLayoutControl to the top, left, bottom, and right of the form.



2. Anchor the axMapcontrol to the top, left, and bottom of the form.



3. Add the following code to the beginning of the Form_Load event:

```
private void Form1_Load(object sender, System.EventArgs e)
{
  //Suppress drawing while resizing
  this.SetStyle(ControlStyles.EnableNotifyMessage,true);
}
```

This method of suppressing resize drawing works by examining the windows messages sent to the form. When a form starts resizing, windows sends the WM_ENTERSIZEMOVE Windows(messge). At this point we suppress drawing to the MapControl and PageLayoutControl and draw using a "stretchy bitmap". When windows sends the WM_EXITSIZEMOVE the form is released from resizing and we resume with a full redraw at the new extent.

4. Add the following constants to the class:

```
public class Form1 : System.Windows.Forms.Form
{
  …
  private const int WM_ENTERSIZEMOVE = 0x231;
  private const int WM_EXITSIZEMOVE = 0x232;
  …
```

5. Add the following code to override the OnNotifyMessage method.

```
protected override void OnNotifyMessage(System.Windows.Forms.Message m)
{

  base.OnNotifyMessage (m);

  if (m.Msg == WM_ENTERSIZEMOVE)
  {
    axMapControl1.SuppressResizeDrawing(true, 0);
    axPageLayoutControl1.SuppressResizeDrawing(true, 0);
  }
  else if (m.Msg == WM_EXITSIZEMOVE)
  {
    axMapControl1.SuppressResizeDrawing(false, 0);
    axPageLayoutControl1.SuppressResizeDrawing(false, 0);
  }
}
```

6. Build and then run the application. Try resizing the Form.

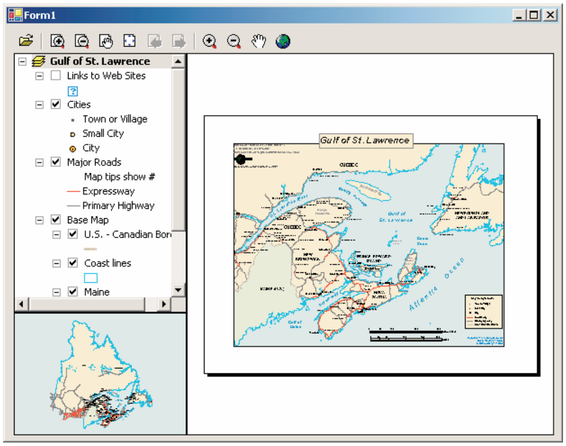
Adding Commands to the ToolbarControl

The *ArcGIS Engine* comes with over 120 commands and tools that work with the *MapControl*, the *PageLayoutControl* and the *ToolbarControl* directly. These commands and tools provide you with a lot of frequently used GIS functionality for map navigation, graphics management and feature selection. You will now add some of these commands and tools to your application.

1. In the Form_Load event add the following code before the load document code.

```
private void Form1_Load(object sender, System.EventArgs e)
{

  string progID;
  //Add generic commands
  progID = "esriControlToolsGeneric.ControlsOpenDocCommand";
  axToolbarControl1.AddItem(progID, -1 , -1, false, 0,
  esriCommandStyles.esriCommandStyleIconOnly);
  //Add PageLayout navigation commands
  progID = "esriControlToolsPageLayout.ControlsPageZoomInTool";
  axToolbarControl1.AddItem(progID, -1, -1, true, 0,
  esriCommandStyles.esriCommandStyleIconOnly);
  progID = "esriControlToolsPageLayout.ControlsPageZoomOutTool";
  axToolbarControl1.AddItem(progID, -1, -1, false, 0,
  esriCommandStyles.esriCommandStyleIconOnly);
  progID = "esriControlToolsPageLayout.ControlsPagePanTool";
  axToolbarControl1.AddItem(progID, -1, -1, false, 0,
  esriCommandStyles.esriCommandStyleIconOnly);
  progID = "esriControlToolsPageLayout.ControlsPageZoomWholePageCommand";
  axToolbarControl1.AddItem(progID, -1, -1, false, 0,
esriCommandStyles.esriCommandStyleIconOnly);
  progID =
"esriControlToolsPageLayout.ControlsPageZoomPageToLastExtentBackCommand";
  axToolbarControl1.AddItem(progID, -1, -1, false, 0,
esriCommandStyles.esriCommandStyleIconOnly);
  progID =
"esriControlToolsPageLayout.ControlsPageZoomPageToLastExtentForwardCommand";
  axToolbarControl1.AddItem(progID, -1, -1, false, 0,
esriCommandStyles.esriCommandStyleIconOnly);
  //Add Map naviagtion commands
  progID = "esriControlToolsMapNavigation.ControlsMapZoomInTool";
  axToolbarControl1.AddItem(progID, -1, -1, true, 0,
esriCommandStyles.esriCommandStyleIconOnly);
  progID = "esriControlToolsMapNavigation.ControlsMapZoomOutTool";
  axToolbarControl1.AddItem(progID, -1, -1, false, 0,
esriCommandStyles.esriCommandStyleIconOnly);
  progID = "esriControlToolsMapNavigation.ControlsMapPanTool";
  axToolbarControl1.AddItem(progID, -1, -1, false, 0,
esriCommandStyles.esriCommandStyleIconOnly);
  progID = "esriControlToolsMapNavigation.ControlsMapFullExtentCommand";
  axToolbarControl1.AddItem(progID, -1, -1, false, 0,
esriCommandStyles.esriCommandStyleIconOnly);

  //Load a pre-authored…
}
```

2. Build and then run the application. The *ToolbarControl* now contains ArcGIS Engine commands and tools that you can use to navigate the map document loaded into the *PageLayoutControl*. Use the page layout commands to navigate around the actual page layout and the map commands to navigate around the data present in the data frames. Use the open document command to browse and load other map documents.



Creating a Popup Menu for the PageLayoutControl

As well as adding ArcGIS Engine commands to the *ToolbarControl* to work with the buddy control, as in the previous step, you can also create popup menus from the ArcGIS Engine commands. You will add a popup menu that works with the *PageLayoutControl* to your application. The popup menu will display whenever the right mouse button is used on the display area of the *PageLayoutControl*.

1. Add the following member variable to the class:

```
public class Form1 : System.Windows.Forms.Form
{
  private ESRI.ArcGIS.ToolbarControl.AxToolbarControl
  axToolbarControl1;
  private ESRI.ArcGIS.TOCControl.AxTOCControl axTOCControl1;
  private ESRI.ArcGIS.MapControl.AxMapControl axMapControl1;
  private ESRI.ArcGIS.PageLayoutControl.AxPageLayoutControl
     axPageLayoutControl1;

  private IToolbarMenu m_ToolbarMenu = new ToolbarMenuClass();   //The
popup menu
  …
```

2. Add the following code to the Form_Load event after the code adding the commands to the *ToolbarControl*, but before the load document code.

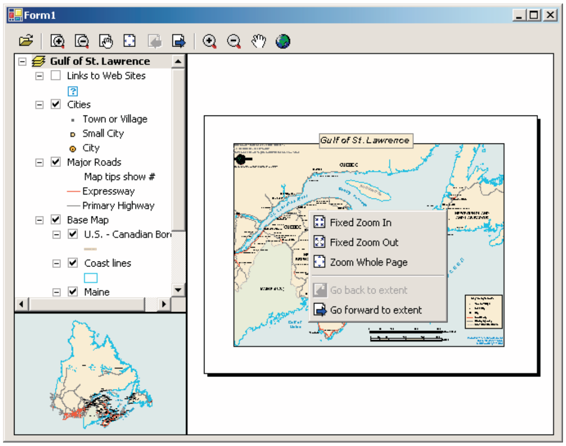
```
private void Form1_Load(object sender, System.EventArgs e)
{
  //Add Map naviagtion commands…
  //Share the ToolbarControl's command pool
  m_ToolbarMenu.CommandPool = axToolbarControl1.CommandPool;
  //Add commands to the ToolbarMenu
  progID = "esriControlToolsPageLayout.ControlsPageZoomInFixedCommand";
  m_ToolbarMenu.AddItem(progID, -1, -1, false,
esriCommandStyles.esriCommandStyleIconAndText);
  progID = "esriControlToolsPageLayout.ControlsPageZoomOutFixedCommand";
  m_ToolbarMenu.AddItem(progID, -1, -1, false,
esriCommandStyles.esriCommandStyleIconAndText);
  progID = "esriControlToolsPageLayout.ControlsPageZoomWholePageCommand";
  m_ToolbarMenu.AddItem(progID, -1, -1, false,
esriCommandStyles.esriCommandStyleIconAndText);
  progID =
"esriControlToolsPageLayout.ControlsPageZoomPageToLastExtentBackCommand";
  m_ToolbarMenu.AddItem(progID, -1, -1, true,
esriCommandStyles.esriCommandStyleIconAndText);
  progID =
"esriControlToolsPageLayout.ControlsPageZoomPageToLastExtentForwardCommand";
  m_ToolbarMenu.AddItem(progID, -1, -1, false,
esriCommandStyles.esriCommandStyleIconAndText);
  //Set the hook to the PageLayoutControl
  m_ToolbarMenu.SetHook(axPageLayoutControl1);

  //Load a pre-authored…
}
```

3. Display the Form in design mode and select axPageLayoutControl1 from the Properties window and display the axPageLayoutControl events. Double click on the OnMouseDown event to add an event handler to the code window.

4. In the axPageLayoutControl_OnMouseDown event add the following code:

```
private void axPageLayoutControl1_OnMouseDown(object sender,
ESRI.ArcGIS.PageLayoutControl.IPageLayoutControlEvents_OnMouseDownEvent
e)
{

  //Popup the ToolbarMenu
  if (e.button == 2)
  {
    m_ToolbarMenu.PopupMenu(e.x,e.y,axPageLayoutControl1.hWnd);
  }
}
```

5. Build and run the application. Right click on the PageLayoutControl's display area to display the popup menu, and navigate around the page layout.



Controlling Label Editing in the TOCControl

By default the *TOCControl* allows users to automatically toggle the visibility of layers and to change map and layer names as they appear in the table of contents. You will add code to prevent users from editing a name, and replacing it with an empty string.

1. Add the following code to the beginning of the Form_Load event.

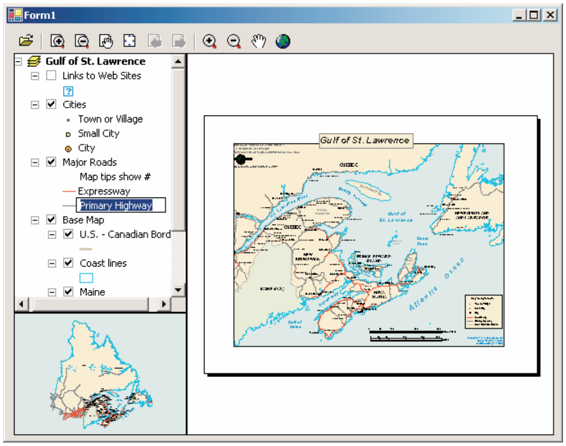
```
private void Form1_Load(object sender, System.EventArgs e)
{
  //Set label editing to manual
  axTOCControl1.LabelEdit = esriTOCControlEdit.esriTOCControlManual;

  //Add generic commands
}
```

2. Display the Form in design mode and select AxTOCControl1 from the Properties window and display the AxTOCControl events. Double click on OnEndLabelEdit to add an event handler to the code window.

3. Add the following code to the axTOCControl_OnEndLableEdit event

```
private void axTOCControl1_OnEndLabelEdit(object sender,
ESRI.ArcGIS.TOCControl.ITOCControlEvents_OnEndLabelEditEvent e)
{
  //If the new label is an empty string then prevent the edit
  string newLabel = e.newLabel;
  if (newLabel.Trim() == "")
  {
    e.canEdit = false;
  }
}
```

4. Build and run the application. To edit a map, layer, heading or legend class label in the *TOCControl*, click on it once, and then click on it a second time to invoke label editing. Try replacing the label with an empty string. You can use the ESC key on the keyboard at any time during the edit to cancel it.



Drawing Shapes on the MapControl

You will now use the *MapControl* as an overview window and draw on its display the current extent of the focus map within the *PageLayoutControl*. As you navigate around the data within the data frame of the *PageLayoutControl* you will see the *MapControl* overview window update.

Navigating around the focus map using the map navigation tools will change the extent of the focus map in the PageLayoutControl and cause the MapControl to update. Navigating around the page layout with the page layout navigation tools will change the extent of the page layout (not the extent of the focus map in the PageLayoutControl), so the MapControl will not update.

1. Add the following member variables to the class:

```
 public class Form1 : System.Windows.Forms.Form
{
  private ESRI.ArcGIS.ToolbarControl.AxToolbarControl axToolbarControl1;
  private ESRI.ArcGIS.TOCControl.AxTOCControl axTOCControl1;
  private ESRI.ArcGIS.MapControl.AxMapControl axMapControl1;
  private ESRI.ArcGIS.PageLayoutControl.AxPageLayoutControl
axPageLayoutControl1;

  private IToolbarMenu m_ToolbarMenu = new ToolbarMenuClass();
  private IEnvelope m_Envelope; //The envelope drawn on the MapControl
  private Object m_FillSymbol; //The symbol used to draw the envelope on
the  MapControl
  private ITransformEvents_VisibleBoundsUpdatedEventHandler
visBoundsUpdatedE; //The PageLayoutControl's focus map events
```

The variable declared as visBoundsUpdatedE is a delegate. A delegate is a class that can hold a reference to a specific method and link this to a specific event. The linking process between the event and the method is sometimes known in .NET as wiring.

2. Create a new function called CreateOverviewSymbol. This is where you will create the symbol used in the *MapControl* to represent the extent of the data in the focus map of the *PageLayoutControl*. Add the following code to the function.

```
private void CreateOverviewSymbol()
{
  //Get the IRGBColor interface
  IRgbColor color = new RgbColor();
  //Set the color properties
  color.RGB = 255;

  //Get the ILine symbol interface
  ILineSymbol outline = new SimpleLineSymbol();
  //Set the line symbol properties
  outline.Width = 1.5;
  outline.Color = color;

  //Get the IFillSymbol interface
  ISimpleFillSymbol simpleFillSymbol = new SimpleFillSymbolClass();
  //Set the fill symbol properties
  simpleFillSymbol.Outline = outline;
  simpleFillSymbol.Style = esriSimpleFillStyle.esriSFSHollow;
  m_FillSymbol = simpleFillSymbol;
}
```

3. Call the CreateOverviewSymbol function from the Form_Load event before the *TOCControl* label editing code.

```
private void Form1_Load(object sender, System.EventArgs e)
{
  //Create symbol used on the MapControl
  CreateOverviewSymbol();

  //Set label editing to manual…
}
```

4. Add the following OnVisibleBoundsUpdated function. This function will be linked to an event raised whenever the extent of the map is changed and is used to set the envelope to the new visible bounds of the map. By refreshing the *MapControl* you force it to redraw the shape on its display.

```
private void OnVisibleBoundsUpdated(IDisplayTransformation sender, bool
sizeChanged)
{
  //Set the extent to the new visible extent
  m_Envelope = sender.VisibleBounds;
  //Refresh the MapControl's foreground phase
  axMapControl1.ActiveView.PartialRefresh(esriViewDrawPhase.
esriViewForeground, null, null);
}
```

5. The default event interface of the *PageLayoutControl* is the IPageLayoutControlEvents. These events do not tell us when the extent of the map within the data frame changes. To do this you will use the

ItransformEvents interface of the *PageLayoutControl*'s focus map. Add the following code to the PageLayoutControl_OnPageLayoutReplaced event handler directly above the load document code.

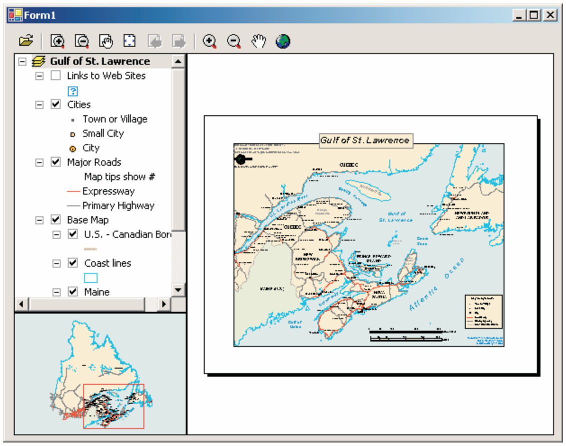
```
private void axPageLayoutControl1_OnPageLayoutReplaced(object sender,
ESRI.ArcGIS.PageLayoutControl.IPageLayoutControlEvents_OnPageLayoutReplacedEvent
e)
{
  //Get the IActiveView of the focus map in the PageLayoutControl
  IActiveView activeView = (IActiveView)
axPageLayoutControl1.ActiveView.FocusMap;
  //Trap the ITranformEvents of the PageLayoutCntrol's focus map
visBoundsUpdatedE = new
ITransformEvents_VisibleBoundsUpdatedEventHandler(OnVisibleBoundsUpdated);
  ((ITransformEvents_Event)activeView.ScreenDisplay.
DisplayTransformation).VisibleBoundsUpdated += visBoundsUpdatedE;
  //Get the extent of the focus map
  m_Envelope = activeView.Extent;

  //Load the same pre-authored map document into the MapControl
  axMapControl1.LoadMxFile(axPageLayoutControl1.DocumentFilename,null,
null);
  //Set the extent of the MapControl to the full extent of the data
  axMapControl1.Extent = axMapControl1.FullExtent;
}
```

6. Display the Form in design mode and select axMapControl1 from the Properties window and display the axMapControl events. Double click on OnAfterDraw to add an event handler to the code window.

7. Add the following code to the axMapControl_OnAfterDraw event handler to draw the envelope with the symbol you created earlier onto the *MapControl*'s display.

```
private void axMapControl1_OnAfterDraw(object sender,
ESRI.ArcGIS.MapControl.IMapControlEvents2_OnAfterDrawEvent e)
{
  if (m_Envelope == null)
  {
    return;
  }

  //If the foreground phase has drawn
  esriViewDrawPhase viewDrawPhase = (esriViewDrawPhase) e.viewDrawPhase;
  if (viewDrawPhase == esriViewDrawPhase.esriViewForeground)
  {
    IGeometry geometry = m_Envelope;
    axMapControl1.DrawShape(geometry, ref m_FillSymbol);
  }
}
```

8. Build and run the application. Use the map navigation tools that you added earlier to change the extent of the focus map in the *PageLayoutControl*. The new extent is drawn on the *MapControl*.



Creating a Custom Tool

Creating custom commands and tools to work with the *MapControl* and *PageLayoutControl* is very much like creating commands for the ESRI ArcMap application that you may have done previously. You will create a custom tool that adds a text element containing today's date to the *PageLayoutControl* at the location of a mouse click. You will however, create the command to work with the *MapControl* and *ToolbarControl* as well as the *PageLayoutControl*.

This scenario's source code is located at <install_location>\DeveloperKit\Samples\Developer_Guide_Scenarios\ArcGIS_Engine\Building_an_ArcGIS_Control_Application\Map_Viewer.

The code for this custom tool is available with the rest of this scenario's source code. If you want to use the custom command directly, rather than creating it yourself, go directly to Step 24.

1. Create a new Visual C# 'Class Library' project from the New project dialog box.
2. Name the project Commands, and browse to a location to save the project.
3. Click on the Project menu and choose Add Reference….
4. In the Add Reference dialog box, check 'ESRI.ArcGIS.Carto', 'ESRI.ArcGIS.Display', 'ESRI.ArcGIS.Geometry', 'ESRI.ArcGIS.System', 'ESRI.ArcGIS.SystemUI', 'ESRI.ArcGIS.Utility', and 'ESRI.ArcGIS.ControlCommands'.
5. Add one class to the project, named AddDateTool.
6. Click on the Project menu and choose Add Existing Item. Browse to the date.bmp from its location in this sample's source code and add it into your project.
7. Click on the date.bmp in the Solution Explorer window to display its properties in the Properties window. Change the Build Action property to Embedded Resource. This bitmap will be used on the face of the command button.

To change a namespace in Visual Basic .NET, right click on the project in the Solution Explorer and select Properties. In the project Property Pages select General and change the Root Namespace. Press OK.

8. Change the namespace of both the AddDateTool to be CSharpDotNETCommands.

```
namespace CSharpDotNETCommands
{
….
```

9. Add the following using directives to the top of the AddDateTool class code window.

```
using System;
using ESRI.ArcGIS.Carto;
using ESRI.ArcGIS.Display;
using ESRI.ArcGIS.Geometry;
using ESRI.ArcGIS.SystemUI;
using ESRI.ArcGIS.esriSystem;
using ESRI.ArcGIS.ControlCommands;
using ESRI.ArcGIS.Utility.BaseClasses;
using System.Runtime.InteropServices;
```

Abstract classes are classes that cannot be instantiated, and frequently contain only partial implementation code, or not implementation at all. They are closely related to interfaces; however, they differ significantly from interfaces in that a class may implement any number of interfaces, but it can inherit from only one abstract class. Inheriting the ESRI BaseTool abstract class will allow you to create commands and tools more quickly and simply than directly implementing the esriSystemUI ICommand and ITool interfaces.

10. Specify that the AddDateTool class inherits from the ESRI BaseTool abstract class. Also add the *sealed* class modifier.

```
public sealed class AddDateTool : BaseTool
```

The sealed class modifier states that a class cannot be inherited from. As this class is not designed for this purpose it is prudent to add this modifier to prevent other classes from inheriting this class.

The class constructor is a method that is called when the class is created. It can be used to set up members of the class. The constructor method has the same name as the class; it differs from other methods, in that it has no return type.

Instead of implementing the Bitmap, Caption, Category, Name, Message and Tooltip methods individually, you can set the values that should be returned from these methods, and rely on the BaseTool class to provide the implementation for these methods. The other members will be left to return the default values as implemented by the BaseTool class.

11. Add the following code to AddDateTool class constructor:

```
public AddDateTool()
{
  //Get an array resources in the assmebly
  string[] res = GetType().Assembly.GetManifestResourceNames();
  //Set the tool properties
  base.m_bitmap = new
System.Drawing.Bitmap(GetType().Assembly.GetManifestResourceStream(res[0]));
  base.m_caption = “Add Date”;
  base.m_category = "CustomCommands";
  base.m_message = "Adds a date element to the page layout";
  base.m_name = "CustomCommands_Add Date";
  base.m_toolTip = "Add date";
}
```

12. Add the following member variable to the AddDateTool class.

```
public sealed class AddDateTool : BaseTool
{
  //The HookHelper object that deals with the hook passed to the OnCreate
event
  private IHookHelper m_HookHelper = new HookHelperClass();
  …
```

13. In the Class View window, navigate to the BaseCommand OnCreate method and right click to display the context menu. Select Add, and then Override to add the property to the code window.

To override properties and methods in Visual Basic .NET, select (Overrides) from the 'Class Name' combo box and the property or method name from the 'Method Name' combo box at the top of the code window.

14. Add the following code to overridden OnCreate method.

```
public override void OnCreate(object hook)
{
  m_HookHelper.Hook = hook;
}
```

15. In the Class View window, navigate to the BaseCommand Enabled property and right click to display the context menu. Select Add, and then Override to add the property to the code window.

16. Add the following code to override the default Enabled value as implemented by the BaseTool class.

The ICommand_OnCreate event is passed a handle or hook to the application that the command will work with. In this case it can be a MapControl, PageLayoutControl or ToolbarControl. Rather than adding code into the OnCreate event to determine the type of hook that is being passed to the command, you will use the HookHelper to handle this.A command or tool needs to know how to handle the hook it gets passed, so a check is needed to determine the type of ArcGIS Control has been passed. The HookHelper is used to hold the hook and return the ActiveView regardless of the type of hook (in this case either a MapControl, PageLayoutControl or ToolbarControl).

```
public override bool Enabled
   {
     get
     {
       //Set the enabled property
       if (m_HookHelper.ActiveView != null)
       {
         return true;
       }
       else
       {
         return false;
       }
     }
   }
```

17. In the Class View window, navigate to the BaseTool OnMouseDown method and right click to display the context menu. Select Add, and then Override to add the property to the code window.

18. Add the following code to the override the default OnMouseDown functionality as implemented by the BaseTool class.

```
public override void OnMouseDown(int Button, int Shift, int X, int Y)
{
  base.OnMouseDown (Button, Shift, X, Y);

  //Get the active view
  IActiveView activeView = m_HookHelper.ActiveView;

  //Create a new text element
  ITextElement textElement = new TextElementClass();
  //Create a text symbol
  ITextSymbol textSymbol = new TextSymbolClass();
  textSymbol.Size = 25;

  //Set the text element properties
  textElement.Symbol = textSymbol;
  textElement.Text = DateTime.Now.ToShortDateString();

  //QI for IElement
  IElement element = (IElement) textElement;
  //Create a page point
  IPoint point = new PointClass();
  point = activeView.ScreenDisplay.DisplayTransformation.ToMapPoint(X,Y);
  //Set the elements geometry
  element.Geometry = point;

  //Add the element to the graphics container
  activeView.GraphicsContainer.AddElement(element, 0);
  //Refresh the graphics
  activeView.PartialRefresh(esriViewDrawPhase.esriViewGraphics, null,
null);
}
```

19. ArcGIS Engine expects the custom command to be a COM class; therefore you must specify that the .NET class you have created is also exposed as a COM class by creating a COM callable wrapper for it. In the Solution Explorer window, right click on the Commands project and select Properties from the context menu.

20. In the project Property Pages dialog box, select Configuration Properties; and then click Build. In the right-hand pane, change the Register for COM Interop property to True. Click OK.

Setting the Register for COM Interop property to True will invoke the Assembly Registration Tool (Regasm.exe). This will add the information about the class to the registry that a COM client would expect to find.

If the Register for COM Interop property is disabled, check that the project is a C# class library type.

A new GUID can be generated by using the GuidGen.exe utility included with Visual Studio .NET or by selecting Create GUID from the Tools menu. The GUID should be specified in the format shown, without curly braces.

21. In the code window of the AddDateTool class add the following code to the beginning of the AddDateTool class declaration, to specify attributes required by COM.

```
[ClassInterface(ClassInterfaceType.None)]
[Guid("D880184E-AC81-47E5-B363-781F4DC4528F")]
```

22. Add the following code to the AddDateTool class after the member variables. The code defines functions that register and unregister the AddDateTool class to the ESRI Controls Commands component category using the categories utility.

```
//Register in the 'ESRI Controls Commands' component category
#region Component Category Registration
[ComRegisterFunction()]
[ComVisible(false)]
static void RegisterFunction(String sKey)
{
  string fullKey = sKey.Remove(0, 18) + @"\Implemented Categories";
  Microsoft.Win32.RegistryKey regKey =
    Microsoft.Win32.Registry.ClassesRoot.OpenSubKey(fullKey, true);
  if (regKey != null)
  {
      regKey.CreateSubKey("{B284D891-22EE-4F12-A0A9-
  B1DDED9197F4}");
  }
}
[ComUnregisterFunction()]
[ComVisible(false)]
static void UnregisterFunction(String sKey)
{
  string fullKey = sKey.Remove(0, 18) + @"\Implemented Categories";
  Microsoft.Win32.RegistryKey regKey =
    Microsoft.Win32.Registry.ClassesRoot.OpenSubKey(fullKey, true);
  if (regKey != null)
```

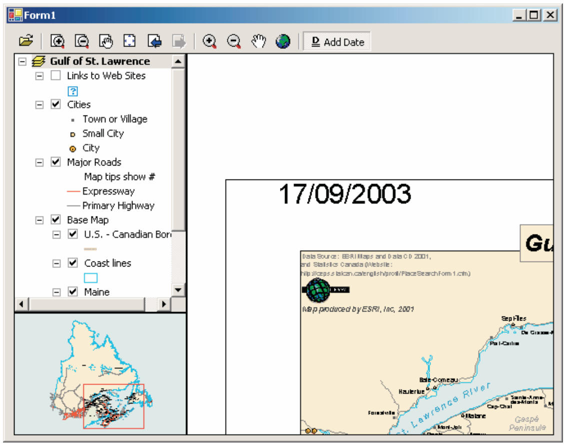
```
    {
      regKey.DeleteSubKey("{B284D891-22EE-4F12-A0A9-B1DDED9197F4}");
    }
  }
  #endregion
```

23. Build the project.

24. In the Visual Studio .NET 'Windows Application' project that you created at the beginning of this scenario, add the following code after the code to add the map navigation commands.

```
private void Form1_Load(object sender, System.EventArgs e)
{
  //Add Map naviagtion commands…

  //Add custom date tool
  progID = "CSharpDotNETCommands.AddDateTool";
  axToolbarControl1.AddItem(progID, -1, -1, true, 0,
   esriCommandStyles.esriCommandStyleIconAndText);

  //Add commands to the ToolbarMenu
}
```

25. Build and run the application and use the AddDateTool to add a text element to the *PageLayoutControl* containing today's date.



Customizing the ToolbarControl

As well as adding ArcGIS Engine commands and tools to the *ToolbarControl* in the Form_Load event, you can also add them by customizing the *ToolbarControl* and using the customize dialog box. To do this you will place the *ToolbarControl* in customize mode and display the customize dialog box.

1. Add the following member variables to the class:

```
public class Form1 : System.Windows.Forms.Form
{
…
  private ITransformEvents_VisibleBoundsUpdatedEventHandler
    visBoundsUpdatedE;
  private ICustomizeDialog m_CustomizeDialog = new
        CustomizeDialogClass(); //The CustomizeDialog used by the ToolbarControl
  private ICustomizeDialogEvents_OnStartDialogEventHandler
    startDialogE; //The CustomizeDialog start event
  private ICustomizeDialogEvents_OnCloseDialogEventHandler
    closeDialogE; //The CustomizeDialog close event
```

Visual Studio .NET provides the ability to specify functions that execute when an assembly exposed for COM interop is registered and unregistered on a system. This allows you to register your class in a component category that the Customize Dialog box will look for.

2. Create a new function called CreateCustomizeDialog. This is where you will create the customize dialog box by adding the following code to the function:

```
private void CreateCustomizeDialog()
{
  //Set the customize dialog box events
  startDialogE = new
  ICustomizeDialogEvents_OnStartDialogEventHandler(OnStartDialog);
  ((ICustomizeDialogEvents_Event)m_CustomizeDialog).OnStartDialog +=
startDialogE;
  closeDialogE = new
  ICustomizeDialogEvents_OnCloseDialogEventHandler(OnCloseDialog);
  ((ICustomizeDialogEvents_Event)m_CustomizeDialog).OnCloseDialog +=
closeDialogE;

  //Set the title
  m_CustomizeDialog.DialogTitle = "Customize ToolbarControl Items";
  //Show the 'Add from File' button
  m_CustomizeDialog.ShowAddFromFile = true;
  //Set the ToolbarControl that new items will be added to
  m_CustomizeDialog.SetDoubleClickDestination(axToolbarControl1);
}
```

The ComVisible attribute is set to false to ensure that this method cannot be called directly by a COM client. It does not affect the method being called when the assembly is registered with COM.

3. Call the CreateCustomizeDialog function from the Form_Load event before the call to the CreateOverviewSymbol sub routine.

```
private void Form1_Load(object sender, System.EventArgs e)
{
  //Create the customize dialog box for the ToolbarControl
  CreateCustomizeDialog();

  //Create symbol used on the MapControl…
}
```

4. Add a check box to the Form and give it the name 'chkCustomize' and the caption 'Customize'.
5. Display the Form in design mode and select chkCustomize from the Properties window and display the chkCustomize events. Double click on CheckedChanged to add an event handler to the code window.
6. Add the following code to the chkCustomize_ CheckedChanged event.

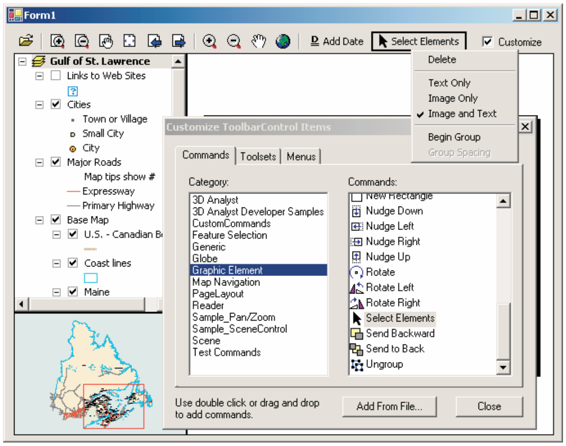
```
private void chkCustomize_CheckedChanged(object sender, System.EventArgs
e)
{
  //Show or hide the customize dialog
  if (chkCustomize.Checked == false)
  {
    m_CustomizeDialog.CloseDialog();
    axToolbarControl1.Customize = false;
  }
  else
  {
    m_CustomizeDialog.StartDialog(axToolbarControl1.hWnd);
    axToolbarControl1.Customize = true;
  }
}
```

7. Add the following OnStartDialog and OnCloseDialog event handlers. These functions will be wired to events raised whenever the customize dialog box is opened or closed.

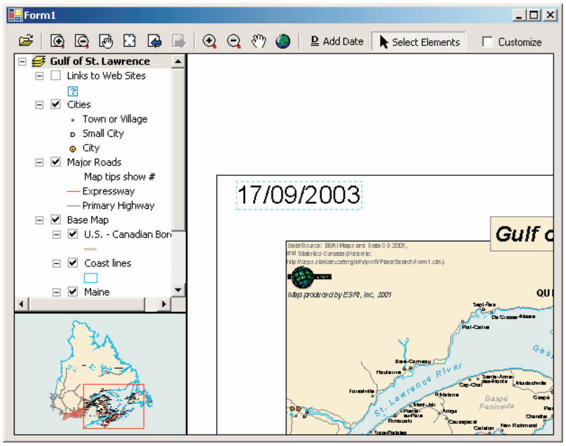
```
private void OnStartDialog()
{
  axToolbarControl1.Customize = true;
}

private void OnCloseDialog()
{
  axToolbarControl1.Customize = false;
  chkCustomize.Checked = false;
}
```

8. Build and run the application and check the customize box to put the *ToolbarControl* into customize mode and open the customize dialog box.

9. On the Commands tab select the Graphic Element category, and double click on the Select Elements command to add it to the *ToolbarControl*. By right clicking on an item on the *ToolbarControl* you can adjust its appearance in terms of style, and grouping.



10. Stop customizing the application. Use the select tool to move the text element containing today's date.



DEPLOYMENT

When developing a stand-alone executable using ESRI ArcObject's it is the responsibility of the application to check and configure the licensing options. The CoClass AoInitialize and the IAoInitialize interface it implements are designed to support license configuration. License initialization must be preformed at application start time, before any ESRI ArcObject functionality is accessed. Failure to do so will resolve in application errors.

To successfully deploy this application onto another machine, the application must configure a license. Firstly, it must check that the product license is available and secondly it must initialize the license. If this license configuration fails, the application cannot run.

1. Add the following member variable to the class.

```
public class Form1 : System.Windows.Forms.Form
{

  …
  private IAoInitialize m_AoInitialize = new AoInitializeClass();
  //The initialization object
  private IToolbarMenu m_ToolbarMenu = new ToolbarMenuClass();
  //The popup menu
  …
}
```

2. Add the following code to the very beginning of the Form_Load event.

```
private void Form1_Load(object sender, System.EventArgs e)
{
  //Create a new AoInitialize object
  if (m_AoInitialize == null)
  {
    System.Windows.Forms.MessageBox.Show("Unable to
   initialize. This application cannot run!");
    this.Close();
  }
  //Determine if the product is available
  esriLicenseStatus licenseStatus = (esriLicenseStatus)
m_AoInitialize.IsProductCodeAvailable(
esriLicenseProductCode.esriLicenseProductCodeEngine);
  if (licenseStatus == esriLicenseStatus.esriLicenseAvailable)
  {
    licenseStatus = (esriLicenseStatus)
    m_AoInitialize.Initialize(esriLicenseProductCode.
esriLicenseProductCodeEngine);
    if (licenseStatus != esriLicenseStatus.esriLicenseCheckedOut)

    {
      System.Windows.Forms.MessageBox.Show("The
   initialization failed. This application cannot
     run!");
      this.Close();
    }
  }
  else
  {
    System.Windows.Forms.MessageBox.Show("The ArcGIS Engine
   product is unavailable. This application cannot run!");
    this.Close();
  }
  //Create the customize dialog box for the ToolbarControl
  …
}
```

3. Display the Form in design mode and select Form1 from the Properties window and display the Form events. Double click on the Closing event to add an event handler to the code window.

4. In the Form_Closing event add the following code:

```
private void Form1_Closing(object sender,
System.ComponentModel.CancelEventArgs e)
{
  //Release COM objects and shut down the AoInitilaize object
  ESRI.ArcGIS.Utility.COMSupport.AOUninitialize.Shutdown();
  m_AoInitialize.Shutdown();
}
```

5. Build the project and build the solution in release mode.

To successfully deploy this application onto a users machine:

- The application's executable and the DLL containing the custom command will need to be deployed onto the users machine. The Assembly Registration Tool (RegAsm.exe) must be used to add information about the custom class to the registry.
- The users machine will need an installation of the ArcGIS Engine Runtime and a standard ArcGIS Engine license.
- The users machine will need an installation of the Microsoft .NET Framework 1.1.

ADDITIONAL RESOURCES

The following resources may help you understand and apply the concepts and techniques presented in this scenario.

- Additional documentation available in the ArcGIS Engine Developer Kit including *ArcGIS Developer Help*, component help, object model diagrams, and samples to help you get started.
- ArcGIS Developer Online—Web site providing the most up-to-date information for ArcGIS Developers including updated samples and technical documents. Go to http://arcgisdeveloperonline.esri.com.
- ESRI online discussion forums—Web sites providing invaluable assistance from other ArcGIS developers. Go to http://support.esri.com and click the User Forums tab.
- Microsoft documentation on the Visual Studio .NET development environment.

This walkthrough is intended for programmers who want to learn more about the Java API in ArcGIS Engine. To get the most out of this scenario you should understand basic Java programming concepts such as classes, inheritance, and using packages. Some familiarity with ArcObjects will also be helpful, although not required. Although this scenario does require conceptual knowledge of the Java language, it does not require a lot of programming experience. The code used in this example provides an easy entry point to learn about the Java API in ArcGIS Engine on a small and simple scale.

Rather than walk through this scenario, you can get the completed application from the samples installation location. The sample is installed as part of the ArcGIS developer samples.

The ArcGIS developer samples are not included in the 'typical' installation of the ArcGIS Engine Developer Kit. If you do not have them installed, rerun the Developer Kit install wizard, choose Custom or Modify, and select the samples feature under Software Developer Kit.

You can find this sample in

`<install_location>\DeveloperKit\Samples\Developer_Guide_Scenarios\ArcGIS_Engine\Building_a_Command_Line_Application\Converting_A_Tin_To_Point_Shapefile`

PROJECT DESCRIPTION

This scenario will cover several aspects of the ArcGIS Engine API. The goal of this exercise is to create a standalone command line application with the ArcGIS Engine Java API. The application will take as input a TIN representation of a surface and create a three-dimensional shapefile representing the interpolated TIN nodes. Once you have completed this scenario, you will understand the techniques required to work with the ArcGIS Engine Java API.

CONCEPTS

Terrain data is collected mostly as a sequence of discrete (X, Y, Z) data points. Digital terrain models (DTM) are generally organized such that mass points lie in a grid pattern or they represent nodes of triangles in an array referred to as triangulated irregular network or TIN. The nodes will be converted to points and used to create a new feature class. This exercise will use the TIN object and the *ITinAdvanced* interface it implements. *ITinAdvanced* provides access to basic properties and is a starting point to the underlying data structure. In addition, the scenario utilizes the *GeometryDef* and *FieldsEdit* objects to populate the newly created feature class.

DESIGN

The application will be written entirely in the Java language. This allows you to write code once on any platform and deploy the application on any supported ArcGIS Engine platform. This scenario will use Microsoft Windows XP as the developer platform, but can easily be followed on any UNIX-based developer platforms.

We will use Ant, a cross-platform Java-based build tool, to build and deploy the scenario. Ant executes tasks implemented as Java classes, which allow it to inherit the platform independence of Java. ArcGIS Engine Developer Kit includes an extended version of Ant called arcgisant. This scenario will use arcgisant, but you are free to use any version of Ant 1.5.x or greater.

REQUIREMENTS

The Java API is not included in the 'typical' installation of the ArcGIS Engine Developer Kit. If you do not have it installed, rerun the Developer Kit install wizard, choose Custom or Modify, and select the Java feature under ArcGIS Engine. In addition, for access to the Javadoc and other Java-specific documentation, select the Java feature under Software Developer Kit.

In order to successfully follow this scenario you need the following (the requirements for deployment are covered later in the Deployment section):

- An installation of the ArcGIS Engine Developer Kit (including Java) with an authorization file enabling it for development use.

- An installation of the Java 2 Platform, Standard Edition (J2SE) Software Development Kit (SDK), preferably 1.4.2 or later. If you do not already have one available, download it from the Java Website at http://java.sun.com/j2se/downloads.html
- A Java IDE of your choice or your favorite text editor.
- An understanding of basic Java programming concepts such as classes, inheritance, and using packages.
- While no experience with other ESRI software is required, previous experience with ArcObjects is helpful.
- A TIN dataset
- Access to the sample data and code that comes with this scenario.

The ArcGIS developer samples are not included in the 'typical' installation of the ArcGIS Engine Developer Kit. If you do not have them installed, rerun the Developer Kit install wizard, choose Custom or Modify, and select the samples feature under Software Developer Kit.

```
<install_location>\DeveloperKit\Samples\Developer_Guide_Scenarios\ArcGIS_Engine
\Building_a_Command_Line_Application\Converting_A_Tin_To_Point_Shapefile
```

Objects from the following libraries will be used:

- DataSourcesFile
- GeoDatabase
- Geometry
- System

In the Java API, their package names se are prefixed by 'com.esri.arcgis'.

IMPLEMENTATION

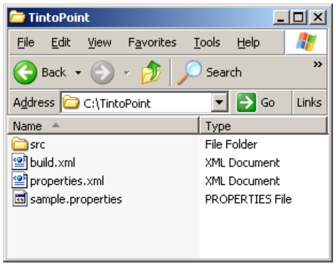
To implement this scenario follow the steps below. This implementation provides you with all the code you will need to successfully complete the scenario. It does not provide step-by-step instructions to develop applications in Java, as it assumes that you have a working knowledge of the development environment already.

Setting up environment variables

Any windows user can add, modify, or remove a user environment variable. Setting such environment variables will make most effective use of this scenario. You will add three environment variables and their respective executable (bin) folders to the global PATH variable.

1. Right-click My Computer, and then click Properties.
2. Click the Advanced tab.
3. Click Environment Variables.
4. Under System Variables click New.
   For Variable name: type 'ARCENGINEHOME'.
   For Variable value: type in the root level ArcGIS Engine install directory (for example, C:\ArcGIS).
5. Click OK.
6. Under System variables single-click on Path and click Edit.
7. Append the following to the beginning of Variable value:
   %ARCENGINEHOME%\bin;
8. Click OK until you have closed all System Properties dialog boxes.

To compile and run applications using the ArcGIS Engine Developer Kit, the PATH environment variable should include paths to ArcGIS/bin and J2SE SDK/bin. In addition, ArcGIS Engine Developer Kit ships with an extended version of Ant called arcgisant*. The path to this tool should also be included.*



C:\TintoPoint

Name	Type
src	File Folder
build.xml	XML Document
properties.xml	XML Document
sample.properties	PROPERTIES File

Folder and file structure required in order for the build scripts to work as desired.

Repeat the steps above to add the following environment variables:

- JAVA_HOME=J2SE SDK install directory
- ANT_HOME=%ARCENGINEHOME\DeveloperKit\tools\arcgisant

Before proceeding with building the application itself, you need to prepare the build scripts. It is vital to set up the build structure as illustrated to the left in order for the scripts to work correctly.

- TintoPoint folder—root folder for the project
- src folder—subfolder containing all of the application's source code
- build.xml—Ant build script file.
- properties.xml—external Ant properties file, which extends the build environment.
- sample.properties—external Java properties file with command line parameters.

The three files—build.xml, properties.xml, and sample.properties—must be created before you can build and deploy the scenario.

Creating the samples.properties file

Let's begin by creating the sample.properties file. This file provides the build script with necessary command line arguments to successfully execute the application created in this exercise. This file will use the *variable=argument* pattern.

1. Create a text file named sample.properties and add the following lines of code to it. Revise the arguments for the variables input.tin.path and output.shape.path to match the paths to your TIN dataset and output shapefile dataset.

```
# TinToPoint
unit.name=TintoPoint
main.class=engine.scenario.analyst3d.TintoPoint
#TinToPoint command line args
input.tin.path=\<path to TIN dataset>
output.shape.path=\<path to generated shapefile dataset>
```

2. Save and close the file.

Creating the properties.xml file

The properties.xml file sets Ant properties for the build environment. Ant properties can be set explicitly or loaded from a file. For simplicity, you will add the dependent properties from a file, properties.xml.

1. Create an XML file named properties.xml and add the following Ant properties:

```
<!– :mode=ant –>

<!– ========================================== –>
<!– load environment variables –>
<!– ========================================== –>
<property environment="env"/>
```

```
<property name="arcengine.home" value="${env.ARCENGINEHOME}" />
<property name="ant.home" value="${engine.home}/developerkit/tools/ant"/>

<!– ========================================== –>
<!– directory mappings –>
<!– ========================================== –>
<property name="root.dir" location="${basedir}"/>
<property name="src.dir" location="src"/>
<property name="build.dir" location="build"/>
<property name="class.dir" location="${build.dir}/classes"/>

<!– ========================================== –>
<!– library dependency settings –>
<!– ========================================== –>
<!– library directory mappings –>
<property name="arcgis.dir"  location="${engine.home}/java"/>
<!– each library has its own unique directory structure –>
<property name="arcgis.subdir" value="opt"/>
<!– jar file mappings –>
<property name="jintegra.jar" location="${arcgis.dir}/jintegra.jar"/>
<property name="arcobjects.jar" location="${arcgis.dir}/${arcgis.subdir}/
    arcobjects.jar"/>
```

2. Save and close the file.

Creating the build.xml file

Ant build scripts are written in XML and contain one project and at least one task. Each project defines one or more targets that combine tasks for execution. In this scenario, the project name is "ArcGIS Engine Developer Scenario" and its build script needs to contain the four private and three public targets, as listed respectively below:

To learn more about private and public targets used in Ant scripts, see the Ant documentation available from the Apache Ant Web site, http://ant.apache.org/.

- init—creates the build directory structure.
- validate-engine—ensures that the ArcGIS Engine Developer environment is properly set.
- compile—compiles the scenario.
- execute—runs the scenario in a separate JVM instance.
- all—the default target, which builds the entire scenario.
- clean—cleans all build products.
- run-scenario—builds and runs the scenario application.

In order to get an idea of the structure of Ant targets, let's have a look at a sample compile target line by line.

```
<target name="compile" depends="validate-engine">
```

As shown in the first line, this target has a name to reference and depends on another target. To ensure a successful build, the 'depends' attribute is used to confirm that the environment is set correctly.

```
          <!– compile the java code from ${src.dir} into ${class.dir} –>
          <javac srcdir="${src.dir}" destdir="${class.dir}">
                    <classpath refid="compile.classpath"/>
```

In the last section of this sample target, one of Ant's core tasks, *'javac'*, is created to compile the files in the source directory and place them in the class directory. The *javac* task supports a classpath task to ensure the classpath settings.

The complete sample target, including closure of the XML tags, then is as shown below:

```
<target name="compile" depends="validate-engine">
          <!– compile the java code from ${src.dir} into ${class.dir} –>
          <javac srcdir="${src.dir}" destdir="${class.dir}">
                    <classpath refid="compile.classpath"/>
          </javac>
</target>
```

The entire build script is not discussed in detail here; to learn more about building Ant scripts see the Ant documentation available from the Apache Ant Web site, http://ant.apache.org/.

Now that you've got a basic understanding of targets and their usage in Ant, let's proceed with creating the build.xml file for this scenario.

1. Create an XML file named build.xml and add the following:

```
<?xml version="1.0"?>
<!DOCTYPE project[
  <!ENTITY properties SYSTEM "file:properties.xml">
]>
<!– :mode=ant –>

<project name="ArcGIS Engine Developer Scenario" default="all"
basedir=".">
 <!– import external XML fragments –>
 &properties;
 <!– import sample properties –>
 <property file="sample.properties"/>

  <path id="compile.classpath">
    <pathelement location="${jintegra.jar}"/>
    <pathelement location="${arcobjects.jar}"/>
  </path>

  <path id="run.classpath">
    <path refid="compile.classpath"/>
    <pathelement location="${class.dir}"/>
  </path>

<!– ========================================== –>
<!– private targets –>
<!– ========================================== –>
 <target name="init">
  <!– create the time stamp –>
  <tstamp/>
  <!– create the build directory structure used by compile –>
    <mkdir dir="${build.dir}"/>
    <mkdir dir="${class.dir}"/>
 </target>

 <target name="validate-engine" depends="init">
  <condition property="engine.available">
    <and>
      <isset property="env.ARCENGINEHOME"/>
    </and>
  </condition>
  <fail message="Missing dependencies: ARCENGINEHOME environment
  variable not correctly set" unless="engine.available"/>
 </target>

 <target name="compile" depends="validate-engine">
  <!– javac resolver needed to run inside of Websphere Studio –>
  <available classname="org.eclipse.core.launcher.Main"
    property="build.compiler"
  value="org.eclipse.jdt.core.JDTCompilerAdapter"
  classpath="${java.class.path}" />
  <!– compile the java code from ${src.dir} into ${class.dir} –>
  <javac srcdir="${src.dir}" destdir="${class.dir}">
    <classpath refid="compile.classpath"/>
  </javac>
 </target>

 <target name="execute" depends="compile" if="input.tin.path">
  <java classname="${main.class}" failonerror="true"
   fork="true">
    <classpath refid="run.classpath"/>
    <!– values must be set correctly in sample.properties –>
      <arg value="${input.tin.path}"/>
      <arg value="${output.shape.path}"/>
  </java>
 </target>

<!– ========================================== –>
<!– public targets –>
<!– ========================================== –>

 <target name="all" depends="compile" description="build everything">
    <echo message="application built"/>
 </target>

 <target name="clean" description="clean all build products">
  <!– delete the ${build} directory trees –>
  <delete dir="${build.dir}"/>
  <echo message="build directory gone!"/>
 </target>

 <target name="run-scenario" depends="execute" description="execute the
  sample with args set in sample.properties"/>
</project>
```

2. Save and close the file.

Testing your build environment

Now that you've set up all the necessary files for building your application, let's test it before proceeding.

1. Open a command prompt and cd into the root folder of your project. For example,

   ```
   cd C:\TintoPoint
   ```

2. You should have all the build script files located in this root directory.
3. Type 'arcgisant' at the command prompt.
4. You should receive output similar to the following:

   ```
   Buildfile: build.xml
    init:
       [mkdir] Created dir: Q:\dop\dev\ant\projects\engine.scenario\build
       [mkdir] Created dir: Q:\dop\dev\ant\projects\
        engine.scenario\build\classes
    validate-engine:
    compile:
   BUILD FAILED
    file:Q:/dop/dev/ant/projects/engine.scenario/build.xml:49: srcdir
   "Q:\dop\dev\ant\projects\engine.scenario\src" does not exist!
   Total time: 1 second
   ```

The build should fail since you do not have any source to build yet. If you look on disk at your project directory, you should notice a build folder was created. This is where all build products will be generated.

Clean up the build with the following command:

```
arcgisant clean
```

This should generate output similar to the following:

```
Buildfile: build.xml
clean:
   [delete] Deleting directory Q:\dop\dev\ant\projects\engine.scenario\build
     [echo] build directory gone!

BUILD SUCCESSFUL
Total time: 1 second
```

Now that your build environment is prepared, youe can proceed with writing the Java source code for the example application.

Creating the TintoPoint main class

Use your favorite text editor or IDE to write your source code.

1. Begin you source code by adding the signature for the main class in this exercise:

   ```
   package engine.scenario.analyst3d;

   public class TintoPoint{}
   ```

The class will eventually have three private static methods and one public main entry point. For the sake of simplicity, private static methods are used to do the work for the application. While some Java developers believe that, when a method can be either static or an instance, an instance method should be utilized, this scenario uses the simplest approach—static methods. Each of these methods will be covered in upcoming sections.

Next, let's look at what imports are required by the class.

The datasourcesfile, geodatabase, geometry, and system Java packages are the equivalent of the DataSourcesFile, GeoDatabase, Geometry, and System libraries of ArcGIS Engine.

As discussed earlier, this scenario uses the datasourcesfile, geodatabase, geometry, and system Java packages.

- datasourcesfile—provides workspace factories and workspaces for vector data formats supported by the geodatabase API. You will be use the *ShapefileWorkspaceFactory* class in this exercise to create your generated 3D shapefile.
- geodatabase—provides all definitions relating to data access including Tins and FeatureClasses. You will be use the *GeometryDef* class to define spatial qualities for your generated feature class.
- geometry—contains the core geometry objects, as well as spatial reference information. In this scenario, you will use the *Point* class as a representation of all the TIN nodes in your input TIN dataset.
- system—contains objects that expose services used by the other libraries within ArcGIS Engine. You will use the use *AoInitializer* object to initialize and un-initialize your application.

2. Below the package declaration you made above, import the classes as shown. Use fully qualified imports so you can see explicitly which classes from the ArcGIS Engine Java API you are working with.

```
import java.io.File;
import java.io.IOException;

import com.esri.arcgis.datasourcesfile.ShapefileWorkspaceFactory;
import com.esri.arcgis.geodatabase.Field;
import com.esri.arcgis.geodatabase.Fields;
import com.esri.arcgis.geodatabase.GeometryDef;
import com.esri.arcgis.geodatabase.IEnumTinNode;
import com.esri.arcgis.geodatabase.IFeatureBuffer;
import com.esri.arcgis.geodatabase.IFeatureClass;
import com.esri.arcgis.geodatabase.IFeatureClassProxy;
import com.esri.arcgis.geodatabase.IFeatureCursor;
import com.esri.arcgis.geodatabase.IFeatureWorkspace;
import com.esri.arcgis.geodatabase.IFeatureWorkspaceProxy;
import com.esri.arcgis.geodatabase.IFieldEdit;
import com.esri.arcgis.geodatabase.IFields;
import com.esri.arcgis.geodatabase.IFieldsEdit;
import com.esri.arcgis.geodatabase.ITinAdvanced;
import com.esri.arcgis.geodatabase.ITinNode;
import com.esri.arcgis.geodatabase.IWorkspaceFactory;
import com.esri.arcgis.geodatabase.Tin;
import com.esri.arcgis.geodatabase.esriFeatureType;
import com.esri.arcgis.geodatabase.esriFieldType;
import com.esri.arcgis.geodatabase.esriTinQualification;
import com.esri.arcgis.geometry.IPoint;
import com.esri.arcgis.geometry.ISpatialReference;
import com.esri.arcgis.geometry.Point;
import com.esri.arcgis.geometry.esriGeometryType;
import com.esri.arcgis.system.AoInitialize;
import com.esri.arcgis.system.EngineInitializer;
import com.esri.arcgis.system.esriLicenseExtensionCode;
import com.esri.arcgis.system.esriLicenseProductCode;
```

Now that the imports have all been made, you can add methods to your class.

Adding the tinToPoint method

This method performs most of the work for the application. It will take, as parameters, a path to your TIN file, the name of the TIN, a path to an output shapefile, and a name for the output shapefile.

1. Create the signature for the *tinToPoint* method as follows:

```
/**
 * @param tinPath - path to input tin data source
 * @param tinName - name of input tin data source
 * @param shapePath - path to output shapefile data source
 * @param shapeFile - name of output shapefile data source
*/
private static void tinToPoint(String tinPath, String tinName,
  String shapePath, String shapeFile){
```

Initially, you need to open your TIN dataset from its file location, which is passed as a parameter to the method, by instantiating a new *Tin* class. The *Tin* class implements many interfaces; you will use methods provided by the *ITinAdvanced* and *IGeoDataset* interfaces. By calling the *init()* method, exposed by the *ITinAdvanced* interface, you can open the specified TIN.

The ITinAdvanced *interface requires an ArcGIS Engine Runtime license with 3D option or an ArcGIS 3D Analyst extension when deployed. You will add code for detecting for this license and checking it out later in this exercise.*

Note about try/catch here.TODO

2. Instantiate the new *Tin* class by adding the following code below the signature for your method.

```
try{
  //Get tin from tin file
  ITinAdvanced tinAdv = new Tin();
  String path = tinPath + File.separator + tinName;
  System.out.println("  - Path to Tin: " + path);
  tinAdv.init(path);

  System.out.println("  - Calculating ... ");

  Tin tin = new Tin(tinAdv);
```

3. Next, get the spatial reference of the *Tin* and send it as a parameter to a *createBasicFields()* method that will be defined in the next section. In addition, you need to send a geometry type point since your resulting feature class will be a point feature class.

```
ISpatialReference tinSpatialRef = tin.getSpatialReference();
IFields fields = createBasicFields(esriGeometryType.esriGeometryPoint,
   false, true, tinSpatialRef);
```

4. Now that the basic fields have been generated, the next step is to create an output shapefile. *WorkspaceFactory* is used as a dispenser of workspaces to create an instance of a *ShapefileWorkspaceFactory* class. The *IFeatureWorkspace* interface is used to access and manage datasets. You will cross-cast to the returned object that implements *IWorkspace* by creating an instance of *IFeatureWorkspaceProxy* class using its object constructor. Finally, your output shapefile is generated using the *createFeatureClass()* method to create a standalone feature class.

```
   // create output shapefile
IWorkspaceFactory wkspFactory = new ShapefileWorkspaceFactory();
   IFeatureWorkspace featureWksp = new IFeatureWorkspaceProxy(
             wkspFactory.openFromFile(shapePath, 0));
   IFeatureClass outFC = new IFeatureClassProxy(
featureWksp.createFeatureClass( shapefile,
        fields,
        null,
        null,
        esriFeatureType.esriFTSimple,
        "Shape",
        ""));
```

5. The final step in creating this method is to populate the newly created feature class with the value of the nodes from your input TIN. The *makeNodeEnumerator()* method returns enumerations of nodes based on an extent and a criteria. The extent used is that of the input TIN dataset and all data inside the TIN is used as the criteria. Once an enumeration object is filled with your TIN nodes, use a *FeatureCursor* as a data access object to iterate over the set of rows in your newly created feature class and a *FeatureBuffer* object to hold the state of the row. Next, create an instance of the *Point* class and populate it with your TIN nodes by instantiating an *ITinNode* interface with the objects returnd by your enumeration. While your node is not null, loop through them and populate your feature class.

```
   // get tin node enum
IEnumTinNode enum = tin.makeNodeEnumerator(tin.getExtent(),
                  esriTinQualification.esriTinInsideDataArea, null);

IFeatureCursor outCursor = outFC.IFeatureClass_insert(true);
IFeatureBuffer outBuffer = outFC.createFeatureBuffer();
IPoint point = new Point();
ITinNode node = enum.IEnumTinNode_next();
while(node != null){
   node.queryAsPoint(point);
   outBuffer.setShapeByRef(point);
   outCursor.insertFeature(outBuffer);
   node = enum.IEnumTinNode_next();
}
```

6. Finish the code for the tintoPoint method by catching any exceptions.

```
System.out.println(" - Path to Generated Shapefile: " +
         shapePath +
         File.separator +
         shapeFile);
}catch (Exception ex) {
  ex.printStackTrace();
  }
}
```

You've now completed the code that results in a new shapefile representing the nodes of the intput TIN on disk.

7. Review the code for the full method below to make sure your's matches.

```
/**
 * @param tinPath - path to input tin data source
 * @param tinName - name of input tin data source
 * @param shapePath - path to output shapefile data source
 * @param shapeFile - name of output shapefile data source
*/
private static void tinToPoint(String tinPath,
         String tinName,
         String shapePath,
         String shapeFile){
try{
  //Get tin from tin file
  ITinAdvanced tinAdv = new Tin();
  String path = tinPath + File.separator + tinName;
  System.out.println("  - Path to Tin: " + path);
  tinAdv.init(path);

  System.out.println("  - Calculating ... ");

  Tin tin = new Tin(tinAdv);

   ISpatialReference tinSpatialRef = tin.getSpatialReference();
   IFields fields = createBasicFields(esriGeometryType.esriGeometryPoint,
           false,
           true,
           tinSpatialRef);
   // create output shapefile
   IWorkspaceFactory wkspFactory = new ShapefileWorkspaceFactory();
   IFeatureWorkspace featureWksp = new IFeatureWorkspaceProxy(
              wkspFactory.openFromFile(shapePath, 0));
   IFeatureClass outFC = new IFeatureClassProxy(
featureWksp.createFeatureClass( shapefile, fields, null, null,
    esriFeatureType.esriFTSimple, "Shape",   “”));
   // get tin node enum
   IEnumTinNode enum = tin.makeNodeEnumerator(tin.getExtent(),
           esriTinQualification.esriTinInsideDataArea,
           null);
   // store node to shapefile
   IFeatureCursor outCursor = outFC.IFeatureClass_insert(true);
   IFeatureBuffer outBuffer = outFC.createFeatureBuffer();
   IPoint point = new Point();
   ITinNode node = enum.IEnumTinNode_next();
   while(node != null){
      node.queryAsPoint(point);
      outBuffer.setShapeByRef(point);
      outCursor.insertFeature(outBuffer);
      node = enum.IEnumTinNode_next();
    }

   System.out.println(„ - Path to Generated Shapefile: " +
         shapePath +
         File.separator +
         shapeFile);
}catch (Exception ex) {
  ex.printStackTrace();
  }
}
```

Creating the basic fields for the 3D shapefile

Now that the method that performs the conversion from TIN to shapefile has been created, the next step is to define the schema structure for the output 3D shapefile. The *tinToPoint()* method defined above uses the *createBasicFields* method you create here to generate a fields object and send it as a parameter to the *createFeatureClass()* method.

1. Create the signature for this method as follows:

```
  /**
   * @param shapeType - a geometry object type
   * @param hasM - m-value precision defined
   * @param hasZ - z-value precision defined
   * @param spatialRef - Spatial Reference
   *
   * @return IFields - a collection of columns in a table
   */

private static IFields createBasicFields(int shapeType, boolean hasM,
  boolean hasZ, ISpatialReference spatialRef){
```

2. Use a *GeometryDef* object to define spatial qualities of your feature class. The most fundamental spatial quality that the method will take as a parameter is the geometry type. Create new fields using the *IFieldsEdit* interface and return this object.

```
try {
  Fields fields = new Fields();
  IFieldsEdit fieldsEdt = fields;
  Field field = new Field();
  IFieldEdit fieldEdt = field;
  GeometryDef geometryDef = new GeometryDef();
```

```
      double dGridSize;
      if (spatialRef.hasXYPrecision()) {
        double[] xmin = {0};
        double[] ymin = {0};
        double[] xmax = {0};
        double[] ymax = {0};
        spatialRef.getDomain(xmin, xmax, ymin, ymax);
        double dArea = (xmax[0] - xmin[0]) * (ymax[0] - ymin[0]);
        dGridSize = Math.sqrt(dArea / 100);
      }else {
        dGridSize = 1000;
      }
      geometryDef.setGeometryType(shapeType);
      geometryDef.setHasM(hasM);
      geometryDef.setHasZ(hasZ);
      geometryDef.setSpatialReferenceByRef(spatialRef);
      geometryDef.setGridCount(1);
      geometryDef.setGridSize(0,dGridSize);

      // add oid field - must come before geometry field
      fieldEdt = new Field();
      fieldEdt.setName("OBJECTID");
      fieldEdt.setAliasName("OBJECTID");
      fieldEdt.setType(esriFieldType.esriFieldTypeOID);
      fieldsEdt.addField(fieldEdt);

      //add Geometry field
      fieldEdt = new Field();
      fieldEdt.setName("SHAPE");
      fieldEdt.setIsNullable(true);
      fieldEdt.setType(esriFieldType.esriFieldTypeGeometry);
      fieldEdt.setGeometryDefByRef(geometryDef);
      fieldEdt.setRequired(true);
      fieldsEdt.addField(fieldEdt);
      return fieldsEdt;
    }catch (IOException ex) {
      ex.printStackTrace();
      return null;
      }
    }
```

Initializing ArcObjects

Every application built with the ArcGIS Engine Developer Kit must initialize ArcObjects at a product level, including any appropriate extension licenses. The *AoInitialize* object is used to accomplish this task. This class initializes the ArcObjects runtime environment and must be the first ArcObjects component created. Two methods will be called from this object:

- *initialize(int product code)*—This method takes an integer value representing a product code. The Java API provides an interface called *esriLicenseProductCode* which exposes static integer fields representing the different ESRI product levels. See the javadoc description for a full list of products.

- *CheckOutExtension(int extensioncode)*—This method takes an integer value representing an extension license code. The Java API provides an interface called *esriLicenseExtensionCode* which exposes static integer fields representing the different ESRI extension products available.

This application requires an ArcGIS Engine Runtime license with 3D Analyst extension license.

1. Create the signature for this method as follows:

```
private static void licenseCheckOut(){}
```

2. Implement the private method.

```
/**
 * Initialize ArcObjects product usage and check out
 * available 3D Analyst extension license
*/
private static void licenseCheckOut(){

  try{
   AoInitialze aoInit = new AoInitialize();
   aoInit.initialze(esriLicenseProductCode.esriLicenseProductCodeEngine);
   aoInit.checkOutExtension(esriLicenseExtensionCode.
                                  esriLicenseExtensionCode3DAnalyst);

  }catch(IOException e){
    System.out.println(„Program Exit: Unable to initialize ArcObjects„);
    System.exit(0);
  }
}
```

Putting it all together

Now that the private methods have all been constructed, you need to create an entry point for your application. This main method will take command line arguments and pass them to the *tinToPoint* method created in earlier steps.

1. First, insert some logic to ensure that you have the correct number of arguments in the form of an if/else code block.

```
/*
 * Description:
 *   Main Method - Application Entry Point
 */
public static void main(String[] args) {

if(args.length != 2){
  System.out.println( "Tin to Point: ArcGIS Engine Developer
Scenario");
  System.out.println("Usage: TintoPoint [Path-to-tin] [Path-to-output-
shapefile]");
  System.exit(0);
}else{
```

```
System.out.println( "Tin to Point: ArcGIS Engine Developer Sample");
String inDataset = args[0];
String outDataset = args[1];
```

This exercise does not implement any logic that determines if the arguments being passed are valid strings or datapaths.This validation process may be beneficial when developing a production application.

This application takes two arguments: an input full path to your TIN dataset and a full path to the generated output shapefile. Once the application determines the correct amount of arguments they can be split into the path and name format required by the *tinToPoint()* method generated earlier.

2. Add in the following code below the if/else code back you just created.

```
 String inDataPath = inDataset.substring( 0,
inDataset.lastIndexOf(File.separator));
 String inDataName = inDataset.substring(
inDataset.lastIndexOf(File.separator) + 1));
 String outDataPath = outDataset.substring( 0,
outDataset.lastIndexOf(File.separator));
 String outDataName = outDataset.substring(
outDataset.lastIndexOf(File.separator) + 1));
```

The next step in creating the main method is to initialize the programming environment. In the ArcGIS Java API, this is handled by calling the static class *EngineInitializer*. This class is a facade class exposed by the Java API to ensure optimal use of native ArcObjects for Java. Following that, the static *licenseCheckOut()* method described above must be called.

3. Add the following code to the main method.

```
EngineInitializer.initilizeEngine();
licenseCheckOut();
```

4. The last step is to call the static worker method *tintoPoint()* and pass in the string parameters.

```
tinToPoint(inDataPath, inDataName, outDataPath, outDataName);
System.out.println("Tin to Point - Done");
}
}
```

The code for the command line application is now complete.

DEPLOYMENT

There are many options to deploying your Java application and while you are free to choose any method you are comfortable with, this scenario utilizes the Ant build scripts you created earlier.

Redo the build steps you tested earlier.

1. Open a command prompt and cd into the root folder of your project. For example,

```
cd C:\TintoPoint
```

2. You should have all the build script files located in this root directory.

3. Type 'arcgisant' at the command prompt.

4. You should receive output similar to the following:

```
Buildfile: build.xml
init
validate-engine:
```

```
compile:
execute:
    [java] Tin to Point: ArcGIS Engine Developer Sample
    [java]   - Path to Tin: Q:\dop\data\imagery\tin\bachtin
    [java]   - Calculating ...
    [java]   - Path to Generated Shapefile:
Q:\dop\data\workspace\newshp.shp
    [java] Tin to Point - Done

run-scenario:

BUILD SUCCESSFUL
Total time: 9 seconds
```

If for any reason your build fails, ensure you have your environment correctly set and parameters correctly set in the sample.properties file. If your build does not compile, ensure that your source code is correct.

TROUBLESHOOTING

If your build returns the following:

```
Buildfile: build.xml
init:
validate-engine:
compile:
execute:
    [java] Tin to Point: ArcGIS Engine Developer Scenario
    [java] Usage: TintoPoint [Path-to-tin] [Path-to-output-shapefile]

run-scenario:

BUILD SUCCESSFUL
```

then your source code has successfully compiled, but you have not provided path variables in the sample.properties file.

If your build returns the following:

```
Buildfile: build.xml
init:
validate-engine:
compile:
execute:
    [java] Tin to Point: ArcGIS Engine Developer Sample
    [java] java.lang.StringIndexOutOfBoundsException: String index out of
range: -1
    [java]   at java.lang.String.substring(String.java:1444)
    [java]   at engine.scenario.analyst3d.TintoPoint.main(Unknown Source)
    [java] Exception in thread "main"

BUILD FAILED
file:Q:/dop/dev/ant/projects/engine.scenario/build.xml:53: Java returned: 1

Total time: 1 second
```

then your source code has compiled, but you have not provided valid variable strings representing your data paths. Ensure that you have provided data paths in the following format:

- C:\\data\\imagery\\tin\\bachtin
- C:\\data\\workspace\\newshp.shp

ADDITIONAL RESOURCES

The following resources may help you understand and apply the concepts and techniques presented in this scenario.

- Additional documentation available in the ArcGIS Engine Developer Kit including *ArcGIS Developer Help*, component help, object model diagrams, and samples to help you get started.
- ArcGIS Developer Online—Web site providing the most up-to-date information for ArcGIS Developers including updated samples and technical documents. Go to http://arcgisdeveloperonline.esri.com.
- ESRI online discussion forums—Web sites providing invaluable assistance from other ArcGIS developers. Go to http://support.esri.com and click the User Forums tab.

Java enjoys a huge community with many resources for its developers. The following is a list of URLs that most developers keep in their toolkit. These links, while correct at publication, are subject to change.

- Sun's Java and JDK FAQ (http://java.sun.com/products/jdk/faq.html)—High-level introductory FAQs about Java.
- Sun's Java (http://java.sun.com/)—The source for Java technology.
- Sun's Java Tutorial (http://java.sun.com/docs/books/tutorials)—On-Line version of the book from Addison-Wesley. Learing all about Java.
- Thinking in Java (http://www.mindview.net/Books/TIJ)—On-Line version fo book from Prentice Hall. Very good for learning about Object-Oriented Programming concepts in Java.

The Apache Ant project is an extremely successful open source project and carries the trademark of many resources.

- The Apache Ant Project (http://ant.apache.org)—this is where the Ant project lives.
- Java Development with Ant (http://www.manning.com/antbook)—excellent book covering Ant 1.5.
- Ant in Anger (http://ant.apache.org/ant_in_anger.html)—this document describes stategies and some basic examples of how to use Ant.
- jGuru (http://www.jguru.com/forums/home.jsp?topic=Ant)—jGuru hosts an interactive Ant discussion forum.

This scenario is designed to introduce the ArcGIS Engine C++ API for cross-platform applications. To get the most out of this scenario you should understand basic C/C++ programming concepts such as the preprocessor, functions, and memory management. Some familiarity with ArcObjects will also be helpful, although not required. Although this scenario does require conceptual knowledge of the C++ language, it does not require a lot of programming experience. The code used in this example provides an easy entry point to learn about the C++ API to ArcGIS Engine on a small and simple scale.

The purpose of this scenario is not to teach how to set up a C++ environment or how to compile on each supported operating system. Throughout this scenario it is assumed that you have a functional C++ environment and know how to compile a C++ program in that environment. What this scenario does provide is the steps to take and the code to write in order to create a command line application which computes the slope of a given digital elevation model.

Rather than walk through this scenario, you can get the completed application from the samples installation location. The sample is installed as part of the ArcGIS developer samples.

The ArcGIS developer samples are not included in the 'typical' installation of the ArcGIS Engine Developer Kit. If you do not have them installed, rerun the Developer Kit install wizard, choose Custom or Modify, and select the samples feature under Software Developer Kit.

You can find this sample in

```
<install_location>\DeveloperKit\Samples\Developer_Guide_Scenarios\ArcGIS_Engine
\Building_a_Command_Line_Application\Computing_the_slope_of_a_raster_dataset
```

PROJECT DESCRIPTION

This scenario covers some aspects of the ArcGIS Engine C++ API. The goal of the RasterSlope scenario is to create a standalone command line application with the ArcGIS C++ API. The application will take as input a digital elevation model (DEM) and will build and persist the slope map raster dataset. Once you have completed this scenario, you will understand the techniques required to work with the ArcGIS Engine C++ API, including using the Spatial Analyst extension. In particular, the scenario covers the following techniques:

Although this scenario steps you through C++ development, solution code is also available in other programming languages, including C#, Java, Visual Basic 6, Visual Basic .NET and Visual C++.

- Programming with the ArcGIS Engine C++ Developer Kit in a standard text editor.
- Parsing command line arguments.
- Enabling extensions: in particular, the Spatial Analyst extension.
- Performing the calculation of slope on a raster dataset.
- Persisting the resultant raster dataset.
- Deploying the application on all platforms supported by the ArcEngine C++ API.

CONCEPTS

Slope datasets are often used as inputs in physical process models. Slope is commonly expressed in degrees or as a percentage. In our slope calculation a parameter (zFactor) can be specified which specifies the number of ground x,y units in 1 z unit. This allows you to create a slope dataset with different z units than the input surface. To build the slope dataset, you will use the *RasterSurfaceOp* class and the ISurfaceOp interface it implements. You will also use the *Raster* class and the IRasterBandCollection interface it implements to persist the resulting raster dataset.

For a more in-depth explanation of slope, see the Burrough and McDonnell reference listed in the 'Additional resources' section at the end of this scenario.

The role of the software is to calculate the slope of a given raster dataset. The user's job is to provide a console at which to run the scenario, as well as a digital elevation model on which to run the scenario. Since this is a cross-platform application, the console can be on any supported platform.

DESIGN

For more detailed information, see the C++ aaplication programming interface section of Chapter 4, 'Developer environments'.

The application will be written entirely in the C++ language. This allows you, as the developer, to write code once on any supported platform and deploy the application on all supported ArcGIS Engine platforms. This scenario uses Microsoft Windows XP as the developer platform and nmake to compile and run the application from the command line, using the .NET 2003 compiler. However, it can easily be followed on any other supported platforms and using any supported build strategy. Both Visual Studio 6.0 and Visual Studio .NET 2003 projects are included with the solution code available in the Developer Kit.

In the design, some safeguards were taken to insure that the application remained cross-platform. They include avoiding function calls and datatypes defined outside the ArcGIS Engine and the C++ API as well as platform-independent path processing. For example, it should not matter whether the user has "/" or "\" as the path separator in the arguments to the application as long as the path separated is used by the operating system on which the application is being executed. In addition, C++ standards need to be followed to avoid compiler dependencies.

REQUIREMENTS

In order to successfully follow this scenario you need the following:

The C++ API is not included in the 'typical' installation of the ArcGIS Engine Developer Kit. If you do not have it installed, rerun the Developer Kit install wizard, choose Custom or Modify, and select the Native C++ feature under Software Developer Kit.

- An installation of the ArcGIS Engine Developer Kit (including Native C++) with an authorization file enabling it for development use.
- A text editor such as Notepad or Microsoft Visual C++.
- A supported C/C++ compiler. This scenario uses the Microsoft Visual C++ Compiler .NET 2003 (7.1). For setup details, see the C++ API section of Chapter 4, 'Developer environments'.
- A configured ArcObjects environment. For setup details, see the C++ API section of Chapter 4, 'Developer environments'.
- ArcSDK.h: the ArcGIS Engine C++ API header file.
- A familiarity with the operating system you have chosen to work on, and a basic foundation in C++ programming.
- While no experience with other ESRI software is required, previous experience with ArcObjects and a basic understanding of maps is advantageous.
- Access to the sample data, solution code, and Makefiles that come with this scenario. This is located at

  `<install_location>\DeveloperKit\Samples\Developer_Guide_Scenarios\ArcGIS_Engine\Building_a_Command_Line_Application\Computing_the_slope_of_a_raster_dataset`
- An ArcGIS Spatial Analyst or ArcGIS Engine Runtime with Spatial option license is required for the application to run once deployed.
- To run the application, you will need a raster dataset.

The ArcGIS developer samples are not included in the 'typical' installation of the ArcGIS Engine Developer Kit. If you do not have them installed, rerun the Developer Kit install wizard, choose Custom or Modify, and select the samples feature under Software Developer Kit.

IMPLEMENTATION

The C++ API section of Chapter 4, 'Developer environments' has a detailed discussion of error checking. You are encouraged to read it for more information.

The implementation below provides you with all the code you will need to successfully complete the scenario. It does not provide step-by-step instructions to compile C++ applications, as it assumes that you already have a working knowledge of your chosen development environment. Error checking has been left out to increase code readability.

This scenario's sample application demonstrates spatial analyst functionality; the full source code is available in the samples included in the ArcGIS Engine Developer Kit. Here are the files discussed in this scenario:

- RasterSlope.cpp—Main C++ source file
- RasterSlope.h—Main C++ header file
- Makefile.Windows.template—nmake utility file template. During this exercise, you will copy this file from the solution code and rename it Makefile.Windows. You will then update it while following the scenario.
- Makefile.Windows—nmake utility file that specifies compiler settings and rules, file dependencies, input arguments, and an execution rule for our application. In the solution code, this file is the completed Makefile for the scenario. As you work through the scenario, the name will refer to the Makefile you are building.
- PathUtilities.cpp—Platform independent path processing helper function implementation file
- PathUtilities.h—Platform independent path processing helper function header file

Creating your build environment

If you are unfamiliar with the nmake utility, see the Microsoft Developer Network (MSDN) for further information.

This scenario uses nmake to build and deploy its application. In order to utilize nmake, you must write a Makefile for it to execute. This scenario is not designed to teach you the basics of project management with the nmake utility. However, this scenario will step you through the parts of the Makefile that must be customized for each application you build.

A template copy of Makefile.Windows, like the one provided in this scenario's solution code, is included in the ArcGIS Engine Developer Kit and can be accessed from the help system under Development Environments -> C++ -> Makefiles as Makefile.Windows.

First, copy Makefile.Windows.template from this scenario's solution code to your coding directory. Once you have the file copied to your coding directory, follow the steps outlined below to prepare it for use:

1. Rename your copy of Makefile.Windows.template to Makefile.Windows.
2. Open the newly renamed Makefile.Windows.
3. Update PROGRAM to be RasterSlope.exe.
4. Update INCLUDEDIRS macro, which contains the include directories to pass to the compiler, to reflect where you installed ArcGIS Engine.
5. Update CPPSOURCES to be RasterSlope.cpp.
6. Update CPPOBJECTS to be RasterSlope.obj.
7. Update the dependencies line from basic_sample.obj to be for RasterSlope.obj and to depend on RasterSlope.cpp and RasterSlope.h.

Replacing all instances of basic_sample *in your Makefile.Windows file with* RasterSlope *will complete steps 3 and 5 through 7 listed at right.*

You are now ready to compile with nmake. When this scenario directs you to do so, you will need to use the "f" flag to specify the name of the Makefile that should be used. At the command line, you will type:

```
nmake /f Makefile.Windows.
```

Setting execution parameters

To use the Makefile to facilitate running the scenario, the parameters need to be stored within it and a target must be set to run the application. To set up your makefile to do this, you need to update the parameters to match the data you wish to process, and the name you want the output to be given.

1. Near the beginning of Makefile.Windows, find the lines:

```
#
# Command line parameters: Edit these parameters so that you can
# easily run the sample by typing "nmake /f Makefile.Windows run".
#
# You will need to:
# (1) Describe parameters here. ex: IN_SHAPEFILE is the input shapefile
# (2) Define parameters below this comment box.
#     ex: IN_SHAPEFILE = "c:\data\shapefile.shp"
# (3) Add the parameters to the run target at the end of this file
#     ex: $(PROGRAM) $(IN_SHAPEFILE)
#
```

2. Below it, add parameters for running this sample. For example, if your input raster is "C:\MyComputer\Rasters\RasterDataset" and your output dataset is going to be named "tempslope" you will add the following lines:

```
IN_RASTER = "C:\MyComputer\Rasters\RasterDataset"
OUT_RASTER = "tempslope"
```

3. At the end of Makefile.Windows there is a run target that currently only executes the program. Update that run target to also pass in the input parameters. If you used the variable names `IN_RASTER` and `OUT_RASTER` as shown in the example above, the run target should now look as follows:

```
#
# Run target: "nmake /f Makefile.Windows run" to execute the application
#


run:
        $(PROGRAM) $(IN_RASTER) $(OUT_RASTER)
```

You are now ready to run the application with nmake. When this scenario directs you to do so, you will need to use the "f" flag to specify the name of the Makefile that should be used. On a Windows system, you will type:

```
nmake /f Makefile.Windows run
```

This will have the same affect as typing "RasterSlope.exe C:\MyComputer\Rasters\RasterDataset tempslope" at the command line.

The command line build tools of Visual Studio (nmake, cl, link, for example) are not available by default. However, a batch file provided by Microsoft makes them available in Windows. This batch file, called vcvars32.bat, must be run each time you open a new command prompt. You can automate this process by either creating a batch file which runs the Visual Studio 6.0 version of vcvars32.bat and opens a command prompt which is ready for development, or by using the Visual Studio .NET 2003 Command Prompt which runs vcvars32.bat for you. For details, see the C++ API section in Chapter 4, 'Developer environments.'

Processing the arguments

The user provides the input and output file information for this application at runtime (either through the Makefile or at the command line). To get the specifics of that information, in order to use it in the program, some argument processing must be done.

1. Create a new file, RasterSlope.h, in your text editor. Place the contents of the file in a #ifndef and #define section. Include a header file so that information (such as a usage message) can be displayed to the user.

```
#ifndef __RASTERSLOPE_ESRISCENARIO_h__
#define __RASTERSLOPE_ESRISCENARIO_h__

#include <iostream>

#endif   // __RASTERSLOPE_ESRISCENARIO_h__
```

2. In another new file, RasterSlope.cpp, begin implementing your slope application. First include the header file you created in the last step. Then start writing the main function. For now, just process the arguments in it. Make sure the correct number of arguments were entered, else print out a usage message and exit. Since the first argument will be the program name, it can be ignored. The second argument is the input data and the third is the resulting slope file. You will check if the arguments passed are valid later in the program's execution.

```
#include “RasterSlope.h”

int main(int argc, char* argv[])
{
  if (argc != 3)
  {
    std::cerr << “Usage: RasterSlope [sourceFile] [outputFile]”
              << std::endl;
    return 0;
  }

  char* source = argv[1];
  char* result = argv[2];

  return 0;
}
```

There are three ways to scope members in a namespace.The following are examples of each using cerr, a member of namespace std:

1. using namespace std;
2. using std::cerr;
3. std::cerr << "Prepend namespace";

The third method is used throughout this scenario.

3. However, you will need to have the path and the file name of the input separately. To get this information, you will create a new file, in which you will place the path parsing utility functions. Start a new file, PathUtilities.h, and declare a helper function to get the parent directory and another to get the file name.

```
#ifndef __PATHUTILITIES_ESRISCENARIO_H__
#define __PATHUTILITIES_ESRISCENARIO_H__

#include <iostream>
#include <ArcSDK.h>
```

```
// Extract the shape file name from the full path of the file
HRESULT GetFileFromFullPath(const char* inFullPath, BSTR* outFileName);

// Remove the file name from the full path and return the directory
HRESULT GetParentDirFromFullPath(const char* inFullPath,
                                 BSTR* outFilePath);

#endif  // __PATHUTILITIES_ESRISCENARIO_H__
```

4. Implement the path utility functions in the new file PathUtilities.cpp.

```
#include "PathUtilities.h"

// Function to remove the file name from the full path and return the
// path to the directory.  Caller is responsible for freeing the memory
// in outFilePath (or pass in a CComBSTR which has been cast to a BSTR
// to let the CComBSTR handle memory management).
HRESULT GetParentDirFromFullPath(const char* inFullPath,
                                 BSTR* outFilePath)
{
  if (!inFullPath || !outFilePath)
    return E_POINTER;

  // Initialize output
  *outFilePath = 0;

  const char *pathEnd = strrchr(inFullPath, '/');      // UNIX
  if (pathEnd == 0)
    pathEnd = strrchr(inFullPath, '\\');               // Windows

  if (pathEnd == 0)
    return E_FAIL;

  int size = strlen(inFullPath) - strlen(pathEnd);
  char *tmp = new char[size+1];
  strncpy(tmp, inFullPath, size);
  *(tmp+size) = '\0';

  CComBSTR bsTmp (tmp);
  delete[] tmp;
  if (!bsTmp)
    return E_OUTOFMEMORY;
  *outFilePath = bsTmp.Detach();

    return S_OK;
}

// Function to extract the file (or directory) name from the full path
// of the file.  Caller is responsible for freeing the memory in
// outFileName (or pass in a CComBSTR which has been cast to a BSTR
// to let the CComBSTR handle memory management).
HRESULT GetFileFromFullPath(const char* inFullPath, BSTR *outFileName)
{
  if (!inFullPath || !outFileName)
    return E_POINTER;

  *outFileName = 0;

  const char* name = strrchr(inFullPath, ‘/’);          // UNIX
  if (name == 0)
    name = strrchr(inFullPath, ‘\\’);                   // Windows

  if (name == 0)
    return E_FAIL;

  name++;
  char* tmp = new char[strlen(name)+1];
  strcpy(tmp, name);

  CComBSTR bsTmp (tmp);
  delete[] tmp;
  if (!bsTmp)
    return E_OUTOFMEMORY;
  *outFileName = bsTmp.Detach();

  return S_OK;
}
```

5. Use the functions in PathUtilities.h to parse the input.

   a. Update RasterSlope.h to include PathUtilities.h so that you can use the functions you wrote above. Find

   ```
   #include <iostream>
   ```

   and below it, add

   ```
   #include “PathUtilities.h”
   ```

   b. In RasterSlope.cpp’s main program, parse the input to get the path and the file name. Find

   ```
   char* result = argv[2];

   return 0;
   ```

   Between these two lines, insert

   ```
   // Parse path
   CComBSTR sourceFilePath;
   CComBSTR sourceFileName;
   HRESULT hr = GetParentDirFromFullPath(source, &sourceFilePath);
   if (FAILED(hr) || sourceFilePath.Length() <= 0)
   {
     std::cerr << “Couldn’t parse source file path.” << std::endl;
     return 0;
   }
   hr = GetFileFromFullPath(source, &sourceFileName);
   if (FAILED(hr) || sourceFileName.Length() <= 0)
   {
   ```

```
    std::cerr << “Couldn’t parse source file name.” << std::endl;
    return 0;
  }
```

6. Update the Makefile to reflect the new PathUtilities.cpp and PathUtilities.h files, including RasterSlope’s dependency upon it.

7. Compile and run the application. It should simply exit and not appear to do anything although it is parsing the arguments.

Accessing ArcGIS Engine

In order to use ArcGIS Engine, it must be initialized and the proper files included. When done using ArcGIS Engine, it must be uninitializd.

1. At the top of RasterSlope.h, but below the inclusions for iostream and PathUtilities.h, add an inclusion for ArcSDK.h. It should now appear as follows:

```
#include <iostream>
#include “PathUtilities.h”
#include <ArcSDK.h>
```

The code shown in gray has already been entered in previous steps. It is given here to illustrate the accurate placement of the code you are adding in this step.

2. AoExit() must be called before the application is exited. This allows portability to supported operating systems which require AoExit() to correctly clean up various ArcGIS Engine and COM elements. Update RasterSlope.cpp’s main() to use this function instead of return.

```
int main(int argc, char* argv[])
{
  if (argc != 3)
  {
    std::cerr << “Usage: RasterSlope [sourceFile] [outputFile]”
                << std::endl;
    AoExit(0);
    ~~return 0;~~
  }

  char* source = argv[1];
  char* result = argv[2];
  ...

  {
    std::cerr << “Couldn’t parse source file path.” << std::endl;
    AoExit(0);
    ~~return 0;~~
  }
  hr = GetFileFromFullPath(source, &sourceFileName);
  if (FAILED(hr) || sourceFileName.Length() <= 0)
  {
    std::cerr << “Couldn’t parse source file name.” << std::endl;
    AoExit(0);
    ~~return 0;~~
  }
  AoExit(0);
  ~~return 0;~~
}
```

3. Write helper functions that initialize and shutdown the engine. These are general functions which you can use in any command line application.

   a. In RasterSlope.h:

```
#include <ArcSDK.h>

bool InitializeWithExtension(esriLicenseProductCode product,
                             esriLicenseExtensionCode extension);
void ShutdownApp(esriLicenseExtensionCode license);
```

   b. At the bottom of RasterSlope.cpp:

```
bool InitializeWithExtension(esriLicenseProductCode product,
                             esriLicenseExtensionCode extension)
{
  ::AoInitialize(0);

  IAoInitializePtr ipInit(CLSID_AoInitialize);
  esriLicenseStatus licenseStatus = esriLicenseFailure;
  ipInit->IsExtensionCodeAvailable(product, extension, &licenseStatus);
  if (licenseStatus == esriLicenseAvailable)
  {
    ipInit->Initialize(product, &licenseStatus);
    if (licenseStatus == esriLicenseCheckedOut)
      ipInit->CheckOutExtension(extension, &licenseStatus);
  }

  return (licenseStatus == esriLicenseCheckedOut);
}

void ShutdownApp(esriLicenseExtensionCode license)
{
  // Scope ipInit so released before AoUninitialize call
  {
    IAoInitializePtr ipInit(CLSID_AoInitialize);
    esriLicenseStatus status;
    ipInit->CheckInExtension(license, &status);
    ipInit->Shutdown();
  }

  ::AoUninitialize();
}
```

It appears that a new instance of AoInitialize is created in ShutdownApp(). However, it is a singleton object and so returns a pointer to the AoInitialize object that was previously created.

Command line applications can be run against any ArcGIS Engine installation—Runtime or Developer Kit—or any installation of the ArcGIS Desktop products, (ArcView, ArcEditor, or ArcInfo). However, this particular application requires a Spatial license in addition to the core license. Depending on the core product license being used , Engine or Desktop, either a Spatial option for the ArcGIS

Any additional extension functionality must use an extension license that matches the core license being used at that time. If the application initially accesses an Engine Runtime license, it must use the 3D or Spatial options for ArcGIS Engine if they are required. If the application initially accesses an ArcGIS Desktop license (ArcView, ArcEditor, or ArcInfo), it must use 3D or Spatial Analyst extension licenses if they are required.

Engine Runtime or an ArcGIS Spatial Analyst extension must also be available. Your application must confirm the availability of and then check out the necessary licenses as required.

4. In RasterSlope.cpp's main(), initialize ArcGIS Engine and set up the licensing for the product. Next, shutdown and uninitialize ArcGIS Engine. These lines of code can be placed after the command line arguments have been processed; that part of the application does not need access to ArcGIS Engine and, if the arguments are invalid, there is no reason to start ArcGIS Engine.

```
if (FAILED(hr) || sourceFileName.Length() <= 0)
{
  std::cerr << "Couldn't parse source file name." << std::endl;
  AoExit(0);
}
if (!InitializeWithExtension(esriLicenseProductCodeEngine,
                             esriLicenseExtensionCodeSpatialAnalyst))
  if (!InitializeWithExtension(esriLicenseProductCodeArcView,
                               esriLicenseExtensionCodeSpatialAnalyst))
    if (!InitializeWithExtension(esriLicenseProductCodeArcEditor,
                                 esriLicenseExtensionCodeSpatialAnalyst))
      if (!InitializeWithExtension(esriLicenseProductCodeArcInfo,
                                 esriLicenseExtensionCodeSpatialAnalyst))
      {
        std::cerr << "Exiting Application: Engine Initialization failed"
                  << std::endl;
        ShutdownApp(esriLicenseExtensionCodeSpatialAnalyst);
        AoExit(0);
      }

// Insert code here

ShutdownApp(esriLicenseExtensionCodeSpatialAnalyst);

AoExit(0);
```

This is a placeholder comment to indicate where additional code will be placed later in the exercise.

5. Compile the application using the nmake utility as you did earlier in the scenario.

6. Run the application. It still appears not to do anything, however now it is also performing the license checking.

Computing the Slope

At this point, you've determined the dataset on which to compute the slope and have accessed ArcGIS Engine. Now, the slope calculation itself can be performed. This action is done in a separate function, *CalculateSlope()*, which is called from *main()*.

1. Place a declaration for *CalculateSlope()* in RasterSlope.h. Give it an HRESULT return type so that it can be used for error checking.

```
void ShutdownApp(esriLicenseExtensionCode license);
HRESULT CalculateSlope(BSTR inPath, BSTR inName, BSTR outFile);
```

2. After the *ShutdownApp()* function in RasterSlope.cpp, add the implementation for *CalculateSlope()*. Place the function only in this step. Upcoming steps will continue to place code into the *CalculateSlope()* function, unless otherwise indicated.

```
HRESULT CalculateSlope(BSTR inPath, BSTR inName, BSTR outFile)
{

}
```

3. Open the input raster workspace.

```
HRESULT CalculateSlope(BSTR inPath, BSTR inName, BSTR outFile)
{
  // Open the workspace
  IWorkspaceFactoryPtr ipWorkspaceFactory
                            (CLSID_RasterWorkspaceFactory);
  IWorkspacePtr ipWorkspace;
  HRESULT hr = ipWorkspaceFactory->OpenFromFile(inPath, 0, &ipWorkspace);
  if (FAILED(hr) || ipWorkspace == 0)
  {
     std::cerr << “Could not open the workspace factory.” << std::endl;
     return E_FAIL;
  }
```

4. Query Interface to get access to the raster-specific workspace functionality and open the input raster dataset.

```
HRESULT hr = ipWorkspaceFactory->OpenFromFile(inPath, 0, &ipWorkspace);
if (FAILED(hr) || ipWorkspace == 0)
{
  std::cerr << “Could not open the workspace factory.” << std::endl;
  return E_FAIL;
}

// Open the raster dataset
IRasterWorkspacePtr ipRastWork(ipWorkspace);
IRasterDatasetPtr ipRastDataset;
hr = ipRastWork->OpenRasterDataset(inName, &ipRastDataset);
if (FAILED(hr) || ipRastDataset == 0)
{
  std::cerr << “Could not open the raster dataset.” << std::endl;
  return E_FAIL;
}
```

5. To perform the slope calculation, use the *ISurfaceOp* interface’s *Slope()* function. To do this you need to access the *ISurfaceOp* interface on the workspace. To set up that workspace, Query Interface to *IRasterAnalysisEnvironment*.

```
hr = ipRastWork->OpenRasterDataset(inName, &ipRastDataset);
if (FAILED(hr) || ipRastDataset == 0)
{
       std::cerr << “Could not open the raster dataset.” << std::endl;
       return E_FAIL;
}
```

```
// Set up the ISurfaceOp interface to calculate slope
IRasterAnalysisEnvironmentPtr ipRastAnalEnv(CLSID_RasterSurfaceOp);
ipRastAnalEnv->putref_OutWorkspace(ipWorkspace);
ISurfaceOpPtr ipSurfOp(ipRastAnalEnv);
```

6. You are now ready to perform the slope calculation and end the *CalculateSlope()* function. Return the HRESULT returned by that function, to indicate if the calculation was successful.

```
ISurfaceOpPtr ipSurfOp(ipRastAnalEnv);
IGeoDatasetPtr ipGeoDataIn(ipRastDataset);
IGeoDatasetPtr ipGeoDataOut;
HRESULT slopeHR = ipSurfOp->Slope(ipGeoDataIn,
                                  esriGeoAnalysisSlopeDegrees,
                                   0,
                                  &ipGeoDataOut);
if (FAILED(slopeHR) || ipGeoDataOut == 0)
{
  std::cerr << “slopeHR = “ << slopeHR << std::endl;
  return slopeHR;
}

return slopeHR;
```

7. *CalculateSlope()* has now been completely implemented and is ready for use. In the main function, after the engine has been initialized, call the *CalculateSlope()* function. Delete the placeholder comment.

```
~~// Insert code here~~

hr = CalculateSlope(sourceFilePath, sourceFileName, CComBSTR(result));
if (FAILED(hr))
  std::cerr << “The slope calculation failed.” << std::endl;
else
  std::wcerr << L“The slope of “ << (BSTR) sourceFileName
             << L“ has been calculated.” << std::endl;

ShutdownApp(esriLicenseExtensionCodeSpatialAnalyst);
```

8. Compile the application and then run it. Notice that the output data’s path information is never used and that the result of the slope calculation is not stored anywhere.

Persisting the result

When the slope is computed, the result is only created in memory. To save it, you must programmatically persist it to disk.

1. Since you cannot create a new raster dataset where one already exists, make sure that the output slope file does not exist yet. This can be done in the *CalculateSlope()* function by trying to open a workspace with the desired name. If such an open is successful, then there is already a dataset with that name.

To perform this check, place the following code after the input dataset is opened in *CalculateSlope()*.

```
hr = ipRastWork->OpenRasterDataset(inName, &ipRastDataset);
if (FAILED(hr) || ipRastDataset == 0)
{
  std::cerr << “Could not open the raster dataset.” << std::endl;
  return E_FAIL;
}

// Check for existence of a dataset with the desired output name.
IRasterDatasetPtr ipExistsCheck;
hr = ipRastWork->OpenRasterDataset(outFile, &ipExistsCheck);
if (SUCCEEDED(hr))
{
  std::cerr << “A dataset with the output name already exists!”
            << std::endl;
  return E_FAIL;
}

// Set up the ISurfaceOp interface to calculate slope
```

2. Once its been determined that no such dataset exists, you can save the one created by the slope operation. The save is done through the *IRasterBandCollection* interface after the slope is computed.

```
HRESULT CalculateSlope(BSTR inPath, BSTR inName, BSTR outFile)
{
  …

  HRESULT slopeHR = ipSurfOp->Slope(ipGeoDataIn,
                                    esriGeoAnalysisSlopeDegrees,
                                    0,
                                    &ipGeoDataOut);
  if (FAILED(slopeHR) || ipGeoDataOut == 0)
  {
    std::cerr << “slopeHR = “ << slopeHR << std::endl;
    return slopeHR;
  }

  // Persist the result
  IRasterBandCollectionPtr ipRastBandColl(ipGeoDataOut);
  IDatasetPtr ipOutDataset;
  ipRastBandColl->SaveAs(outFile, ipWorkspace, CComBSTR(L”GRID”),
                       &ipOutDataset);

      return slopeHR;
}
```

3. Compile and run the application. Browse to where the slope data was created. A new ESRI grid was generated with your output name.

DEPLOYMENT

The final part of the development process is your application's successful deployment to an enduser's machine. Doing so requires the following:

- An installation of ArcGIS Engine Runtime with Spatial option on the user machine.
- A copy of the application's executable, created at compile time, residing on the enduser's machine.

Once these requirements are in place, your enduser will be able to create a slope file for any dataset just by typing the following at the command line:

```
RasterSlope inputRaster outputRaster
```

The output slope file is created in the same directory as its parent raster file.

where `inputRaster` is the full path (including file name) to the raster data file and `outputRaster` is the name of the output slope file that to be created.

Alternatively, your enduser can use the nmake utility to run the sample. To do so, an appropriate Makefile with the correct arguments must be made and the following entered at the command line:

```
nmake /f Makefile.Windows run
```

ADDITIONAL RESOURCES

The following resources may help you understand and apply the concepts and techniques presented in this scenario.

- Additional documentation available in the ArcGIS Engine Developer Kit including *ArcGIS Developer Help*, component help, object model diagrams, and samples to help you get started.
- ArcGIS Developer Online—Web site providing the most up-to-date information for ArcGIS Developers including updated samples and technical documents. Go to http://arcgisdeveloperonline.esri.com.
- ESRI online discussion forums—Web sites providing invaluable assistance from other ArcGIS developers. Go to http://support.esri.com and click the User Forums tab.
- *Principles of Geographical Information Systems*. Peter A. Burrough and Rachel A. McDonnell. Oxford University Press. 1998.

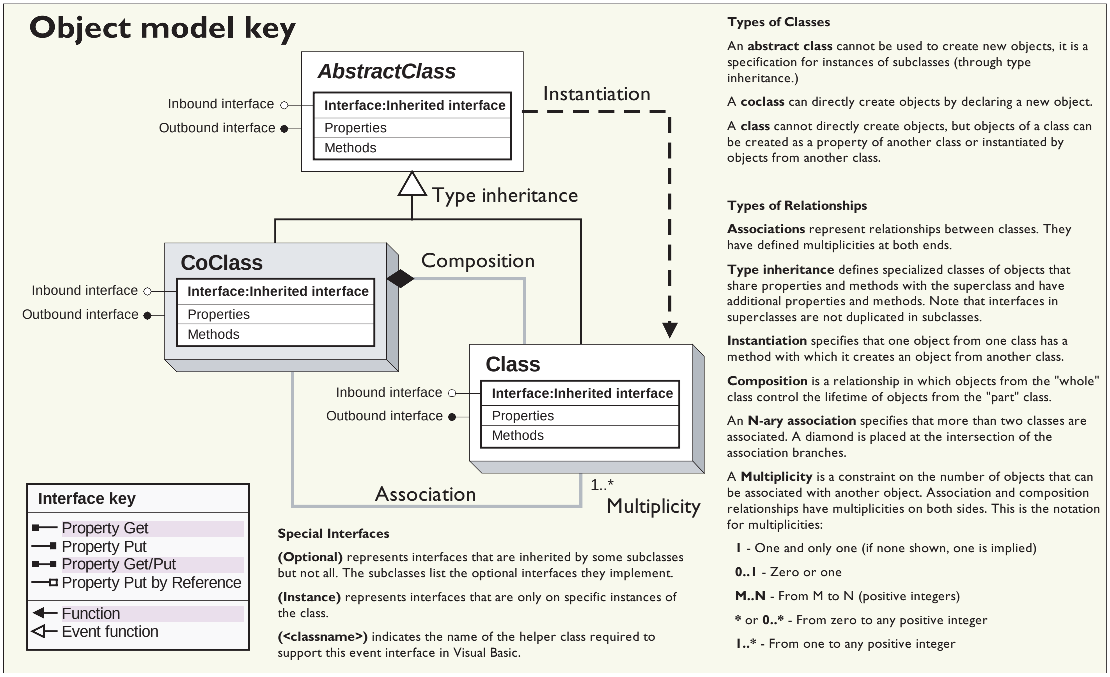
A

Reading the object model diagrams

The ArcObjects object model diagrams are an important supplement to the information you receive in object browsers. This chapter describes the diagram notation used throughout this book and in the object model diagrams that are accessed through ArcGIS Developer Help.

The diagram notation used in this book and the ArcObjects object model diagrams is based on the Unified Modeling Language (UML) notation, an industry-diagramming standard for object-oriented analysis and design, with some modifications for documenting COM-specific constructs.

The object model diagrams are an important supplement to the information you receive in object browsers. The development environment, Visual Basic or other, lists all of the classes and members but does not show the structure or relationships of those classes. These diagrams complete your understanding of the ArcObjects components.



Types of Classes

An **abstract class** cannot be used to create new objects, it is a specification for instances of subclasses (through type inheritance.)

A **coclass** can directly create objects by declaring a new object.

A **class** cannot directly create objects, but objects of a class can be created as a property of another class or instantiated by objects from another class.

Types of Relationships

Associations represent relationships between classes. They have defined multiplicities at both ends.

Type inheritance defines specialized classes of objects that share properties and methods with the superclass and have additional properties and methods. Note that interfaces in superclasses are not duplicated in subclasses.

Instantiation specifies that one object from one class has a method with which it creates an object from another class.

Composition is a relationship in which objects from the "whole" class control the lifetime of objects from the "part" class.

An **N-ary association** specifies that more than two classes are associated. A diamond is placed at the intersection of the association branches.

A **Multiplicity** is a constraint on the number of objects that can be associated with another object. Association and composition relationships have multiplicities on both sides. This is the notation for multiplicities:

1 - One and only one (if none shown, one is implied)

0..1 - Zero or one

M..N - From M to N (positive integers)

* or **0..*** - From zero to any positive integer

1..* - From one to any positive integer

Special Interfaces

(Optional) represents interfaces that are inherited by some subclasses but not all. The subclasses list the optional interfaces they implement.

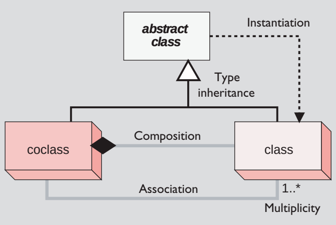
(Instance) represents interfaces that are only on specific instances of the class.

(<classname>) indicates the name of the helper class required to support this event interface in Visual Basic.

Object Model Diagram key showing the types of ArcObjects and the relationships between them.

CLASSES AND OBJECTS

There are three types of classes shown in the UML diagrams: abstract classes, coclasses, and classes.



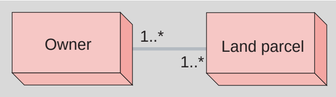
A *coclass* represents objects that you can directly create using the object declaration syntax in your development environment. In Visual Basic, this is written with the *Dim pFoo As New FooObject* syntax.

A *class* cannot directly create new objects, but objects of a class can be created as a property of another class or by functions from another class.

An *abstract class* cannot be used to create new objects; it is a specification for subclasses. An example is that a "line" could be an abstract class for "primary line" and "secondary line" classes. Abstract classes are important for developers who wish to create a subclass of their own since it shows which interfaces are required and which are optional, for the type of classs they are implementing. Required interfaces must be implemented on any subclass of the abstract class to ensure the new class behaves correctly in the ArcObjects system.

RELATIONSHIPS

Among abstract classes, coclasses, and classes, there are several types of class relationships possible.



In this diagram, an owner can own one or many land parcels, and a land parcel can be owned by one or many owners.

Associations represent relationships between classes. They have defined multiplicities at both ends.

A *Multiplicity* is a constraint on the number of objects that can be associated with another object. This is the notation for multiplicities:

1—One and only one. Showing this multiplicity is optional; if none is shown, "1" is implied.

0..1—Zero or one

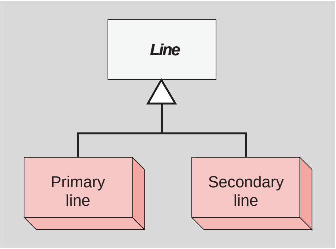
M..N—From M to N (positive integers)

* or 0..*—From zero to any positive integer

1..*—From one to any positive integer

TYPE INHERITANCE

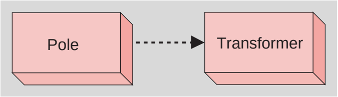
Type inheritance defines specialized classes that share properties and methods with the superclass and have additional properties and methods.



This diagram shows that a primary line (creatable class) and secondary line (creatable class) are types of a line (abstract class).

INSTANTIATION

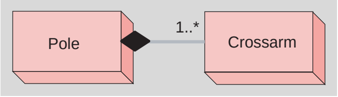
Instantiation specifies that one object from one class has a method with which it creates an object from another class.



A pole object might have a method to create a transformer object.

COMPOSITION

Composition is a stronger form of aggregation in which objects from the “whole” class control the lifetime of objects from the “part” class.

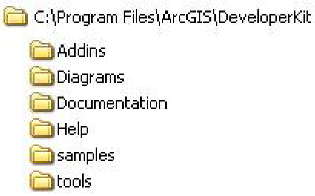


A pole contains one or many crossarms. In this design, a crossarm cannot be recycled when the pole is removed. The pole object controls the lifetime of the crossarm object.

B

ArcGIS developer resources

ESRI has created two essential resources for ArcGIS developers: the ArcGIS Software Developer Kit and ArcGIS Developer Online (http://ArcGISDeveloperOnline.esri.com).



A typical SDK installation

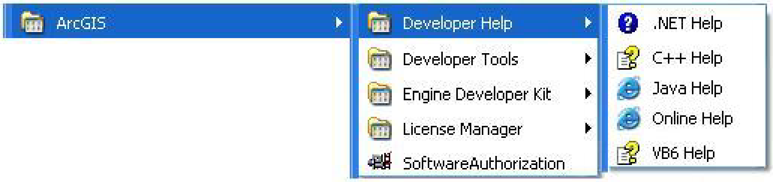
The ArcGIS Software Developer Kit (SDK) is the collection of diagrams, utilities, add-ins, samples, and documentation geared to help developers implement custom ArcGIS functionality.

ARCGIS DEVELOPER HELP SYSTEM

The ArcGIS Developer Help System is the gateway to all the SDK documentation including help for the add-ins, developer tools, and samples; in addition, it serves as the complete syntactical reference for all object libraries.

Each supported API has a version of the help system that works in concert with it. Regardless of the API you choose to use, you will see the appropriate library reference syntax and have a help system that is integrated with your development environment. For example, if you are a Visual Basic 6 developer you will use ArcGISDevHelp.chm which has the VB6 syntax and integrates with the VB6 IDE thereby providing F1 help support in the Code Window.

The help systems reside in the DeveloperKit\Help folder but are typically launched from the start menu or F1 help in Visual Basic 6 and Visual Studio.NET 2003. The graphic below shows the start menu options for opening the help systems.

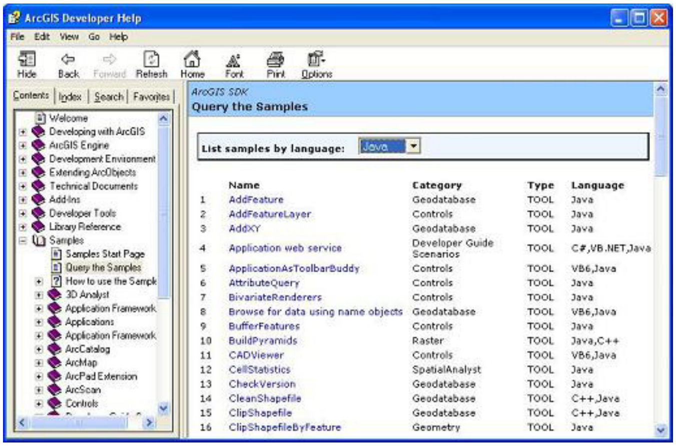


SAMPLES

The ArcGIS Developer Kit contains over 600 samples many of which are written in several languages. The samples are described in the help system and the source code and project files are installed in the DeveloperKit\samples folder. The help system's table of contents for the samples section mirrors the samples directory structure.

The help system organizes samples by functionality. For example, all the Geodatabase samples are grouped under Samples\Geodatabase. Most first tier groupings are further subdivided. You can also find samples in the SDK using the 'Query the Samples' topic in the help system which lists all the samples alphabetically; in addition, you can sort the list by language. For example, you can elect to only list the available Java samples.

Installing the samples source code and project files is an option in the install. The samples are installed under the ArcGIS\DeveloperKit\samples folder. If you don't have this folder on your computer, you can rerun the install program and check on Samples under Developer Kit.



ArcGIS SDK

Query the Samples

List samples by language: Java

	Name	Category	Type	Language
1	AddFeature	Geodatabase	TOOL	Java
2	AddFeatureLayer	Controls	TOOL	Java
3	AddXY	Geodatabase	TOOL	Java
4	Application web service	Developer Guide Scenarios	TOOL	C#,VB.NET,Java
5	ApplicationAsToolbarBuddy	Controls	TOOL	VB6,Java
6	AttributeQuery	Controls	TOOL	Java
7	BivariateRenderers	Controls	TOOL	Java
8	Browse for data using name objects	Geodatabase	TOOL	VB6,Java
9	BufferFeatures	Controls	TOOL	Java
10	BuildPyramids	Raster	TOOL	Java,C++
11	CADViewer	Controls	TOOL	VB6,Java
12	CellStatistics	SpatialAnalyst	TOOL	Java
13	CheckVersion	Geodatabase	TOOL	Java
14	CleanShapefile	Geodatabase	TOOL	C++,Java
15	ClipShapefile	Geodatabase	TOOL	C++,Java
16	ClipShapefileByFeature	Geometry	TOOL	Java

DEVELOPER TOOLS

The ArcGIS Developer Tools are executables that ESRI has provided to facilitate your ArcObjects development. You may find some of these tools essential. For example, if you are a Visual Basic 6 desktop developer you will likely use the Categories.exe tool to register components in component categories.

Each of the developer tools is installed in the DeveloperKit\tools folder except for the Component Category Manager, which is located in the ArcGIS\bin folder. Please refer to *ArcGIS Developer Help* for a detailed discussion of each tool and instructions for their use.

Tools available with each ArcGIS Developer Kit

- Component Categories Manager—Registers components within a specific component category.
- Fix Registry Utility—Fixes corruptions in the Component Categories section of the registry.
- GUID Tool—Generates GUIDs, in registry format, for use within source code.
- Library Locator—Identifies object library containing a specified interface, coclass, enumeration, or structure.

Additional tools available in the Desktop Developer Kit

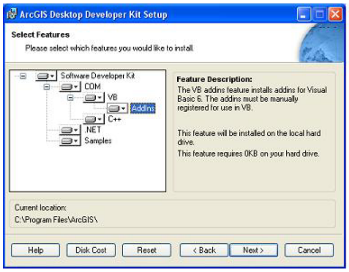
- ESRI Object Browser—Advanced object browser.
- Extract VBA—Extracts VBA code from a corrupt map document (mxd).

ADD-INS

The ESRI add-ins automate some of the tasks performed by the software engineer when developing with ArcObjects, as well as provide tools that make debugging code easier. ESRI provides add-ins for the Visual Basic 6 IDE and the Visual Studio.NET IDE.

Visual Basic 6 add-ins

The following Visual Basic 6 add-ins are available but only installed if you select them during the installation process:


ArcGIS Desktop Developer Kit Setup

Select Features

Please select which features you would like to install.

Feature Description:
The VB addins feature installs addins for Visual Basic 6. The addins must be manually registered for use in VB.

This feature will be installed on the local hard drive.

This feature requires 0KB on your hard drive.

Current location:
C:\Program Files\ArcGIS\

Installation dialog box for selecting the Visual Basic add-ins.

- ESRI Align Controls with Tab Index—Ensures control creation order matches tab index.
- ESRI Automatic References—Automatically adds ArcGIS library references.
- ESRI Code Converter—Converts projects from ArcGIS 8.X to ArcGIS 9.X.
- ESRI Command Creation Wizard—Facilitates the creation of commands and tools.
- ESRI Compile and Register—Aids in compiling components and registering these in desired component categories.
- ESRI ErrorHandler Generator—Automates the generation of error handling code.

- ESRI ErrorHandler Remover—Removes the error handlers from the source files.
- ESRI Interface Implementer—Automatically stubs out implemented interfaces.
- ESRI Line Number Generator—Adds line numbers to the appropriate lines within source files.
- ESRI Line Number Remover—Removes the line numbers from source files.

Visual Studio.NET add-ins

The following .NET add-ins are automatically installed during setup if a version of Visual Studio.NET 2003 is detected:

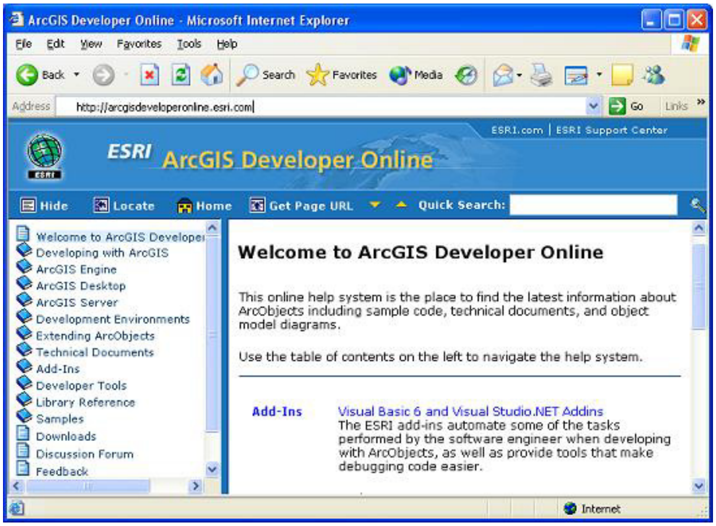
- ESRI Component Category Registrar—Stubs out registration functions to enable self component category registration.
- ESRI Guid Generator—Inserts a GUID attribute.

ArcGIS Developer Online Website

ArcGIS Developer Online is the place to find the most up-to-date ArcGIS 9 developer information including sample code, technical documents, object model diagrams, and the complete object library reference.

The web site is a reflection of the ArcGIS Developer Help system except it is online and therefore more current. The web site has some additional features including an advanced search utility that enables you to control the scope of your searches. For example, you can create a search that only scans the library reference portion of the help system.

Visit the site today (http://arcgisdeveloperonline.esri.com).


ArcGIS Developer Online - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Search Favorites Media
Address http://arcgisdeveloperonline.esri.com
Go Links
ESRI.com | ESRI Support Center
ESRI
ArcGIS Developer Online
Hide Locate Home Get Page URL Quick Search:
Welcome to ArcGIS Developer
Developing with ArcGIS
ArcGIS Engine
ArcGIS Desktop
ArcGIS Server
Development Environments
Extending ArcObjects
Technical Documents
Add-Ins
Developer Tools
Library Reference
Samples
Downloads
Discussion Forum
Feedback
Internet

Welcome to ArcGIS Developer Online

This online help system is the place to find the latest information about ArcObjects including sample code, technical documents, and object model diagrams.

Use the table of contents on the left to navigate the help system.

Add-Ins — Visual Basic 6 and Visual Studio.NET Addins
The ESRI add-ins automate some of the tasks performed by the software engineer when developing with ArcObjects, as well as provide tools that make debugging code easier.

C

Glossary

The following is a glossary of common terms used throughout this book. While it is not meant to be an all-encompassing list, its should provide you with a quick reference to ArcGIS Engine-specific terminology.

abstract class A specification for subclasses that is often shown on object model diagrams to help give structure to the diagram. An abstract class is not defined in a type library and cannot be instantiated.

Active Server Pages A Microsoft server-side scripting environment that can be used to create and run dynamic, interactive Web server applications, which are typically coded in JavaScript or VBScript. An ASP file contains not only text and HTML tags, smiilar to standard Web documents, but also commands written in a scripting language, which can be carried out on the server.

Active Template Library A set of C++ template classes, designed to be small, fast, and extensible.

add-in An extension to a development environment that performs a custom task. ESRI provides various developer add-ins as part of the ArcGIS developer kit.

ADF Application Developer Framework. The set of custom Web controls and templates that can be used to build Web applications that communicate with a GIS server. ArcGIS Server includes an ADF for both .NET and Java.

ADF runtime The components required to run an application built with the ADF. See also ADF.

apartment A group of threads working within a process that work within the same context. See also MTA, STA, thread, TNA.

API See application programming interface.

application programming interface A set of routines, protocols, and tools that application developers use to build or customize a program or set of programs. APIs make it easier to develop a program by providing building blocks for a preconstructed interface instead of requiring direct programming of a device or piece of software. They also guarantee that all programs using a common API will have similar interfaces. APIs can be built for programming languages such as C, COM, Java, and so on.

application Web service A Web service that solves a particular problem, for example, a Web service that finds all of the hospitals within a certain distance of an address. An application Web service can be implemented using the native Web service framework of a Web server, for example, an ASP.NET Web service (WebMethod) or Java Web service (Axis).

ArcGIS Server Web service A Web service processed and executed from within an ArcGIS Server. Each Web service is a distinct HTTP endpoint (URL). Administrators can expose MapServer and GeocodeServer objects as generic ArcGIS Server Web services for access across the Internet. See also Web service catalog.

arcgisant The command, provided with the Java ADF, that starts the Apache ANT tool that builds and deploys Web applications. See also ADF.

ArcObjects A library of software components that makes up the foundation of ArcGIS. ArcGIS Desktop, ArcGIS Engine, and ArcGIS Server are all built on top of the ArcObjects libraries.

ASCII American Standard Code for Information Interchange. The de facto standard for the format of text files in computers and on the Internet. Each alphabetic, numeric, or special character is represented with a 7-bit binary number (a string of seven 1s and 0s). 128 possible characters are defined.

ASP See Active Server Pages.

ASP.NET A programming framework built on the Common Language Runtime (CLR) that can be used on a server to build Web applications in any programming language supported by .NET. See also Active Server Pages.

assembly A package of software and its associated resources. Typically, an ArcGIS Win32 assembly will include executables and DLLs, object libraries, registry files, and help files for a unit of software. A .NET assembly is a unit of software built with a .NET language that uses the .NET Framework and the CLR to execute.

ATL See Active Template Library.

authentication The process of obtaining identification credentials, such as a name and password, from a user and validating those credentials against some authority. If the credentials are valid, the entity that submitted the credentials is considered an authenticated identity. Authentication can be used to determine whether an entity has access to a given resource.

.bat file Sometimes referred to as a batch file, a file that contains commands that can be run in a command window. It is used to perform repetitive tasks and to run scheduled commands.

big endian A computer hardware architecture in which, within a multibyte numeric representation, the most significant byte has the lowest address and the remaining bytes are encoded in decreasing order of significance. See also little endian.

binary Any file format for digital data encoded as a sequence of bits (1s and 0s) but not consisting of a sequence of printable characters (ASCII format). The term is often used for executable machine code, such as a DLL or EXE file that contains information that can be directly loaded or exectued by the computer.

by value A way of passing a parameter to a function such that a temporary copy of the value of the parameter is created. The function makes changes to this temporary copy, which is discarded after the function exits. If the parameter is a reference to an underlying object, any changes made to the underlying object will be preserved after the function exits.

C++ A common object-oriented programming language, with many different implementations designed for different platforms.

Cascading Style Sheets A standard for defining the layout or presentation of an HTML or XML document. Style information includes font size, background color, text alignment, and margins. Multiple stylesheets may be applied to “cascade” over previous style settings, adding to or overriding them. The World Wide Web Consortium (W3C) maintains the CSS standard. See also World Wide Web Consortium.

CASE Computer-aided software engineering. A category of software that provides a development environment for programming teams. CASE systems offer tools to automate, manage, and simplify the development process. Complex tasks that often require many lines of code are simplified with CASE user interfaces and code generators.

class A template for a type of object in an object-oriented programming language. A class may be considered to be a set of objects that share a common structure and behavior.

class identifier A COM term refering to the globally unique number that is used by the system registry and the COM framework to identify a particular coclass. See also GUID.

client An application, computer, or device in a client/server model that makes requests to a server.

cloning The process of creating a new instance of a class with the same state as an existing instance.

CLR Common Language Runtime. The execution engine for .NET Framework applications, providing services such as code loading and execution and memory management.

CLSID See class identifier.

coclass A template for an object that can be instantiated in memory.

COM See Component Object Model.

COM contract The COM requirement that interfaces, once published, cannot be altered.

COM interface A grouping of logically related virtual functions, implemented by a server object, allowing a client to interact with the server object. Interfaces form the basis of COM's communication between objects and the basis of the COM contract.

COM-compliant language A language that can be used to create COM components.

command Any class in an ArcGIS system that implements the ICommand interface and can therefore be added to a menu or toolbar in an ArcGIS application.

command bar A toolbar, menu bar, menu, or context menu in an ArcGIS application.

command line An onscreen interface in which the user types in commands at a prompt. In geoprocessing, any tool added to the ArcToolbox™ window can be run from the command line.

component A binary unit of code that can be used to create COM objects.

component category A section of the registry that can be used to categorize classes by their functionality. Component categories are used extensively in ArcGIS to allow extensibility of the system.

Component Category Manager An ArcGIS utility program (Categories.exe) that can be used to view and manipulate component category information.

Component Object Model A binary standard that enables software components to interoperate in a networked environment regardless of the language in which they were developed. Developed by Microsoft, COM technology provides the underlying services of interface negotiation, lifecycle management (determining when an object can be removed from a system), licensing, and event services (putting one object into service as the result of an event that has happened to another object). The ArcGIS system is created using COM objects.

computer-aided software engineering See CASE.

container account The operating system account that server object container processes run as, which is specified by the GIS server post installation utility. Objects running in a server container process have the same access rights to system resources as the container account.

container process A process in which one or more server objects is running. Container processes run on SOC machines and are started and shut down by the SOM.

Content Standard for Digital Geospatial Metadata A publication authored by the Federal Geographic Data Committee (FGDC) that specifies the information content of metadata for a set of digital geospatial data. The purpose of the standard is to provide a common set of terminology and definitions for concepts related to the metadata. All U.S. government agencies (federal, state, and local) that receive federal funds to create metadata must follow this standard.

control A component with a user interface. In ArcGIS, the term often refers to the MapControl, PageLayoutControl, TOCControl, ToolbarControl, or ArcReaderControl, which are parts of ArcGIS Engine.

control points See control.

creation time The time it takes to initialize an instance of a server object when server objects are created in the GIS server either as a result of the server starting or in response to a request for a server object by a client.

CSDGM See Content Standard for Digital Geospatial Metadata.

CSS See Cascading Style Sheets.

custom Functionality provided or created by a party who is not the original software developer.

data type The attribute of a variable, field, or column in a table that determines the kind of data it can store. Common data types include character, integer, decimal, single, double, and string.

database management system A set of computer programs that organizes the information in a database according to a conceptual schema and provides tools for data input, verification, storage, modification, and retrieval.

database support The proprietary database platforms supported by a program or component.

DBMS See database management system.

DCOM Distributed Component Object Model. Extends COM to support communication among objects on different computers on a network.

debug To test a program or component in order to determine the cause of faults.

deeply stateful application An application that uses the GIS server to maintain application state by changing the state of a server object or its related objects. Deeply stateful applications require non-pooled server objects.

default interface When a COM object is created, the interface that is returned automatically if no other interface is specified. Most ArcObjects classes specify IUnknown as the default interface.

deployment The installation of a component or application to a target machine.

developer sample A sample contained in the ArcGIS Developer Help system.

development environment A software product used to write, compile, and debug components or applications.

device context Represents a surface that can be drawn to, for example, a screen, bitmap, or printer. In ArcGIS, the Display abstract class is used to abstract a device context.

display Often used to refer to subclasses of the Display abstract class. For example, "when drawing to the display" means when drawing to any of the display coclasses; "the display pipeline" refers to the sequence of calls made when drawing occurs.

DLL See dynamic link library.

dockable window A window that can exist in a floating state or be attached to the main application window.

dynamic link library Modules of code containing a set of routines that are called from procedures. A DLL is loaded and linked to an application at runtime by its calling modules (EXE or DLL).

early binding A technique that an application uses to access an object. In early binding, an object's properties and methods are defined from a class, instead of being checked at runtime as in late binding. This difference often gives early binding performance benefits over late binding. See also late binding.

EJB See Enterprise JavaBeans.

EMF Enhanced Metafile. A spool file format used in printing by the Windows operating system.

Enterprise JavaBeans The server-side component architecture for the J2EE platform. EJB enables development of distributed, transactional, secure, and portable Java applications.

EOBrowser An ArcGIS utility application that can be used to investigate the contents of object libraries.

event handling Sinking an event interface raised by another class.

executable file A binary file containing a program that can be implemented or run. Executable files are designated with a .exe extension.

extension In ArcGIS, an optional software module that adds specialized tools and functionality to ArcGIS Desktop. ArcGIS Network Analyst, StreetMap, and ArcGIS Business Analyst are examples of ArcGIS extensions.

Federal Geographic Data Committee An organization established by the United States Federal Office of Management and Budget responsible for coordinating the development, use, sharing, and dissemination of surveying, mapping, and related spatial data. The committee is comprised of representatives from federal and state government agencies, academia, and the private sector. The FGDC defines spatial data metadata standards for the United States in its Content Standard for Digital Geospatial Metadata and manages the development of the National Spatial Data Infrastructure (NSDI).

FGDC See Federal Geographic Data Committee.

framework The existing ArcObjects components that comprise the ArcGIS system.

GDB See geodatabase.

GDI Graphical Device Interface. A standard for representing graphical objects and transmitting them to output devices, such as a monitor. GDI generally refers to the Windows GDI API.

GeocodeServer An ArcGIS Server software component that provides programmatic access to an address locator and performs single and batch address matching. It is designed for use in building Web services and Web applications using ArcGIS Server.

geodatabase An object-oriented data model introduced by ESRI that represents geographic features and attributes as objects and the relationships between objects but is hosted inside a relational database management system. A geodatabase can store objects, such as feature classes, feature datasets, nonspatial tables, and relationship classes.

geometry The measures and properties of points, lines, and surfaces. In a GIS, geometry is used to represent the spatial component of geographic features. An ArcGIS geometry class is one derived from the Geometry abstract class to represent a shape, such as a polygon or point.

geoprocessing tool An ArcGIS tool that can create or modify spatial data, including analysis functions (overlay, buffer, slope), data management functions (add field, copy, rename), or data conversion functions.

GIS server The components of ArcGIS Server that host and run server objects. A GIS server consists of a server object manager and one or more server object containers.

GUID Globally Unique Identifier. A string used to uniquely identify an interface, class, type library, or component category. See also class identifier.

hexadecimal A number system using base 16 notation.

HKCR HKEY_CLASSES_ROOT registry hive. A Windows registry root key that points to the HKEY_LOCAL_MACHINE\Software\Classes registry key. It displays essential information about OLE and association mappings to support drag-and-drop operations, Windows shortcuts, and core aspects of the Windows user interface.

HRESULT A 32-bit integer returned from any member of a COM interface indicating success or failure, often written in hexadecimal notation. An HRESULT can also give information about the error that occured when calling a member of a COM interface. Visual Basic translates HRESULTS into errors; Visual C++ developers work directly with HRESULT values.

IDE See integrated development environment.

IDispatch A generic COM interface that has methods allowing clients to ask which members are supported. Classes that implement IDispatch can be used for late binding and ID binding.

IDL See Interface Definition Language.

IID Interface Identifier. A string that provides the unique name of an interface. An IID is a type of Globally Unique Identifier. See also GUID.

impersonation A process by which a Web application assumes the identity of a particular user and thus gains all the privileges to which that user is entitled.

implement Regarding an interface, to provide code for each of the members of an interface (the interface is defined separately).

inbound interface An interface implemented by a class, on which a client can call members. See also outbound interface.

inheritance In object-oriented programming, the means to derive new classes or interfaces from existing classes or interfaces. New classes or interfaces contain all the methods and properties of another class or interface, plus additional methods and properties. Inheritance is one of the defining characteristics of an object-oriented system.

in-process Within the process space of a client application, a class contained in a DLL is in-process, as objects are loaded into the process space of the client EXE. A component contained in a separate EXE is out-of-process.

integrated development environment A software development tool for creating applications, such as desktop and Web applications. IDEs blend user interface design and layout tools with coding and debugging tools, which allows a developer to easily link functionality to user interface components.

Interface Definition Language A language used to define COM interfaces. The Microsoft implementation of IDL may be referred to as MIDL or Microsoft IDL.

IUnknown All COM interfaces inherit from the IUnknown interface, which controls object lifetime and provides runtime type support.

JavaServer Faces A framework for building user interfaces for Java Web applications. JSF is designed to ease the burden of writing and maintaining applications that run on a Java application server and render their user interfaces back to a target client.

JavaServer Pages A Java technology that enables rapid development of platform-independent Web-based applications. JSP separates the user interface from content generation, enabling designers to change the overall page layout without altering the underlying dynamic content.

JavaServer Pages Standard Tag Library A Java technology that encapsulates core functionality common to many Web-based applications as simple tags. JSTL includes tags for structural tasks such as iteration and conditionals, manipulation of XML documents, internationalization and locale-sensitive formatting, and SQL.

JSF See JavaServer Faces.

JSP See JavaServer Pages.

JSTL See JavaServer Pages Standard Tag Library.

late binding A technique that an application uses for determining data type at runtime using the IDispatch interface, rather than when the code is compiled. Late binding is generally used by scripting languages. See also early binding.

LIBID Library Identifier. A type of GUID consisting of a unique string assigned to a type library. See also GUID.

library In object-oriented programming, generic, platform independent term indicating a logical grouping of classes. ArcGIS is composed of approximately 50 libraries. Although the term library refers to a conceptual grouping of ArcGIS types, libraries do have multiple representations on disk: one per development environment. In COM, OLBs contain all the type information; in .NET, Assemblies contain the type information; and in Java, JAR files contain the type information.

license The grant to a party of the right to use a software package or component.

little endian A computer hardware architecture in which, within a multibyte numeric representation, the least significant byte has the lowest address and the remaining bytes are encoded in increasing order of significance. See also big endian.

macro A computer program, usually a text file, containing a sequence of commands that are executed as a single command. Macros are used to perform commonly used sequences of commands or complex operations.

map document In ArcMap, the file that contains one map; its layout; and its associated layers, tables, charts, and reports. Map documents can be printed or embedded in other documents. Map document files have a .mxd extension.

MapServer An ArcGIS Server software component that provides programmatic access to the contents of a map document on disk and creates images of the map contents based on user requests. It is designed for use in building map-based Web services and Web applications using ArcGIS Server.

marshaling The process that enables communication between a client object and server object in different apartments of the same process, between different processes, or between different processes on different machines by specifying how function calls and parameters are to be passed over these boundaries.

members Refers collectively to the properties and methods, or functions, of an interface or class.

memory leak When an application or component allocates a section of memory and does not free the memory when finished with it, it is said to have a memory leak; the memory cannot then be used by any other application.

MTA Multiple threaded apartment. An apartment that can have multiple threads running. A process can only have one MTA. See also apartment, STA, thread, TNA.

network 1. A set of edge, junction, and turn elements and the connectivity between them, also known as a logical network. In other words, an interconnected set of lines representing possible paths from one location to another. A city streets layer is an example of a network. 2. In computing, a group of computers that share software, data, and peripheral devices, as in a LAN or WAN.

object In object-oriented programming, an instance of a class.

Object Definition Language Similar to Interface Definition Language but used to define the objects contained in an object library. See also Interface Definition Language, object library.

object library A binary file that stores information about a logical collection of COM objects and their properties and methods in a form that is accessible to other applications at runtime. Using a type library, an application or browser can determine which interfaces an object supports and invoke an object's interface methods.

object model diagram A graphical representation of the types in a library and their relationships.

object pooling The process of precreating a collection of instances of classes, such that the instances can be shared between multiple application sessions at the request level. Pooling objects allows the separation of potentially costly initialization and aquisition of resources from the actual work the object does. Pooled objects are used in a stateless manner.

object-oriented programming A programming model in which developers define the data type of a data structure as well as the functions, or types of operations, that can be applied to the data structure. Developers can also create relationships between objects. For example, objects can inherit characteristics from other objects.

OCX See OLE custom control.

ODL See Object Definition Language.

OGIS Open Geodata Interoperability Specification. A specification, developed by the Open GIS Consortium, to support interoperability of GIS systems in a heterogeneous computing environment.

OLB See object library.

OLE Object Linking and Embedding. A distributed object system and protocol from Microsoft that allows applications to exchange information. Applications using OLE can create compound documents that link to data in other applications. The data can be edited from the document without switching between applications. Based on the Component Object Model, OLE allows the development of reusable objects that are interoperable across multiple applications.

OLE Custom Control Also known as an ActiveX Control, an OLE custom control is contained in a file with the extension .ocx. The ArcGIS controls are ActiveX Controls.

OleView A utility, available as part of Microsoft Visual Studio, that can be used to view type information stored in a type library or object library or inside a DLL.

out-of-process Within the process space of a client application, a component contained in an EXE is out-of-process; instantiated classes are loaded into the process space of the EXE in which they are defined rather than into that of the client. See also in-process.

outbound interface An interface implemented by a class, on which that object can make calls to its clients; analogous to a callback mechanism. See also inbound interface.

PDF Portable Document Format. A proprietary file format from Adobe that creates lightweight text-based, formatted files for distribution to a variety of operating systems.

performance A measure of the speed at which a computer system works. Factors affecting performance include availability, throughput, and response time.

persistence The process by which information indicating the current state of an object is written to a storage medium such as a file on disk. In ArcObjects, persistence is achieved via the standard COM interfaces IPersist and IPersistStream or the ArcObjects interface IPersistVariant.

pixel type See data type.

platform A generic term often referring to the operating system of a machine. May also refer to a programming language or development environment, such as COM, .NET, or Java.

plug-in data source An additional read-only data source provided by either ESRI or a third party developer. It may be a data source forming part of the core ArcObjects or an extension.

PMF See Published Map File.

ProgID A string value, stored in the system registry, identifying a class by library and class name, for example, *esriCarto.FeatureLayer*. The ProgID registry key also contains the human-readable name of a class, the current version number of the class, and a unique class identifier. ProgIDs are used in VB object instantiation. See also class identifier, IID.

property page A user interface component that provides access to change the properties of an object or objects.

proxy object A local representaion of a remote object, supporting the same interfaces as the remote object. All interaction with the remote object from the local process is forced via the proxy object. A local object makes calls on the members of a proxy object as if it were working directly with the remote object.

Published Map File A file exported by the Publisher extension that can be read by ArcReader. Publisher Map Files end with a .pmf extension.

query interface A client may request a reference to a different interface on an object by calling the QueryInterface method of the IUnknown interface.

raster A spatial data model that defines space as an array of equally sized cells arranged in rows and columns. Each cell contains an attribute value and location coordinates. Unlike a vector structure, which stores coordinates explicitly, raster coordinates are contained in the ordering of the matrix. Groups of cells that share the same value represent geographic features. See also vector.

recycling The process by which objects in an object pool are replaced by new instances of objects. Recycling allows for objects that have become unusable to be destroyed and replaced with fresh server objects and to reclaim resources taken up by stale server objects.

reference A pointer to an object, interface, or other item allocated in memory. COM objects keep a running total of the references to themselves via the IUnknown interface methods AddRef and Release.

Regedit A utility, part of the Windows operating system, that allows you to view and edit the system registry.

register To add information about a component to the system registry, generally performed using RegSvr32.

registry Stores information about system configuration for a Windows machine. COM uses the registry extensively, storing details of COM components including ProgIDs and ClassIDs, file location of the binary code, marshaling information, and categories in which they participate.

registry file A file containing information in Windows Registry format. Double clicking a .reg file in Windows will enter the information in the file to the system registry. Often used to register components to component categories.

RegSvr32 A Windows utility that can add information about a component to the system registry. A component must be registered before it can be used.

rehydrate To reinstantiate an object and its state from persisted storage.

render To draw to a display. The conversion of the geometry, coloring, texturing, lighting, and other characteristics of an object into a display image.

runtime environment The host that provides the services required for compiled code to execute. The Service Control Manager is effectively the runtime environment for COM. The Visual Basic Virtual Machine (VBVM) is the runtime environment that runs Visual Basic code.

scalable A system that does not show negative effects when its size or complexity grows greater.

SCM Service Control Manager. An administrative tool that enables the creation and modification of system services. It effectively serves as the runtime environment for COM.

script A set of instructions in plain text, usually stored in a file and interpreted, or compiled, at runtime. In geoprocessing, scripts can be used to automate tasks, such as data conversion, or generate geodatabases and can be run from their scripting application or added to a toolbox. Geoprocessing scripts can be written in any COM-compliant scripting language, such as Python, JScript, or VBScript.

serialization A form of persistence, in which an object is written out in sequence to a target, usually a stream. See also persistence.

server 1. A computer in a network that is used to provide services, such as access to files or e-mail routing, to other computers in the network. Servers may also be used to host Web sites or applications that can be accessed remotely. 2. An item that provides functionality to a client—for example, a COM component or object to a user application using components or to a database client utility using a database on a server machine.

server account The operating system account that the server object manager service runs as. The server account is specified by the GIS server post installation utility.

server context A space on the GIS server where a server object and its associated objects are running. A server context runs within a server container process. A developer gets a reference to a server object through the server object's server context and can create other objects within a server object's context.

server directory A location on a file system used by a GIS server for temporary files that are cleaned up by the GIS server.

server object A coarse-grained ArcObjects component that manages and serves a GIS resource, such as a map or a locator. A server object is a high-level object that simplifies the programming model for doing certain operations and hides the fine-grained ArcObjects that do the work. Server objects also have SOAP interfaces, which makes it possible to expose server objects as Web services that can be consumed by clients across the Internet.

server object isolation Describes whether server objects share processes with other server objects. Server objects with high isolation run dedicated processes, whereas server objects with low isolation share processes with other server objects of the same type.

server object type Defines what a server object's initialization parameters are and what methods and properties it exposes to developers. At ArcGIS version 9.0, there are two server object types: MapServer and GeocodeServer.

session state The process by which a Web application maintains information across a sequence of requests by the same client to the same Web application.

shallowly stateful application An application that uses the session state management capabilities of a Web server to maintain application state and makes stateless use of server objects in the GIS server. Shallowly stateful applications can use pooled server objects.

singleton A class for which there can only be one instance in any process.

smart pointer A Visual C++ class implementation that encapsulates an interface pointer, providing operators and functions that can make working with the underlying type easier and less error prone.

SOAP Simple Object Access Protocol. An XML-based protocol developed by Microsoft/Lotus/IBM for exchanging information between peers in a decentralized, distributed environment. SOAP allows programs on different computers to communicate independently of an operating system or platform by using the World Wide Web's HTTP and XML as the basis of information exchange. SOAP is now a W3C specification. See also XML, World Wide Web Consortium.

SOC Server object container. A process in which one or more server objects is running. SOC processes are started and shut down by the SOM. The SOC processes run on the GIS server's container machines. Each container machine is capable of hosting multiple SOC processes. See also SOM.

SOM Server object manager. A Windows service that manages the set of server objects that are distributed across one or more server object container machines. When an application makes a connection to an ArcGIS Server over a LAN, it is making a connection to the SOM. See also SOC.

SQL See Structured Query Language.

STA Single threaded apartment. An apartment that only has a single thread. User interface code is usually placed in an STA. See also apartment, MTA, thread, TNA.

standalone application An application that runs by itself, not within an ArcGIS application.

state The current data contained by an object.

stateful operation An operation that makes changes to an object or one of its associated objects—for example, removing a layer from a map. See also stateless operation.

stateless An object that stores no state data in between member calls.

stateless operation An operation that does not make changes to an object—for example, drawing a map. See also stateful operation.

stream A mode of data delivery in which objects provide data storage. Stream objects can contain any type of data in any internal structure. See also persistence.

Structured Query Language A syntax for defining and manipulating data from a relational database. Developed by IBM in the 1970s, SQL has become an industry standard for query languages in most relational database management systems.

SXD Scene Document. A document saved by ArcScene™ that has the extension .sxd.

synchronization The process of automatically updating certain elements of a metadata file.

target computer A computer to which an application is deployed.

thread A process flow through an application. An application can have many threads. See also apartment, MTA, STA, TNA.

TNA Thread neutral apartment. An apartment that has no threads permanently associated with it; threads enter and leave the apartment as required. See also apartment, MTA, STA, thread.

tool A command that requires interaction with the user interface before an action is performed. For example, with the Zoom In tool, you must click or draw a box over the geographic data or map before it is redrawn at a larger scale. Tools can be added to any toolbar.

type inheritance A kind of inheritance in which an interface may inherit from a parent interface. A client may call the child interface as if it were the parent, as all the same members are supported.

type library A collection of information about classes, interfaces, enumerations, and so on, that is provided to a compiler for inclusion in a component. Type libraries are also used to allow features such as IntelliSense to function correctly. Type libraries usually have the extension .tlb.

UI User interface. The portion of a computer's hardware and software that facilitates human interaction. The UI includes items that can be displayed on screen, and interacted with by using the keyboard, mouse, video, printer, and data capture.

UML Unified Modeling Language. A graphical language for object modeling. See also CASE.

URL Uniform Resource Locator. A standard format for the addresses of Web sites. A URL looks like this: www.esri.com. The first part of the address indicates what protocol to use, while the second part specifies the IP address or the domain name where the Web site is located.

usage time The amount of time between when a client gets a reference to a server object and when the client releases it.

utility COM object A COM object that encapsulates a large number of fine-grained ArcObjects method calls and exposes a single coarse-grained method call. Utility COM objects are installed on a GIS server and called by server applications to minimize the round trips between the client application and the GIS server. See also Component Object Model.

variant A data type that can contain any kind of data.

VB Visual Basic. A programming language developed by Microsoft based on an object-oriented form of the BASIC language and intended for application development. Visual Basic runs on Microsoft Windows platforms.

VBA Visual Basic for Applications. The embedded programming environment for automating, customizing, and extending ESRI applications, such as ArcMap and ArcCatalog. It offers the same tools as Visual Basic in the context of an existing application. A VBA program operates on objects that represent the application and can be used to create custom symbols, workspace extensions, commands, tools, dockable windows, and other objects that can be plugged in to the ArcGIS framework.

VBVM Visual Basic Virtual Machine. The runtime environment used by Visual Basic code when it runs.

vector 1. A coordinate-based data model that represents geographic features as points, lines, and polygons. Each point feature is represented as a single coordinate pair, while line and polygon features are represented as ordered lists of vertices. Attributes are associated with each feature, as opposed to a raster data model, which associates attributes with grid cells. 2. Any quantity that has both magnitude and direction. See also raster.

virtual directory A directory name, used as a URL, that corresponds to a physical directory on a Web server.

Visual C++ A Microsoft implementation of the C++ language, which is used in the Microsoft application Visual Studio, producing software that can be used on Windows machines.

W3C See World Wide Web Consortium.

wait time The amount of time it takes between a client requesting and receiving a server object.

Web application An application created and designed specifically to run over the Internet.

Web application template A file that contains a user interface as well as all the code and necessary files to use as a starting point for creating a new customized Web application. ArcGIS Server contains a number of Web application templates.

Web control The visual component of a Web form that executes its own action on the server. Web controls are designed specifically to work on Web forms and are similar in appearance to HTML elements.

Web form Based on ASP.NET technology, Web forms allow the creation of dynamic Web pages in a Web application. Web forms present their user interface to a client in a Web browser or other device but generally execute their actions on the server.

Web server A computer that manages Web documents, Web applications, and Web services and makes them available to the rest of the world.

Web service A software component accessible over the World Wide Web for use in other applications. Web services are built using industry standards such as XML and SOAP and thus are not dependent on any particular operating system or programming language, allowing access through a wide range of applications.

Web service catalog A collection of ArcGIS Server Web services. A Web service catalog is itself a Web service with a distinct endpoint (URL) and can be queried to obtain the list of Web services in the catalog and their URLs. See also ArcGIS Server Web service.

World Wide Web Consortium An organization that develops standards for the World Wide Web and promotes interoperability between Web technologies, such as browsers. Members from around the world contribute to standards for XML, XSL, HTML, and many other Web-based protocols.

WSDL Web Service Description Language. The standard format for describing the methods and types of a Web service, expressed in XML.

XMI See XML Metadata Interchange.

XML Extensible Markup Language. Developed by the World Wide Web Consortium, XML is a standard for designing text formats that facilitates the interchange of data between computer applications. XML is a set of rules for creating standard information formats using customized tags and sharing both the format and the data across applications.

XML Metadata Interchange A standard produced by the Object Management Group that specifies how to store a UML model in an XML file. ArcGIS can read models in XMI files.

XSL Extensible Style Language. A set of standards for defining XML document presentation and transformation. An XSL stylesheet may contain information about how to display tagged content in an XML document, such as font size, background color, and text alignment. An XSL stylesheet may also contain XSLT code that describes how to transform the tagged content in an XML document into an output document with another format. The World Wide Web Consortium maintains the XSL standards. See also XML, World Wide Web Consortium.

XSLT Extensible Style Language Transformations. A language for transforming the tagged content in an XML document into an output document with another format. An XSL stylesheet contains the XSLT code that defines each transformation to be applied. Transforming a document requires the original XML document, an XSL document containing XSLT code, and an XSLT parser to execute the transformations. The World Wide Web Consortium maintains the XSLT standard. See also XML, XSL, World Wide Web Consortium.